Assessment Cover Sheet

This Assessment Cover Sheet is only to be attached to hard copy submission of assessments.

ASSESSMENT DETAILS

Unit title	IoT Programming	Tutorial /Lab Group	1	Office use only
Unit code	SWE30011	Due date	4 June 2021	
Name of lecturer/tutor	Dr. Mark Tee Kit Tsun			
Assignment title	Assessment 3: Group Assignment			Faculty or school date stamp

STUDENT(S) DETAILS

	Student Name(s)	Student ID Number(s)
(1)	Wong Jun Jie	101215892
(2)	Karyn Chong Huei Xien	101215944
(3)	Jasmin Chu Ze Kee	101215779
(4)	Chan Kwang Yung	101215067
(5)		
(6)		

DECLARATION AND STATEMENT OF AUTHORSHIP

1. I/we have not impersonated, or allowed myself/ourselves to be impersonated by any person for the purposes of this assessment.
2. This assessment is my/our original work and no part of it has been copied from any other source except where due acknowledgement is made.
3. No part of this assessment has been written for me/us by any other person except where such collaboration has been authorised by the lecturer/tutor concerned.
4. I/we have not previously submitted this work for this or any other course/unit.
5. I/we give permission for my/our assessment response to be reproduced, communicated, compared and archived for plagiarism detection, benchmarking or educational purposes.

I/we understand that:

6. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to exclusion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.

Student signature/s

I/we declare that I/we have read and understood the declaration and statement of authorship.

(1)		(4)	
(2)		(5)	
(3)	JASMIN	(6)	

Further information relating to the penalties for plagiarism, which range from a formal caution to expulsion from the University is contained on the Current Students website at **https://www.swinburne.edu.my/current-students/manage-course/exams-results-assessment**

Copies of this form can be downloaded from the Student Forms web page at **https://www.swinburne.edu.my/current-students/manage-course/exams-results-assessment/how-to-submit-work.php**

Table of Contents

Table of Contents **2**

Presentation Video Link **4**

Google Drive Link and Details **4**

1.0 Introduction **5**

1.1 Background 5

1.2 System Proposed 5

2.0 Conceptual Design **7**

2.1 Software Block Diagram 7

2.2 Hardware Block Diagram 7

2.3 Complete Diagram 8

3.0 Task Breakdown **9**

4.0 Implementation **11**

4.1 Smart Lighting System 11

4.2.1 System Images 11

4.2.2 System Script 12

4.2 Auto Radiator 15

4.2.1 System Images 15

4.2.2 System Scripts 17

4.3 Smart Trash Can 21

4.3.1 System Images 21

4.3.2 System Scripts 23

4.4 Auto Plant Watering 31

4.4.1 System Images 31

4.4.2 System Scripts 32

4.5 Edge servers 35

4.6 Communication protocols 35

5.0 Website Details **38**

5.1 Smart Lighting System Dashboard 39

5.2 Auto Radiator Dashboard 40

5.3 Smart Trash Can Dashboard 41

5.4 Auto Watering System Dashboard 42

6.0 Cloud Computing: Beebotte **43**

7.0 User Manual **45**

7.1 Smart Lighting System 45
7.2 Auto Radiator System 46
7.3 Smart Trash Can System 48
7.4 Auto Watering System 50

8.0 Limitations 52

9.0 Resources 53
9.1 Software 53
9.2 Online Tutorials and Guides 53

10.0 Appendix 55
10.1 Smart Lighting System 55
10.1.1 a3.ino 55
10.1.2 a3.py 57
10.2 Auto Radiator 59
10.2.1 Assignment3.ino 59
10.2.2 Assignment3.py 62
10.3 Smart Trash Can 65
10.3.1 bin_arduino_Assignment3.ino 65
10.3.2 binLog.py 69
10.4 Auto Plant Watering 72
10.4.1 groupTo.ino 72
10.4.2 main_database.py 74

Presentation Video Link

VIDEO LINK: https://www.youtube.com/watch?v=Gzr9cRiL5Ag

Google Drive Link and Details

LINK:
https://drive.google.com/drive/folders/1OnK9uFWVNY-qmPXXTRqdTHPLq41g-eY_?usp=sharing

EdgeDevice_1 (Smart_Lighting_System)

- a3.ino
- a3.py

EdgeDevice_2 (Auto_Radiator)

- Assignment3.ino
- Assignment3.py

EdgeDevice_3 (Smart_Trash_Can)

- bin_arduino_Assignment3.ino
- binLog.py

EdgeDevice_4 (Auto_Plant_Watering)

- groupTo.ino
- main_database.py

1.0 Introduction

1.1 Background

A home hub is a compilation of IoT devices linked together to form a smart collection of components in a house. For example, the door, air conditioning, lighting, and thermostat can be linked together, and each uses its own sensors and actuators to perform tasks smartly and be controlled through a shared hub. All these devices will work individually, collect their own data, and perform their own tasks while simultaneously uploading their data into the cloud. For example, the air conditioning will turn off if it gets too hot, and the lights will turn on when it gets too dark, with both these events being independent of each other. In this group project, the team will develop a home hub containing multiple IoT nodes to automate and simplify daily mundane tasks.

1.2 System Proposed

The system proposed is a home hub system that consists of 4 components, a Smart Trash Can component, a Smart Lighting System, an Auto Watering Unit for houseplants, and an Automatic Radiator. The components will all be made using the Arduino Uno microcontroller, with Raspberry Pi microcomputer as an edge device, with the data collected being passed into a cloud cloud, which is (Chosen platform). The details of the modules are as below:

<u>The Smart Trash</u>
Sensors:
-Motion Sensor to Open/Close if motion is detected
-Distance Sensor to check how much trash is filled
-RFID scanner
Actuator:
-Servo Motor to mock opening and closing
-LED to indicate if the trash is full or not

<u>Smart Lighting System</u>
Sensors:
-LM35 Temperature Sensor
-LDR Light Sensor
Actuator:
-LED to mock lighting

<u>Auto Plant Watering</u>
Sensors:
-DHT11 Temperature-Humidity sensor
-LDR
Actuators:
-Servo Motor to mock watering plants

Auto Radiator

Sensors:

-Sound-sensor to turn on and off via clap sounds

-Temperature sensor to automate on and off

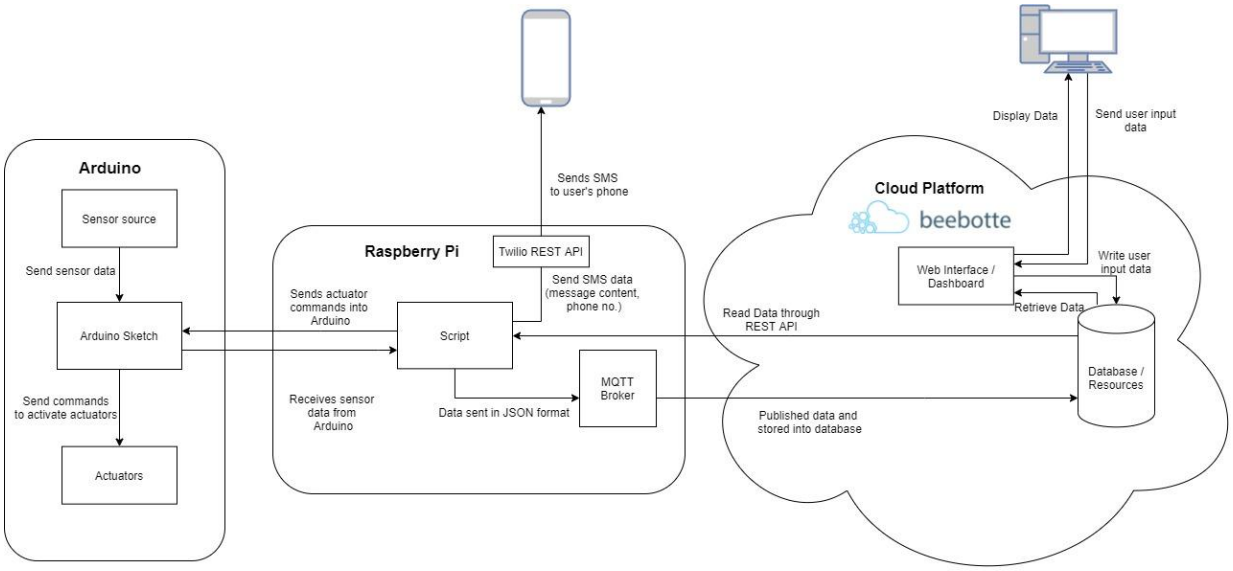
Actuators:

-Relay

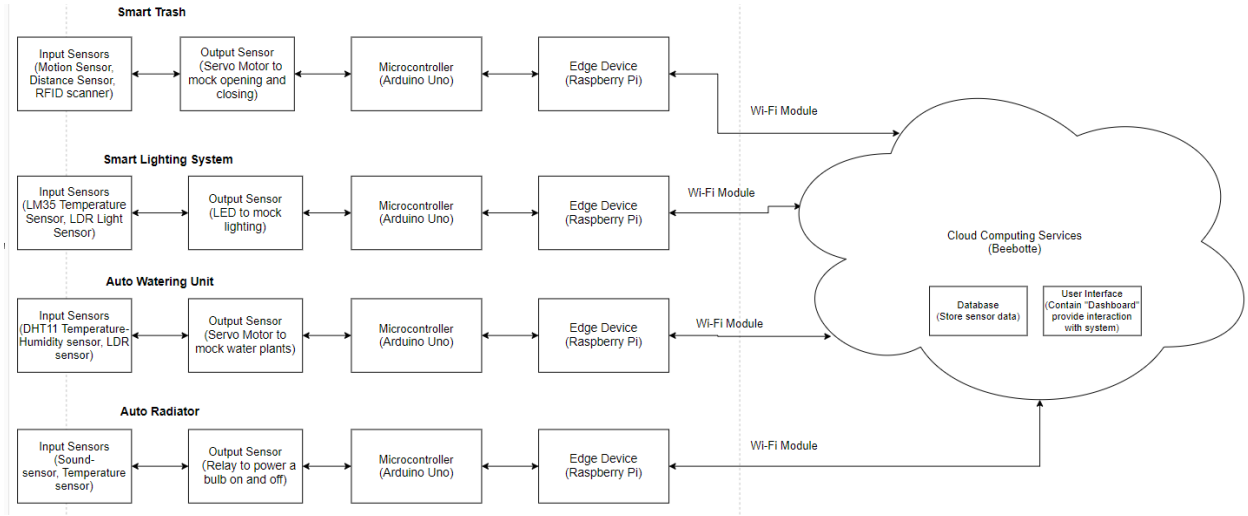
-Bulb

2.0 Conceptual Design

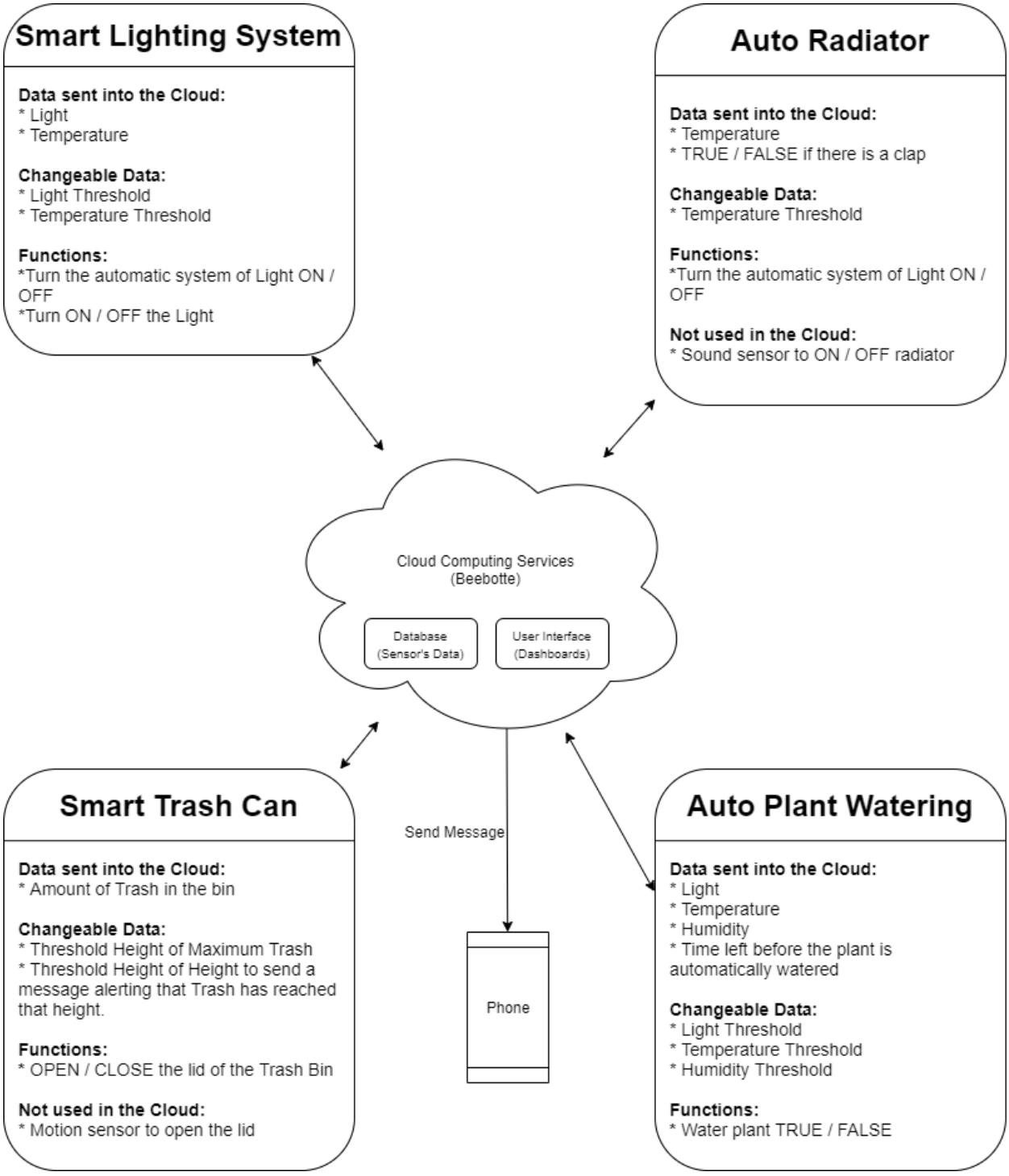
2.1 Software Block Diagram



2.2 Hardware Block Diagram



2.3 Complete Diagram



3.0 Task Breakdown

Team Member	Work Done
Wong Jun Jie (101215892)	● Researched and tutored the implementation of Beebotte Cloud Service. ● Researched and tutored the implementation of Twilio SMS RESTful API ● Wrote the Background and System Proposed. ● Contributed the entirety of 4.2 (Auto Radiator) ● Contributed the entirety of 5.0 Website Details ● Contributed to the entirety of 6.0 Cloud Computing ● Contributed to the “Delay use in Python Script” in the 8.0 Limitation Section ● Contribution in 9.0 Resources (mainly Beebotte and Twilio SMS). ● Full implementation of the Auto Radiator system (scripts and hardware)
Jasmin Chu Ze Kee (101215779)	● Create Google Drive link and details ● Create hardware block diagram of the system ● Explanation of the entire section 4.1(Smart Lighting System) ● Explanation of the entire section 4.5 Edge Servers ● Explanation of the entire section 4.6 Communication Protocols ● Contribution in 8.0 Limitations ● Contribution in 9.0 Resources ● Implementation of Smart Lighting System (arduino, script, cloud service)
Chan Kwang Yung (101215067)	● Created software block diagram of the system ● Written implementation of Smart Trash Can System in 4.3 ● Written all of 7.0 User Manual

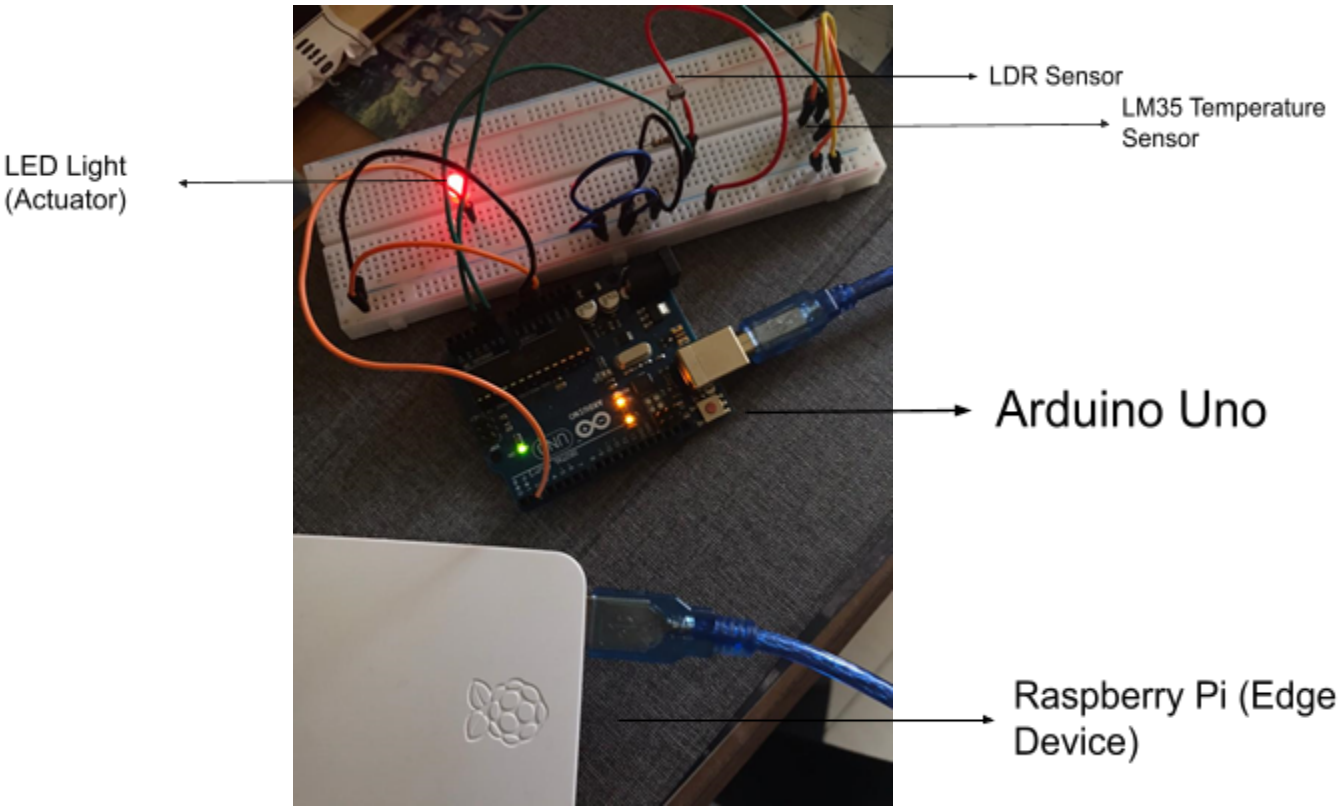
	● Contribution in 8.0 Limitations ● Contribution in 9.0 Resources ● Implemented the Smart Trash Can System (arduino + script + cloud service)
Karyn Chong Huei Xien (101215944)	● Created the complete diagram of the system ● Wrote about the implementation of the Auto Plant Watering system in 4.4 ● Contributed by making the Complete Diagram ● Contributed to the 9.0 Resources ● Editing the video for the team members showing their systems. ● Implementation of the Auto Plant Watering system (both script and code)

4.0 Implementation

4.1 Smart Lighting System

With the Smart Lighting System, you can turn the lights on and off with the touch of a button. The user can also turn on the automation function which allows the user to input a threshold to trigger the lighting system automatically. Temperature sensor (LM35) and light intensity sensor (LDR) will act as the sensors and LED light will be the main actuator for the whole project. When the detected surrounding temperature or light intensity is lower than the threshold set by the user, the sensors will trigger the LED light (switch on) in this system.

