Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación

Embedded platforms and communications for IoT

Plant Monitoring System

Implementation of the embedded platform for plant monitoring IoT system using the B-L072Z-LRWAN1 ARM mbed-based platform

Supervisor:

Guillermo Azuara
Miguel Chavarrías
Vicente Hernández
Eduardo Juárez
Gonzalo Rosa

Authors:

Corpaci Ana Daria
Morano Stefano

Madrid, December 1, 2024

Contents

1 Introduction

This **plant monitoring system** project is designed to track and manage the environmental conditions necessary for optimal **plant growth**. By integrating a variety of sensors, including light, temperature, humidity, and soil moisture sensors, the system collects real-time data to monitor plant health. The collected data is then processed and can be used to automate or alert users to take action, ensuring that plants receive the **ideal conditions** for growth. With its user-friendly interface and efficient sensor network, this system aims to provide valuable insights into plant care while promoting healthier plants and more efficient use of resources.

2 Overview

The ***B-L072Z-LRWAN1*** is a development board designed for IoT applications with LoRaWAN connectivity. It features the STM32L072CZ microcontroller, a low-power ARM Cortex-M0+ processor, and the SX1276 transceiver for LoRa communication.

The board includes an integrated ST-LINK/V2-1 debugger, making programming and debugging straightforward, and a variety of interfaces such as *UART*, *SPI*, and *I2C* for connecting peripherals. With its ultra-low power consumption, compact design, and support for LoRaWAN protocol stacks, the B-L072Z-LRWAN1 is ideal for IoT projects.

The **datasheet** can be found here [6].

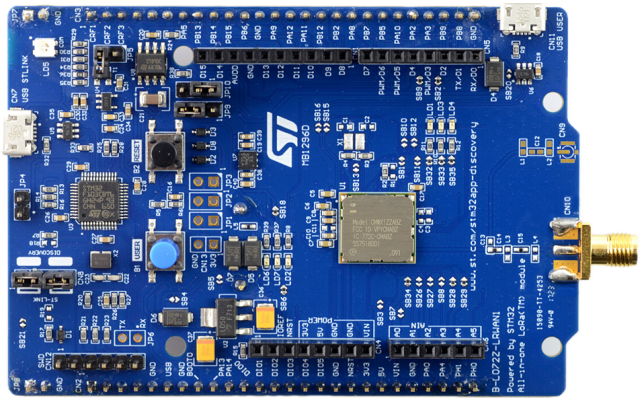

Figure 1: B-L072Z-LRWAN1

2.1 Sensors

2.1.1 Light Sensor

The **HW5P-1** sensor is a compact industrial device used to detect objects or materials without physical contact. Its operation depends on its sensing technology. The **photoelectric sensor** emits a light beam and sense objects based on reflected or interrupted light. It provides an **analog output voltage** proportional to the light detected by the sensor.

The **datasheet** can be found on the Digi_Key Electronics Site [4].

Figure 2: HW5P-1

2.1.2 Soil Mosture Sensor

The SparkFun **Soil Moisture Sensor** measures the water content in **soil** to help determine when plants need **watering**. It works by passing a small electrical current through two exposed metal probes. When the soil is moist, it **conducts electricity** more effectively, causing a measurable change in the sensor's output. Conversely, dry soil **reduces conductivity**.

The **datasheet** can be found on the SparkFun site [9].

Figure 3: Soil Mosture Sensor

2.1.3 Temperature and Humidity sensor

The **Si7021** is a digital humidity and temperature sensor known for its high accuracy and energy efficiency. It measures humidity by using a **polymer dielectric layer** whose capacitance changes with moisture levels. Temperature is measured using an internal band-gap temperature sensor. These measurements are processed by an onboard ADC and provided via an **I2C interface**.

The **datasheet** can be found on [5].

Figure 4: Si7021

2.1.4 Accelerometer

The **MMA8451Q** is a 3-axis digital accelerometer that measures acceleration forces to detect motion, orientation, and vibration. It operates using a **microelectromechanical** system (MEMS) where tiny structures inside the sensor deform under acceleration, changing the

capacitance between plates. These changes are processed into digital signals and output via an **I2C or SPI interface**.

The **datasheet** can be found on [8].

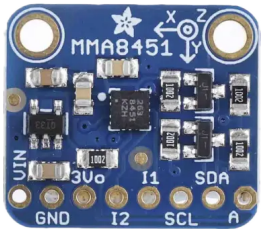

Figure 5: MMA8451Q

2.1.5 RGB Color sensor

The **TCS3472** is a digital color sensor that detects red, green, blue, and clear light levels. It uses an **array of photodiodes** with filters for each color, converting light into electrical signals. These signals are processed by an onboard ADC and provided via an **I2C interface**. The sensor adjusts for varying lighting conditions with built-in IR compensation and gain control.

The **datasheet** can be found on [3].

