Design and Analysis of IoT Software Systems: Assignment 1

SOFE 4610U

Devante Wilson
100554361

Emin Avanessian-Zadeh
100563462

Ebrahim Merchant
100559443

Sikandar Shahbaz
100566271

Cherlyne Santhirarajan
100540234

Gaith Haddad
100553215

***Abstract*—This document describes the setup process for interacting with the Texas Instruments CC2650 SensorTag. It also describes how to run the application so that the process can be replicated.**

Keywords—setup, CC2650, SensorTag, interact

INTRODUCTION

The Texas Instruments CC2650 Sensortag allows quick and easy prototyping of IoT systems. Users can get started with Wi-Fi, Bluetooth low energy, Sub-1 GHz and IEEE 802.15.4 based protocols (e.g. 6LowPAN, ZigBee, etc.).

ThingsBoard is an example of an open source IoT platform that provides APIs to store and retrieve data from things using the HTTP protocol over the Internet or via a Local Area Network. The platform enabled the creation of our sensor logging application with data visualization.

In this assignment, our team used the Raspberry Pi, SensorTag, and the Thingsboard IoT platform to read, store and display sensor data (see figure below). The Raspberry Pi reads various sensor data from the SensorTag. We decided to store Light, Ambient or IR Temperature, Barometer values, and Humidity. The data was stored in a local database (using sqlite3 on RPi) before it is sent to the online platform for processing.

I. SETUP PROCESS

A. *Thingsboard Setup*

Create an account for Thingsboard to get started. Under the “devices” tab, create a new device for the demo. Copy the access token and paste into the python script - there will be a URL in the code used to perform the POST request.

Once the POST request is sent from the script later on, view the “latest telemetry”. Select the incoming data A button will be highlighted displaying “show on widget”. A new screen will be displayed showing a drop down menu of different widgets. For viewing all the data in a chart, the “chart” widget should be selected. You can now add this chart to the dashboard.

Under the “Dashboard” tab, the widget created from beforehand can be shown to view the data. Alternatively, the digital gauge widgets may also be used to display the incoming sensor data

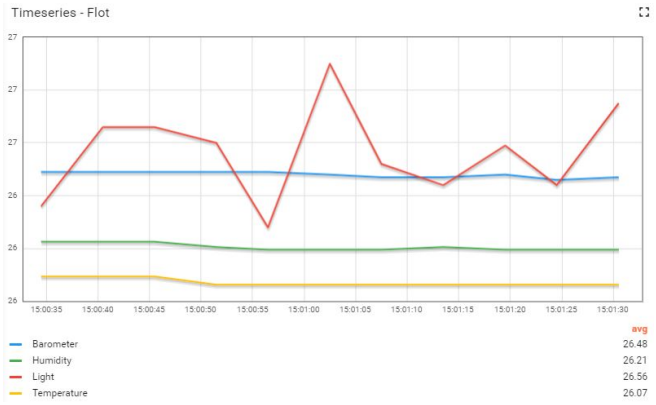


Fig. 1. All sensor data being displayed in a chart widget

B. *Connecting the SensorTag*

Press the left button on the SensorTag if the green LED is not blinking - this action means standby and searching for a connection.

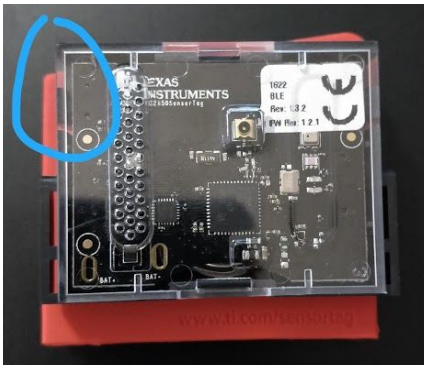


Fig.2. Showing location of button

C. *Running the script*

To run the script, open an IDE of your choice. Open the python script from the file location on the RPi.

Note: If you have a different SensorTag than that of our team, you will have to change the MAC address in the script.

Run the application. You should see that the blinking LED on the SensorTag stopped blinking - this

indicates that communication is taking place between the RPI and the SensorTag.

You may notice that the console displays the light value in Lux, this is simply to manually monitor the light value for the bonus portion of the assignment.

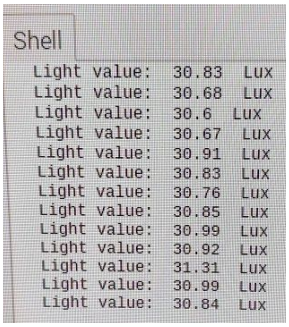


Fig.4. Monitoring the light value in the console

D. Viewing the Local Database

In the folder where the script is located, a local database file will be held; every time the sensor data is being read from the SensorTag, the data is being parsed and store inside the database for temporary storage.

After a certain period, the data will be pushed to Thingsboard via HTTP POST request.