4.2.1 System Images



4.2.2 System Script

The first script being explained is the Arduino sketch, which is named “a3.ino”.

a3

```
const int ledPin = 2;
const int ldrPin = A0;
const int tmpPin = A1;
float temp;
unsigned long previousTime = 0;
const unsigned long eventInterval = 10000;
char buffer[30];

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(ldrPin, INPUT);
  pinMode(tmpPin, INPUT);
}
```

The setup is the first section of the script. Constant variables are also defined in this section. The setup function is then filled up with the serial start and pin mode configuration for each pin.

```
void loop() {
  unsigned long currentTime = millis();
  if (currentTime-previousTime >= eventInterval){
    int lightData = analogRead(A0);
    int tempData = analogRead(A1);
    tempData = -(tempData * 110)/1023;
    //Serial.println(lightData);
    //Serial.println(tempData);

    sprintf(buffer,"IOT Reading (ldr,tmp): %04d,%04d",lightData,tempData);
    Serial.println(buffer);
    previousTime=currentTime;
  }
```

Data collecting is in the second session within the loop function. When the timer reaches the eventInterval, which is set to 10000 seconds in the example above, it reads the temperature from the LM35 and LDR sensors, sends the current system status and temperature to the Raspberry Pi over serial, and then resets the timer. At this point, the temperature has been computed and converted to degrees celsius before sending to the edge side.

```
int ldrStatus = analogRead(A0);
int tmpStatus = analogRead(A1);

  if (Serial.available()>0){
   int lightStatus = Serial.parseInt();
   switch (lightStatus)
   {
    case 1:
      digitalWrite(2, HIGH);
      break;
    case 2:
      digitalWrite(2, LOW);
      break;
    default:
       break;
      }
   }
```

The third section waits for a serial input and turns on the LED light when the “1” integer is detected and turns it off when the “2” integer is detected.

The second script being discussed is the Python Script, a3.py, and is as below:

a3.py

```
import serial
import time
import paho.mqtt.client as mqtt
import json

from twilio.rest import Client
from beebotte import *


# PERSONAL TESTING api key and secret key
# API_KEY_AUTOLIGHT = 'gTYdN2z6boA2abUcyBjwodf1'
#SECRET_KEY_AUTOLIGHT = 'DvkUcfpV3fikrkowpjJaaVubOrQpXjor'

# GROUP api key and secret key
API_KEY_AUTOLIGHT = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY_AUTOLIGHT = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

#Twilio SMS REST Information
account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

# Setting the resources
bbt = BBT(API_KEY_AUTOLIGHT, SECRET_KEY_AUTOLIGHT)
light_resourse = Resource(bbt, 'Smart_Lighting_System', 'light_intensity_value')
light_button = Resource(bbt, 'Smart_Lighting_System', 'light_onoff')
light_input = Resource(bbt, 'Smart_Lighting_System', 'light_input')
light_auto_onoff = Resource(bbt, 'Smart_Lighting_System', 'light_auto_onoff')
temp_resourse = Resource(bbt, 'Smart_Lighting_System', 'temp_value')
temp_input = Resource(bbt, 'Smart_Lighting_System', 'temp_input')
ser = serial.Serial('/dev/ttyACM0', 9600)
```

Importing the necessary libraries, such as serial for serial data reading, time for delays, paho for mqtt communication, and JSON for data processing, is the initial step of the script. The Twilio REST library, which is used with the Client, is also included, as is Beebotte's API. The API keys and Secret Keys for Beebotte, as well as Twilio's SMS id and token, are defined next. The API and secret keys are then used to construct a Beebotte object. Finally, resources created in Beebotte are defined here so that data can be written into Beebotte in the future.

```
def on_connect(client, data, flags, rc):
    client.subscribe("Smart_Lighting_System/light_intensity_value")
    client.subscribe("Smart_Lighting_System/light_onoff")
    client.subscribe("Smart_Lighting_System/light_input")
    client.subscribe("Smart_Lighting_System/light_auto_onoff")

    client.subscribe("Smart_Lighting_System/temp_value")
    client.subscribe("Smart_Lighting_System/temp_input")
```

MQTT function on_connect. Allows the client to subscribe to the channels that are relevant to them. The channels are set according to Beebotte's "channel/resource" format.

```
def on_message(client, data, msg):

    count=0

    print(msg.topic + " " + str(msg.payload))
    read_light_auto_button_onoff = light_auto_onoff.read(limit=1)
    if (read_light_auto_button_onoff[0]['data']==False):
        read_light_button_onoff = light_button.read(limit=1)
        if(read_light_button_onoff[0]['data'] == True):
            ser.write(b"1")
            print("Manual On")
        elif(read_light_button_onoff[0]['data'] == False):
            ser.write(b"2")
            print("Manual Off")
        else:
            print("No Match")
        time.sleep(5)
```

MQTT function on_message. It first uses the Beebotte API to read first data from the resource "light_auto_onoff" and then outputs the message in the console to view current status. If the device is not in automatic mode (which means it is manual mode), it will actively check the status from Beebotte. If the manual button is True then the LED will be switched on, if False then the LED will be off.

```
elif(read_light_auto_button_onoff[0]['data']==True):
    read_light_input = light_input.read(limit=1)
    read_light_resourse = light_resourse.read(limit=1)

    read_temp_input = temp_input.read(limit=1)
    read_temp_resourse = temp_resourse.read(limit=1)

    if(read_light_resourse[0]['data'] <= read_light_input[0]['data'] and read_temp_resour
        ser.write(b"1")

        count+=1
        if(count==1):
            # SMS script
            smsclient = Client(account_sid, auth_token)
            mailbody = ("We detect the light is on")
            message = smsclient.messages.create(body=mailbody, from_ = "+12018905970",
            to = "+601110679126")

        print("Auto On")


    elif(read_light_resourse[0]['data'] >read_light_input[0]['data'] and read_temp_resour
        ser.write(b"2")
        count=0
        print("Auto Off")
    else:
        print("No Match")
    time.sleep(5)
else:
    print(read_light_auto_button_onoff[0]['data'])
```

If the device is in automated mode, it will actively receive data from the LM35 and LDR stored inside Beebotte from the sensor and compare it to the user-inputted threshold value. If it's lower, it'll send a serial byte to the Arduino to turn the LED on and vice versa. After that, it will create a Twilio Client and send a message to the user to inform them that there are changes from the system.The SMS is sent using Twilio's REST API.

```
# Setup MQTT Client
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# PERSONAL TESTING token
#client.username_pw_set("token:token_WbenCRZsqINb08va")

# GROUP token
client.username_pw_set("token:token_CmG3yJSa0ZaiaPKs")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()
```

Setting up the MQTT client to send messages to Beebotte. The token for the Smart Lighting System's channel is set automatically, and it connects to Beebotte's mqtt site via the 1883 port. The loop then begins.

```
while True:
    if ser.in_waiting > 0:

        ln = ser.readline()
        #light=int(ln);
        #temp=int(ln);

        # Reading data
        light=int(ln[23:27])
        temp=int(ln[28:32])
        #print(light)
        #print(temp)

        light_dictionary={"data":light,"write":True}
        temp_dictionary={"data":temp,"write":True}

        light_json = json.dumps(light_dictionary)
        temp_json = json.dumps(temp_dictionary)

        client.publish("Smart_Lighting_System/light_intensity_value", light_json)
        client.publish("Smart_Lighting_System/temp_value", temp_json)
```

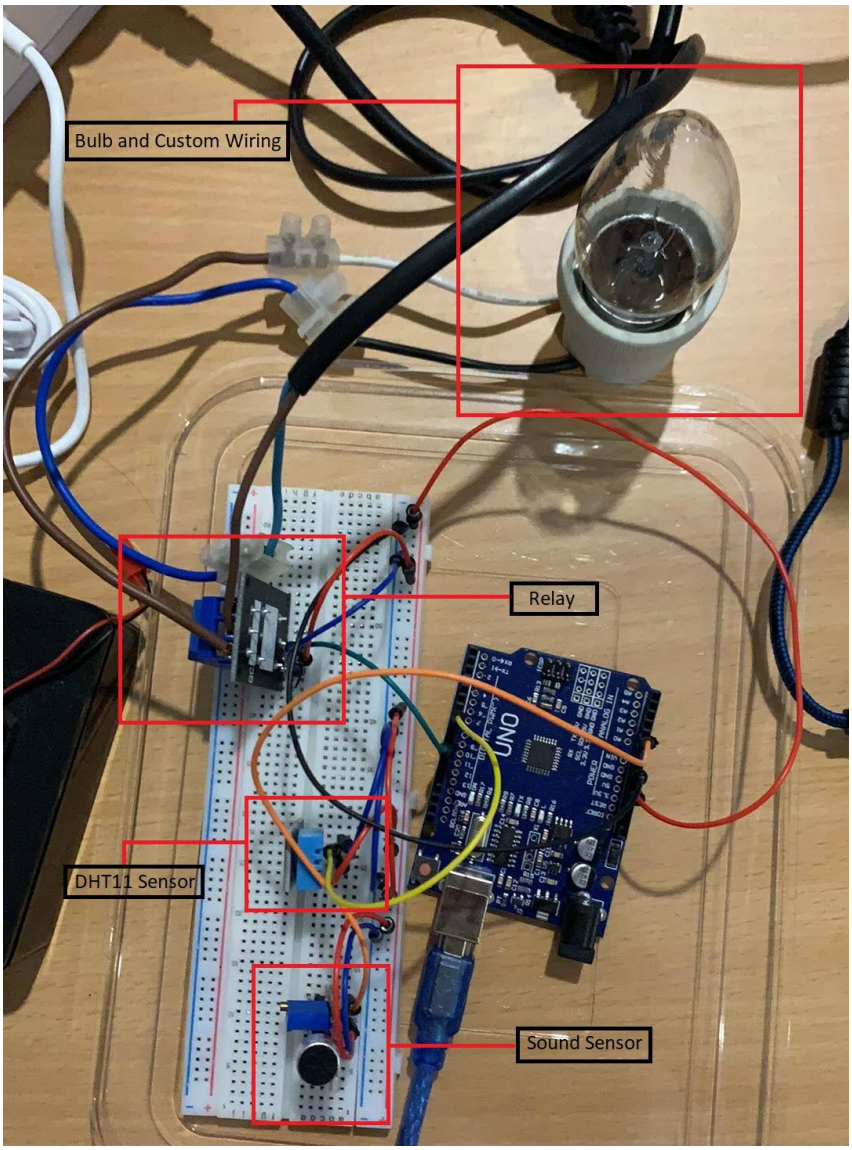
Because Beebotte only accepts JSON data when using MQTT as a bridge, it will read the data from the LDR and LM35 from the Beebotte and save it in a dictionary in JSON format. MQTT is then used to publish the data to the appropriate channels.

4.2 Auto Radiator

4.2.1 System Images

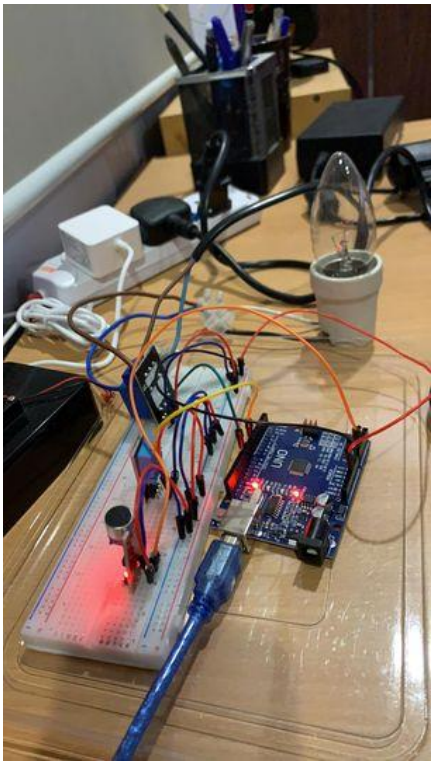
The automatic radiator is a radiator hooked up to a few sensors that allows for automation. First up is the sound sensor, which listens for a loud clap. Once a spike in volume is detected, the radiator will turn on. Another function for the auto radiator is its automatic mode, which takes in the temperature around the room, and turns the radiator on or off based on the user set threshold.

For the prototype, the components it contains are the KY-037 Sound Sensor, the DHT11 humidity and temperature sensor, a relay which is connected to a bulb to simulate the radiator. Below is a picture of the system:

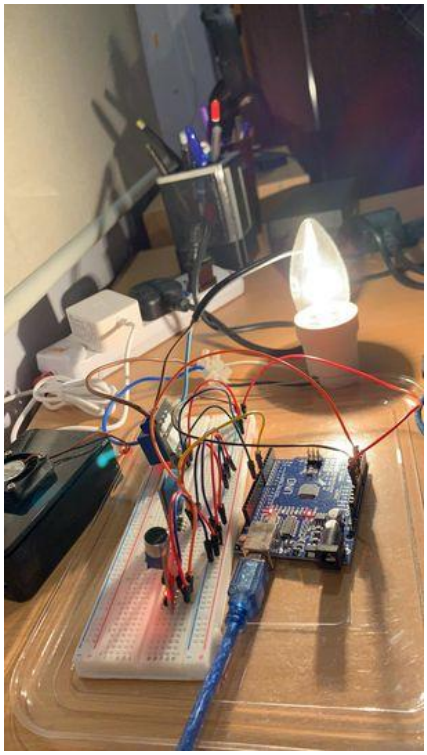


The auto radiator has 2 states of operation, when it is on, or when it is off. They are shown as below:

When the Autoradiator is off



When the Autoradiator is on



4.2.2 System Scripts

The first script being explained is the Arduino sketch, which is named “Assignment3.ino”.

Assignment3

```
#include <dht.h>
dht DHT;

unsigned long previousTime = 0;
unsigned int temp = 0;
int count = 0;
unsigned int soundData;
const unsigned long eventInterval = 10000;
unsigned int input = 0;
String stat = "off";

#define HTSENSOR 7
#define RELAY 10
void setup() {
  Serial.begin(9600);
  pinMode(RELAY, OUTPUT);
  pinMode(HTSENSOR, INPUT);
  digitalWrite(10, LOW);
}
```

The first part of the script is the setup. This is where the DHT sensor object is created to track the temperature, and the other variables are initialised. Constant variables for pins are also defined here. Next, the setup function is filled in with the serial starting, pin mode configuration for each pin, and setting the default state of the relay.

```
void loop() {
  //Data Collection
  unsigned long currentTime = millis();
  int soundData = analogRead(A0);
  if(soundData > 200)
  {
    //Serial.println(soundData);
    if(stat == "off")
    {
      stat = "on";
    }
    else if(stat == "on")
    {
      stat = "off";
    }
    int chk = DHT.read11(HTSENSOR);
    temp = DHT.temperature;
    Serial.print("<");
    Serial.print(stat);
    Serial.print(",");
    Serial.print(temp);
    Serial.println(">");
  }
```

The second part here is the manual control using the sound sensor. Firstly, the sound sensor would read the data, if it detects a spike of more than 200 (The sensor is calibrated to pick up silent as ~40), it will change the status from on to off or vice versa, then it will read the temperature data from the DHT11 sensor, and wraps it in a <status, temperature> format, and sends that as serial data to the Raspberry Pi.

```
  if (currentTime - previousTime >= eventInterval)
  {
    int chk = DHT.read11(HTSENSOR);
    temp = DHT.temperature;
    Serial.print("<");
    Serial.print(stat);
    Serial.print(",");
    Serial.print(temp);
    Serial.println(">");
    previousTime = currentTime;
  }
```

The third part is the automatic data collection. When the timer hits the eventInterval, which is set as 10000 seconds as above, it will take the temperature from the DHT11 sensor, and sends the current status of the system and temperature as serial to the Raspberry pi, then it resets the timer.

```
if(Serial.available() > 0)
{
  input = Serial.parseInt();
  switch(input)
  {
    case 1:
      digitalWrite(RELAY, HIGH);
      stat = "on";
      break;
    case 2:
      digitalWrite(RELAY, LOW);
      stat = "off";
      break;
    default:
      break;
  }
}
```

The fourth part waits for a serial input, and turns the relay on when the 1 integer is detected, and off when the 2 integer is detected. It also changes the status to on if it is on, and off if it is off.

