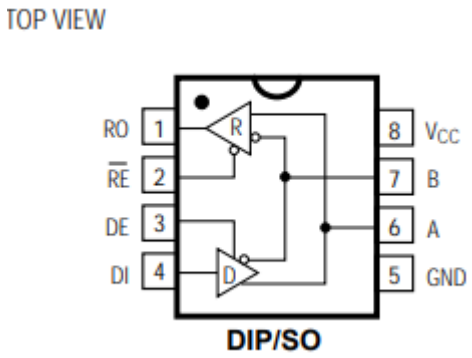
Report

The purpose of the project is to get information from a 7-in-1 soil quality sensor about the parameters of the soil and send them to a IoT platform where the data is processed into graphs, tables. The data is sent first to a microcontroller that controls the data and instruction flow of the sensor. Then, the microcontroller transmits the data via Wi-Fi to a Raspberry Pi that acts as an access point and that has the IoT platform installed on it. A client can see the data if it is connected to the same Wi-Fi network as the Raspberry Pi.

The sensor used is JXBS-3001-NPK-RS[2] that is powered by a supply voltage of 5V and has two channels (A and B) that are used to transmit data and receive instructions from the microcontroller. The sensor though sends the data using the RS485 protocol which leads to a misinterpretation of the data if the microcontroller receives the information directly from the sensor. This is the reason why a transceiver is used between the sensor and the microcontroller. The transceiver used is MAX3483 [3]. It is powered up by 3.3V and it ensures that the data from the sensor is correctly transmitted to the microcontroller in a manner that the last one can interpret the information correctly. In the next figure there can be seen the internal structure of this transceiver:



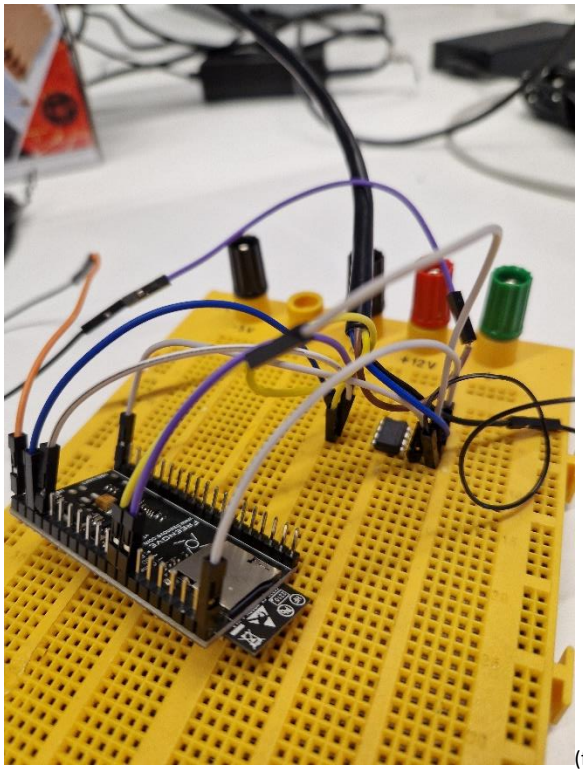
(figure 1)

The pins *RE* and *DE* control the flow (receive/transmit) and on *RO* and *DI* is the actual data that is transmitted/received.

The microcontroller used is an ESP32-S3-WROOM Board [1]. This can be powered up by a 5V voltage supply, it supplies the transceiver and controls the sensor. In the table below there can be seen how the connections with the transceiver are made:

Transceiver pin	*Microcontroller pin*
RE (receive enable)	13
DE (data enable)	12
RO (receive output)	16
DI (data input)	17

The next figure shows the connections made:



(figure 2)

The code [2] to program the microcontroller is written in C language using Arduino IDE. The main idea is that the microcontroller will send to the sensor a request for data for every parameter: nitrogen, phosphorus, potassium, ph, moisture, temperature, electrical conductivity. A string of bytes (that is found in the datasheet of the sensor) will be send as a data request for each parameter and as response the microcontroller also receives a string of bytes that needs to be interpreted by the code in order to find the value of each parameter. For example, for nitrogen the following format of the string of bytes is given in datasheet [2]:

Inquiry frame

Address Code	Function Code	Register Start address	Register Length	CRC_L	CRC_H
0x01	0x03	0x00 0x1e	0x00 0x01	0XE4	0x0C

Answer Frames (e.g. reading that nitrogen content is 32 mg/kg)

Address Code	Function Code	Effective Number of bytes	Nitrogen Value	CRC_L	CRC_H
0x01	0x03	0x02	0x00 0x20	0Xb9	0x9C

(figure 3)

Consequently, there have been initialized seven constant byte strings for each inquiry for the parameters (*nitro[]* for this example) and seven functions that return the value for each parameter. Next, as example, the function for nitrogen (all the other functions are similar with it):

```
int nitrogen() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(nitro, sizeof(nitro)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  serialFlush();
  delay(100);
  return inter;
}
```

(figure 4)

In this function there can be seen that the lines *DE* and *RE* need to be put *HIGH* to transmit the string of bytes to the sensor and after the transmission is finished, they are put back in *LOW. SerialPort* refers to the lines through which the transmission/reception is made, so *RO* and *DI*. After that, each byte that sends the sensor is read and the format is also checked (if the first byte is 0x01, according to the answer frame from the datasheet). A second check is also made, verifying if the first two bytes correspond to the answer frame and if not, the reception is repeated. From the answer frame there can be seen which bytes represent the value of the parameter. In this case, there is a high byte and a low byte, so a "concatenation" is made in the local variable *inter* that is returned by the function. A flush of the buffer needs to be made before leaving the function, to prepare it for the next transmission. The function *serialFlush()* is presented below:

```
void serialFlush(){//function for flushing the buffer
  while(SerialPort.available() > 0)
    char t = SerialPort.read();
}
```

(figure 5)

Next, the connection between the microcontroller and the Raspberry Pi is made. First, a Raspberry Pi 4 B has been configured [7], so an OS has been installed on it (in this case, Raspbian has been installed [8]). It has followed its configuration as an access point, fixing the Wi-Fi credentials [9]. In the code, the Wi-Fi connection of the microcontroller has been also introduced using the specific library *WiFi.h* [5]. The Wi-Fi credentials of the network (SSID and password) emitted by the Raspberry Pi were set:

```
// declaration of wi-fi credentials
const char* wifiSSID = "V9061";
const char* wifiPassword = "3*04Xe91";
```

(figure 6)

And the function *connectWiFi()* realizes the connection to this network:

```
void connectWifi() {
  Serial.println("Connecting To Wifi");
  WiFi.begin(wifiSSID, wifiPassword);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("Wifi Connected");
  /*Serial.println(WiFi.SSID());
  Serial.println(WiFi.RSSI());
  Serial.println(WiFi.macAddress());
  Serial.println(WiFi.localIP());
  Serial.println(WiFi.gatewayIP());
  Serial.println(WiFi.dnsIP());
  */

}
```

(figure 7)

In the next step, *Thingsboard* (IoT platform) needs to be installed on the Raspberry Pi. To do this, *Docker* needs to be installed first in order to ensure a more efficient way of managing large data sets, as is the case in this project [10]. After finishing installing *Docker*, *Thingsboard* will be installed [11]. With the platform installed on the Raspberry Pi, the code needs to be changed to facilitate the transmission to the platform using the MQTT protocol [4].

The MQTT protocol is based practically on 3 pillars: publisher (which is the ensemble formed by the sensor and the microcontroller), broker (which is the IoT platform) and subscriber (client). So, to establish the transmission to *Thingsboard* using MQTT, the specific libraries *ThingsBoard.h* and *TBPubSubClient.h* need to be used [5]. The Thingsboard credentials are also set (*tbHost* – IP address where the platform is located and *tbToken* – token of the device used, sensor), as well as 2 specific objects need to be created:

```
// Thingsboard credentials
String tbHost = "demo.thingsboard.io";//adresa ip raspberry pi + port
String tbToken = "z75yg74YU1TBB4pMC1ld";

WiFiClient client;
ThingsBoard tb(client);
PubSubClient mqtt(client);
```

(figure 8)

These objects are used further for the connection with the server through the MQTT-specific port 1883:

```
connectWifi();
mqtt.setServer(tbHost.c_str(), 1883);
mqtt.setCallback(on_message);
```