Figure 6: TCS3472

2.1.6 GPS

The Adafruit Ultimate **GPS Breakout Board** (version 3) is a compact GPS module that provides precise location, speed, and time data. It operates by receiving signals from **GPS satellites** and calculating **position** through trilateration. The onboard MTK3339 chipset processes these signals with high sensitivity and outputs data in **NMEA** format via **UART**. It can be expanded with an **external antenna**.

The **datasheet** can be found on [2].

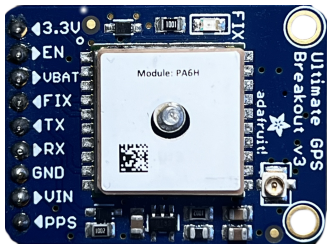

Figure 7: Ultima GPS Breakout

2.2 Other Components

2.2.1 Led RGB

An RGB LED is a **light-emitting diode** that combines red, green, and blue light to create various colors. It is commonly used in displays and lighting effects, with color and brightness controlled digitally.

The **datasheet** can be found on [1].

Figure 8: RGB Led

2.2.2 Resistor

A resistor is an electronic component that limits or regulates the **flow of electrical current** in a circuit. It is used to control voltage and current, providing specific resistance values to ensure proper operation of electronic devices.

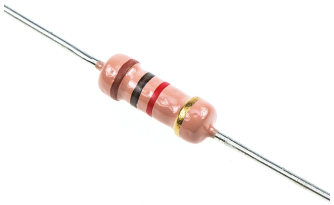

Figure 9: Resistor

2.3 IDE

Mbed Keil Studio Cloud is an online **integrated** development environment (IDE) designed for embedded systems development. It allows developers to write, compile, and debug code for **microcontrollers** directly in the cloud. The platform supports a wide range of microcontroller architectures and offers features like code autocompletion, version control, and collaboration tools. Mbed Keil Studio Cloud simplifies the development process by providing access to **libraries**, **device-specific configurations**, and **debugging tools**, enabling efficient prototyping and development of IoT and embedded applications.

The online **Mbed Keil Studio Cloud compiler** can be found at [7].

3 Project Requirements

The requirements of the project work as a guideline to implement a fully functional and useful system to monitor the health of a plant. The system will track various environmental and physical parameters crucial for plant well-being, including:

- **Climate Temperature adn Humidity**: Monitoring ambient temperature, as well as air humidity, to ensure both align with the plant's optimal conditions
- **Light Intensity**: Recording light levels, as different plants have varying light requirements for photosynthesis and growth
- **Soil Moisture**: Measuring soil hydration to prevent overwatering or underwatering
- **Leaf Color Analysis**: Evaluating leaf color, as it is one of the most reliable indicator of the plant's overall health.
- **Location and Movement**: Tracking the plant's position and detecting acceleration or displacement. This is be particularly relevant for decorative plants that may be at risk of falling due to external factors such as strong air currents, interaction with pets or children, etc.

The system features 3 distinct modes, which can be toggled by the user using the user button. Below is a brief overview of each mode.

3.1 Test Mode

In this mode, the main focus is testing all the different sensors to make sure they are working properly and are reliable. Therefore, every 2 seconds new readings from the sensors are done and shown, to constantly monitor if there is any problem with the whole system created.

```
+-------- TEST MODE --------+

SOIL MOSTURE: 0.15%
LIGHT: 0.00%
GPS: #Sats: 3 Lat(UTC): 40.243389 N Long(UTC): 3.417413 W Altitude: 98.3 m GPS time: 21:18:18
COLOR SENSOR: Clear = 33, Red = 16, Green = 9, Blue = 8 -- Dominant color: RED
ACCELLEROMETERS: X_axis = -2.54 m/s², Y_axis = 1.47 m/s², Z_axis = 7.85 m/s��
TEMP/HUM: Temperature = 14.7°C,         Relative Humidity = 74.5%

-----------------------------------------------------------------------------------------------
```

Figure 10: Output in TEST Mode

Moreover, this mode colors the LED in the dominant color, **R**ed, **G**reen, **B**lue, based on the one read from the sensor analyzing the leaves' color.

Additionally, to indicate the current operational mode to the user during testing, **LED1** on the board will remain ON until the mode is changed.

3.2 Normal Mode

Through this mode, the system will measure and display data from sensors every 30 seconds. **LED2** will be ON when the system is operating in the normal mode.

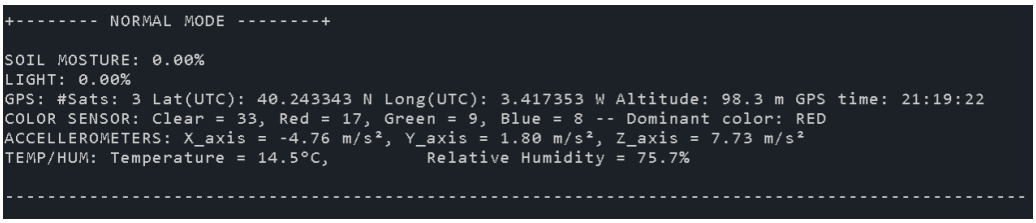

Figure 11: Output in NORMAL mode

In addition to the test mode, this one offers a summary of the values read from the sensors every hour. In the final solution of the system we kept the interval to 30 seconds for testing and debugging reasons.