The second script being discussed is the Python Script, Assignment3.py, and is as below:

Assignment3.py *

```python
import serial
import time
import paho.mqtt.client as mqtt
import json

from twilio.rest import Client
from beebotte import *

#Beebotte Keys
API_KEY = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

#Twilio SMS REST Information
account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

#Setting up the resources
bbt = BBT(API_KEY, SECRET_KEY)
statres = Resource(bbt, 'autoradiator', 'onoff')
tempres = Resource(bbt, 'autoradiator', 'temp')
autores = Resource(bbt, 'autoradiator', 'autobutton')
thresholdres = Resource(bbt, 'autoradiator', 'threshold')
ser = serial.Serial('/dev/ttyUSB0', 9600)
```

The first part of the script is importing the relevant libraries such as serial for serial data reading, time for delays, paho for mqtt communication and JSON for data transformation. The Twilio REST library is also included, using the Client, and Beebotte's API is also imported here. Next, Beebotte's API keys and Secret Keys are defined, as well as Twilio's SMS id and token. After that, a Beebotte object is created with the API and secret keys. Lastly, resources defined in Beebotte are defined here to write data into Beebotte later on. The serial is also set here.

```python
#MQTT on_connect
def on_connect(client, data, flags, rc):
    client.subscribe("autoradiator/onoff")
    client.subscribe("autoradiator/temp")
    client.subscribe("autoradiator/autobutton")
    client.subscribe("autoradiator/threshold")
```

MQTT function on_connect. Allows the client to subscribe to the relevant channels. The channels are named that way due to Beebotte's "channel/resource" format.

```python
#MQTT on_message
def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))
    autorecord = autores.read(limit = 1)
    if(autorecord[0]['data'] == False):
        soundrecord = statres.read(limit = 1)
        if(soundrecord[0]['data'] == True):
            ser.write(b"1")
            print("On")
            time.sleep(5)
        elif(soundrecord[0]['data'] == False):
            ser.write(b"2")
            print("Off")
            time.sleep(5)
        else:
            print("No Match")
```

MQTT function on_message. It prints the message in the console for view, and first, using the Beebotte API to read 1 data from the resource "autores". The record returned is a dictionary inside a list, hence, it is read as shown. If the device is not in automatic mode, it will actively check the status from Beebotte, if it is on, it will send a serial byte back to turn the relay on, and if it is off, it will send a serial byte back to turn the relay off.

```python
    elif(autorecord[0]['data'] == True):
        #Automation Script
        #Get the threshold data, and set it turn on and off accordi
        thresholdrecord = thresholdres.read(limit = 1)
        temprecord = tempres.read(limit = 1)
        if(temprecord[0]['data'] < thresholdrecord[0]['data']):
            ser.write(b"1")
            print("Auto On")
            time.sleep(5)
        elif(temprecord[0]['data'] >= thresholdrecord[0]['data']):
            ser.write(b"2")
            print("Auto Off")
            time.sleep(5)
        else:
            print("No Match")
```

An extension from the previous one, if the device is in automatic mode, it will actively read data from the temperature stored inside Beebotte from the sensor, and compare it to the threshold value the user inputted. If it is more, it will turn the relay off by sending a serial byte to the Arduino, simulating turning the radiator off and vice versa.

```python
#Set up MQTT Client
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

#Setting channel configurations
client.username_pw_set("token:token_rSnUqiQ9WIZ4MSd2")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()
```

Setting up the MQTT client to write to Beebotte later on. The username and password set is the token for the auto radiator's channel, and it then connects to Beebotte's mqtt site using the 1883 port. It then starts the loop.

Assignment3.py * ✖

```
oristatus = "off"
while True:
    #Collects data if serial is in waiting
    if ser.in_waiting > 0:
        ln = ser.readline()
        #Cleans the input and separates
        ln_refined = ln.decode("utf-8").strip('\r\n')
        lnlist = ln_refined.split(",")
        status = lnlist[0].strip("<")
        temperature = lnlist[1].strip(">")
        temp = int(temperature)
        #Prepares the input for beebotte
        if(status == "on"):
            boolstat = True
        elif(status == "off"):
            boolstat = False
```

Here, the original status is set for SMS use later. It then checks if there is serial data in waiting, and if there is, it reads the line and cleans the data. It then splits it and puts it into 2 pieces of data, which is the temperature, and the status, which is turned into a boolean value.

```
#If the current status is not the original status, send an SMS
if(status != oristatus):
    smsclient = Client(account_sid, auth_token)
    mailbody = "Status of AutoRadiator changed from " + oristatus + " to " + status
    message = smsclient.messages.create(body = mailbody,
                        from_ = "+12018905970",
                        to = "+601110679126")
    message.sid
    oristatus = status
```

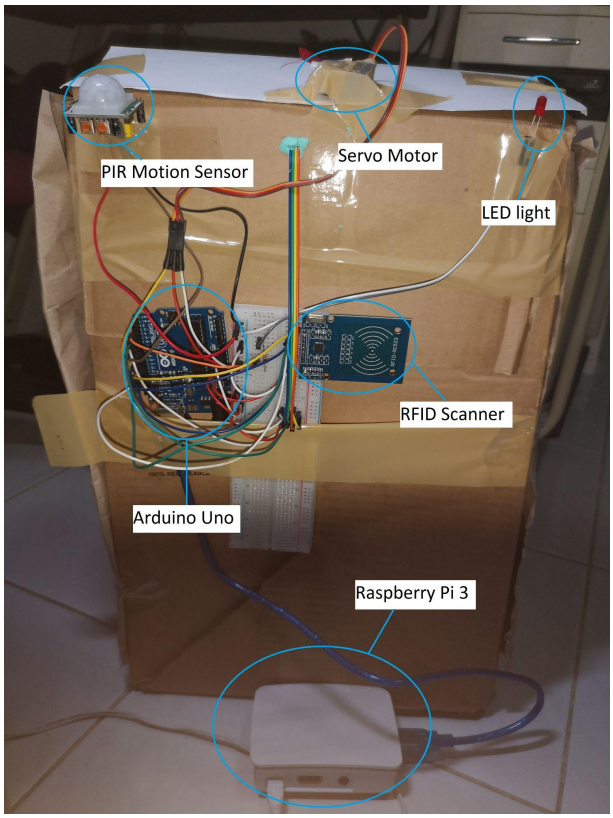
If the status is not the same as the original status, create a Twilio Client, and create a SMS to notify of the change. The SMS is then sent using Twilio's REST API, and the current status is the new original status.

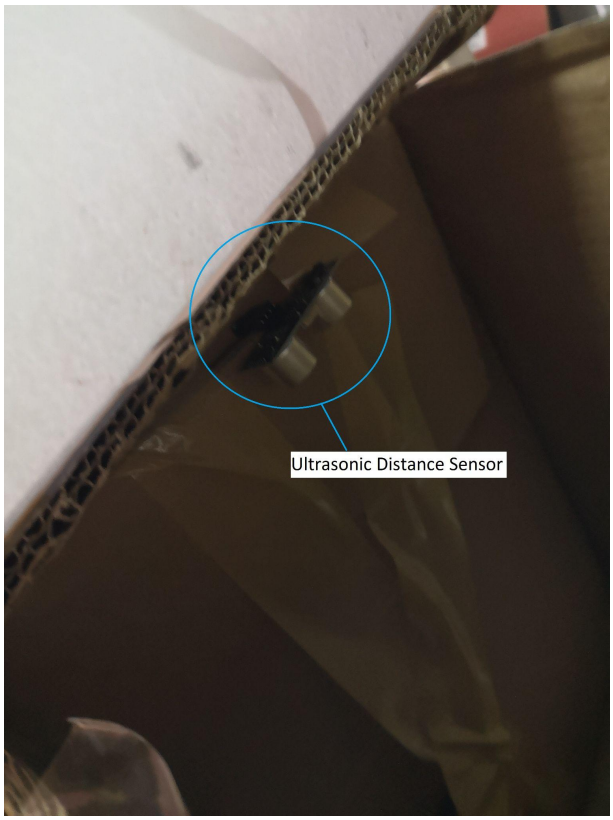
```
#Convert data to dict then JSON for BeeBotte
booldict = {"data": boolstat, "write": True}
tempdict = {"data": temp, "write": True}
booljson = json.dumps(booldict)
tempjson = json.dumps(tempdict)

#Publish the Data
client.publish("autoradiator/onoff", booljson)
client.publish("autoradiator/temp", tempjson)
```

The last section of the script, firstly it converts the data into a dictionary as seen, with an extra attribute "write" = True to tell Beebotte that this data is persistent. The data is then JSONfied as Beebotte only takes in JSON data when using MQTT as a bridge. The data then uses MQTT to publish to their relevant channels.

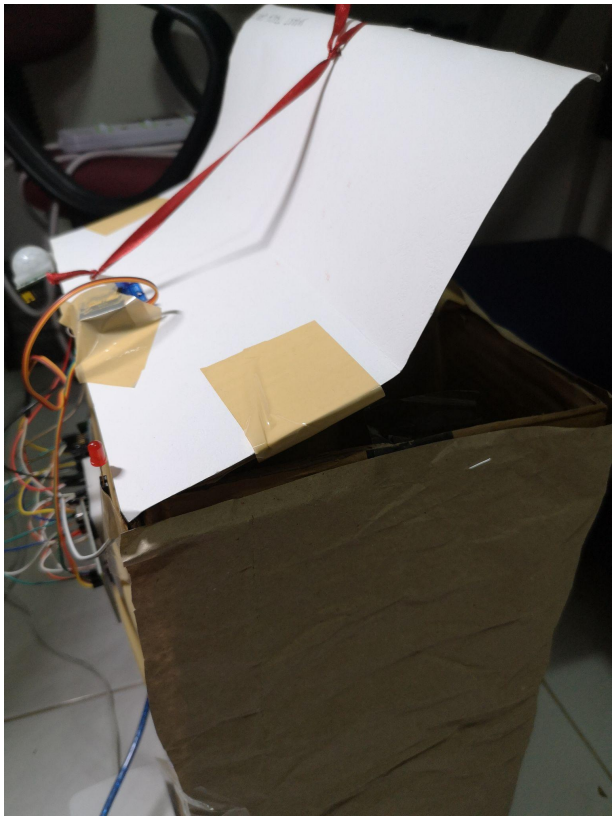
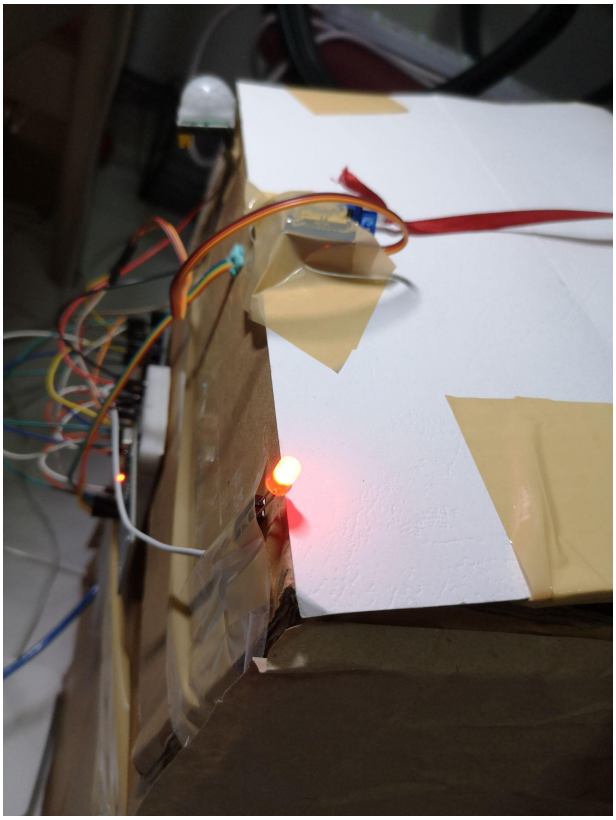
4.3 Smart Trash Can

4.3.1 System Images





The above two images show the hardware setup of the Smart Trash Can system. The system works with two actuators and three input sensors. One through the HC-SR501 PIR Motion Sensor situated at the top left of the trash bin at the left picture. This sensor will detect any motion through infrared and actuate the SG90 Servo motor if motion is detected. The other sensor that can open and close the trash lid is the RC522 RFID Scanner in the middle of the trash can in the left picture. The last input sensor is the HC0SR04 Ultrasonic Distance Sensor which will detect the distance between itself and the distance with the surface of any object that it is facing. As mentioned before, one of the actuators is the SG90 Servo motor which will be responsible for open and closing the lid. The other actuator is the LED light which will indicate if the trash surpasses the threshold set by the system or not. The last two components are the Arduino UNO which acts as the microcontroller and the Raspberry Pi 3 where the Arduino UNO is connected to it through a USB port.

The above two pictures are to demonstrate how the actuators will perform if the input is sensed and the conditions are met.

4.3.2 System Scripts

Firstly, the Arduino Sketch is to be addressed.

```
#include <Servo.h>
Servo servoMain;
int trigpin = 4;
int echopin = 3;
int distance;
float duration;
float cm;
#include "SPI.h"
#include "MFRC522.h"

unsigned int pinStatus = 0;
unsigned int thresholdPercentage = 0;
int currentThreshold = 20;
int height = 45;
int displayPercentage = 0;
int displayThreshold = 0;
#define SS_PIN 10
#define RST_PIN 9
#define SP_PIN 8
#define LED 2
#define PIR 6

MFRC522 rfid(SS_PIN, RST_PIN);

MFRC522::MIFARE_Key key;

void setup() {
  servoMain.attach(7);
  pinMode(trigpin, OUTPUT);
  pinMode(echopin, INPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  pinMode(LED,OUTPUT);
  digitalWrite(LED,LOW);
  pinMode(PIR, INPUT);
  servoMain.write(180);
}
```

The first section of the Arduino Sketch is the setup process. This is where the majority of the pin numbers are set, some libraries are included for the sensors and actuators, and some variables are defined here as well for later use.

The setup function will be responsible for setting up the pin modes, the initial state of the actuators and sensors as well as setting up the pin number for the servomotor.

```
void loop() {
  if (digitalRead(PIR) == HIGH && servoMain.read()!= 0){
    servoMain.write(0);
    delay(5000);
    servoMain.write(180);
    readDistance();

  }
  if (Serial.available() > 0)
  {
    String data = Serial.readStringUntil('\n');

    String firstWord = getValue(data, ' ', 0);
    String secondWord = getValue(data, ' ', 1);
    if (firstWord == "S"){
      if (secondWord == "1"){
        servoMain.write(0);
      }else if (secondWord == "2"){
        servoMain.write(180);
        readDistance();
      }
    }else if (firstWord == "T"){
      currentThreshold = 100 - secondWord.toInt();
      readDistance();
    }else if (firstWord == "H"){
      height = secondWord.toInt();
      readDistance();
    }
  }
```

In the loop function, this is where most of the sensors and actuators are being managed. The first if block is about the motion sensor, what it basically means is that if the motion sensor returns a positive feedback indicated by HIGH, and the servo motor is in a closed position, then turn the servo motor to open the trash lid and wait for 5 seconds. After 5 seconds, return the servo motor to its original position which is at 180 degrees. The readDistance is a custom function that is used with the ultrasonic sensor which will be covered in the next screenshot.

The next if block is for serial communication with the Raspberry Pi. If there is something in the serial port, Arduino will read it and run different functionalities based on the data read. If the data is of a string such as “S 1” or “S 2”, it will control the servo motor to open or close respectively. If the data starts with a “T” that comes with a number following it, it will change the threshold based on the number input. If the data starts with “H” instead, then it will change the height of the trash bin that is used to calculate the percentage of the filling rate of the trash.

```
void readDistance(){
  digitalWrite(trigpin, LOW);
  delay(2);
  digitalWrite(trigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigpin, LOW);

  duration = pulseIn(echopin, HIGH);

  cm = (duration / 58.82);

  distance = cm;
  int currentPercentage = cm / height*100;
  displayPercentage = 100 - currentPercentage;
  displayThreshold = 100 - currentThreshold;

  if (displayPercentage < 0){
    displayPercentage = 0;
  }
   if (displayThreshold < 0){
    displayThreshold = 0;
  }
   if (height <0){
    height = 0;
  }
  Serial.print(displayPercentage);
  Serial.print(" ");
  Serial.print(displayThreshold);
  Serial.print(" ");
  Serial.println(height);
  if (currentPercentage <= currentThreshold){
    digitalWrite(LED, HIGH);
  }else{
    digitalWrite(LED, LOW);
  }
}
```

This block of code is a function for reading the distance between the ultrasonic sensor and the surface of anything that it is facing. How it works is that the ultrasonic sensor will send a sound in front of it and it will measure how long it takes for that sound to travel back to it. The duration is then converted into a distance in cm. The distance is then calculated and processed to become a percentage value. This function will also be the one that sends output into the Raspberry Pi. It will send the percentage value, the threshold value and the height value through the serial port. The LED light computation is also here to detect if the trash is over the threshold. This function is usually used at the end of places where the the servo motor will close the lid, meaning to say that every time the lid is closed, then the data will be sent.

```
if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial())
  return;

MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);

if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
  piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
  piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
  Serial.println(F("Your tag is not of type MIFARE Classic."));
  return;
}

String strID = "";
for (byte i = 0; i < 4; i++) {
  strID +=
  (rfid.uid.uidByte[i] < 0x10 ? "0" : "") +
  String(rfid.uid.uidByte[i], HEX) +
  (i!=3 ? ":" : "");
}
strID.toUpperCase();



if (strID != "" && servoMain.read() == 0){
  Serial.print(strID);
  Serial.print(" ");
  Serial.print(displayPercentage);
  Serial.print(" ");
  Serial.println(displayThreshold);
  servoMain.write(180);
  readDistance();
}else if(strID != "" && servoMain.read() == 180){
  servoMain.write(0);
}
rfid.PICC_HaltA();
rfid.PCD_StopCryptol();
```

The block of code above is regarding the RFID Scanner. The scanner will detect if the card scanned is compatible with it and then convert the id from the data scanned and turn it into a more readable id. If there is an id and the servo motor is in an opened position, then data of the id and the percentage and threshold will be sent through the serial port, the servo motor will turn to a closed position and the readDistance function is run again.. Alternatively, if there is an id and the servo motor is in a closed state, then turn the servo motor to the open state and wait for the next input.

```
String getValue(String data, char separator, int index)
{
  int found = 0;
  int strIndex[] = {0, -1};
  int maxIndex = data.length()-1;

  for(int i=0; i<=maxIndex && found<=index; i++){
    if(data.charAt(i)==separator || i==maxIndex){
        found++;
        strIndex[0] = strIndex[1]+1;
        strIndex[1] = (i == maxIndex) ? i+1 : i;
    }
  }

  return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

The last block of code is just a simple function used to split the data input from Raspberry Pi.

As for the Raspberry Pi side. Only one Python script is in use for this project.

```
import serial
import time
import json
import paho.mqtt.client as mqtt

from beebotte import *
from twilio.rest import Client

API_KEY = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"
```

The above screenshot is to include all the necessary libraries to run this project. The api keys for BeeBotte and Twillio are also placed here.

```
bbt = BBT(API_KEY, SECRET_KEY)
user = Resource(bbt, 'Smart_Trash_Can', 'username')
height = Resource(bbt, 'Smart_Trash_Can', 'height')
percentage = Resource(bbt, 'Smart_Trash_Can', 'percentage')
threshold = Resource(bbt, 'Smart_Trash_Can', 'threshold')
lid = Resource(bbt, 'Smart_Trash_Can', 'lidControl')
collect = Resource(bbt, 'Smart_Trash_Can', 'collect')
manual = Resource(bbt, 'Smart_Trash_Can', 'manual')
changeThreshold = Resource(bbt, 'Smart_Trash_Can', 'changeThreshold')
changeHeight = Resource(bbt, 'Smart_Trash_Can', 'changeHeight')
```

The above screenshot is where a connection with the cloud server is established. In the first line, a Beebotte object is created using the api key and secret key that was mentioned before. The rest of the code is used to establish a connection with the resources inside of the Beebotte Cloud server. In order to connect to the different resources inside the channel, these objects are to be created.