(figure 9)

The *on_message()* function is used when the platform sends a message to the microcontroller (which is not the case for this project). There can be seen that a JSON string is received (as this is the format through which messages are transmitted via MQTT) and decomposed in order to be displayed [5]:

```
void on_message(const char *topic, byte *payload, unsigned int length) {
  Serial.println("On message");
  char json[length + 1];
  strncpy(json, (char *)payload, length);
  json[length] = '\0';

  Serial.print("Topic: ");
  Serial.println(topic);
  Serial.print("Message: ");
  Serial.println(json);
}
```

(figure 10)

The *tbReconnect()* is called in the main *loop()* function and ensures that the connection to the platform is preserved and if not, it tries to reconnect to the platform. In such way, there can be avoided transmission of data if the platform is not available:

```
void tbReconnect(){
  while (!tb.connected()){
    if (WiFi.status() != WL_CONNECTED)
      connectWifi();

    Serial.println("connecting to thingsboard ... ");
    if (tb.connect(tbHost.c_str(), tbToken.c_str())){
      Serial.println("Thingsboard Connected!");
      }
    else{
      Serial.println("Thingsboard connection failed");
      Serial.println("Retrying in 5 seconds...");
      delay(5000);
    }
  }
}
```

(figure 11)

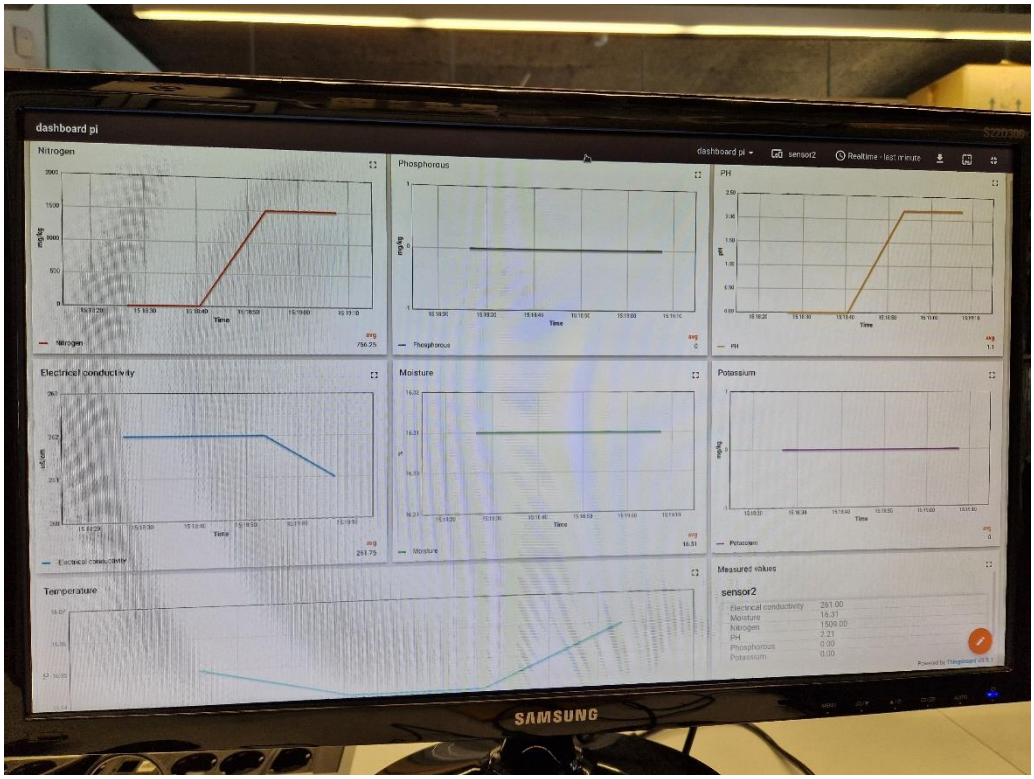
The effective transmission is made after all the 7 functions have been executed and each function has returned in a local variable the value of the parameter. The transmission to *Thingsboard* is made using the function *sendTelemetry(),* specific to the Thingsboard *tb* object [5].

```
Serial.println("Sending data to Thingsboard");

tb.sendTelemetryInt("Nitrogen", nitrovar);
tb.sendTelemetryInt("Phosphorous", phosvar);
tb.sendTelemetryInt("Potassium", potavar);
tb.sendTelemetryFloat("PH", phvar);
tb.sendTelemetryFloat("Moisture", moistvar);
tb.sendTelemetryFloat("Temperature", tempvar);
tb.sendTelemetryInt("Electrical conductivity", ecvar);
tb.loop();
mqtt.loop();
```