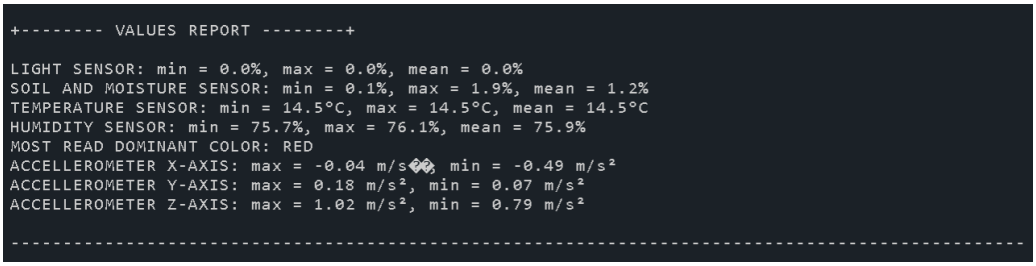

Figure 12: Output of summary of values in NORMAL mode

In case of any malfunctions of the sensors, the RGB led will be colored in a different color, as explained in the table below.

RGB Color	Issue
RED	Temperature sensor not working Temperature value <-10 Temperature value >50
WHITE	Humidity sensor not working Humidity value <25 Humidity value >75
CYAN	Light sensor value <0 Light sensor value >100
YELLOW	Moisture sensor value <0 Moisture sensor value >100
PURPLE	RGB color sensor not working
BLUE	Accelerometer sensor not working

3.3 Advanced Mode

The third and final mode extends the functionality of the system by signaling issues with the plant health through the RGB led. By default, when switching to this mode, the led is off. It continues to be off as long as no issues arise in the system, such as no error or out-of-range values read from the sensors.

The mode is operational every 3 seconds. When it is active, readings to the sensor are performed. If any issue is detected, it is added in a FIFO data structure. The system then uses a 0.5-second delay to illuminate the LED with a specific color corresponding to the identified problem.

3.4 Personal Advanced Features

We added 2 extra features to the system to enhance its functionality and simplify debugging.

The first feature is a ”night mode” capability. To ensure accurate readings from the RGB sensor, the system checks the current value of the light sensor before capturing data. If the light level is below a predefined threshold (**simulating night time conditions**) the LED on the RGB sensor is turned ON to provide sufficient illumination for accurate color measurement.

Secondly, in the case of the advanced mode, on top of the led alarm system, all the sensors health data will be displayed on the serial port, as shown in the image below.

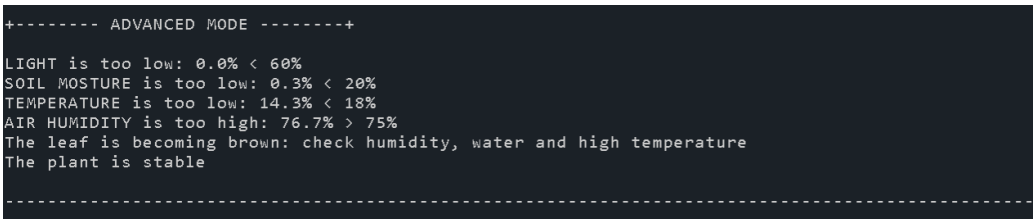

Figure 13: Output in ADVANCED mode

4 Hardware Design

The **final hardware** assembly integrates all the sensors described in section 2, ensuring seamless functionality.

In the diagram, the red dotted line represents connections to the power supply: the right side links to the **5V** pins, while the left side connects to the **3.3V** pins of the board. The blue dotted line corresponds to the connections to the board's ground pin.