```
arduino = serial.Serial('/dev/ttyACM0', 9600)
def on_connect(client, data, flags, rc):
    client.subscribe("Smart_Trash_Can/username")
    client.subscribe("Smart_Trash_Can/height")
    client.subscribe("Smart_Trash_Can/percentage")
    client.subscribe("Smart_Trash_Can/threshold")
    client.subscribe("Smart_Trash_Can/lidControl")
    client.subscribe("Smart_Trash_Can/collect")
    client.subscribe("Smart_Trash_Can/manual")
    client.subscribe("Smart_Trash_Can/changeThreshold")
    client.subscribe("Smart_Trash_Can/changeHeight")
```

The above code is used to connect the Raspberry Pi to the Arduino through the serial port as well as establish a connection to the Cloud Server through MQTT. This would be mostly used to publish data to the server for the remainder of the codes for this segment.

```
def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))
    manualOverride = manual.read(limit = 1)
    changeHeightOverride = changeHeight.read(limit = 1)
    changeThresholdOverride = changeThreshold.read(limit = 1)
    if (manualOverride[0]['data'] == True):
        record = lid.read(limit = 1)
        print(record[0]['data'])
        if(record[0]['data'] == True):
            arduino.write(b"S 1\n")
        elif(record[0]['data'] == False):
            arduino.write(b"S 2\n")
        else:
            print("No Match")
```

```
        arduino.flushOutput()
        time.sleep(5)
    if (changeThresholdOverride[0]['data'] == True):
        thresholdValue = threshold.read(limit=1)
        arduino.write(b"T "+str(thresholdValue[0]['data']).encode()+b"\n")
        arduino.flush()
        time.sleep(5)
    if (changeHeightOverride[0]['data'] == True):
        heightValue = height.read(limit=1)
        arduino.write(b"H "+str(heightValue[0]['data']).encode()+b"\n")
        arduino.flush()
        time.sleep(5)
```

Moving on would be the MQTT on_message function which is mainly used to read the latest data from the Beebotte server if it receives a message in the server. The main use of this function in this context is to provide manual override to the system to let the user be able to change values or control the bin if they want. The if statements are mainly used to differentiate whether or not to enable or disable the function of sending data through the serial port. If the latest data from the Beebotte server is true, then run the following codes that will activate or change different rules of the system based on what is activated. The features that can be changed or controlled is the opening and closing of the lid, the changing of height and the threshold.

```
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.username_pw_set("token:token_KSLevGDpDVcT4dWu")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()
```

The above code is used to start the connection through MQTT with both the functions mentioned above. The next line of code is to place a token of the trash can channel into the MQTT client which will enable the MQTT client to connect to said channel. It is then connected to the cloud server through the 1883 port and the loop has begun.

```
while 1:
    if arduino.in_waiting > 0:
        print(arduino.in_waiting)
        line = arduino.readline().decode("utf-8")
        x = line.split()
        arduino.flushInput()
        arduino.flushOutput()
        print(x)
        if (not(x[0].isdigit())):
            userList = user.read(limit = 2)
            print(userList[0]['data'])
            name = ""
            for users in userList:
                userID = users['data'].split(" ")
```

The while loop begins here and the first segment would of the Raspberry Pi reading input from the arduino. If there is input, then read the line and split it into different segments for better processing. The following if statement is to check whether the input is from the RFID or from the closing motion of the trash lid. If the input is from the RFID, which will not be integers, then run a code which is the following:

```
for users in userList:
    userID = users['data'].split(" ")
    if (userID[0] == str(x[0])):
        name = userID[1]
collectDict ={"data":name, "write": True}
percentageDict ={"data":int(x[1]), "write": True}
thresholdDict ={"data":int(x[2]), "write": True}

percentageJSON = json.dumps(percentageDict)
thresholdJSON = json.dumps(thresholdDict)
collectJSON = json.dumps(collectDict)

client.publish("Smart_Trash_Can/percentage", percentageJSON)
client.publish("Smart Trash Can/threshold", thresholdJSON)
 client.publish("Smart_Trash_Can/collect", collectJSON)

 smsclient = Client(account_sid, auth_token)
 mailbody = name + " has collected the rubbish"
 message = smsclient.messages.create(body = mailbody,
                         from_ = "+12018905970",
                         to = "+601110679126")
 message.sid

 arduino.flushInput()
 arduino.flushOutput()
 time.sleep(5)
```

The code will be check the database from the cloud server to see if this id exists, if it does, then log the person's name into a variable for later use. It will then start preparing the data in JSON format to be sent to the cloud server and then is sent through the client.publish() function. Different data is sent to different resources inside the channel.

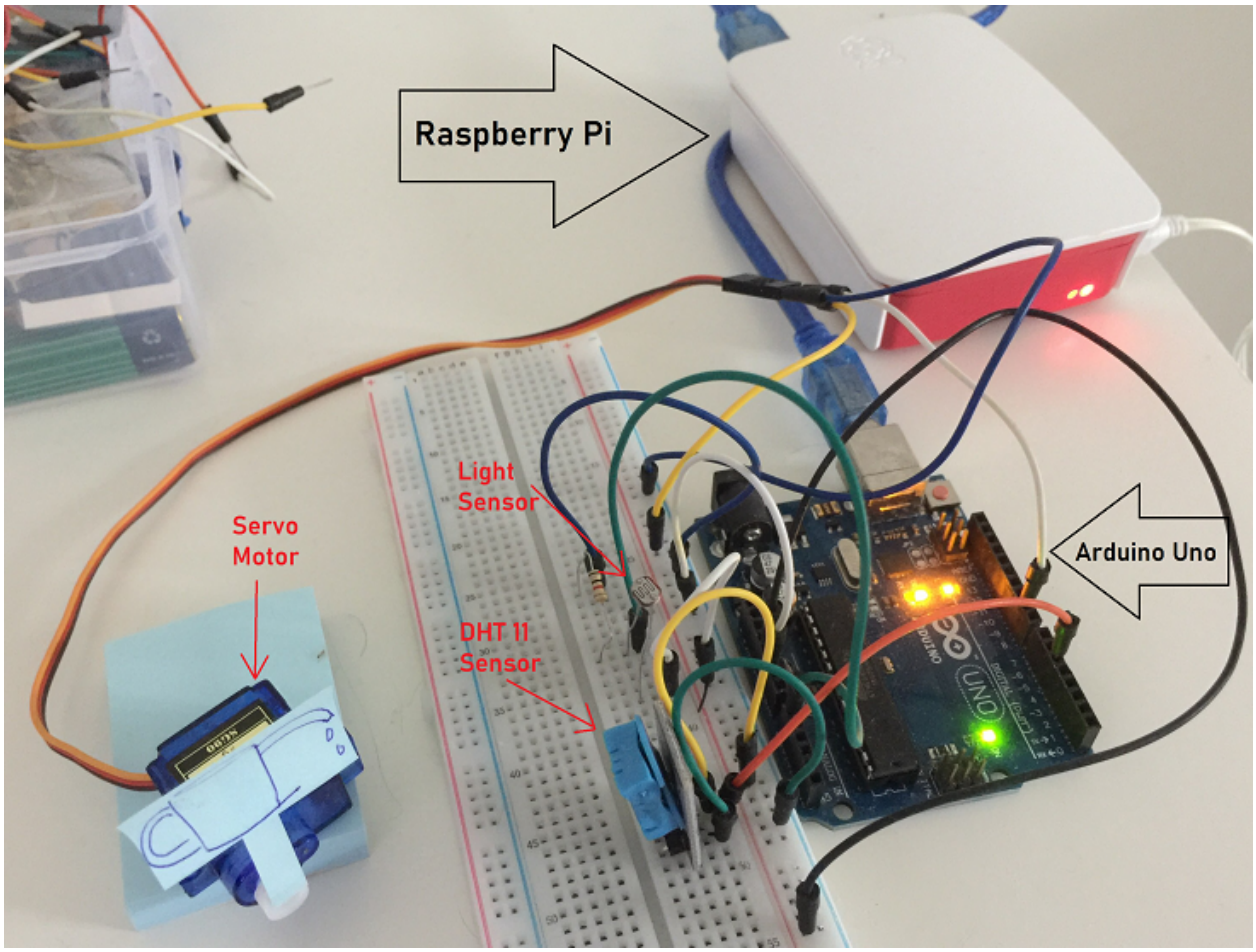
Other than that, it will also run a code to send an sms message to the user (in this case Jun Jie) through a Twilio API. The message content will generally be informing the user that a person has collected the trash since the RFID's use is to log people who collect trash.

```
else:
    percentageDict ={"data":int(x[0]), "write": True}
    thresholdDict ={"data":int(x[1]), "write": True}
    heightDict ={"data":int(x[2]), "write": True}

    percentageJSON = json.dumps(percentageDict)
    thresholdJSON = json.dumps(thresholdDict)
    heightJSON = json.dumps(heightDict)

    client.publish("Smart_Trash_Can/percentage", percentageJSON)
    client.publish("Smart_Trash_Can/threshold", thresholdJSON)
    client.publish("Smart_Trash_Can/height", heightJSON)
    arduino.flush()
    time.sleep(5)
```

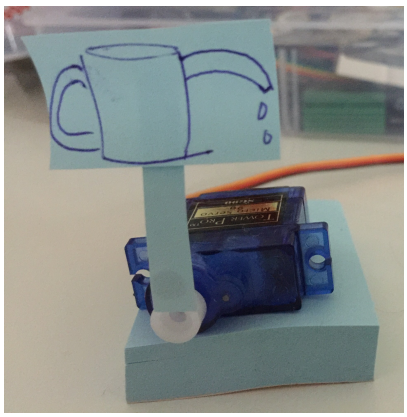
The last segment of the code will be the else statement from the previous RFID if statement. Basically, it means that if the data is numbers, then save the data into the cloud server. Similarly to the one before, the data would be prepared in a JSON format and then published into the Beebotte server through the MQTT client.

4.4 Auto Plant Watering

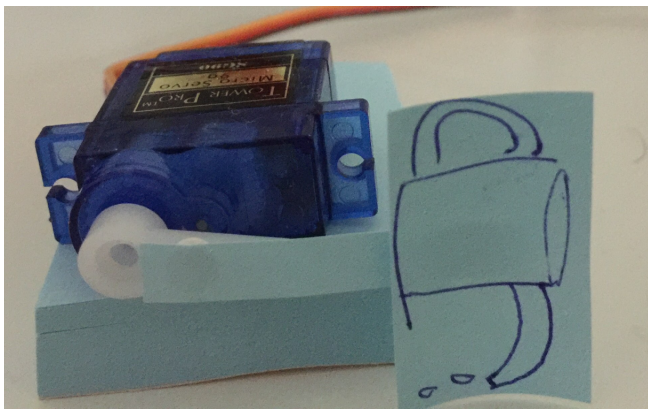
4.4.1 System Images



The Servo Motor when the “Watering System” is turned off



The Servo Motor when the “Watering System is turned on”



4.4.2 System Scripts

groupTo | Arduino 2:1.0.5+dfsg2-4.1

File Edit Sketch Tools Help

groupTo

```
#include <dht.h>
#include <Servo.h>

#define dht_apin 5 //Analog Pin sensor is connected to

dht DHT;
int light;
int input;
unsigned long previousTime = 0;
const unsigned long eventInterval = 10000;
Servo servo_test;

void setup() {
  Serial.begin(9600);
  servo_test.attach(9);
}
```

The beginning of the script is about including all of the needed libraries and initialising the variables, including the constant variables. It also sets up the servo motor which would be used in this project.

This part of the script also initialises which pin is attached to the servo motor.

```
void loop() {
  unsigned long currTime = millis();

  if (currTime - previousTime >= eventInterval) //10 Seconds Pass
  {
    DHT.read11(dht_apin);
    light = analogRead(A0);

    Serial.print(light);
    Serial.print(",");
    Serial.print((int)DHT.humidity);
    Serial.print(",");
    Serial.print((int)DHT.temperature);
    Serial.println(",");
    previousTime = currTime;
  }
```

The second part of the script is when 10 seconds have passed, the data received by the two sensors, Light Sensor and DHT11 (Temperature and Humidity Sensor), it will send the data to the RaspberryPi in an array. After that it would change the previousTime data into the currTime data, so that it would keep increasing as the system runs.

```
  if (Serial.available() > 0)
  {
    input = Serial.parseInt();
    switch(input)
    {
      case 1:
        servo_test.write(0);
        delay(10000);
        servo_test.write(100);
        break;

      default:
        break;
    }
  }
}
```

This last part of the script is about how to control the servo motor, it receives data from the RaspberryPi and if it is the correct input. The servo motor would rotate to the set angle. After 10 seconds, it would rotate back to its original angle.

That's all for my Arduino Script code.

```python
import serial
import time
import json
from beebotte import *
import paho.mqtt.client as mqtt
from twilio.rest import Client

device = '/dev/ttyACM0'
arduino = serial.Serial(device, 9600)
valueLight = ''
lightVal = 0
counter = 1000
COUNT = 1000
thresLight = 400
thresHumid = 30
thresTemp = 25
```

The first section of my python code was to initialise and import the libraries needed for the system.

As well as stating which port is the Arduino connected to the Raspberry Pi.

```
#Twilio SMS REST Information
account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

smsclient = Client(account_sid, auth_token)
mailbody = ("Water system has been turned on")
message = smsclient.messages.create(body=mailbody, from_ = "+12018905970",
to = "+601110679126")
```

This part of the code is initializing how the system sends a message to the given phone number, first getting the account's sid and authorized token and connecting to the web page where everything is controlled. Following that is inputting what message I want to send to the phone. The message would then be compiled and sent to the phone number.

```
API_KEY = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

bbt = BBT(API_KEY, SECRET_KEY)

light_re = Resource(bbt, 'Auto_Plant_Watering', 'Light')
temp_re= Resource(bbt, 'Auto_Plant_Watering', 'Humidity')
humid_re= Resource(bbt, 'Auto_Plant_Watering', 'Temperature')
button_re= Resource(bbt, 'Auto_Plant_Watering', 'Water_Plant')
counter_re= Resource(bbt, 'Auto_Plant_Watering', 'Counter')
tLight_re= Resource(bbt, 'Auto_Plant_Watering', 'Threshold_Light')
tHumid_re= Resource(bbt, 'Auto_Plant_Watering', 'Threshold_Humid')
tTemp_re= Resource(bbt, 'Auto_Plant_Watering', 'Threshold_Temp')

def on_connect(client, data, flags, rc):
    client.subscribe("Auto_Plant_Watering/Light")
    client.subscribe("Auto_Plant_Watering/Humidity")
    client.subscribe("Auto_Plant_Watering/Temperature")
    client.subscribe("Auto_Plant_Watering/Water_Plant")
    client.subscribe("Auto_Plant_Watering/Counter")
    client.subscribe("Auto_Plant_Watering/Threshold_Light")
    client.subscribe("Auto_Plant_Watering/Threshold_Humid")
    client.subscribe("Auto_Plant_Watering/Threshold_Temp")

def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect


client.username_pw_set("token:token_DXqA4dui7GEZjYFM")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()
```

This code connects the system to the Beebotte webpage, and then will send all the relevant data into the website according to the names.

The MQTT function on_connect is to allow the client to subscribe to the channels in the webpage.

The MQTT function on_message is to print the message sent by the system.

The username and the password is then set up to connect to the Beebotte's webpage using the 1883 port.

Finally it starts the loop.

```
while 1:
    while(arduino.in_waiting == 0):
        pass

    line = arduino.readline()

    values = str(line)[2:].split(",")


    light = values[0]
    humid = values[1]
    temp = values[2][:2]

    # Counter

    thresLight = tLight_re.read(limit= 1)[0]['data']
    if (int(light) > thresLight):
        counter -= 1

    thresHumid = tHumid_re.read(limit= 1)[0]['data']
    if (int(humid) < thresHumid):
        counter -= 1

    thresTemp = tTemp_re.read(limit= 1)[0]['data']
    if (int(temp) > thresTemp):
        counter -= 1

    counter -= 1

    print(str(thresLight) + "," + str(thresHumid) + "," + str(thresTemp))

    print(counter)
```

The code here is to get the sensor's data from the Arduino, which was sent in an array. After that, the code splits it apart. The data for the light, humidity, and temperature would then be compared to the threshold values placed for the data.

Depending on the data received and threshold, the system would either speed up the counter by minus 1-3 from the current counter or continue with just minus 1. Each time the Raspberry receives data and compares it, the counter would go down either way.

```
    if (counter <= 0):
        arduino.write(b'1')
        message.sid
        counter = COUNT

    # Watering Plant

    waterIn = button_re.read(limit = 1)
    if (waterIn[0]['data'] == True):
        arduino.write(b'1')
        message.sid
        counter = COUNT
```

This part of the code is about how the system tells the Arduino to “water the plant.” The first part is automatic watering, where after the Counter reaches 0, the plant will get watered, and the Counter gets restarted.

The second part is the manual control. When the button on the Beebotte website is clicked, the Boolean data will be sent to the Raspberry Pi, which will then send the code to “water the plant.”

