Assignment 5 report

Part 1: Temperature collection from sensor (Through out 2 days)

#Code for DHT temperature and humidity sensor and LCD1606 output

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <DHT.h>
#include<LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
#define LED 2  // Indicating the wifi connect at the start of the program
#define DHT_PIN 7
#define DHT_TYPE DHT11

DHT dht(DHT_PIN, DHT_TYPE);

const char* SSID = "VGU_Student_Guest";
const char* PASSWORD = "";
const char* ID = "2fdac31a-b52e-4ea4-a2f6-4e6629111b7e";
const char* BROKER = "test.mosquitto.org";
String CLIENT_TELEMETRY_TOPIC = String(ID) + "/telemetry";
String CLIENT_NAME = String(ID) + "temperature_sensor_client";

const long interval = 60000 * 60; // Set delay to an hour every measurement
unsigned long publish_time = 0;

void connectWiFi() {
```

```
    Serial.println("\nConnecting to WiFi!");
    WiFi.begin(SSID, PASSWORD);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    digitalWrite(LED, HIGH);
    delay(2000);
    digitalWrite(LED, LOW);
    Serial.println("\nConnected to WiFi!");
}

WiFiClient espClient;
PubSubClient client(espClient);

void reconnectMQTTClient() {
    while (!client.connected()) {
        Serial.println("Connecting to MQTT ...");
        if (client.connect(CLIENT_NAME.c_str())) {
            Serial.println("MQTT Connected again!");
        } else {
            Serial.println("Retrying in 5 seconds - failed, client.state=");
            Serial.println(client.state());
            delay(5000);
        }
    }
}

void createMQTTClient() {
    client.setServer(BROKER, 1883);
    reconnectMQTTClient();
}

void setup() {
    Serial.begin(9600);
    pinMode(LED, OUTPUT);
```

```
    connectWiFi();
    dht.begin();
    lcd.begin();
    lcd.backlight();
    lcd.setCursor(0, 0);
    lcd.print("Hello, Trung!");
    createMQTTClient();
}

void loop() {
    publish_time++;
    reconnectMQTTClient();
    client.loop();

    float temperature = dht.readTemperature();
    float humidity = dht.readHumidity();
    if (isnan(humidity) || isnan(temperature)) {
        Serial.println("DHT not correctly working!");
        return;
    }

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Publish: " + String(publish_time));

    String displayText = "H:" + String(humidity) + "% T:" + String(temperature) + "
    lcd.setCursor(0, 1);
    lcd.print(displayText);

    DynamicJsonDocument doc(1024);
    doc["temperature"] = temperature;
    String telemetry;
    serializeJson(doc, telemetry);

    Serial.print("Telemetry #");
    Serial.print(publish_time);
```

```
    Serial.print(": ");
    Serial.println(telemetry);

    client.publish(CLIENT_TELEMETRY_TOPIC.c_str(), telemetry.c_str());

    delay(interval);
}
```

⇒ This ESP32-based program measures temperature and humidity using a **DHT11 sensor** and displays the data on an **I2C LCD screen**. It connects to a **WiFi network** and sends the sensor readings to an **MQTT broker (test.mosquitto.org)** for remote monitoring. The device subscribes to a command topic, allowing it to receive and process remote instructions. Data is formatted as JSON before being transmitted. The system operates on an hourly interval

#Code of **temperature-sensor-server** to collect and save data to **temperature.csv** file

```
from os import path
import json
import csv
from datetime import datetime
import paho.mqtt.client as mqtt

id = "2fdac31a-b52e-4ea4-a2f6-4e6629111b7e"

client_telemetry_topic = id + "/telemetry"
client_name = id + 'temperature_sensor_server'
temperature_file_name = 'temperature.csv'
fieldnames = ['date', 'temperature']

if not path.exists(temperature_file_name):
    with open(temperature_file_name, mode="w") as csv_file:
```

```
        writer = csv.DictWriter(csv_file, fieldnames= fieldnames)
        writer.writeheader()

# Callback function when a message is received
def handle_telemetry(client, userdata, message):
    try:
        payload = json.loads(message.payload.decode())
        print("Message received: ", payload, " | ")
    except json.JSONDecodeError:
        print("Error decoding JSON:", message.payload.decode())
    with open(temperature_file_name, mode='a') as temperature_file:
        temperature_writer = csv.DictWriter(temperature_file, fieldnames=fieldname
        temperature_writer.writerow({'date' : datetime.now().astimezone().replace(n

# Callback function when connected to the broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
        client.subscribe(client_telemetry_topic)
    else:
        print(f"Failed to connect, return code {rc}")

# Setup MQTT client
mqtt_client = mqtt.Client(client_name)
mqtt_client.on_connect = on_connect
mqtt_client.on_message = handle_telemetry

# Connect to the MQTT broker
try:
    mqtt_client.connect("test.mosquitto.org", 1883, 60)
except Exception as e:
    print("Failed to connect to broker:", e)
    exit(1)