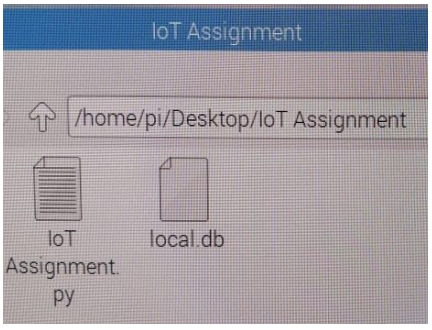
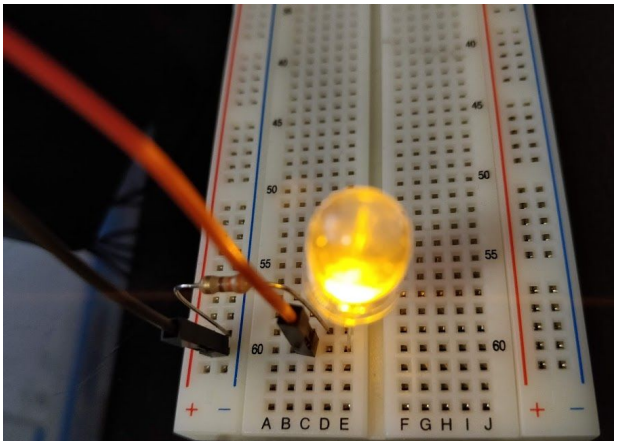


Fig.5. Viewing the local database and script

E. Bonus Assignment Task

To install the LED onto the breadboard, a 330 ohm resistor is required. Connect the male-female jumper wires onto the breadboard as shown in the following figure.



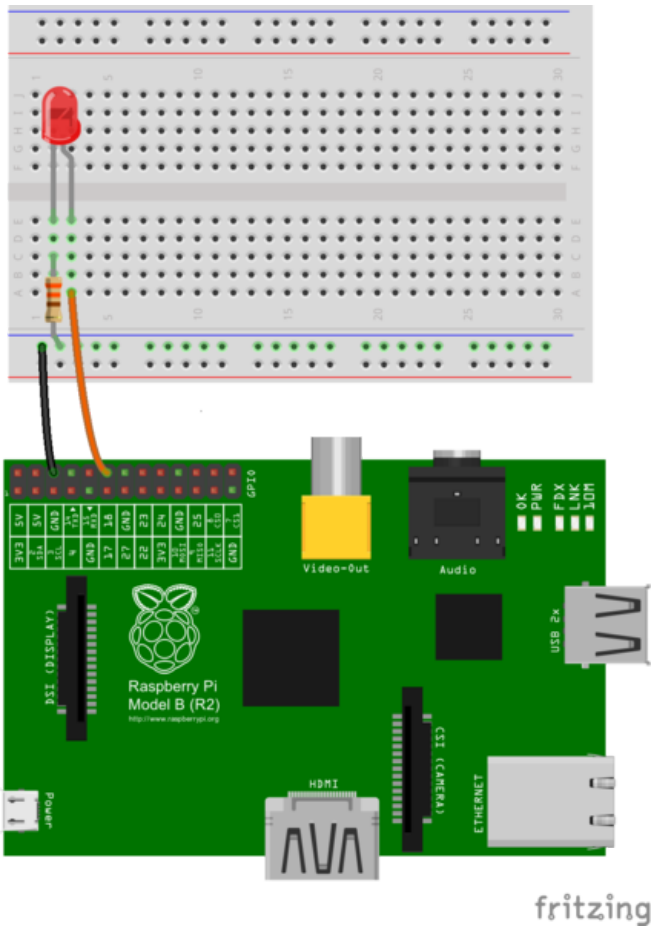


Fig.5. Wiring the electrical component for the LED

A jumper wire will lead to a GPIO pin of your choice (remember which pin number), and the other wire will lead to a ground pin on the Raspberry Pi.

Note: Depending on which GPIO pin the jumper wire was connected to, the script will have to be modified to reflect the same pin number.

In addition to the regular lab tasks explained previously, there was an additional bonus task assigned to us. We successfully accomplished this task inside of the same python script.

The instructed task was to toggle the LED once the sensor had detected a low ambient light (in Lux).

The function toggle() was created to set the GPIO pin high when the ambient light was below 15 Lux; then similarly set the GPIO pin to low when an ambient light value was detected above 15 Lux.

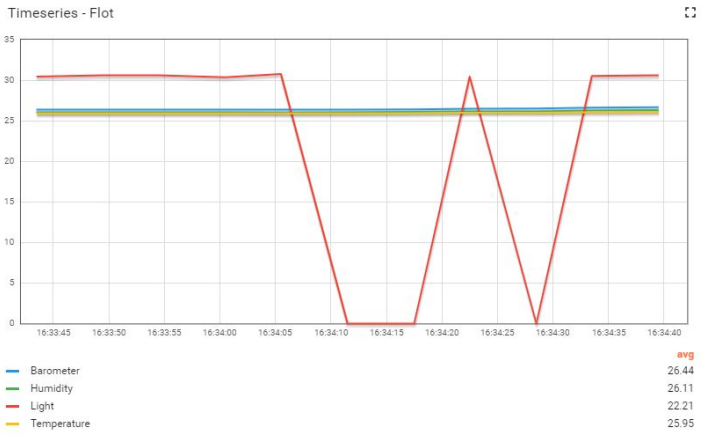


Fig.6. Monitoring the Light Value via Thingsboard

The previous figure shows some dips in the chart on Thingsboard; this represents one of our team members physically covering the SensorTag completely. Thus, the light value dipped below 15 Lux, indicating a dark environment and the LED subsequently illuminated.

