Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Curso Verão com Ciência

Report

Integration in the work plans of the Dragonfly and SmarterCW projects

Supervisors: Dr. Henrique Pinho, Dr. Manuel Barros, Dr. Carlos Ferreira

Work done by:

André Teixeira, nº 20577

Vasim Tana, nº 20896

TOMAR

September 2021

FCT Fundação para a Ciência e a Tecnologia

Ci2.ipt Centro de Investigação em Cidades Inteligentes

Summary

Rivers and lakes are among the most important natural resources in our ecosystem, providing water for human consumption and agriculture activities. However, these water resources face major threats due to contamination by pollutants of various types. Thus, it is important to ensure their preservation, providing safe consumption for human water supply and sustainable management of water resources.

This work is divided among two projects, the first focuses on the development and testing of a new monitoring unit for a robotic surface-water vehicle, and in the second project, it includes the development and testing of a new monitoring unit for data acquisition from sensors for water quality assessment.

Keywords: Water quality, pH, dissolved oxygen, conductivity, Sensors, Time based Data Bases, InfluxDB, MQTT, Raspberry Pi

Trabalho financiado pela Fundação para a Ciência e a Tecnologia no âmbito do programa Verão com a Ciência.

INDEX

FIGURE INDEX

GLOSSARY

ALU – Autonomous Locomotion Units

FCT – Fundação para a Ciência e Tecnologia

IOT – Internet Of Things

IPA – Isopropyl Alcohol

IPT – Instituto Politécnico de Tomar

MQTT – MQ Telemetry Transport

PCB – Printed Circuit Board

PETG – Polyethylene Terephthalate Glycol

PLA – Polylactic Acid

ROV – Remotely Operated Vehicle

SBC – Single Board Computer

1 Introduction

This document reports the experimental forum activities developed during the summer course "Verão com ciência", financed by the Foundation for Science and Technology (FCT), and implemented in the facilities of the Polytechnic Institute of Tomar (IPT), namely the laboratories integrated in the Center for Research in Intelligent Cities (Ci2), which took place in September 2021.

The course aimed to Participation in I&D activities, to be developed within the Ci2 Research Unit and its laboratories, through the integration in the work plans of the Dragonfly and SmarterCW projects. The work plan includes the development and testing of a new monitoring unit for a robotic surface-water vehicle, and in the second project, it includes the development and testing of a new monitoring unit for data acquisition from sensors for water quality assessment.

1.1 Summary of Work Developed

The main objectives for this course are as follows:

- Develop a standard PCB design for both projects;
- Develop the code to read sensor data, store it on a database and develop a platform to visualize it;
- Developing and installing containers for the sensors into the constructed wetlands;
- Testing the system.

2 Implementation

The activities in this course were carried out in the laboratories of the Polytechnic Institute of Tomar. This work provided an opportunity to get in touch with both projects the Dragonfly and the SmarterCW, to complement and improve both projects.

Overall, both projects have one main objective in common, which is to read data from sensors for water quality proposes using an acquisition unit based on a microcontroller capable of communicating whit the sensors and then deliver this data to the user.

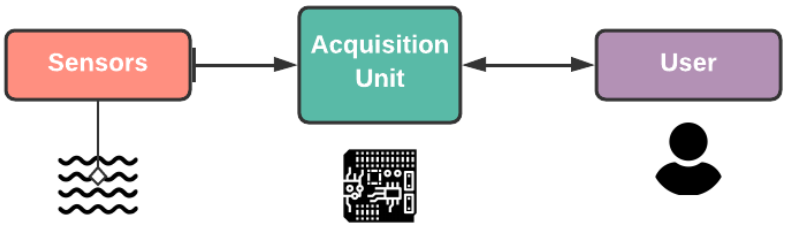

Figure 1 - Base diagram for data acquisition

2.1 Dragonfly project

The main objective of the Dragonfly project is to develop a robotic water quality system constituted by several monitoring units (Dragonfly), that working in network, and due to its mobility allow in a quick and easy manner the pinpointed identification of pollution sources.