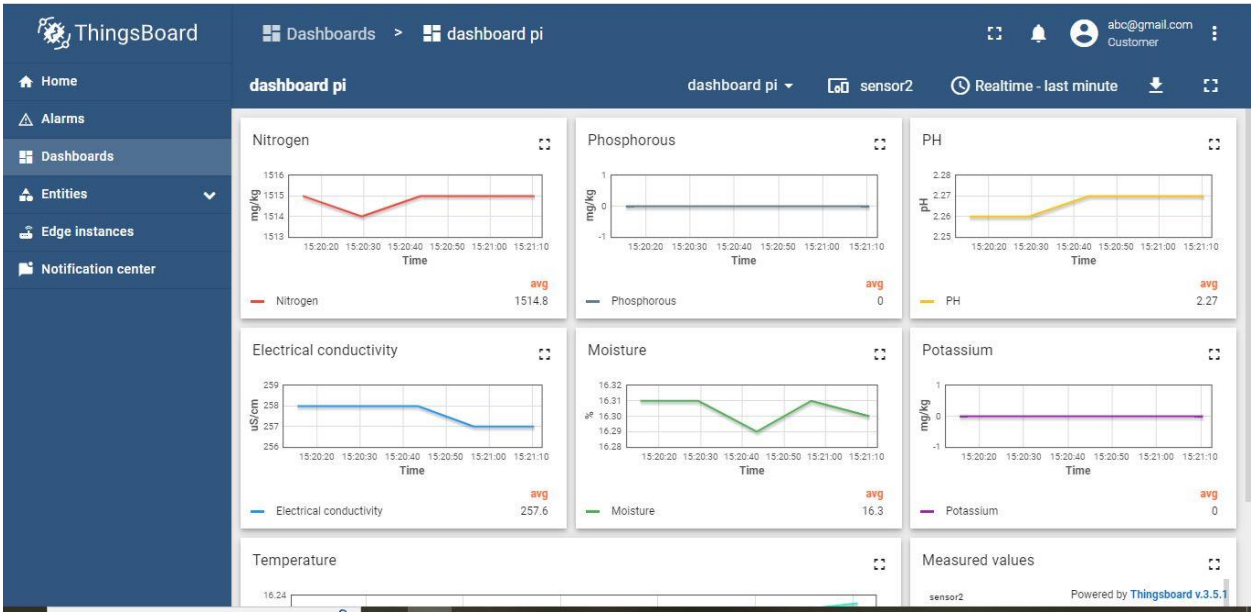
(figure 12)

Afterwards, the *Thingsboard* platform on the Raspberry Pi is configured [6], assigning an administrator, dashboard, device. As the platform runs on the localhost, the *tbHost* credential needs to be changed to the localhost IP address of the Raspberry Pi (can be found by running the command *ifconfig* and the *eth0* IP address represents the address of the localhost). After configuration, a client has been added and a dashboard assigned to him. The client needs to be connected to the same Wi-Fi network as the Raspberry Pi emits. The next figure shows how Thingsboard from the tenant point of view:



(figure 13)

And also the customer point of view:



(figure 14)