F. Code Overview

The first step our team performed to implement the application was to gather the libraries we would require.

```
import sqlite3
import datetime
import requests
import RPi.GPIO as GPIO
import subprocess, sys
import time
import json
```

Fig.7. Monitoring the Light Value via Thingsboard

Next came the task to define a main section of the script. We defined the command we needed to connect to the Sensortag via its MAC address. We then piped the command to a subprocess that will perform it. We connect to the local database and also define the database table name.

```
# command to run
cmd = "sensortag -T -H -B -L 24:71:89:BB:FB:04"

# create a process to pipe a command to the command line
p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE)
# define dictionary to hold values from command
dict_values = {}
# connect to local database
conn = sqlite3.connect("local.db")
# define database table name
table_name = "Sensor_Data"
count = 0
```

Fig.8. Connecting to the SensorTag via MAC address

```
# get values and connect to the local database
try:
    while True:
        # read the contents of the command line
        out = p.stdout.readline()
        line = out.decode("utf-8")

        # save values in a dictionary
        if "Temp" in line:
            dict_values["Temperature"] = getTemp(line)
        elif "Humidity" in line:
            dict_values["Humidity"] = getHumidity(line)
        elif "Barometer" in line:
            dict_values["Barometer"] = getBarometer(line)
        elif "Light" in line:
            dict_values["Light"] = getLight(line)
            dict_values["timestamp"] = str(datetime.datetime.now())
            count += 1
            send_data_live(dict_values)
            # add dictionary to local DB
            #add_data_local_db(conn, dict_values, table_name)
        # if count == 2:
        #     count = 0
        #     send_data_thingsboard(conn)
```

We then had to read the contents of the command line to and then connect and send the data to the local database.

The code below shows the functions used to parse each piece of data from each sensor type. The code following this simply sends the parsed data (inside of the dictionary list), in JSON format via a HTTP request

```
# receive temperature (ambient: values[0], IR temp: values[1])
def getTemp(line):
    index = line.find("(")
    line = line[index+1:len(line)-2]
    values = line.split(", ")
    value = float(values[0])
    print ("Temp value: ", value)
    if value > 25:
        toggle(True)
    else:
        toggle(False)
    return values[0]

# parse humidity data
def getHumidity(line):
    index = line.find("(")
    line = line[index+1:len(line)-2]
    values = line.split(", ")
    return values[0]

# parse light data
def getLight(line):
    line = line.replace("Light:  ", "")
    return line

# parse barometer data
def getBarometer(line):
    index = line.find("(")
    line = line[index+1:len(line)-2]
```

```
# send data via HTTP POST request
def send_data_thingsboard(conn):
    URL = 'http://demo.thingsboard.io/api/v1/T7CVaWF4kbbMtPHJGGjf/telemetry'
    cur = conn.cursor()
    query = "SELECT * FROM Sensor_Data"
    cur.execute(query)
    list_dict = []
    result = cur.fetchall()
    for array in result:
        dict = {}
        dict["Temperature"] = array[0]
        dict["Light"] = array[1]
        dict["Barometer"]= array[2]
        dict["timestamp"]= array[3]
        dict["Humidity"] = array[4]
        list_dict.append(dict)

    r = requests.post(URL, data = json.dumps(list_dict))
    if r.status_code == 200:
        query = "DELETE FROM Sensor_Data"
        cur.execute(query)
        return True
```

When the time came to do the bonus task, we developed a function to toggle the LED on/off. But first, we had to set the GPIO mode and setup the proper GPIO pin for the LED.

```
# main function
if __name__ == '__main__':
    # define GPIO setup
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(4, GPIO.OUT)
```

```
# toggle LED
def toggle(state):
    if state:
        GPIO.output(4, GPIO.HIGH)
    else:
        GPIO.output(4, GPIO.LOW)
```

Fig. 9 & 10 Toggling the state of the LED

Conclusion

By the end of the assignment tasks, our team successfully learned how to interact with the SensorTag to build a simple IoT system. We gained hands on experience with wiring the LED, RPi, and breadboard together. Furthermore, we learned how to interface to the SensorTag to continuously read the particular sensor data. The raw data was parsed and fed into a local database before being sent by HTTP POST request for use with data visualization with digital gauges and charts via the Thingsboard IoT platform.

References

[1] https://demo.thingsboard.io/devices