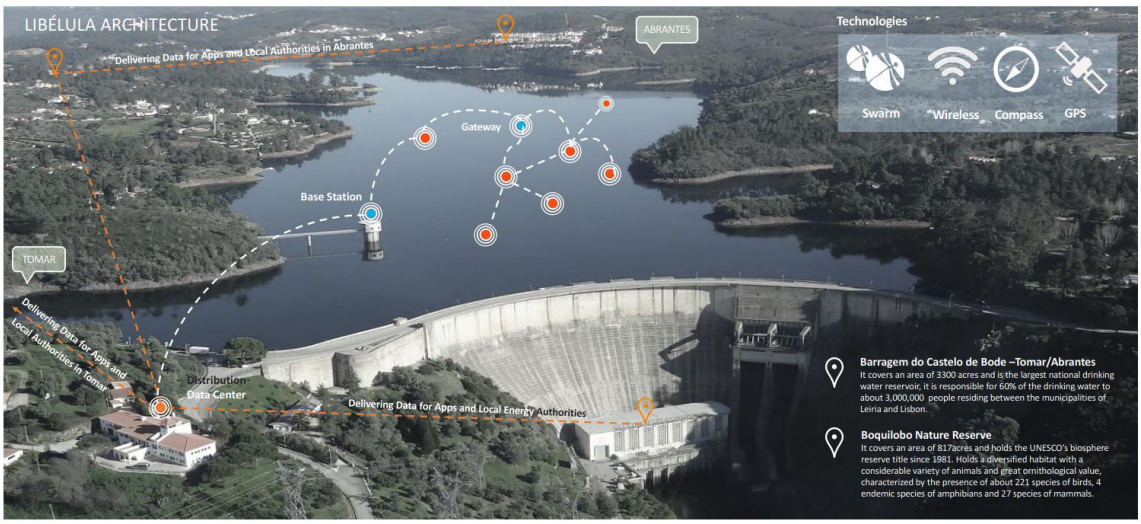

Figure 2 - Dragonfly architecture on Barragem de Castelo de Bode

Conceptually, putting aside the Technical Energy Management Unit (EMU) on , the system could be divided in four parts: 1: The mechanical part: Autonomous Locomotion Units (ALU´s) with all the technical specifications at the electrical and mechanical levels considering locomotion chain/transportation specifications of the ALU´s, these unit could be boats, submarine ROVs (Remotely Operated Vehicle); 2: Sensing Unit (SU) allowing data acquisition remotely and in real time being, composed by probes installed in ALU´s for measuring physical and chemical parameters of water; 3: Communication Unit (CU) for telecommunications, that is used to communicate between ALU and the ALU BASE, which in turn communicates with the Central Control and Management Unit (CMCU); 4: The CMCU receives the information of the IoT platform, stores it and makes it available.

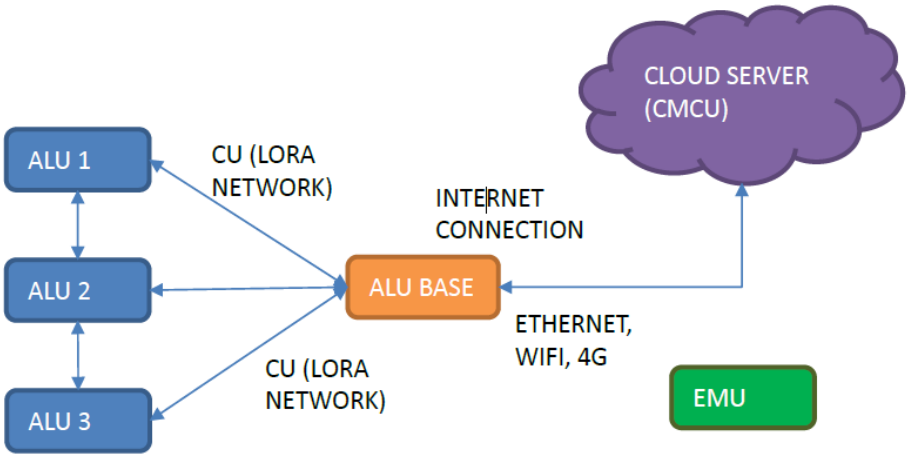

Figure 3 - Solution diagram for the Dragonfly system

The ALU Base can be seen in the pictures of Figure 4, although this unit will also include its's sensors and monitoring interface, this is the main unit that receives all the data from the other mobile units, processes the information for pollution tracking and sends all the data to a cloud server.

Figure 4 - ALU base station

2.2 SmarterCW project

The SmarterCW project entails a smarter monitorization of constructed wetlands to treat wastewater and increase its efficacy and the quality of the treated water.

By developing a system capable of acquire water quality data from sensors and store all this information in a database, it allows the implementation of data analysis tools for better monitoring and understanding of complex wastewater treatment systems.

Figure 5 - Constructed wetland located on IPT

The system presented in Figure 5 has a water input that supplies a constant stream of water that is stored in the water reservoir seen in the next image and the water is driven by a peristaltic pump.