References

1. https://github.com/Freenove/Freenove_ESP32_S3_WROOM_Board/blob/main/Datasheet/ESP32-S3%20Pinout.pdf
2. https://forum.arduino.cc/t/7-in-one-multiparameter-soil-sensor-with-esp32/1138788
3. https://www.alldatasheet.com/view.jsp?Searchword=Max3485%20datasheet&gclid=CjwKCAjw5MOlBhBTEiwAAJ8e1vLfe2mt2LoH8O7-bjy5-ctt9wtTHZtiU02kzMoh-4SlNodgnxK2JRoCFHAQAvD_BwE
4. https://info.hivemq.com/mqtt-essentials?utm_source=adwords&utm_campaign=&utm_term=mqtt&utm_medium=ppc&hsa_tgt=kwd-296477891479&hsa_cam=17295502954&hsa_src=g&hsa_net=adwords&hsa_kw=mqtt&hsa_ad=618110679575&hsa_grp=138486496202&hsa_ver=3&hsa_acc=3585854406&hsa_mt=e&gad=1&gclid=CjwKCAjwlJimBhAsEiwA1hrp5qAGlmRjWhAj9qBFqKu4w8HMb6fKuuIIzsSiE7UwEEictj8LuUN2cBoC3kcQAvD_BwE
5. https://github.com/benfany505/ESP32IOT_Youtube/blob/master/src/main.cpp
6. https://thingsboard.io/docs/user-guide/dashboards/
7. https://www.tomshardware.com/how-to/set-up-raspberry-pi
8. https://www.raspberrypi.com/software/
9. https://www.tomshardware.com/how-to/raspberry-pi-access-point
10. https://www.simplilearn.com/tutorials/docker-tutorial/raspberry-pi-docker
11. https://hub.docker.com/r/thingsboard/tb-postgres/