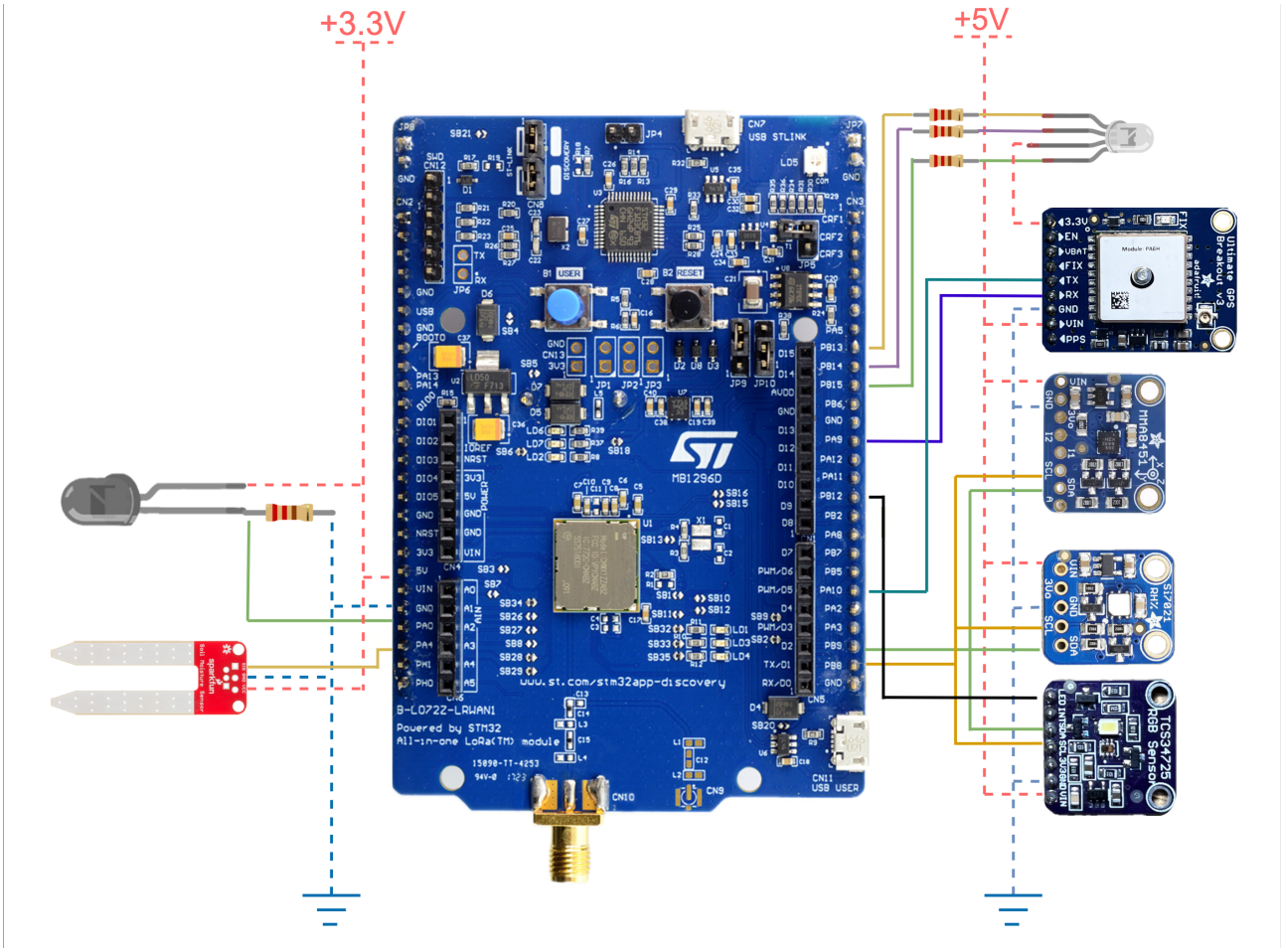

Figure 14: Hardware design and electric scheme

1. **Digital and Analog Sensors**

   - **RGB Led**

     The power pin of the **RGB LED** is connected to the **3.3V** output of the GPS sensor. The color control pins are connected as follows: *PB13* for red, *PB14* for green and *PB15* for blue.

   - **Light Sensor**

     The power pin of the **light sensor** is connected to the **3.3V** power line, while its ground pin is linked to the *PA0* pin of the board. Additionally, a resistor is connected in parallel between the ground of the board and the ground pin of the sensor.

   - **Soil Moisture Sensor**

The **VCC** pin of the sensor is connected to the **3.3V** power line, while the ground pin is connected to the ground line. The signal pin is wired to the *PA4* pin of the board.

2. **Serial Sensor**

   - **GPS Sensor**

     The **GPS** sensor's Vin pin is connected to the **5V** power line, and its ground pin is connected to the ground line. The **RX** pin of the GPS is connected to the *PA9* pin of the board (Serial 2 TX), while the **TX** pin is connected to the *PA10* pin of the board (Serial 2 RX). This configuration is crucial for the proper functioning of the GPS sensor [refer to section 5.3 for details]. Additionally, the **3.3V** output pin of the GPS sensor is connected to the power pin of the **RGB LED**.

3. **I2C Sensors**

   - **Temperature and humidity sensor**

     The power pin of the **Si7021** sensor is connected to the **5V** power line, and its ground pin is connected to the ground line. The **SDA** pin of the sensor is linked to the **SDA** pin of the board (*PB9*), while the **SCL** pin is connected to the **SCL** pin of the board (*PB8*).

   - **Accelerometer**

     The power pin of the **MMA8451Q** sensor is connected to the **5V** power line, while its ground pin is connected to the ground line. The **SDA** pin of the sensor is connected to the **SDA** pin of the board (*PB9*), and the **SCL** pin is connected to the **SCL** pin of the board (*PB8*).