```
lightdict = {"data": int(light), "write": True}
lightjson = json.dumps(lightdict)

humiddict = {"data": int(humid), "write": True}
humidjson = json.dumps(humiddict)

tempdict = {"data": int(temp), "write": True}
tempjson = json.dumps(tempdict)

buttondict = {"data": False, "write": True}
buttonjson = json.dumps(buttondict)

countdict = {"data": counter, "write": True}
countjson = json.dumps(countdict)

tLightdict = {"data": thresLight, "write": True}
tLightjson = json.dumps(tLightdict)

tHumiddict = {"data": thresHumid, "write": True}
tHumidjson = json.dumps(tHumiddict)

tTempdict = {"data": thresTemp, "write": True}
tTempjson = json.dumps(tTempdict)

client.publish("Auto_Plant_Watering/Light", lightjson)
client.publish("Auto_Plant_Watering/Humidity", humidjson)
client.publish("Auto_Plant_Watering/Temperature", tempjson)
client.publish("Auto_Plant_Watering/Water_Plant", buttonjson)
client.publish("Auto_Plant_Watering/Counter", countjson)
client.publish("Auto_Plant_Watering/Threshold_Light", tLightjson)
client.publish("Auto_Plant_Watering/Threshold_Humid", tHumidjson)
client.publish("Auto_Plant_Watering/Threshold_Temp", tTempjson)
```

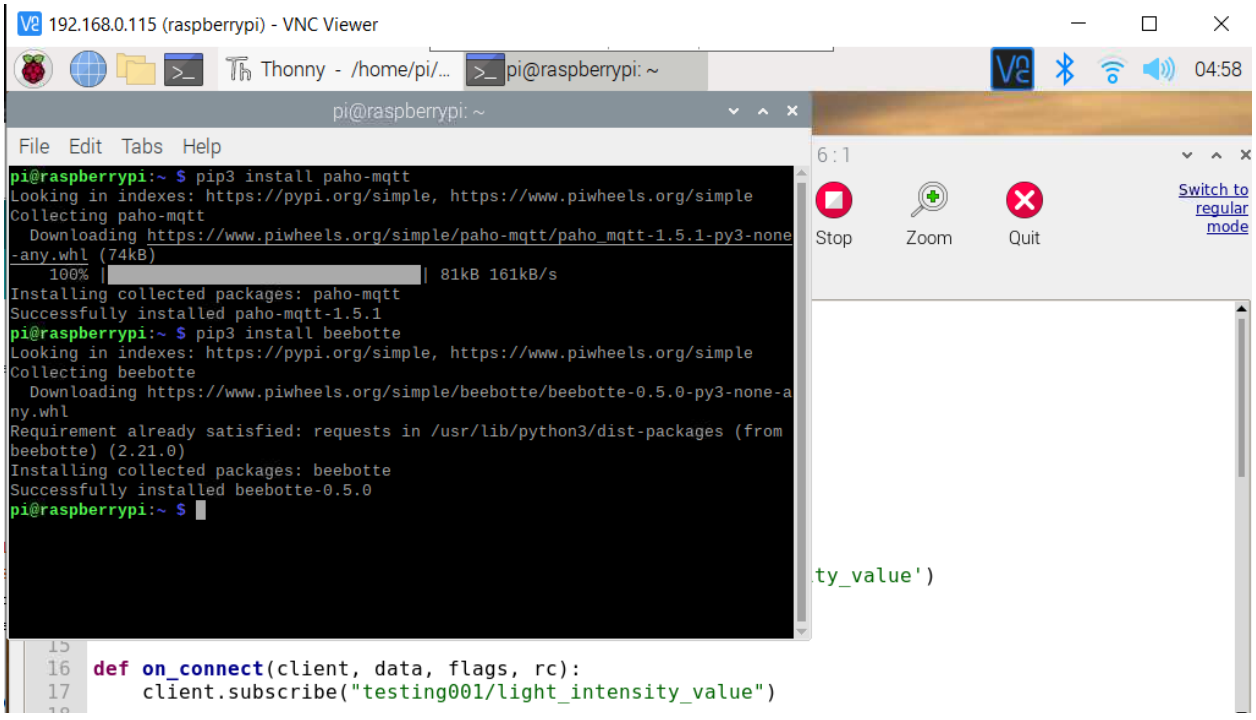
This is the final part of my python code, where it turns all the data into a dictionary, sending it to the Beebotte that the data is persistent. The data is then turned into JSON as Beebotte only accepts JSON data if the user is using MQTT to publish.

4.5 Edge servers

Because Beebotte only accepts JSON data when using MQTT as a bridge, it will read the data of the sensor from the Beebotte and save it in a dictionary in JSON format. MQTT is then used to publish the data to the appropriate channels. All the sensing system will send the read data from the Arduino into the edge server which is the Raspberry Pi. After that, the data will be stored inside the edge server and sent to the cloud through the communication protocols.

4.6 Communication protocols

The communication protocol used by the group to allow the communication between the edge server and the cloud side is MQTT. It is an OASIS-recognized Internet of Things communications protocol (IoT). It's built as a super-lightweight publish/subscribe messaging transport that's perfect for linking faraway devices with minimal code and network resources. It's a publish/subscribe protocol that lets devices at the network's edge publish to a broker. Clients connect to the broker, who then acts as a middleman between the two devices. Each device has the ability to subscribe to or register for specific topics.



```
import serial
import time
import paho.mqtt.client as mqtt
import json

from twilio.rest import Client
from beebotte import *

# will need to change when come to group combination
# PERSONAL api key and secret key
# API_KEY_AUTOLIGHT = 'gTYdN2z6boA2abUcyBjwodf1'
#SECRET_KEY_AUTOLIGHT = 'DvkUcfpV3fikrkowpjJaaVubOrQpXjor'

# GROUP api key and secret key
API_KEY_AUTOLIGHT = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY_AUTOLIGHT = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'
```

Importing the necessary libraries, such as serial for serial data reading, time for delays, paho for mqtt communication, and JSON for data processing, is the initial step of the script. The Twilio REST library, which is used with the Client, is also included, as is Beebotte's API. The API keys and Secret Keys for Beebotte, as well as Twilio's SMS id and token, are defined next. The API and secret keys are then used to construct a Beebotte object.

```
bbt = BBT(API_KEY_AUTOLIGHT, SECRET_KEY_AUTOLIGHT)
light_resourse = Resource(bbt, 'Smart_Lighting_System', 'light_intensity_value')
light_button = Resource(bbt, 'Smart_Lighting_System', 'light_onoff')
light_input = Resource(bbt, 'Smart_Lighting_System', 'light_input')
light_auto_onoff = Resource(bbt, 'Smart_Lighting_System', 'light_auto_onoff')

temp_resourse = Resource(bbt, 'Smart_Lighting_System', 'temp_value')
temp_input = Resource(bbt, 'Smart_Lighting_System', 'temp_input')
```

After that, connect the cloud side and the edge server by using the API and SECRET keys. Ensure that we connect to the right channel and correct resources of our system. For example, in this case there is a Channel named “Smart_Lighting_System” and has resources such as “light_intensity_value”, “light_onoff”, “light_input”, “light_auto_onoff”, “temp_value” and “temp_input”. Finally, resources created in Beebotte are defined here so that data can be written into Beebotte in the future. The serial is also set in this session.

```
def on_connect(client, data, flags, rc):
    client.subscribe("Smart_Lighting_System/light_intensity_value")
    client.subscribe("Smart_Lighting_System/light_onoff")
    client.subscribe("Smart_Lighting_System/light_input")
    client.subscribe("Smart_Lighting_System/light_auto_onoff")

    client.subscribe("Smart_Lighting_System/temp_value")
    client.subscribe("Smart_Lighting_System/temp_input")
```

MQTT function on_connect. Allows the client to subscribe to the channels that are relevant to them. The channels are set according to Beebotte's "channel/resource" format.

```
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# will need to change when come to group combination
# PERSONAL token
#client.username_pw_set("token:token_WbenCRZsqINb08va")

# GROUP token
client.username_pw_set("token:token_CmG3yJSa0ZaiaPKs")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()
```

Setting up the MQTT client to send messages to Beebotte. We use mqtt as a subscription protocol and allow edge-of-network devices to publish to a broker. Clients connect to this broker, which then allows communication between the edge device and cloud side. Each device can subscribe, or register, to a particular channel/resource pair. The token for the Smart Lighting System's channel is set automatically, and it connects to Beebotte's mqtt site via the 1883 port. The loop then begins.

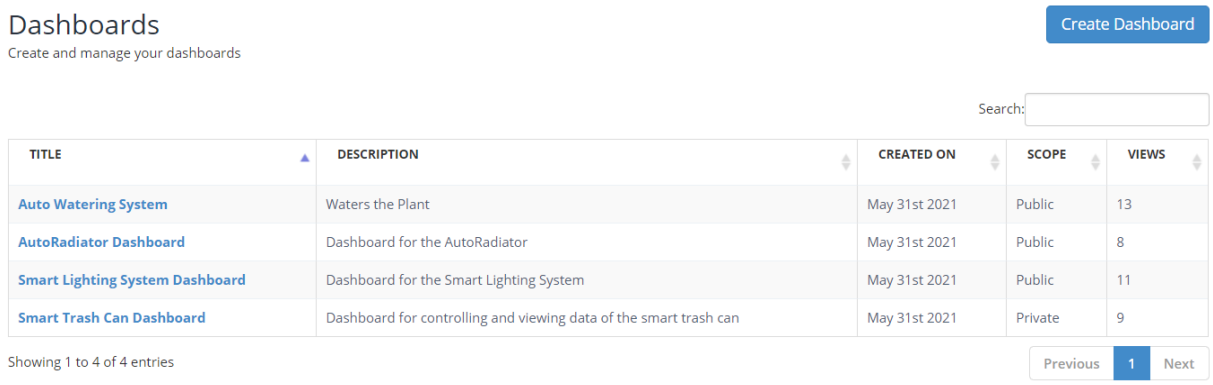
```
    #Convert data to dict then JSON for BeeBotte
    booldict = {"data": boolstat, "write": True}
    tempdict = {"data": temp, "write": True}
    booljson = json.dumps(booldict)
    tempjson = json.dumps(tempdict)

    #Publish the Data
    client.publish("autoradiator/onoff", booljson)
    client.publish("autoradiator/temp", tempjson)
```

Lastly, through the use of MQTT, data can also be published to their relevant channel/resource pair. As seen below, the data is first changed to a dictionary with a new field to signify that the data should be written, and persistent. Then the data is JSONfied to allow it to be transferred using MQTT into Beebotte. The last step is to publish the data into their relevant channel/resource pait, such as the temperature JSON to the temperature resource, and the status JSON to the on/off resource.

5.0 Website Details

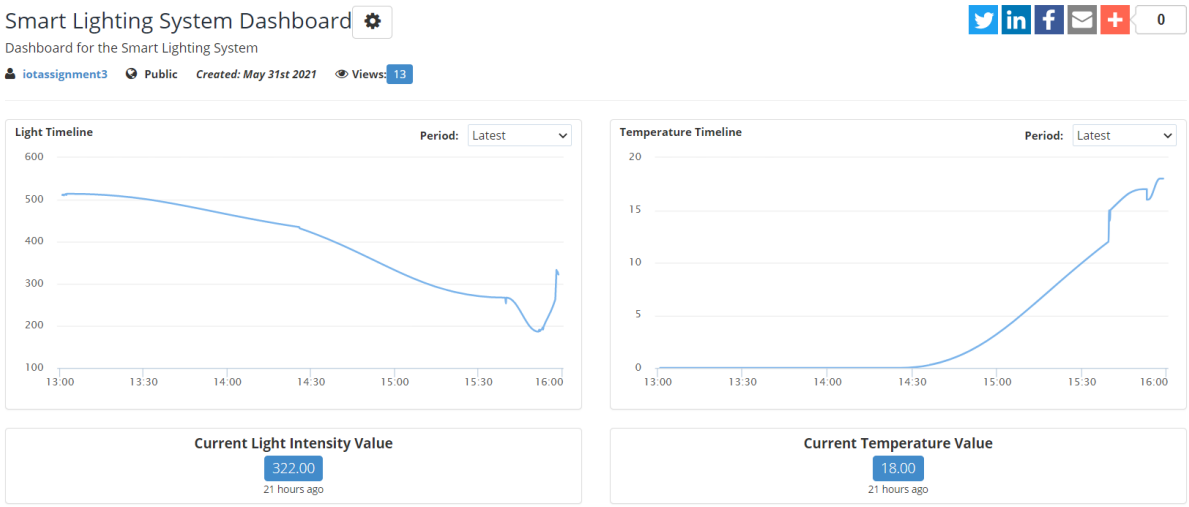
The web application used to take control of this home hub system is Beebotte's dashboard. Beebotte supports the creation of multiple pages or dashboards for each channel. The resources and widgets in each dashboard can then be linked to any channels' resources to be displayed in the dashboards. User inputs such as buttons and input fields can also be added to interact with Beebotte's cloud system, which is used in the home hub.

Dashboards

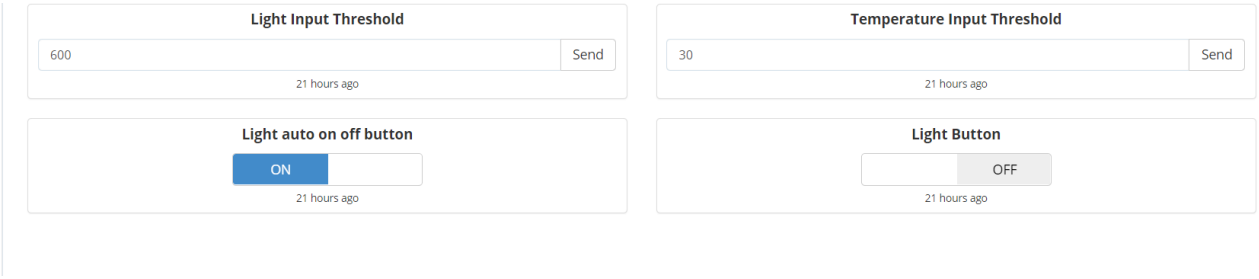


5.1 Smart Lighting System Dashboard

The first dashboard being showcased is the Smart Lighting System's Dashboard. The dashboard is as below:

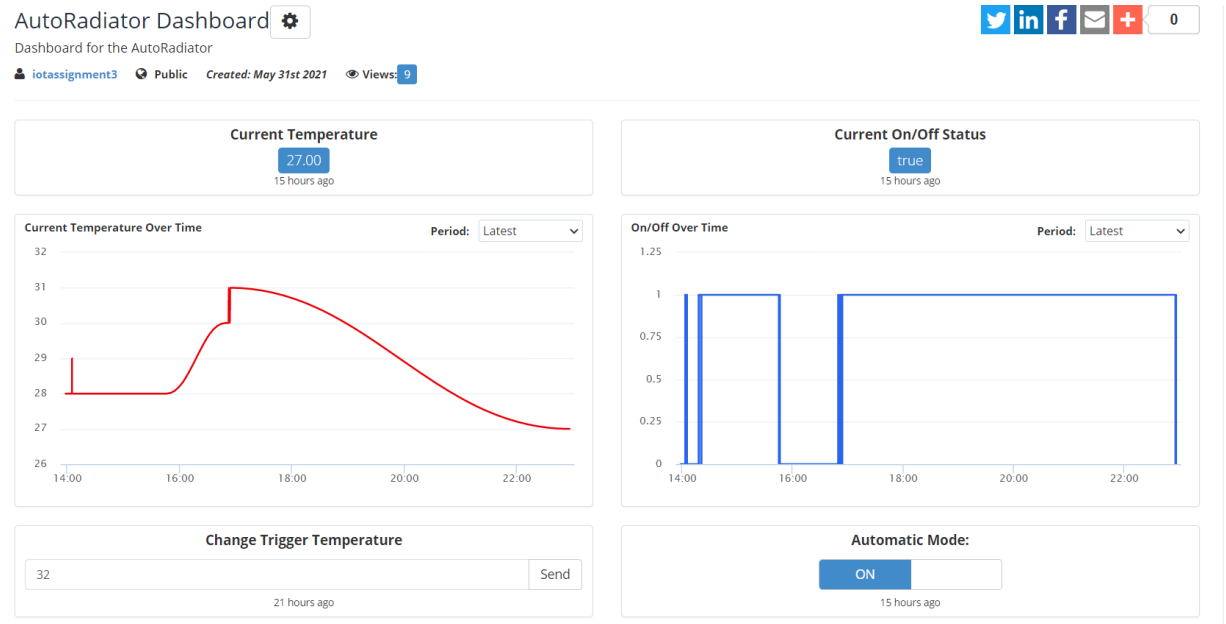


The first segment is the display, where a graph on the light intensity over time and the temperature over time is displayed. Below that are two basic displays for the user to look at the values stored, which are light intensity and the current temperature value. The temperature value is recorded in degree Celcius, whereas for the light intensity value, the higher it is, the brighter it is. The second segment is a user control segment. It consists of user input fields, which is used to set threshold values for light and temperature for the automation to check against, and two buttons, one is to turn on automation mode, and the other is to turn on the lights manually.



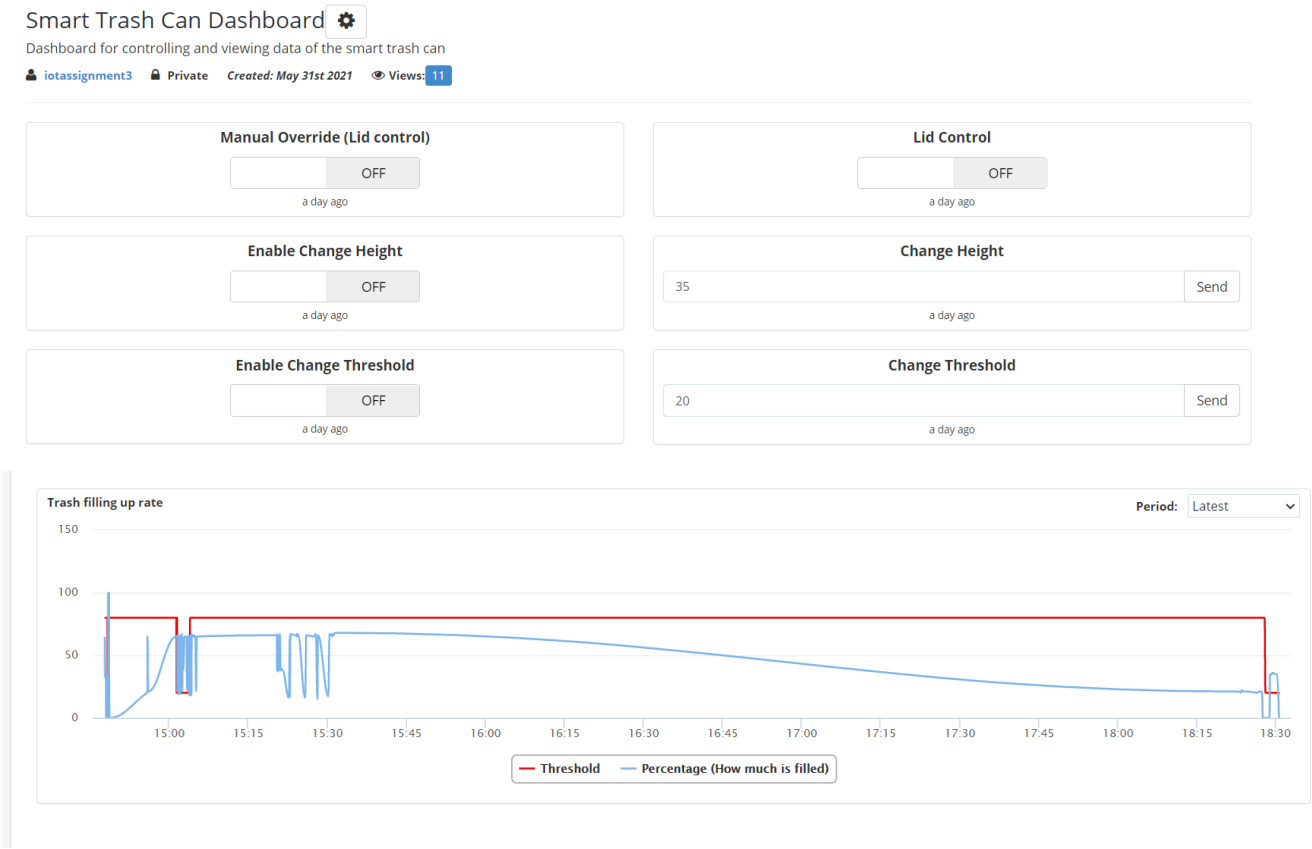
5.2 Auto Radiator Dashboard

The next dashboard is for the automatic radiator. It also comes in two segments, which is the display and the user input. The display consists of 4 widgets, which are the temperature and status display, to check the temperature of the area and the current status of the radiator. After that, there are two graphs to read two kinds of data over time, which is 1) The temperature over time to see the temperature trend and 2) The on off status over time to see how long is the radiator turned on or off. After that, there is an input and a button for the user input segment. The input field is used to enter a temperature for the radiator to trigger at. The button is to turn on automatic mode, if it is on, it will check the data with a threshold and turn it off and on from there, and if it is off, the system will listen for claps to turn the radiator on or off.



5.3 Smart Trash Can Dashboard

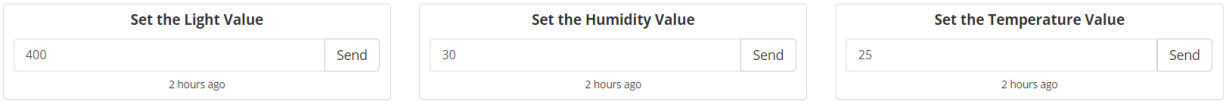
The third dashboard being shown is the Smart Trash Can Dashboard, which contains much more user input functions, and a single multi-line graph depicting the trash filled up rate against the threshold, to check when the trash can is filled and check how many times it surpasses the threshold. The user input contains four buttons, the first being the manual override button to open the lid from the dashboard, the lid control to enable the motion sensor to open the lid, the enable height change button to enable the changing of the height of the trash in the bin and the enable threshold change button to enable the changing of threshold values. Only two user input fields are present, which are to change the height of the trash manually, and to change the threshold of the trash to be checked against.



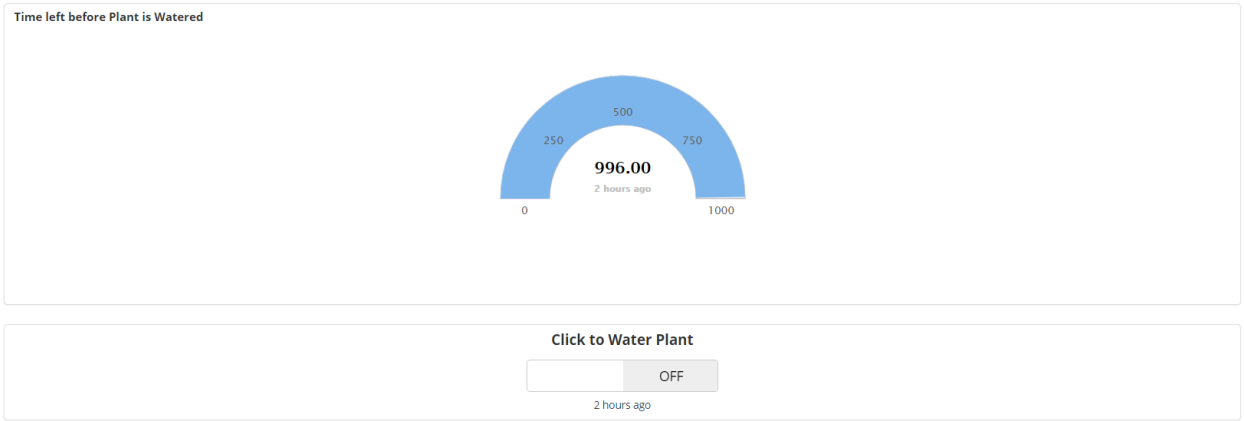
5.4 Auto Watering System Dashboard

The last dashboard made for this home hub system is the auto watering dashboard. It contains multiple time graphs for the data collected by this module, a timer that counts down towards the next time the plant is going to be watered, and four user inputs, which are the threshold setting for the three data being collected, and a manual button to water the plant.

The first segment is as above, which tracks the following data received from the sensors on the arduino: Light intensity, Humidity and Temperature. Light intensity is tracked in the sensor's own unit, with the higher the value, the brighter the light exposed. Humidity is tracked using the DHT11 sensor, and is in percentage. Lastly is the temperature from the DHT11 sensor as well, which is tracked in degree Celcius.



The second segment is the user input segment, which is where the thresholds are set. The threshold input for light intensity, humidity and temperature value is set here for the automation to check the sensor data against them. The last two widgets on this dashboard is a timer display, which displays the time left before the plant is watered again, and a button to manually water the plant.



6.0 Cloud Computing: Beebotte

The Home Hub system uses the Beebotte Cloud Service as its Cloud Computing Component. Beebotte, according to its website (https://beebotte.com/) is a Cloud Platform Service which supports RESTful APIs, websockets and MQTT. The components of the Home Hub system uses, as stated before, MQTT to communicate with Beebotte through Beebotte's Bridging capabilities with MQTT. It incorporates Amazon's AWS's redundant architecture for high availability and is free to be used with limitations set.

Throughout the components of the home hub system, the data passed from the sensors, to the edge devices eventually reach Beebotte's cloud storage through the use of MQTT. Some details on how Beebotte stores data, it stores them in a JSON format that contains an ID, the data itself and the information for Beebotte's REST API requests. This can be seen as below:

```
[{'_id': '60b63a3fdb20a07c4d761a12', 'data': 26, 'ts': 1622555199811, 'wts': 1622555199811}
]
```

(From left to right, the data id, the data itself, Beebotte REST API request codes)
To store data into the Beebotte cloud, the data has to be formatted as follows for it to be marked as persistent in the Beebotte cloud storage.

```
#Convert data to dict then JSON for BeeBotte
booldict = {"data": boolstat, "write": True}
tempdict = {"data": temp, "write": True}
booljson = json.dumps(booldict)
tempjson = json.dumps(tempdict)
```

To send data to Beebotte's cloud storage, MQTT was used. Firstly, the client is first asked to subscribe to a topic. In Beebotte's format, a topic is written as a channel of a resource, like below:

```
client.subscribe("Smart_Lighting_System/temp_value")
```

This sets the client to subscribe to the topic Smart_Lighting_System/temp_value, allowing it to receive all the data from the Smart_Lighting_System channel's temp_value resource.

After subscription, the client has to publish to their subscribed topic. The data needed to be published is formatted as reported above, then published to their respective channels. This can be seen as below:

```
#Publish the Data
client.publish("autoradiator/onoff", booljson)
client.publish("autoradiator/temp", tempjson)
```

The data is published to the topic autoradiator/onoff and autoradiator/temp, which would be saved in the Beebotte cloud storage, and appear in the autoradiator channel's onoff resource and temp resource respectively.

To read the data from Beebotte's cloud storage, Beebotte's REST API was used. As per their webguide for Beebotte's cloud storage functions (*REST API* 2015), Beebotte uses the GET, POST and DELETE requests to provide ways to retrieve data from the cloud storage. The home

hub system utilises these functions to read data from the cloud storage to act on. As seen below, the read function from Beebotte's API is used to retrieve the data, and used in Python to act on.