# Keep the connection alive and process messages
```

```
mqtt_client.loop_forever()
```

⇒ This Python script functions as a cloud-based MQTT subscriber that listens for temperature telemetry data from an ESP32 sensor via the MQTT broker (`test.mosquitto.org`). It processes incoming messages, extracts temperature readings, and logs them into a CSV file (`temperature.csv`) with timestamps. This enables remote data collection and storage for monitoring temperature over 2 day (from 30/3/2025 to 1/4/2025).

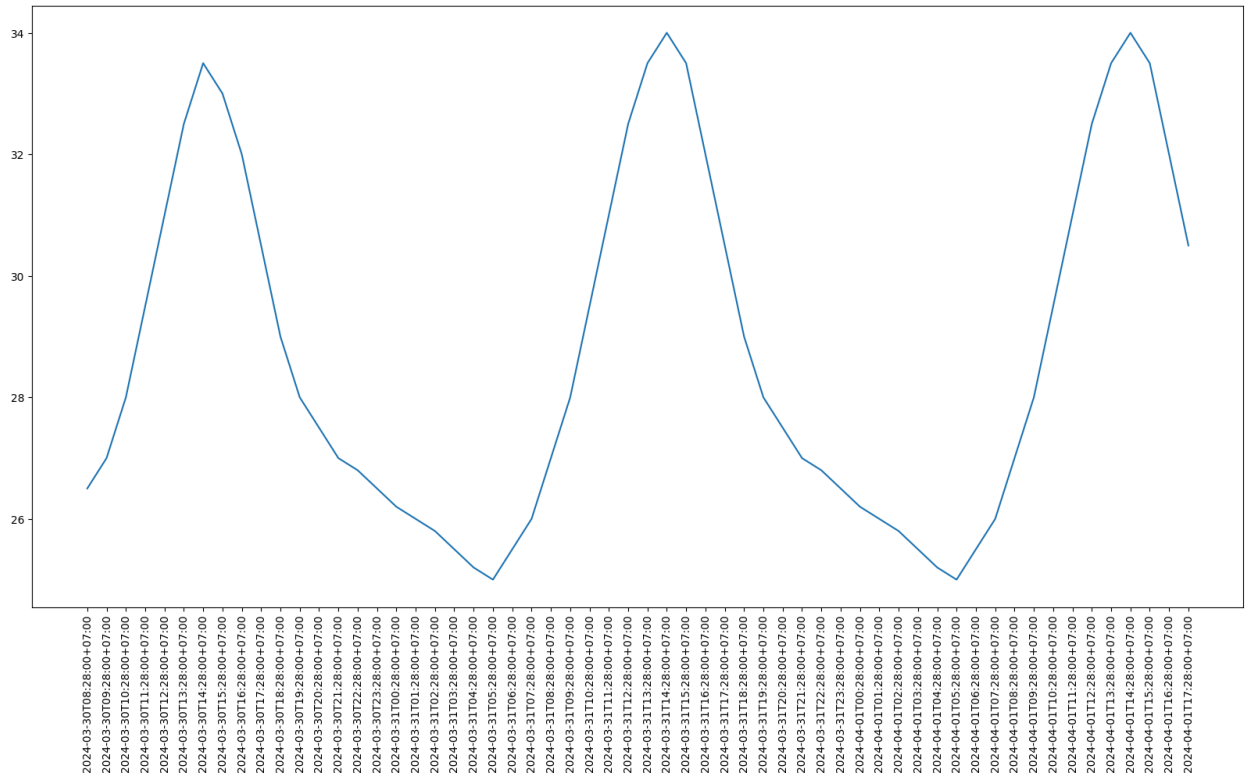
Part 2: Analysis of data to calculate GDD

The result Temperature.csv can be access in the **source code directory** in the github repo.

A part of the data is below:

47	2024-04-01T05:28:00+07:00	25.0
48	2024-04-01T06:28:00+07:00	25.5
49	2024-04-01T07:28:00+07:00	26.0
50	2024-04-01T08:28:00+07:00	27.0
51	2024-04-01T09:28:00+07:00	28.0
52	2024-04-01T10:28:00+07:00	29.5
53	2024-04-01T11:28:00+07:00	31.0
54	2024-04-01T12:28:00+07:00	32.5
55	2024-04-01T13:28:00+07:00	33.5
56	2024-04-01T14:28:00+07:00	34.0
57	2024-04-01T15:28:00+07:00	33.5
58	2024-04-01T16:28:00+07:00	32.0
59	2024-04-01T17:28:00+07:00	30.5

The temperature can now be plotted on a graph.



The GDD can be calculated using the standard GDD equation

markdown

```python
def calculate_gdd(row):
    return ((row['temperature_max'] + row['temperature_min']) / 2) - base_temperature

# Calculate the GDD for each row
min_max_by_date['gdd'] = min_max_by_date.apply (lambda row: calculate_gdd(row), axis=1)

# Print the results
print(min_max_by_date[['date', 'gdd']].to_string(index=False))
```

0.0s

Python

```
      date  gdd
2024-03-30 20.0
2024-03-31 19.5
2024-04-01 19.5
```

+ Code + Markdown

⇒ The remaining code of Jupiter notebook can be access in the **source code folder**