Figure 6 - Water reservoir

The wetland tank also has a water output from which the water is drained, and it allows installation of a container with sensors as seen in the next images.

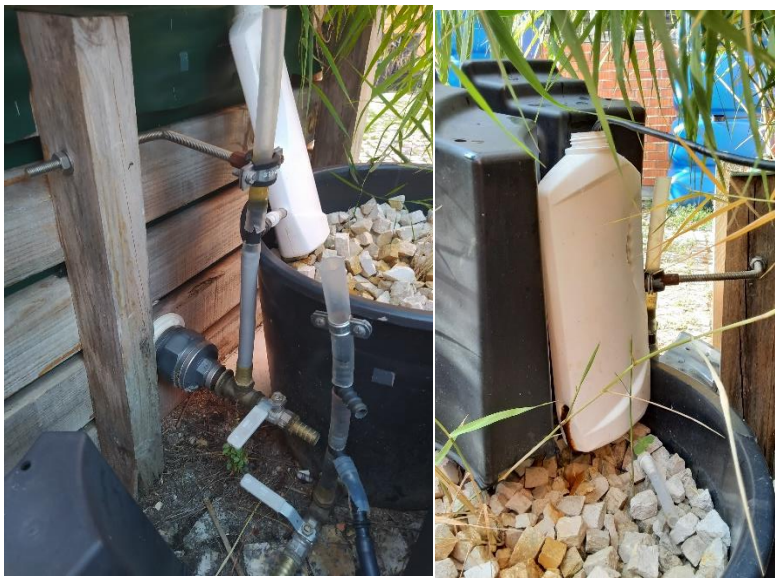

Figure 7 - System water output from different angles

For this project the main objectives are to develop a monitoring unit capable of read sensor data, in this case for water quality proposes, and send this data to a database that could later be visualized from a platform. For this, certain specifications had to be meet, for instances the PCB for the monitoring unit must be compatible with different types of power supplies in order to adapt to the Dragonfly project, while using batteries or be integrated in the SmarterCW system powered from the electric grid or from a dedicated solar panel for a more remote application for example.