   - **RGB Color Sensor**

     The power pin of the **TCS3472** sensor is connected to the **5V** power line, and the ground pin is connected to the ground line. The **SDA** pin of the sensor is connected to the board's **SDA** pin (*PB9*), while the **SCL** pin is connected to the board's **SCL** pin (*PB8*). Additionally, the *LED pin* of the sensor is connected to *PB12*, enabling the use of the internal LED [refer to section 3.4 for details].

5 Code Development

Our **implementation** involves the creation of **two** distinct **classes**, each managed by its own dedicated **thread**. This design ensures that each class operates independently and efficiently, allowing seamless parallel processing. By assigning a thread to each class, we **maximize** the **responsiveness and reliability** of the system, ensuring the sensors function optimally. This approach represents the most effective way to manage and operate the sensors in our system.

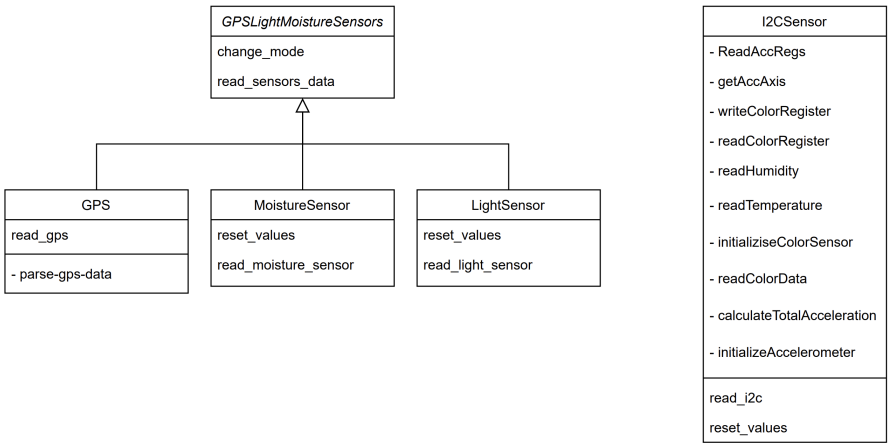

Figure 15: Diagram of the classes

The code can be found on the Github Page [10].

5.1 Classes

5.1.1 I2C Sensors Class

As 3 out of the 6 sensors communicate with the board through I2C bus, we decided to define these ones in a separate class, whose methods will be run by a separate thread, to serialize the access to the bus. This design decision will assure that there will not be any conflicts when reading or writing data from and to any of these sensors.

Below, we give a short overview of how the class is used to implement all of the sensors.

- **Accelerometer Sensor**

  Firstly, we defined the addresses that are used to manipulate the sensor, as depicted in the image below.

```
// Accelerometer address
#define ACC_I2C_ADDRESS 0x1D << 1
#define ACC_CTRL_REG 0x2A
#define REG_OUT_X_MSB 0x01
#define REG_OUT_Y_MSB 0x03
#define REG_OUT_Z_MSB 0x05
```

Figure 16: Accelerometer Sensor Addresses

Two more main methods have been implemented inside the class for manipulating the sensor. These are ***initializeAccelerometer()***, to set the sensor from standby mode to wake mode, as well as ***getAccAxis()***, called 3 times to get the acceleration on X, Y and Z axis.

- **RGB color Sensor**

  Similar to the accelerometer sensor, firstly we defined the necessary registers to work with the sensor.

```
#define TCS34725_ADDRESS 0x29 << 1
#define TCS34725_ENABLE 0x00
#define TCS34725_ENABLE_PON 0x01
#define TCS34725_ENABLE_AEN 0x02
#define TCS34725_ATIME 0x01
#define TCS34725_CONTROL 0x0F
#define TCS34725_COMMAND_BIT 0x80
#define TCS34725_CDATAL 0x14
#define TCS34725_RDATAL 0x16
#define TCS34725_GDATAL 0x18
#define TCS34725_BDATAL 0x1A
```

Figure 17: RGB Color Sensor Addresses

  For this sensor, we defined and implemented ***initializeColorSensor()***. It calls another method, ***writeColorRegister()*** 4 times. First time to enable the sensor, by enabling PON (Power ON) bit. Next, we also set AEN to 1 (enabled), which means the sensor can start performing color measurements. Lastly, we call the method to set integration time to minimum time (2.4ms) and then to amplify the incoming light by a factor of 16. The later is useful when working in environments with low light, as it will allow the sensor to detect faint light sources more effectively.