```
#MQTT on_message
def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))
    autorecord = autores.read(limit = 1)
    if(autorecord[0]['data'] == False):
        soundrecord = statres.read(limit = 1)
        if(soundrecord[0]['data'] == True):
            ser.write(b"1")
            print("On")
            time.sleep(5)
        elif(soundrecord[0]['data'] == False):
            ser.write(b"2")
            print("Off")
            time.sleep(5)
        else:
            print("No Match")
```

The data from the Beebotte cloud storage can be viewed using the table widget from the Beebotte dashboards. The table created is seen as below:

Database Table

RESOURCE	DATA	WHEN
autoradiator.temp	26	42 minutes ago
autoradiator.temp	27	a day ago
autoradiator.temp	27	a day ago
autoradiator.temp	27	a day ago
autoradiator.temp	27	a day ago
autoradiator.temp	27	a day ago
autoradiator.temp	27	a day ago
autoradiator.temp	31	a day ago
autoradiator.temp	30	a day ago
autoradiator.temp	31	a day ago

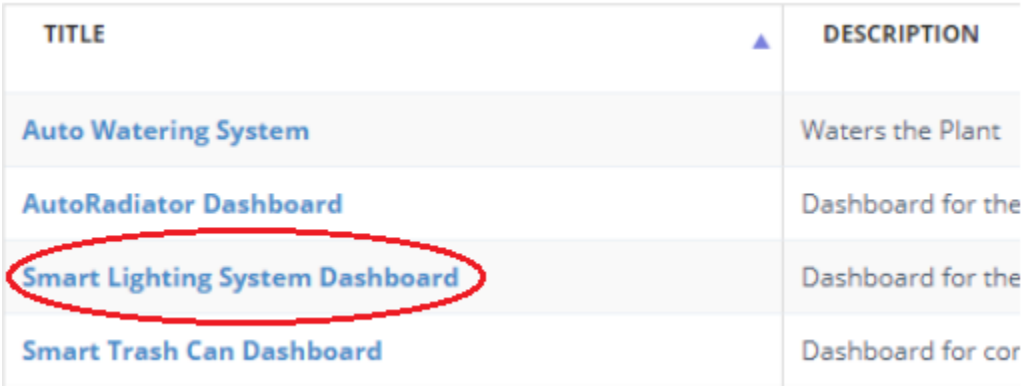
Showing 1 to 10 of 64 entries

Previous 1 2 3 4 5 6 7 Next

7.0 User Manual

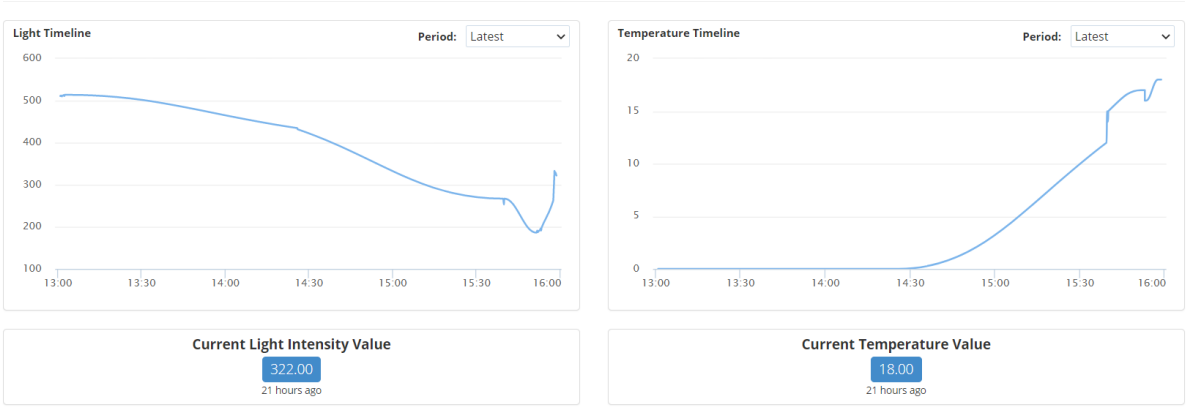
7.1 Smart Lighting System

1. To control the smart lighting system, firstly navigate to the dashboard.

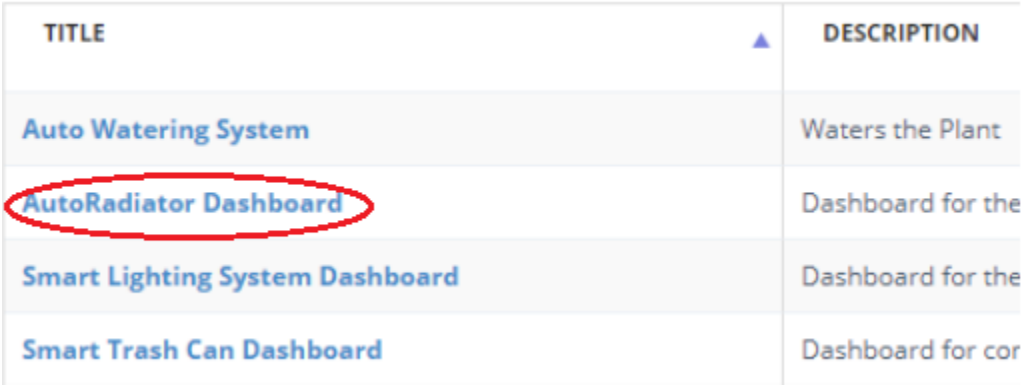


2. There are three ways to activate the light in the system, one being through manual control over the dashboard, the other by automation through sensing the surrounding temperature and light intensity and activating based on the thresholds set.
3. To activate the lights manually, the user will simply need to click on the “Light Button” on the right side as shown below.

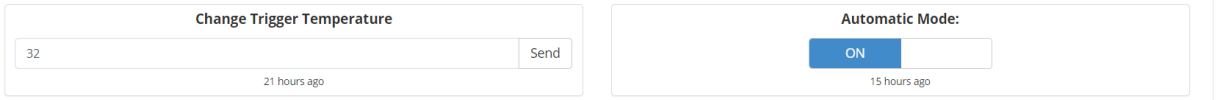
4. Once the button is toggled on, the light will shine up.
5. To turn on automation mode, simply click on the “Light auto on off button” located at the left side. The dashboard will indicate whether the automation mode is on or not through the toggle button’s status.
6. The user can also view the statistics of the light intensity and the temperature on the dashboard.



7.2 Auto Radiator System

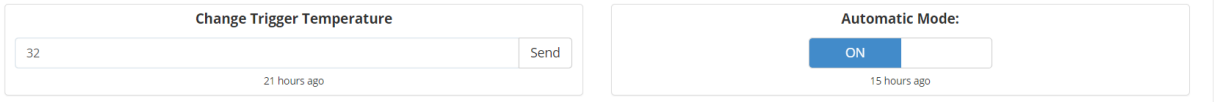
1. The auto radiator has two ways of functioning, one being manually switching on the radiator through sound detection (clapping sound), and the other through automation which will automatically switch on or off based on the current temperature of the surroundings.
2. To control the radiator, simply navigate to the Auto Radiator Dashboard.
3. 

4. Then, at the bottom of the dashboard there are two widgets that the user can interact with as shown below.



5. To enable switching on the radiator manually, simply turn off the automation mode located on the right side. Once the automation mode is off, the user can manually control the radiator through making sounds such as clapping.

6. If the user wants to change back to automation mode, simply click on the “Automatic Mode” button again to turn the automation mode back on.
7. To change the temperature of when the radiator triggers, simply change the textbox value located at the left side as shown below. After clicking on send, the system will change accordingly.



8. The user can also view statistics such as the temperature and on off times through a timeframe. The user can also view the current temperature and current on or off status located at the top of the graphs.

Current Temperature

27.00

15 hours ago

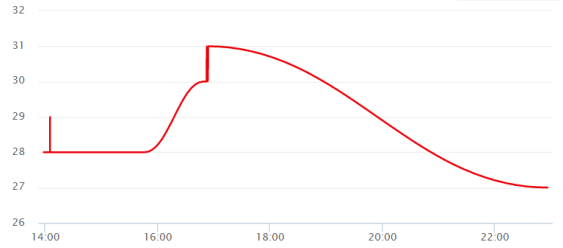
Current On/Off Status

true

15 hours ago

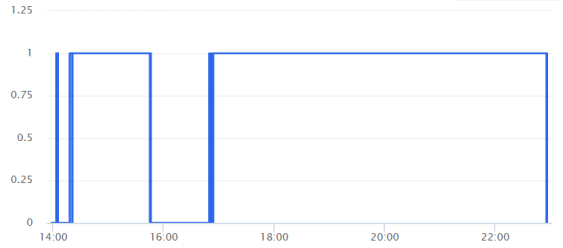
Current Temperature Over Time

Period: Latest



On/Off Over Time

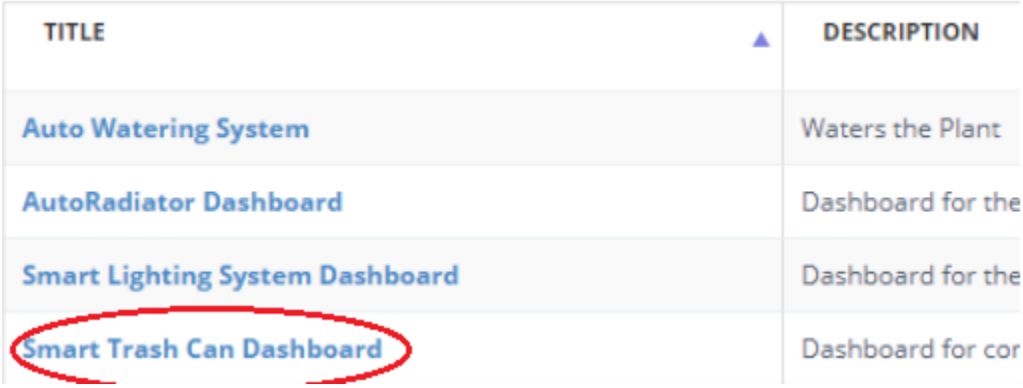
Period: Latest



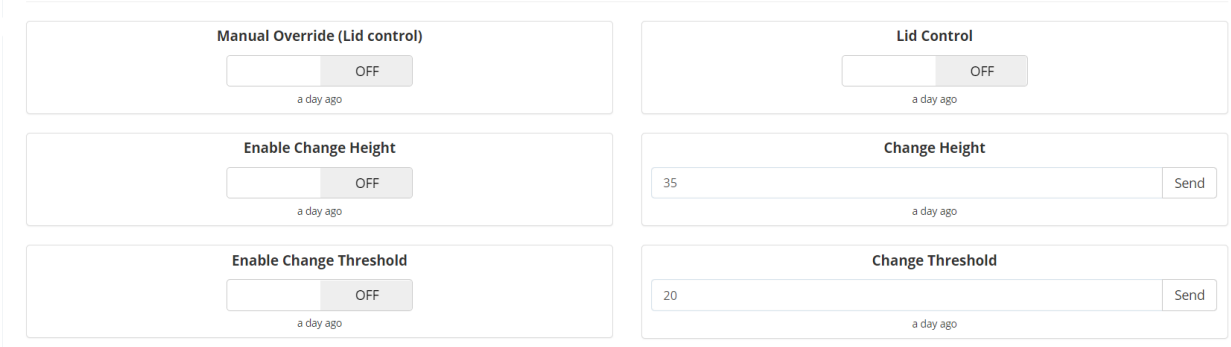
7.3 Smart Trash Can System

1. In order to throw trash away, the user can wave their hands over the motion sensor located at the top of the trash can.

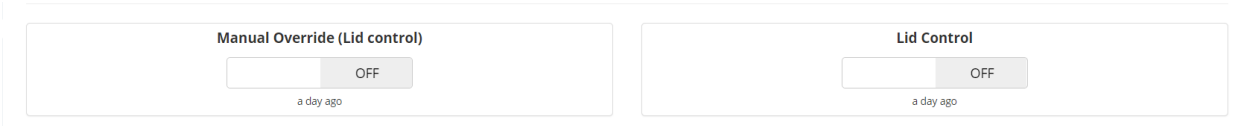
2. The trash can lid will open for 5 seconds, leaving time for the user to throw away the trash.
3. After the 5 seconds are up, the trash lid will close.
4. If the trash is over a certain limit or threshold, the LED light will light up telling the user to throw away the trash.
5. To collect trash, simply scan the RFID card at the scanner located at the back of the trash can. Once successfully scanned, the lid will be opened for the user to empty out all the trash.
6. Once the user finishes emptying out the trash, simply scan the card again to close the lid.
7. To control the trash can through a web interface, firstly navigate to the dashboard in BeeBotte.



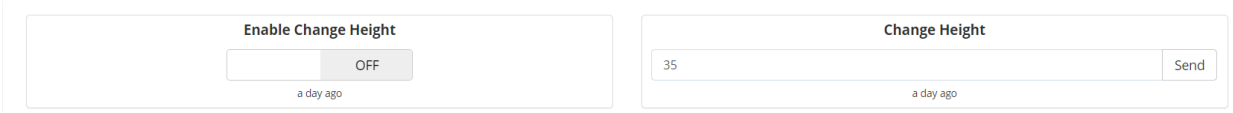
8. In the dashboard, the user can manually control the opening and closing of the lid, change the height factor of the trash can and change the trigger level of trash to be accumulated.



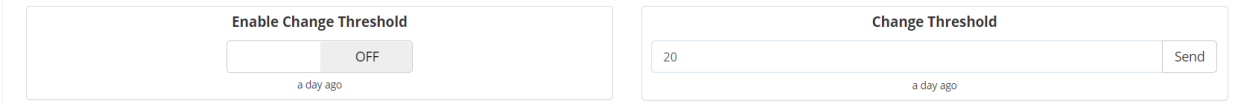
9. To control the opening and closing of the lid, the user is to turn on the manual override button on the left to put the system into manual override mode.



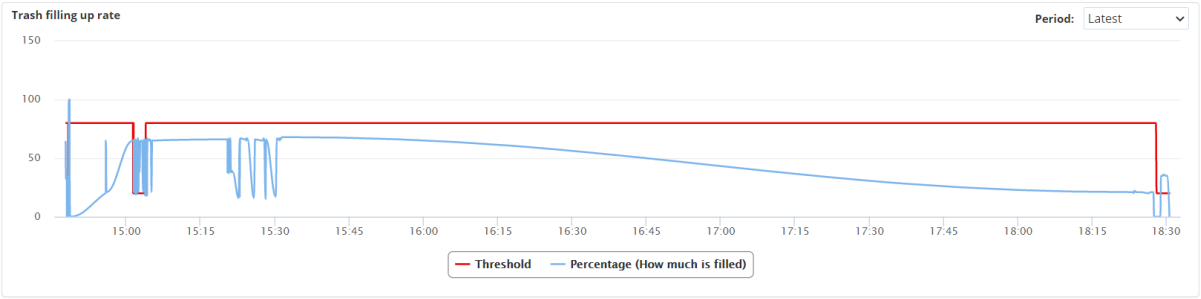
10. After that, the user can control the lid by toggling the lid control button on the right to be on for open and off for close.
11. If the trash can that the user is using is of a different height, the user can change its height throw toggling on the “Enable Change Height” button as show below:



12. After toggling on the “Enable Change Height” button, the user can freely change the height of the system through the text box located at the right side as shown in the above screenshot.A number will be pre-filled to let the user know what is the current value of the height. After inputting a number, the user can click send to send the data into the system.
13. To change the threshold of the system, the user may navigate to the “Enable Change Threshold” section as shown in the below screenshot.



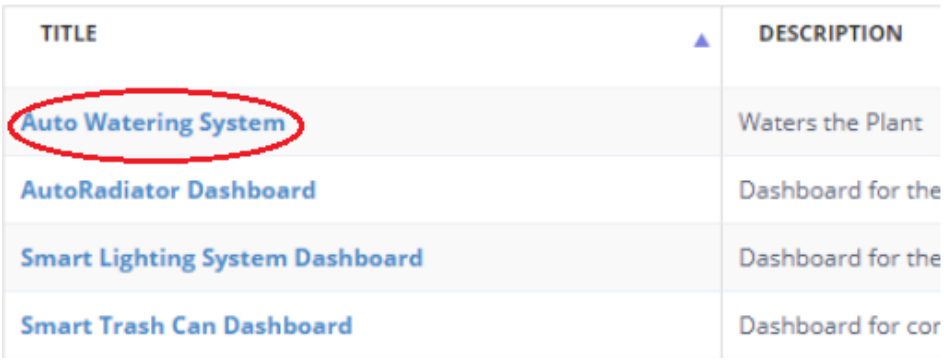
14. After enabling the change threshold button, the user can freely change the threshold through the text box located at the right side. A number will be pre-filled to let the user know what is the current value of the threshold. Once a number is inputted, click on send to send the data into the system.
15. The user can also view the rate of which the trash is filled up and emptied as shown in the screenshot below.



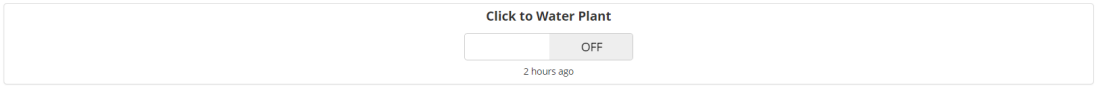
16. The red line shows the threshold value while the blue line shows the percentage of how full the trash is.

7.4 Auto Watering System

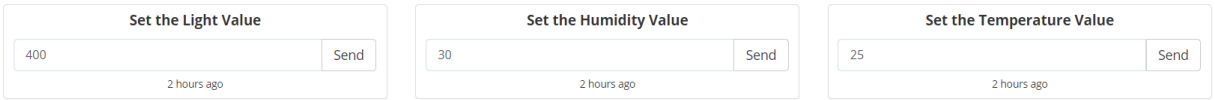
1. The majority of the system is automatic without needing any external user input on the hardware side.
2. The watering system will detect the amount of light, humidity and temperature of the surrounding and water of the plant based on the data it collects.
3. Most manual things can be done through the dashboard.
4. Navigate to the Auto Watering Dashboard



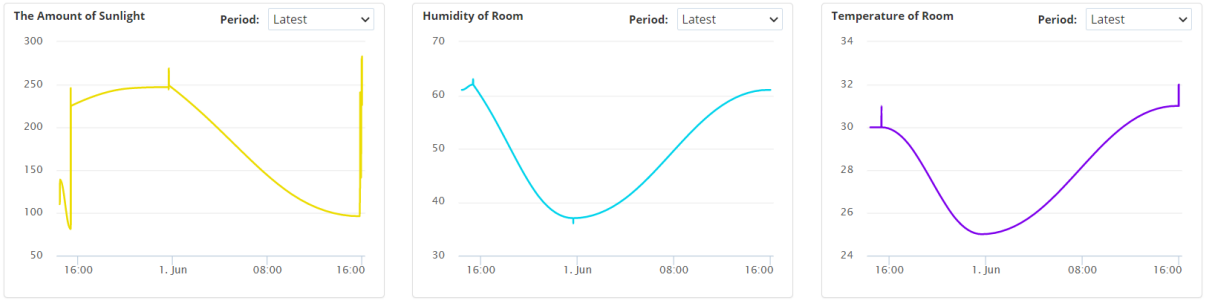
5. To manually activate the watering function, scroll to the bottom and click on the button as shown below.



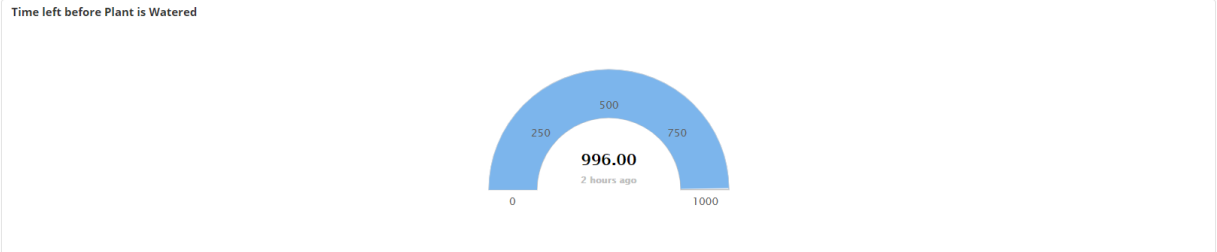
6. Once clicked, the button will be toggled on and the system will water the plant. After the system is done watering, the button will revert back to the initial off state.
7. To change the threshold values for the light, humidity and temperature sensor, simply navigate to the section as shown in the screenshot below and change the values inside the textbox and click on send.



8. The user can also view the statistics of the amount of sunlight, humidity and the temperature through the graphs in the dashboard.



9. A time is also supplied to tell the user when the next watering is.



8.0 Limitations

Storage Issues

Inbound communications are now limited to 16 KB in size by Beebotte. A message is defined as a 1KB block of publish/subscribe data. Each block of 64KB outbound data is billed as 1 message in the read API. MQTT and Websockets connect, disconnect, subscribe, and unsubscribe calls are also considered messages.

Messaging Limitations

Using the free version of Twilio's SMS RESTful is a major limitation to communication, with only the bound number being able to receive SMS messages on the system. This can be rectified by purchasing Twilio's subscription service, which allows for unlimited SMS numbers and the ability to choose the system's own fixed number to send to. Other than that, the messaging is also capped at US$15 for the duration of the project as part of Twilio's testing version, which can be fixed by subscribing to Twilio.

The use of Delays in Python Scripts

To not overload Beebotte with data, delays had to be added to the Python Scripts, which would stop the whole of the code from running, which is a major limitation as it is not as concurrent and fluid as actual IoT systems.

Hardware Limitations

The team was unable to obtain all the necessary hardware for the intended system and therefore had to resort to using substitutes to simulate the result. For example, in the auto radiator, the team does not have an actual radiator to make use of and therefore has to be substituted with a light bulb to simulate the actuation of producing heat. Another example would be the watering system where the servo motor is used to simulate the watering process.

9.0 Resources

9.1 Software

- Arduino IDE
- VNC Viewer
- Thonny Python IDE
- Beebotte Cloud Service

9.2 Online Tutorials and Guides

‘Arduino lesson – Sound detection sensor’ 2017, *Osoyoo*, 26 July, viewed 1 June 2021, <http://osoyoo.com/2017/07/26/arduino-lesson-sound-detection-sensor/#:~:text=On%20the%20top%20of%20the,starts%20blinking%20with%20the%20rhythm.>.

Beebotte at a glance 2015, Beebotte, 15 January, viewed 1 June 2021, <https://beebotte.com/overview>.

Beebotte MQTT support 2015, Beebotte, 22 September, viewed 1 June 2021, <https://beebotte.com/docs/mqtt>.

Bright Sparks NZ 2014, ‘Flush Serial buffer’, *Raspberry Pi Forums*, 2 July, viewed 25 May 2021, <https://www.raspberrypi.org/forums/viewtopic.php?t=80857>.

Chris, 2015, “Using a TMP36 temperature sensor with Arduino”, BC Robotics, 7 July, viewed 29 May 2021, <https://bc-robotics.com/tutorials/using-a-tmp36-temperature-sensor-with-arduino/>.

doniyor 2013, ‘Accessing JSON elements’, *Stack Overflow*, 21 April, viewed 26 May 2021, <https://stackoverflow.com/questions/16129652/accessing-json-elements/46306694>.

‘Guide for Relay Module with Arduino’ 2016, *Random Nerd Tutorials*, 21 December, viewed 1 June 2021, <https://randomnerdtutorials.com/guide-for-relay-module-with-arduino/>.

Harkins, O, 2014, “How do I split an incoming string?”, 29 April, viewed on 29 May 2021,<https://arduino.stackexchange.com/questions/1013/how-do-i-split-an-incoming-string>.

Haslam, Z 2017, ‘DHT11 Temperature/Humidity Sensor’, *Project Hub*, viewed 20 May 2021, <https://create.arduino.cc/projecthub/techno_z/dht11-temperature-humidity-sensor-98b03b>.

Led Control n.d., Beebotte, viewed 28 May 2021, <https://beebotte.com/tutorials/led_control>.

Monitor temperature and humidity with Beebotte 2015, Beebotte, 15 January, viewed 1 June 2021, <https://beebotte.com/tutorials/monitor_humidity_and_temperature_with_raspberrypi>.

Read data 2015, Beebotte, 22 September, viewed 1 June 2021, <https://beebotte.com/docs/read>.

REST API 2015, Beebotte, 23 September, viewed 1 June 2021, <https://beebotte.com/docs/restapi#:~:text=beebotte%20implements%20a%20REST%20API,providing%20thus%20a%20convenient%20wrapper.>.

Riggs, B 2020, ‘Use Python to send notifications to your phone during model training’, *Towards Data Science*, 24 November, viewed 1 June 2021, <https://towardsdatascience.com/use-python-to-send-notifications-to-your-phone-during-model-training-123a9aa257a2>.

10.0 Appendix

10.1 Smart Lighting System

10.1.1 a3.ino

```
const int ledPin = 2;
const int ldrPin = A0;
const int tmpPin = A1;
float temp;
unsigned long previousTime = 0;
const unsigned long eventInterval = 10000;
char buffer[30];

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(ldrPin, INPUT);
  pinMode(tmpPin, INPUT);
}

void loop() {
  unsigned long currentTime = millis();
  if (currentTime-previousTime >= eventInterval){
    int lightData = analogRead(A0);
    int tempData = analogRead(A1);
    tempData = -(tempData * 110)/1023;
    //Serial.println(lightData);
    //Serial.println(tempData);

    sprintf(buffer,"IOT Reading (ldr,tmp): %04d,%04d",lightData,tempData);
    Serial.println(buffer);
    previousTime=currentTime;
  }

  int ldrStatus = analogRead(A0);
  int tmpStatus = analogRead(A1);

    if (Serial.available()>0){
     int lightStatus = Serial.parseInt();
     switch (lightStatus)
     {
      case 1:
        digitalWrite(2, HIGH);
        break;
      case 2:
```

```
        digitalWrite(2, LOW);
        break;
      default:
         break;
        }
      }
}
```

10.1.2 a3.py

```
import serial
import time
import paho.mqtt.client as mqtt
import json

from twilio.rest import Client
from beebotte import *

# PERSONAL TESTING api key and secret key
# API_KEY_AUTOLIGHT = 'gTYdN2z6boA2abUcyBjwodf1'
#SECRET_KEY_AUTOLIGHT = 'DvkUcfpV3fikrkowpjJaaVubOrQpXjor'

# GROUP api key and secret key
API_KEY_AUTOLIGHT = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY_AUTOLIGHT = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

#Twilio SMS REST Information
account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

# Setting the resources
bbt = BBT(API_KEY_AUTOLIGHT, SECRET_KEY_AUTOLIGHT)
light_resourse = Resource(bbt, 'Smart_Lighting_System', 'light_intensity_value')
light_button = Resource(bbt, 'Smart_Lighting_System', 'light_onoff')
light_input = Resource(bbt, 'Smart_Lighting_System', 'light_input')
light_auto_onoff = Resource(bbt, 'Smart_Lighting_System', 'light_auto_onoff')
temp_resourse = Resource(bbt, 'Smart_Lighting_System', 'temp_value')
temp_input = Resource(bbt, 'Smart_Lighting_System', 'temp_input')
ser = serial.Serial('/dev/ttyACM0', 9600)

def on_connect(client, data, flags, rc):
    client.subscribe("Smart_Lighting_System/light_intensity_value")
    client.subscribe("Smart_Lighting_System/light_onoff")
    client.subscribe("Smart_Lighting_System/light_input")
    client.subscribe("Smart_Lighting_System/light_auto_onoff")

    client.subscribe("Smart_Lighting_System/temp_value")
    client.subscribe("Smart_Lighting_System/temp_input")


def on_message(client, data, msg):

    count=0

    print(msg.topic + " " + str(msg.payload))
    read_light_auto_button_onoff = light_auto_onoff.read(limit=1)
```

```
    if (read_light_auto_button_onoff[0]['data']==False):
        read_light_button_onoff = light_button.read(limit=1)
        if(read_light_button_onoff[0]['data'] == True):
            ser.write(b"1")
            print("Manual On")
        elif(read_light_button_onoff[0]['data'] == False):
            ser.write(b"2")
            print("Manual Off")
        else:
            print("No Match")
        time.sleep(5)

    elif(read_light_auto_button_onoff[0]['data']==True):
        read_light_input = light_input.read(limit=1)
        read_light_resourse = light_resourse.read(limit=1)

        read_temp_input = temp_input.read(limit=1)
        read_temp_resourse = temp_resourse.read(limit=1)

        if(read_light_resourse[0]['data'] <= read_light_input[0]['data'] and
read_temp_resourse[0]['data'] <= read_temp_input[0]['data']):
            ser.write(b"1")

            count+=1
            if(count==1):
                # SMS script
                smsclient = Client(account_sid, auth_token)
                mailbody = ("We detect the light is on")
                message = smsclient.messages.create(body=mailbody, from_ =
"+12018905970",
                to = "+601110679126")

            print("Auto On")

        elif(read_light_resourse[0]['data'] >read_light_input[0]['data'] and
read_temp_resourse[0]['data'] >read_temp_input[0]['data']):
            ser.write(b"2")
            count=0
            print("Auto Off")
        else:
            print("No Match")
        time.sleep(5)
    else:
        print(read_light_auto_button_onoff[0]['data'])


# Setup MQTT Client
client = mqtt.Client()
```

```
client.on_connect = on_connect
client.on_message = on_message

# PERSONAL TESTING token
#client.username_pw_set("token:token_WbenCRZsqINb08va")

# GROUP token
client.username_pw_set("token:token_CmG3yJSa0ZaiaPKs")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()

while True:
    if ser.in_waiting > 0:

        ln = ser.readline()
        #light=int(ln);
        #temp=int(ln);

        # Reading data
        light=int(ln[23:27])
        temp=int(ln[28:32])
        #print(light)
        #print(temp)

        light_dictionary={"data":light,"write":True}
        temp_dictionary={"data":temp,"write":True}

        light_json = json.dumps(light_dictionary)
        temp_json = json.dumps(temp_dictionary)

        client.publish("Smart_Lighting_System/light_intensity_value", light_json)
        client.publish("Smart_Lighting_System/temp_value", temp_json)
```

10.2 Auto Radiator

10.2.1 Assignment3.ino

```
#include <dht.h>
dht DHT;

//Initiate variables
unsigned long previousTime = 0;
unsigned int temp = 0;
int count = 0;
unsigned int soundData;
```

```
const unsigned long eventInterval = 10000;
unsigned int input = 0;
String stat = "off";