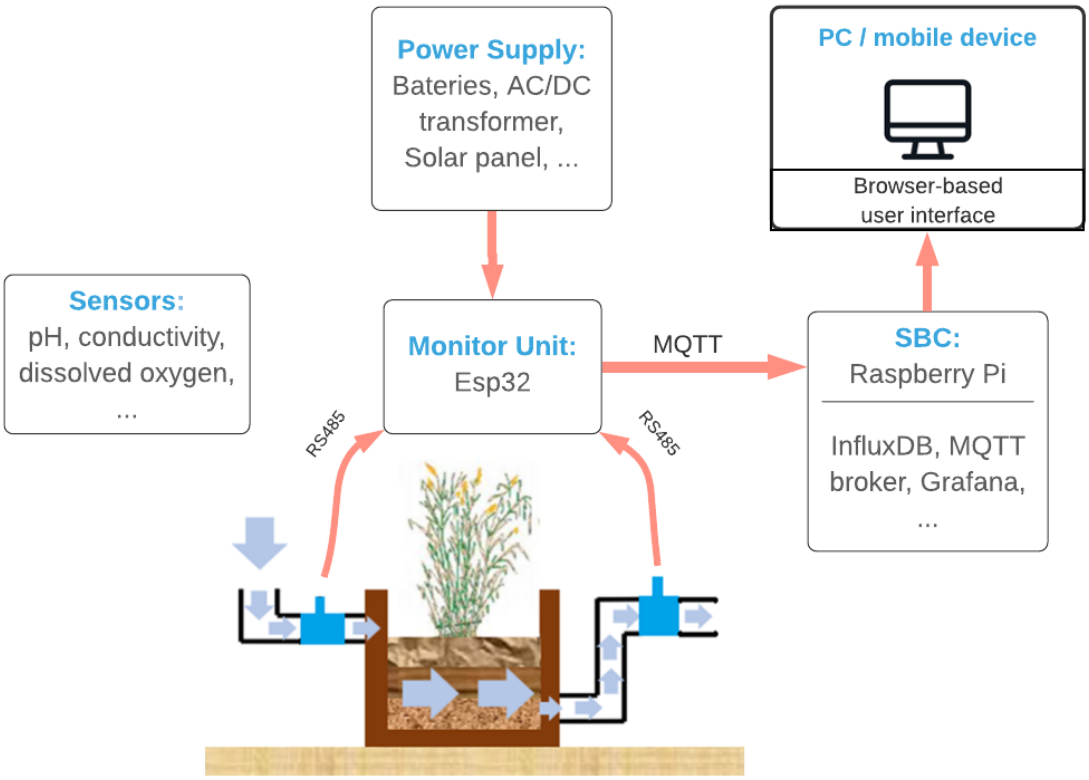

Figure 8 - Diagram of the project

Each monitoring unit or "node" will be based on a ESP32 microcontroller; by utilising the Incorporated Wi-Fi capability of the ESP32 it is possible to send the data from the sensors using the MQTT protocol to a main unit, SBC (Single Board Computer), creating a distributed system comprised of one or more nodes. This main unit, responsible to receive the data from all the nodes and save it to the database will be a Raspberry Pi.

This type of arrangements is optimal for scalability because if the need to add more sensors arrive, that can be easily accomplished by integrating another node to the setup.

Regarding the software and platforms used in this application, the influxDB will be used to store all the data and the Grafana platform will be integrated to allow the visualisation of the stored information.

2.3 Sensors

Since most water quality sensors could be applied for both projects, the sensors utilized for the SmarterCW adaptation were the same the *Dragonfly* project utilizes, this way the process of implementation was easier, and it made possible to develop a modular unit capable of directly operate or be easily adapted for both project environments.

The sensors present here are responsible to acquire a set of parameters with direct correlation to the water quality and by monitoring these values it will be possible to study the system.

Conductivity sensor Dissolved Oxygen sensor PH sensor

Figure 9 - Sensors Aqualabo

That said, these were the available sensors for this project, conductivity, dissolved Oxygen and PH sensors from *Aqualabo.* In addition, all the sensors named before having integrated temperature measurement for temperature compensation on the readings for better accuracy

All the sensors for water quality are capable of store calibration data and so allow a “plug and play” use without the necessity of constant recalibration. The communication protocol is based on the Universal Modbus RS485 and can easily be configured to accommodate a big number of sensors on the same bus network.

In the next image it’s displayed a practical example of the connection arrangement of the probes and sensors for the RS485 bus. All sensors are added to the network following a daisy chain

connection and for compliance with this communication protocol it is necessary to add termination resistors at both ends of the bus with the value of 120Ω.

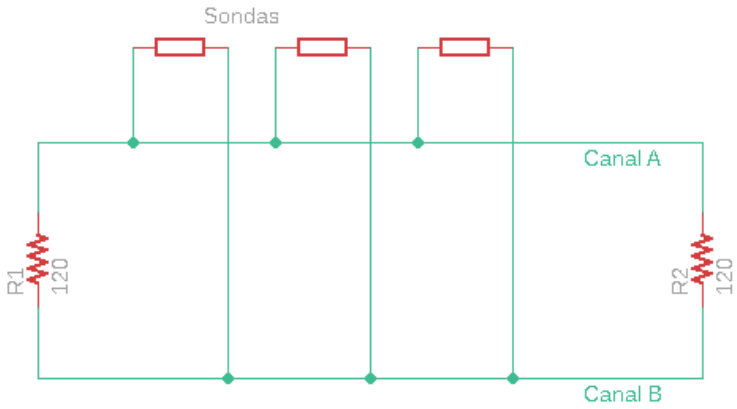

Figure 10 – Sensor connection diagram

2.3.1 Conductivity sensor (C4E)

This working principal of this conductivity sensor is based on 4 electrodes in its body, a primary pair of electrodes in graphite and a secondary pair in platinum, an alternating current of constant voltage is applied between the primary pair. The secondary pair allow of regulate the voltage imposed to the primary's electrodes to reflect of the fouling. The voltage measured between the primary's electrodes is in function of the resistance of place and so, of the conductivity.

Figure 11 - Conductivity sensor (C4E)