- **Temperature & Humidity Sensor**

  In the case of this sensor, we work with the following addresses:

```
#define SI7021_ADDRESS 0x40 << 1
#define CMD_MEASURE_HUMIDITY 0xF5
#define CMD_MEASURE_TEMPERATURE 0xF3
```

Figure 18: Temperature & Humidity Sensor Addresses

  0xF5 and **0xF3** are the addresses for reading the humidity and the temperature data. For both readings, we used the **NO HOLD** mode. Since all I2C sensors are handled by the same thread, the mode is not that relevant or crucial. Nevertheless, a benefit may come from avoiding unnecessarily holding the bus active.

  Main methods to interact with the sensor are ***readHumidity()*** and ***readTemperature()***.

5.1.2 Analog and Serial Sensors Class

This class is responsible for managing the reading of **serial** and **analog values**. It serves as a central controller, instantiating three separate classes, each dedicated to handling specific types

of data. The primary function of this class is to sequentially **read data** from the three classes whenever the **main** function is called.

```
void GPSLightMoistureSensors::read_sensors_data() {
    while (true) {
        uint32_t flags = ThisThread::flags_wait_any(0x1, true);
        if (mode != 4)
            gps.read_gps();
        moisture.read_moisture_sensor();
        light.read_light_sensor();
    }
}
```

Figure 19: Snippet code of the main function in the class

- **GPS Class**

  The GPS class handles data acquisition from the GPS module. It includes the *read_gps()* function, which reads data from the **GPS buffer** (configured at a baud rate of **9600**). The data is then parsed to identify the "*GPGGA*" string, extracting critical information such as Latitude, Longitude, Orientation, Time, Altitude, and the number of satellites in view. All this information is stored in a structured string called *gps_data* for further use.

- **Soil Mosture Sensor Class**

  This class is dedicated to monitoring **soil moisture levels**. It includes the *reset_values()* function, which clears all stored data to prepare for accurate reporting. The *read_moisture_sensor()* function reads the analog input from the sensor and converts it into a percentage value, providing an intuitive measure of soil moisture.

- **Light Sensor Class**

  The Light Sensor class manages the collection of **light intensity** data. Similar to the Soil Moisture Sensor class, it includes a *reset_values()* function to clear stored data for **report generation**. The *read_light_sensor()* function reads analog input from the light sensor and translates it into a percentage value, representing the relative intensity of ambient light.

This modular design ensures **clear organization** and efficient handling of **different sensor** data, making the system flexible and easy to maintain.

5.2 Threads

5.2.1 Main

The main thread is responsible for managing serial communication over USB, initialized at **115200** baud. This thread oversees serial printing, which is controlled by a **Ticker**. The tick interval dynamically adjusts whenever the system mode changes. Before switching modes, all relevant values are **reset** to prevent conflicts and ensure accurate calculations for hourly reports.

```
GPSLightMoistureSensors gpsLightMoisture(GPS_TX_PIN, GPS_RX_PIN, 9600, LIGHT_PIN, MOISTURE_PIN);
I2CSensor I2CSensor(I2C_SDA, I2C_SCL, LIGHT_RGB_PIN);

Thread i2cThread(osPriorityNormal, 1024), gpsLightMoistureThread(osPriorityNormal, 1024);
Ticker ticker, summarize_ticker, color_ticker;

void onClick(void) { button_pressed = true; }
void print_func(void) { print_test = true; }
void print_summ(void) { print_summarize = true; }
void advanced_func(void) { advanced_mode = true; }
void color_sequence(void) { color_seq = true; }
```

Figure 20: Snippet code of the Threads and Ticker function declaration

Mode switching is handled via the **Blue Button** on the board (*PB2*), cycling through the available modes in sequence. Each Ticker also wakes up the two working threads by setting flags at the start of every Ticker function. Below is a detailed breakdown of the system's modes:

- **Test Mode**

  In **Test Mode**, the board activates **LED 1** to indicate its status. The serial printing interval is set to **2 seconds**, and the system outputs the real-time values of all connected sensors to the serial buffer. If any sensor **malfunctions**, the output explicitly notes the issue with a message such as "*sensor_name* is not available", allowing for easy identification of problems during testing.

```
i2cThread.flags_set(0x1);
gpsLightMoistureThread.flags_set(0x1);
print_test = false;
printf("%s\n", gpsLightMoisture.moisture.moisture_sensor_data);
printf("%s\n", gpsLightMoisture.light.light_sensor_data);
printf("%s\n", gpsLightMoisture.gps.gps_data);
printf("%s", I2CSensor.RGBData);
```

Figure 21: Snippet code of the main serial printing function

- **Normal Mode**

  In **Normal Mode**, **LED 2** is illuminated. The printing interval is extended to 30 seconds to provide less frequent updates. Additionally, a separate **Ticker** with a 1-hour interval generates detailed reports of sensor performance. These reports include the *minimum*, *maximum*, and *average* **values** recorded for each sensor during the period, ensuring comprehensive monitoring over time. This mode is designed for regular operation with a focus on long-term data aggregation and monitoring. The led will be on if any of the sensor will **malfunction** or their parameters are **out of range**.