Code Annex

```
#include <Wire.h>
#include <SoftwareSerial.h>
#include <WiFi.h>           // WiFi control for ESP32
#include <ThingsBoard.h>    // ThingsBoard SDK
#include <TBPubSubClient.h>
#include <ArduinoJson.h>

#define RE 13
#define DE 12

// declaration of wi-fi credentials
const char* wifiSSID = "V9061";
const char* wifiPassword = "3*04Xe91";

// Thingsboard credentials
String tbHost = "192.168.15.56";//adresa ip raspberry pi
String tbToken = "z75yg74YU1TBB4pMC1ld";

WiFiClient client;
ThingsBoard tb(client);
PubSubClient mqtt(client);

// functions declarations
void connectWifi();
void tbReconnect();
void on_message(const char *topic, byte *payload, unsigned int length);

//inquiry format for the soil characteristics
const byte nitro[] = {0x01, 0x03, 0x00, 0x1E, 0x00, 0x01, 0xE4, 0x0C};
const byte phos[] = {0x01, 0x03, 0x00, 0x1F, 0x00, 0x01, 0xB5, 0xCC};
const byte pota[] = {0x01, 0x03, 0x00, 0x20, 0x00, 0x01, 0x85, 0xC0};
const byte soil_ph[] = {0x01, 0x03, 0x00, 0x06, 0x00, 0x01, 0x64, 0x0B};
const byte soil_moist[] = {0x01, 0x03, 0x00, 0x12, 0x00, 0x01, 0x24, 0x0F};
const byte temp[] = {0x01, 0x03, 0x00, 0x12, 0x00, 0x02, 0x64, 0x0E};
const byte ec[] = {0x01, 0x03, 0x00, 0x15, 0x00, 0x01, 0x95, 0xCE};

byte values[8];
SoftwareSerial SerialPort(16,17);  // Use SoftwareSerial for RS485
communication
```

```
void setup() {
  Serial.begin(9600);
  SerialPort.begin(4800); // Set the baud rate and pins for SoftwareSerial
  pinMode(RE, OUTPUT);
  pinMode(DE, OUTPUT);
  connectWifi();
  mqtt.setServer(tbHost.c_str(), 1883); // set the server for MQTT
communication using specific port 1883
  mqtt.setCallback(on_message);
  delay(1000);
}

void loop() {
  //variables to store the received data from sensor
  int nitrovar, phosvar, potavar, ecvar;
  float phvar, moistvar, tempvar;

  if (!tb.connected()){
    tbReconnect();
  }

  nitrovar=nitrogen();
  phosvar=phosphorous();
  potavar=potassium();
  phvar=ph();
  moistvar=moist();
  tempvar=stemp();
  ecvar=econd();

  Serial.print("Nitrogen: ");
  Serial.print(nitrovar);
  Serial.println(" mg/kg");

  Serial.print("Phosphorous: ");
  Serial.print(phosvar);
  Serial.println(" mg/kg");

  Serial.print("Potassium: ");
  Serial.print(potavar);
  Serial.println(" mg/kg");

  Serial.print("Soil pH: ");
  Serial.print(phvar);
  Serial.println(" pH");
```

```
  Serial.print("Soil Moisture: ");
  Serial.print(moistvar);
  Serial.println(" %");

  Serial.print("Temperature: ");
  Serial.print(tempvar);
  Serial.println(" C");

  Serial.print("Electrical Conductivity: ");
  Serial.print(ecvar);
  Serial.println(" uS/cm");

  Serial.println();

  Serial.println("Sending data to Thingsboard");

  tb.sendTelemetryInt("Nitrogen", nitrovar);
  tb.sendTelemetryInt("Phosphorous", phosvar);
  tb.sendTelemetryInt("Potassium", potavar);
  tb.sendTelemetryFloat("PH", phvar);
  tb.sendTelemetryFloat("Moisture", moistvar);
  tb.sendTelemetryFloat("Temperature", tempvar);
  tb.sendTelemetryInt("Electrical conductivity", ecvar);
  tb.loop();
  mqtt.loop();

  delay(1000);
}

int nitrogen() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(nitro, sizeof(nitro)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
```

```
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  serialFlush();
  delay(100);
  return inter;
}

int phosphorous() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(phos, sizeof(phos)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++){
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  serialFlush();
  delay(100);
  return inter;
}

int potassium() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(pota, sizeof(pota)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
```

```
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  serialFlush();
  delay(100);
  return inter;
}

double ph() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(soil_ph,sizeof(soil_ph))==8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  int interfint=inter/100;
  int interffrac=inter%100;
  serialFlush();
  delay(100);
  return (double)interfint+(double)interffrac/100;
}

double moist() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
```

```
    delay(10);
    if (SerialPort.write(soil_moist, sizeof(soil_moist)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  int interfint=inter/100;
  int interffrac=inter%100;
  serialFlush();
  delay(100);
  return (double)interfint+(double)interffrac/100;
}

double stemp() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(temp, sizeof(temp)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 9; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[5] * 256 + values[6];
  int interfint=inter/100;
  int interffrac=inter%100;
```

```
  serialFlush();
  delay(100);
  return (double)interfint+(double)interffrac/100;
}

int econd() {
  do{
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (SerialPort.write(ec, sizeof(ec)) == 8){
      digitalWrite(DE, LOW);
      digitalWrite(RE, LOW);
      for (int i = 0; i < 7; i++) {
        values[i] = SerialPort.read();
        //Serial.print(values[i], HEX);
        //Serial.println();
        delay(10);
        if (values[0]!=0x01)//jump over first transmission if it is wrong
          i--;
      }
    }
  }while(values[0]!=0x01 || values[1]!=0x03);

  int inter = values[3] * 256 + values[4];
  serialFlush();
  delay(100);
  return inter;
}

void serialFlush(){//function for flushing the buffer
  while(SerialPort.available() > 0)
    char t = SerialPort.read();
}

void tbReconnect(){
  while (!tb.connected()){
    if (WiFi.status() != WL_CONNECTED)
      connectWifi();

    Serial.println("connecting to thingsboard ... ");
    if (tb.connect(tbHost.c_str(), tbToken.c_str())){
      Serial.println("Thingsboard Connected!");
      }
    else{
```

```
      Serial.println("Thingsboard connection failed");
      Serial.println("Retrying in 5 seconds...");
      delay(5000);
    }
  }
}

void connectWifi() {
  Serial.println("Connecting To Wifi");
  WiFi.begin(wifiSSID, wifiPassword);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("Wifi Connected");
  /*Serial.println(WiFi.SSID());
  Serial.println(WiFi.RSSI());
  Serial.println(WiFi.macAddress());
  Serial.println(WiFi.localIP());
  Serial.println(WiFi.gatewayIP());
  Serial.println(WiFi.dnsIP());
  */

}

void on_message(const char *topic, byte *payload, unsigned int length) {
  Serial.println("On message");
  char json[length + 1];
  strncpy(json, (char *)payload, length);
  json[length] = '\0';

  Serial.print("Topic: ");
  Serial.println(topic);
  Serial.print("Message: ");
  Serial.println(json);
}
```