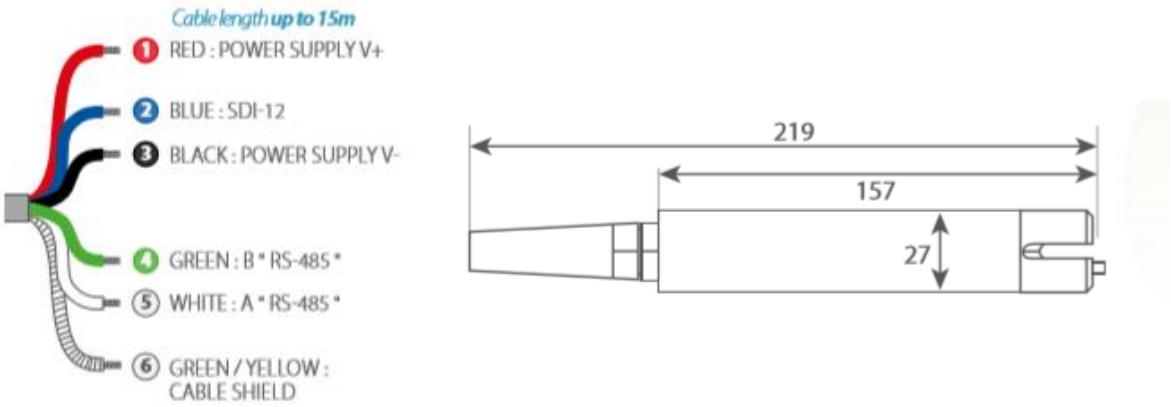

Figure 12 - C4E sensor wiring information and dimensions

2.3.2 Dissolved oxygen sensor (OPTOD)

The OPTOD (Optical Dissolved Oxygen) sensor is based in optic luminescent technology and is approved by the international method ASTM D888-05. Whiteout constant calibration required and a low power consumption technology this sensor attends to the major demands of field work for both short and long-term campaigns.

Figure 13 - Dissolved oxygen sensor (OPTOD)

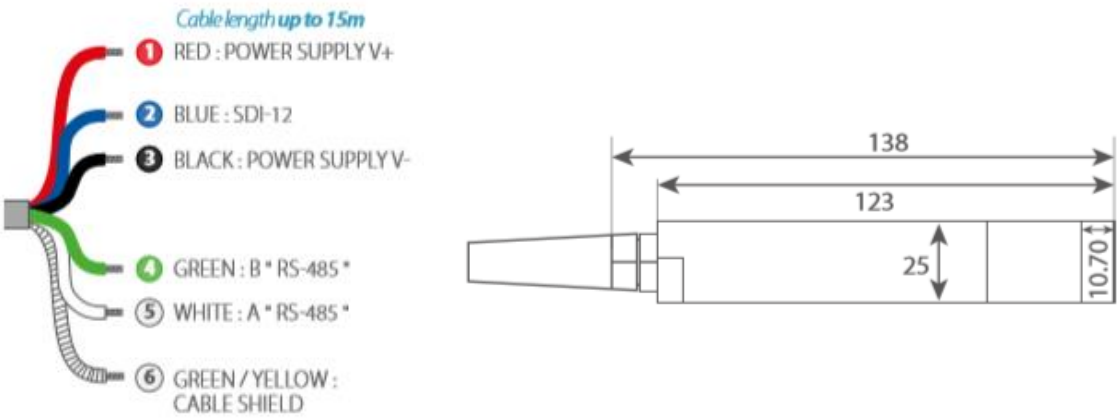

Figure 14 - OPTOD sensor wiring information and dimensions

2.3.3 PH sensor

The PH sensor has been designed to perform under hard conditions from pure mountains water with conductivity as low as 20 µS/cm, lakes and rivers (100 – 2000 µS/cm), seawater with conductivities of 50 mS/cm and to wastewater with conductivity higher than 200 mS/cm.