- **Advanced Mode**

  In **Advanced Mode**, **LED 3** signals its activation. Here, the serial printing interval returns to 2 seconds, but with added functionality to address system alarms. The system continuously monitors sensor **parameters** and identifies any **deviations** or **errors**. When an issue is detected, corresponding color indicators are added to the *FIFO* queue for the RGB LED, highlighting specific problems in real-time [as

explained in section 6]. Additionally, all detected issues are logged into the **serial buffer**, providing a detailed account of any **anomalies** for review.

5.2.2 I2C Sensors Thread

As mentioned in section 5.1.1, access to the I2C bus is serialized using a single dedicated thread, ***i2cThread***, to manage all connections, readings and writings. This thread runs the **read_i2c()** method defined in the I2CSensor class. The coordination between threads is performed through a wait-signal synchronization paradigm. This ensures that the main thread signals the other threads after displaying the most recent sensor values, using the signaling method ***flags_set(0x1)***.

On the I2C thread side, it remains blocked on the call to ***ThisThread::flags_wait_any(0x1, true)***. Upon receiving the flag, the I2C thread initializes the accelerometer, retrieves the X, Y and Z acceleration values, activates the color sensor and reads the clear, red, green and blue values. Finally, it collects humidity and temperature data. All these readings are stored in public variables, making them accessible to the main thread for display, with the frequency of the current operational mode.

5.2.3 Analog/Serial Thread

Similar to the I2C thread behavior, we create and start the **gpsLightMoistureThread** thread, to handle the other 3 sensors. It performs the actions implemented in the ***read_sensors_data()*** of the GPSLightMoistureSensors class.

In a while loop, the thread is firstly blocked by the ***ThisThread::flags_wait_any(0x1, true)*** call, until the main thread is ready to get new values from the sensors. When it gets the signal, the thread reads the gps data, followed by moisture and light sensors.

The choice to manage interactions with these three sensors using a single thread is driven by the fact that analog and serial communications involve significantly lower overhead, allowing readings to be performed almost instantly.

5.3 Problems detected

During the development and implementation of the system, we found two significant problems. We successfully addressed these issues, ensuring the system's functionality and reliability. The following sections provide a explanation of these challenges and the solutions we implemented to overcome them.

5.3.1 Serialize I2C

One of the challenges faced during the implementation was **the need to serialize access to the I2C bus**. The I2C protocol requires that only one operation occurs on the bus at any given time, as simultaneous operations can lead to collisions or data corruption. In our system, 3 sensors share the same I2C bus. Therefore, we need a mechanism to coordinate their interactions.

To address this, we **designed a dedicated I2C thread to manage all sensor communication**. This approach allows us to serialize the sequence of operations on the bus, to retrieve

data from the accelerometer, color sensor and environmental sensor without introducing latency or risking data integrity.

5.3.2 GPS Serial Conflict

Another significant issue was **operating the serial communication of the GPS module**. During testing, we observed a conflict when the GPS shared the serial interface with the serial output, resulting in data interruptions and erroneous readings. The root cause was identified as overlapping read and write operations on the same interface, causing unpredictable behavior.

To resolve this, we **switched the GPS TX and RX pins to PA_9 and PA_10**. By doing this, we separated the serial output and GPS communication, the later using the Serial 2 of the board.

6 Advanced Specification

6.1 Alarms and Sensor status

In the advanced mode, as previously described in section 3.3, we have implemented a logical **FIFO** (First-In-First-Out) **input system** to handle two specific cases.

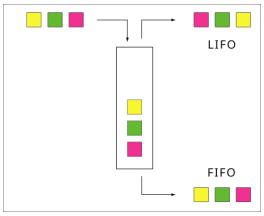

Figure 22: FIFO and LIFO differences

The first case addresses the monitoring of sensor functionality. If a sensor **malfunction**s—such as encountering an error while reading **registers** from an **I2C** sensor or if the **GPS** serial buffer is **empty**—the board will set the *sensor_working* Boolean variable to false. In this state, the system will not perform any checks on the sensor's **parameters** until it is verified to be functioning correctly.

In the second case, if the sensor is operating properly, we have designed a **custom parameter** to evaluate the plant's status based on data from each sensor. If any sensor's readings exceed predefined **thresholds** or deviate from expected values, the system will set the *sensor_alarm* boolean variable to true, signaling an issue with the plant's condition.

```
(gpsLightMoisture.light.alarm) ? add_queue(BLUE) : remove_queue(BLUE);
(gpsLightMoisture.moisture.alarm) ? add_queue(YELLOW) : remove_queue(YELLOW);
(I2CSensor.temperature_alarm) ? add_queue(RED) : remove_queue(RED);
(I2CSensor.humidity_alarm) ? add_queue(GREEN) : remove_queue(GREEN);
(I2CSensor.color_alarm) ? add_queue(PURPLE) : remove_queue(PURPLE);
(I2CSensor.accellerometer_alarm) ? add_queue(WHITE) : remove_queue(WHITE);
```