#define HTSENSOR 7
#define RELAY 10
//Setup Script
void setup() {
  Serial.begin(9600);
  pinMode(RELAY, OUTPUT);
  pinMode(HTSENSOR, INPUT);
  digitalWrite(10, LOW); //Set relay default to off
}

void loop() {
  //Data Collection
  unsigned long currentTime = millis();
  int soundData = analogRead(A0);
  //If there is a clap here, change state and send serial data
  if(soundData > 200)
  {
    //Serial.println(soundData);
    if(stat == "off")
    {
      stat = "on";
    }
    else if(stat == "on")
    {
      stat = "off";
    }
    int chk = DHT.read11(HTSENSOR);
    temp = DHT.temperature;
    Serial.print("<");
    Serial.print(stat);
    Serial.print(",");
    Serial.print(temp);
    Serial.println(">");
  }

  //Standard time based data collection
  if (currentTime - previousTime >= eventInterval) //if 10 seconds have passed
  {
    int chk = DHT.read11(HTSENSOR);
    temp = DHT.temperature;
    Serial.print("<");
    Serial.print(stat);
    Serial.print(",");
    Serial.print(temp);
```

```
    Serial.println(">");
    previousTime = currentTime;
  }

  //Control
  if(Serial.available() > 0)
  {
    input = Serial.parseInt();
    switch(input)
    {
      case 1:
        digitalWrite(RELAY, HIGH);
        stat = "on";
        break;
      case 2:
        digitalWrite(RELAY, LOW);
        stat = "off";
        break;
      default:
        break;
    }
  }
}
```

10.2.2 Assignment3.py

```
import serial
import time
import paho.mqtt.client as mqtt
import json

from twilio.rest import Client
from beebotte import *

#beebotte Keys
API_KEY = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

#Twilio SMS REST Information
account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

#Setting up the resources
bbt = BBT(API_KEY, SECRET_KEY)
statres = Resource(bbt, 'autoradiator', 'onoff')
tempres = Resource(bbt, 'autoradiator', 'temp')
autores = Resource(bbt, 'autoradiator', 'autobutton')
thresholdres = Resource(bbt, 'autoradiator', 'threshold')

ser = serial.Serial('/dev/ttyUSB0', 9600)

#MQTT on_connect
def on_connect(client, data, flags, rc):
    client.subscribe("autoradiator/onoff")
    client.subscribe("autoradiator/temp")
    client.subscribe("autoradiator/autobutton")
    client.subscribe("autoradiator/threshold")

#MQTT on_message
def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))
    autorecord = autores.read(limit = 1)
    if(autorecord[0]['data'] == False):
        soundrecord = statres.read(limit = 1)
        if(soundrecord[0]['data'] == True):
            ser.write(b"1")
            print("On")
            time.sleep(5)
        elif(soundrecord[0]['data'] == False):
            ser.write(b"2")
            print("Off")
            time.sleep(5)
```

```
        else:
            print("No Match")
    elif(autorecord[0]['data'] == True):
        #Automation Script
        #Get the threshold data, and set it turn on and off accordingly
        thresholdrecord = thresholdres.read(limit = 1)
        temprecord = tempres.read(limit = 1)
        if(temprecord[0]['data'] < thresholdrecord[0]['data']):
            ser.write(b"1")
            print("Auto On")
            time.sleep(5)
        elif(temprecord[0]['data'] >= thresholdrecord[0]['data']):
            ser.write(b"2")
            print("Auto Off")
            time.sleep(5)
        else:
            print("No Match")

#Set up MQTT Client
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

#Setting channel configurations
client.username_pw_set("token:token_rSnUqiQ9WIZ4MSd2")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()
oristatus = "off"
while True:
    #Collects data if serial is in waiting
    if ser.in_waiting > 0:
        ln = ser.readline()
        #Cleans the input and separates
        ln_refined = ln.decode("utf-8").strip('\r\n')
        lnlist = ln_refined.split(",")
        status = lnlist[0].strip("<")
        temperature = lnlist[1].strip(">")
        temp = int(temperature)
        #Prepares the input for beebotte
        if(status == "on"):
            boolstat = True
        elif(status == "off"):
            boolstat = False

        #If the current status is not the original status, send an SMS
        if(status != oristatus):
            smsclient = Client(account_sid, auth_token)
```

```
            mailbody = "Status of AutoRadiator changed from " + oristatus + " to " +
status
            message = smsclient.messages.create(body = mailbody,
                                    from_ = "+12018905970",
                                    to = "+601110679126")
            message.sid
            oristatus = status

        #Convert data to dict then JSON for beebotte
        booldict = {"data": boolstat, "write": True}
        tempdict = {"data": temp, "write": True}
        booljson = json.dumps(booldict)
        tempjson = json.dumps(tempdict)

        #Publish the Data
        client.publish("autoradiator/onoff", booljson)
        client.publish("autoradiator/temp", tempjson)
```

10.3 Smart Trash Can

10.3.1 bin_arduino_Assignment3.ino

```
#include <Servo.h>
Servo servoMain;
int trigpin = 4;
int echopin = 3;
int distance;
float duration;
float cm;
#include "SPI.h"
#include "MFRC522.h"

unsigned int pinStatus = 0;
unsigned int thresholdPercentage = 0;
int currentThreshold = 20;
int height = 45;
int displayPercentage = 0;
int displayThreshold = 0;
#define SS_PIN 10
#define RST_PIN 9
#define SP_PIN 8
#define LED 2
#define PIR 6

MFRC522 rfid(SS_PIN, RST_PIN);

MFRC522::MIFARE_Key key;

void setup() {
  servoMain.attach(7);
  pinMode(trigpin, OUTPUT);
  pinMode(echopin, INPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  pinMode(LED,OUTPUT);
  digitalWrite(LED,LOW);
  pinMode(PIR, INPUT);
  servoMain.write(180);
}

void loop() {
  if (digitalRead(PIR) == HIGH && servoMain.read()!= 0){
    servoMain.write(0);
    delay(5000);
```

```
  servoMain.write(180);
  readDistance();

}
if (Serial.available() > 0)
{
  String data = Serial.readStringUntil('\n');

  String firstWord = getValue(data, ' ', 0);
  String secondWord = getValue(data, ' ', 1);
  if (firstWord == "S"){
    if (secondWord == "1"){
      servoMain.write(0);
    }else if (secondWord == "2"){
      servoMain.write(180);
      readDistance();
    }
  }else if (firstWord == "T"){
    currentThreshold = 100 - secondWord.toInt();
    readDistance();
  }else if (firstWord == "H"){
    height = secondWord.toInt();
    readDistance();
  }
}
if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial())
  return;

MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);

if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
  piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
  piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
  Serial.println(F("Your tag is not of type MIFARE Classic."));
  return;
}

String strID = "";
for (byte i = 0; i < 4; i++) {
  strID +=
  (rfid.uid.uidByte[i] < 0x10 ? "0" : "") +
  String(rfid.uid.uidByte[i], HEX) +
  (i!=3 ? ":" : "");
}
strID.toUpperCase();
```

```
  if (strID != "" && servoMain.read() == 0){
    Serial.print(strID);
    Serial.print(" ");
    Serial.print(displayPercentage);
    Serial.print(" ");
    Serial.println(displayThreshold);
    servoMain.write(180);
    readDistance();
  }else if(strID != "" && servoMain.read() == 180){
    servoMain.write(0);
  }
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}

void readDistance(){
  digitalWrite(trigpin, LOW);
  delay(2);
  digitalWrite(trigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigpin, LOW);

  duration = pulseIn(echopin, HIGH);

  cm = (duration / 58.82);

  distance = cm;
  int currentPercentage = cm / height*100;
  displayPercentage = 100 - currentPercentage;
  displayThreshold = 100 - currentThreshold;

  if (displayPercentage < 0){
    displayPercentage = 0;
  }
   if (displayThreshold < 0){
    displayThreshold = 0;
  }
   if (height <0){
    height = 0;
  }
  Serial.print(displayPercentage);
  Serial.print(" ");
  Serial.print(displayThreshold);
  Serial.print(" ");
  Serial.println(height);
  if (currentPercentage <= currentThreshold){
    digitalWrite(LED, HIGH);
  }else{
```

```
    digitalWrite(LED, LOW);
  }
}

String getValue(String data, char separator, int index)
{
  int found = 0;
  int strIndex[] = {0, -1};
  int maxIndex = data.length()-1;

  for(int i=0; i<=maxIndex && found<=index; i++){
    if(data.charAt(i)==separator || i==maxIndex){
        found++;
        strIndex[0] = strIndex[1]+1;
        strIndex[1] = (i == maxIndex) ? i+1 : i;
    }
  }

  return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

10.3.2 binLog.py

```
import serial
import time
import json
import paho.mqtt.client as mqtt

from beebotte import *
from twilio.rest import Client

API_KEY = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

bbt = BBT(API_KEY, SECRET_KEY)
user = Resource(bbt, 'Smart_Trash_Can', 'username')
height = Resource(bbt, 'Smart_Trash_Can', 'height')
percentage = Resource(bbt, 'Smart_Trash_Can', 'percentage')
threshold = Resource(bbt, 'Smart_Trash_Can', 'threshold')
lid = Resource(bbt, 'Smart_Trash_Can', 'lidControl')
collect = Resource(bbt, 'Smart_Trash_Can', 'collect')
manual = Resource(bbt, 'Smart_Trash_Can', 'manual')
changeThreshold = Resource(bbt, 'Smart_Trash_Can', 'changeThreshold')
changeHeight = Resource(bbt, 'Smart_Trash_Can', 'changeHeight')

arduino = serial.Serial('/dev/ttyACM0', 9600)
def on_connect(client, data, flags, rc):
    client.subscribe("Smart_Trash_Can/username")
    client.subscribe("Smart_Trash_Can/height")
    client.subscribe("Smart_Trash_Can/percentage")
    client.subscribe("Smart_Trash_Can/threshold")
    client.subscribe("Smart_Trash_Can/lidControl")
    client.subscribe("Smart_Trash_Can/collect")
    client.subscribe("Smart_Trash_Can/manual")
    client.subscribe("Smart_Trash_Can/changeThreshold")
    client.subscribe("Smart_Trash_Can/changeHeight")

def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))
    manualOverride = manual.read(limit = 1)
    changeHeightOverride = changeHeight.read(limit = 1)
    changeThresholdOverride = changeThreshold.read(limit = 1)
    if (manualOverride[0]['data'] == True):
        record = lid.read(limit = 1)
        print(record[0]['data'])
        if(record[0]['data'] == True):
```

```
            arduino.write(b"S 1\n")
        elif(record[0]['data'] == False):
            arduino.write(b"S 2\n")
        else:
            print("No Match")
        arduino.flushOutput()
        time.sleep(5)
    if (changeThresholdOverride[0]['data'] == True):
        thresholdValue = threshold.read(limit=1)
        arduino.write(b"T "+str(thresholdValue[0]['data']).encode()+b"\n")
        arduino.flush()
        time.sleep(5)
    if (changeHeightOverride[0]['data'] == True):
        heightValue = height.read(limit=1)
        arduino.write(b"H "+str(heightValue[0]['data']).encode()+b"\n")
        arduino.flush()
        time.sleep(5)
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.username_pw_set("token:token_KSLevGDpDVcT4dWu")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()

while 1:
    if arduino.in_waiting > 0:
        print(arduino.in_waiting)
        line = arduino.readline().decode("utf-8")
        x = line.split()
        arduino.flushInput()
        arduino.flushOutput()
        print(x)
        if (not(x[0].isdigit())):
            userList = user.read(limit = 2)
            print(userList[0]['data'])
            name = ""
            for users in userList:
                userID = users['data'].split(" ")
                if (userID[0] == str(x[0])):
                    name = userID[1]
            collectDict ={"data":name, "write": True}
            percentageDict ={"data":int(x[1]), "write": True}
            thresholdDict ={"data":int(x[2]), "write": True}

            percentageJSON = json.dumps(percentageDict)
            thresholdJSON = json.dumps(thresholdDict)
```

```
    collectJSON = json.dumps(collectDict)

    client.publish("Smart_Trash_Can/percentage", percentageJSON)
    client.publish("Smart_Trash_Can/threshold", thresholdJSON)
    client.publish("Smart_Trash_Can/collect", collectJSON)

    smsclient = Client(account_sid, auth_token)
    mailbody = name + " has collected the rubbish"
    message = smsclient.messages.create(body = mailbody,
                            from_ = "+12018905970",
                            to = "+601110679126")
    message.sid

    arduino.flushInput()
    arduino.flushOutput()
    time.sleep(5)

else:
    percentageDict ={"data":int(x[0]), "write": True}
    thresholdDict ={"data":int(x[1]), "write": True}
    heightDict ={"data":int(x[2]), "write": True}

    percentageJSON = json.dumps(percentageDict)
    thresholdJSON = json.dumps(thresholdDict)
    heightJSON = json.dumps(heightDict)

    client.publish("Smart_Trash_Can/percentage", percentageJSON)
    client.publish("Smart_Trash_Can/threshold", thresholdJSON)
    client.publish("Smart_Trash_Can/height", heightJSON)
    arduino.flush()
    time.sleep(5)
```

10.4 Auto Plant Watering

10.4.1 groupTo.ino

```
#include <dht.h>
#include <Servo.h>

#define dht_apin 5 //Analog Pin sensor is connected to

dht DHT;
int light;
int input;
unsigned long previousTime = 0;
const unsigned long eventInterval = 10000;
Servo servo_test;

void setup() {
  Serial.begin(9600);
  servo_test.attach(9);
}

void loop() {
  unsigned long currTime = millis();

  if (currTime - previousTime >= eventInterval) //10 Seconds Pass
  {
    DHT.read11(dht_apin);
    light = analogRead(A0);

    Serial.print(light);
    Serial.print(",");
    Serial.print((int)DHT.humidity);
    Serial.print(",");
    Serial.print((int)DHT.temperature);
    Serial.println(",");
    previousTime = currTime;
  }

  if (Serial.available() > 0)
  {
    input = Serial.parseInt();
    switch(input)
    {
      case 1:
        servo_test.write(0);
        delay(10000);
        servo_test.write(100);
```

```
        break;

      default:
        break;
    }
  }
}
```

10.4.2 main_database.py

```
import serial
import time
import json
from beebotte import *
import paho.mqtt.client as mqtt
from twilio.rest import Client

device = '/dev/ttyACM0'
arduino = serial.Serial(device, 9600)
valueLight = ''
lightVal = 0
counter = 1000
COUNT = 1000
thresLight = 400
thresHumid = 30
thresTemp = 25

#Twilio SMS REST Information
account_sid = "ACdf1050604f3971f8fb2e9b8fa02bc924"
auth_token = "355c59fa5b02a66e5c11305d3f40e10d"

smsclient = Client(account_sid, auth_token)
mailbody = ("Water system has been turned on")
message = smsclient.messages.create(body=mailbody, from_ = "+12018905970",
to = "+601110679126")

API_KEY = 'MNdVZntUkkrD4ONHPw070RKX'
SECRET_KEY = '7aRoJUz4hOyrdzojpYhe3WPJcYOQFVEE'

bbt = BBT(API_KEY, SECRET_KEY)

light_re = Resource(bbt, 'Auto_Plant_Watering', 'Light')
temp_re= Resource(bbt, 'Auto_Plant_Watering', 'Humidity')
humid_re= Resource(bbt, 'Auto_Plant_Watering', 'Temperature')
button_re= Resource(bbt, 'Auto_Plant_Watering', 'Water_Plant')
counter_re= Resource(bbt, 'Auto_Plant_Watering', 'Counter')
tLight_re= Resource(bbt, 'Auto_Plant_Watering', 'Threshold_Light')
tHumid_re= Resource(bbt, 'Auto_Plant_Watering', 'Threshold_Humid')
tTemp_re= Resource(bbt, 'Auto_Plant_Watering', 'Threshold_Temp')

def on_connect(client, data, flags, rc):
    client.subscribe("Auto_Plant_Watering/Light")
    client.subscribe("Auto_Plant_Watering/Humidity")
    client.subscribe("Auto_Plant_Watering/Temperature")
    client.subscribe("Auto_Plant_Watering/Water_Plant")
```

```
    client.subscribe("Auto_Plant_Watering/Counter")
    client.subscribe("Auto_Plant_Watering/Threshold_Light")
    client.subscribe("Auto_Plant_Watering/Threshold_Humid")
    client.subscribe("Auto_Plant_Watering/Threshold_Temp")

def on_message(client, data, msg):
    print(msg.topic + " " + str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect

client.username_pw_set("token:token_DXqA4dui7GEZjYFM")
client.connect("mqtt.beebotte.com", 1883, 60)

client.loop_start()

while 1:
    while(arduino.in_waiting == 0):
        pass

    line = arduino.readline()

    values = str(line)[2:].split(",")


    light = values[0]
    humid = values[1]
    temp = values[2][:2]

    # Counter

    thresLight = tLight_re.read(limit= 1)[0]['data']
    if (int(light) > thresLight):
        counter -= 1

    thresHumid = tHumid_re.read(limit= 1)[0]['data']
    if (int(humid) < thresHumid):
        counter -= 1

    thresTemp = tTemp_re.read(limit= 1)[0]['data']
    if (int(temp) > thresTemp):
        counter -= 1

    counter -= 1

    print(str(thresLight) + "," + str(thresHumid) + "," + str(thresTemp))

    print(counter)
```

```
if (counter <= 0):
    arduino.write(b'1')
    message.sid
    counter = COUNT

# Watering Plant

waterIn = button_re.read(limit = 1)
if (waterIn[0]['data'] == True):
    arduino.write(b'1')
    message.sid
    counter = COUNT

lightdict = {"data": int(light), "write": True}
lightjson = json.dumps(lightdict)

humiddict = {"data": int(humid), "write": True}
humidjson = json.dumps(humiddict)

tempdict = {"data": int(temp), "write": True}
tempjson = json.dumps(tempdict)

buttondict = {"data": False, "write": True}
buttonjson = json.dumps(buttondict)

countdict = {"data": counter, "write": True}
countjson = json.dumps(countdict)

tLightdict = {"data": thresLight, "write": True}
tLightjson = json.dumps(tLightdict)

tHumiddict = {"data": thresHumid, "write": True}
tHumidjson = json.dumps(tHumiddict)

tTempdict = {"data": thresTemp, "write": True}
tTempjson = json.dumps(tTempdict)

client.publish("Auto_Plant_Watering/Light", lightjson)
client.publish("Auto_Plant_Watering/Humidity", humidjson)
client.publish("Auto_Plant_Watering/Temperature", tempjson)
client.publish("Auto_Plant_Watering/Water_Plant", buttonjson)
client.publish("Auto_Plant_Watering/Counter", countjson)
client.publish("Auto_Plant_Watering/Threshold_Light", tLightjson)
client.publish("Auto_Plant_Watering/Threshold_Humid", tHumidjson)
client.publish("Auto_Plant_Watering/Threshold_Temp", tTempjson)
```