Figure 23: Snippet code of the alarm checking

6.2 Blinking RGB Led

The main thread checks these boolean variables every **2 seconds**, following the thread logic explained in section 5.2.1. Depending on the status of the boolean variables, the system will either **add** or **remove** corresponding color indicators to the **FIFO RGB LED**, signaling the status of the system.

```
if (color_seq){
    color_seq = false;
    if (counter == size){
        color_ticker.detach();
        counter = 0;
        led_rgb = OFF;
    } else  {
        led_rgb = alarm[counter];
        counter ++;
    }
```

Figure 24: Snippet code of the blinking led

The LED blinking functionality is controlled using a **Ticker** that reads the next value from the queue every *500ms*. If the queue reaches its **maximum** size, the ticker will stop and wait for the next 3-second interval before starting again.

The accelerometer alarm is triggered based on the total **magnitude** calculation, which is evaluated using the following equation:

$$\text{magnitude} = \sqrt{x^2 + y^2 + z^2}$$

If the result exceeds a threshold of 1, it indicates that the plant is **moving** and the system will set the alarm to true, signaling a potential **issue**.

```
float I2CSensor::calculateTotalAcceleration(float x, float y, float z) {
    return sqrt(x * x + y * y + z * z);
}
```

Figure 25: Snippet code of the total magnitude calculation

For clarity, the **table** below outlines the colors associated with each **specific issue** or **alarm condition**.

RGB Color	Issue
BLUE	Light sensor value <0.2 Light sensor value >1
YELLOW	Moisture sensor value <20 Moisture sensor value >60
RED	Temperature sensor not working Temperature value <18 Temperature value >25
GREEN	Humidity sensor not working Humidity value <25 Humidity value >75
PURPLE	RGB color sensor not working Leaves are turning yellow **(RED value <GREEN value *and* BLUE value <80** Leaves are turning brown **(RED value <GREEN value *and* BLUE value <30)**
WHITE	Accelerometer sensor not working Plant is falling or tilting

7 Final Results

We managed to build a reliable and fully functioning system, by following all the requirements, as well as having reliable hardware integration, efficient software execution and functional usability. Some of the key achievements that we identified are the following:

- **Sensor Integration and Accuracy**

  All the sensors were successfully integrated into the system. Through the implementation of the **TEST** mode, we tested if the sensors are working properly and if they provide consistent and accurate readings.

- **System Modes Functionality**

  All the 3 operational mode, **Test, Normal and Advances**, are successfully implemented and operate as expected, in respect with their requirements.

  On top of these, we included personalized features such as **"night mode"** for improved RGB sensor accuracy and enhanced **serial reporting** for the advanced mode. We consider these features useful, as they reinforce the practicality of the design.

- **Issues Resolved**

  Challenges such as I2C bus serialization and GPS serial conflicts were effectively solved, to reinforce the system's robustness.

In conclusion, this system's provides real-time monitoring and alert capabilities and enables users to maintain optimal conditions for plant growth with minimal effort. The multi-mode enables users to interact with the system in various ways, from basic diagnostics in Test Mode to comprehensive data summaries and alerts in Advanced Mode. These features make the system an efficient and user-friendly tool for plant care.

References

[1] Components 101. Rgb led datasheet. https://components101.com/diodes/rgb-led-pinout-configuration-circuit-datasheet, 2024. Accessed: 2024-11-18.

[2] Adafruit. Gps sensor datasheet. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf, 2024. Accessed: 2024-11-18.

[3] Adafruit. Rgb sensor datasheet. https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf, 2024. Accessed: 2024-11-18.

[4] Digit Key. light sensor datasheet. https://www.digikey.co.th/htmldatasheets/production/2935517/0/0/1/ambient-light-detector-photosensitive-sensor.html, 2024. Accessed: 2024-11-18.

[5] Silicon Labs. Temperature humidity sensor datasheet. https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf, 2024. Accessed: 2024-11-18.

[6] ARM mbed. Disco-l072cz-lrwan1 datasheet. https://os.mbed.com/platforms/ST-Discovery-LRWAN1/#technical-references, 2024. Accessed: 2024-11-18.

[7] ARM Mbed. Ide site. https://studio.keil.arm.com, 2024. Accessed: 2024-11-18.

[8] NXP. Accelerometer datasheet. https://www.nxp.com/docs/en/data-sheet/MMA8451Q.pdf, 2024. Accessed: 2024-11-18.

[9] SparkFun. Soil mosture sensor datasheet. https://cdn.sparkfun.com/datasheets/Sensors/Biometric/SparkFun_Soil_Moisture_Sensor.pdf, 2024. Accessed: 2024-11-18.

[10] Ana Daria Corpaci Stefano Morano. Github project code. https://github.com/stefano-morano/plant_monitoring_system, 2024. Accessed: 2024-11-18.