Figure 15 – PH sensor

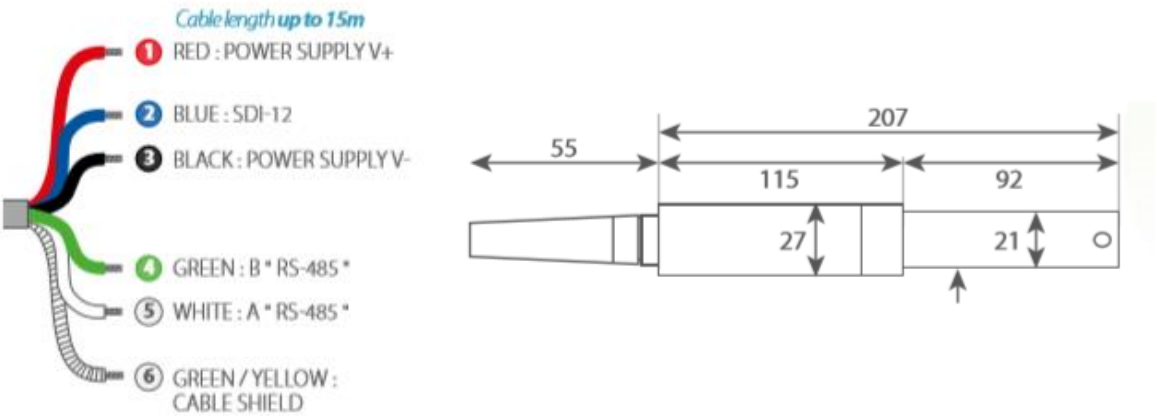

Figure 16 – PH sensor wiring information and dimensions

2.3.4 Sensor holder

To hold the sensor in place, a 3D model was develop using Fusion 360, fig xx. The objective of the sensor holder is to keep the sensors from the reaching the bottom of the container and secure them in position inside the container. The senser holder were designed for the sensors to be fitted into position without any mechanism to be adjusted. The diameter of the hole is 0.5mm larger than the diameter of the sensers.

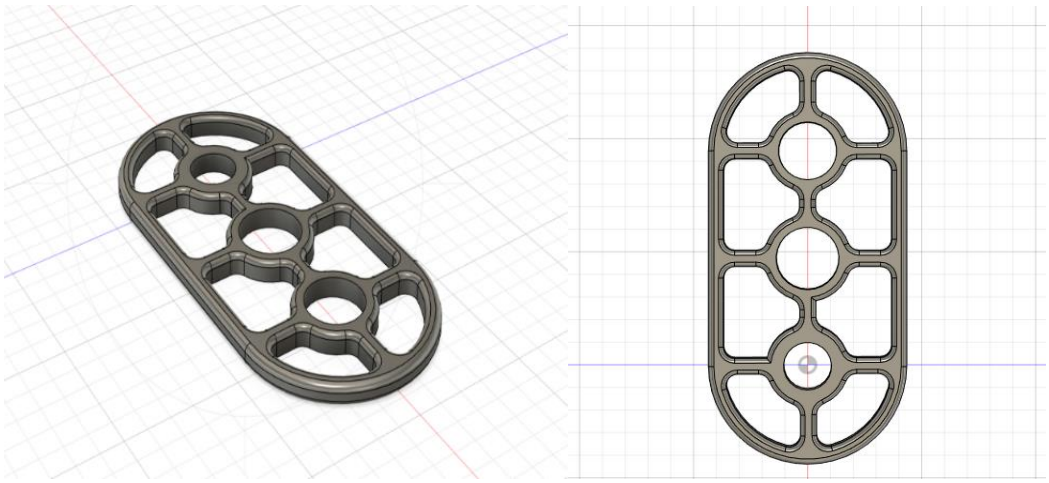

Figure 17 – 3D model on the Fusion 360 software

To be stable in position, the Sensors will be wrapped with a layer of Teflon that thickens their diameter and holds them in position. This method has been chosen due to its simplicity and efficiency, as well as the fact that this model is a protype and not the final design. In this case the material used while printing was a PLA filament.

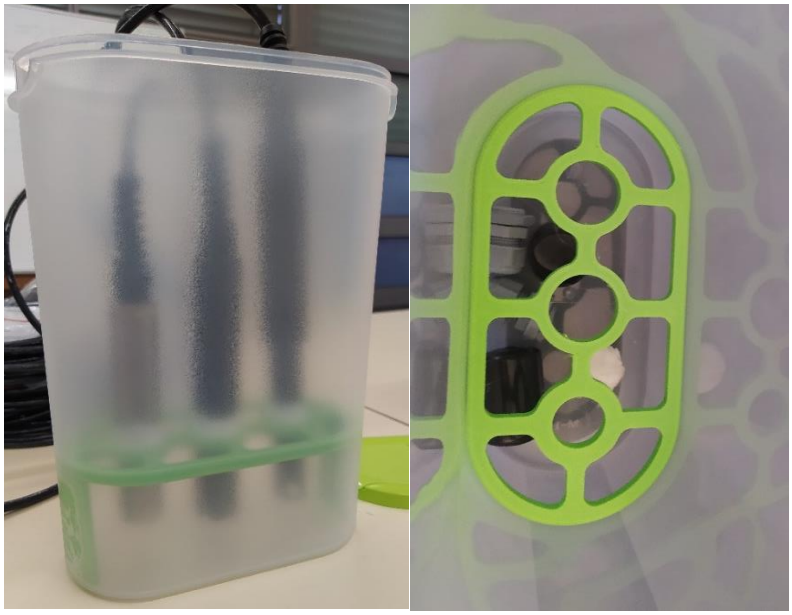

Figure 18 - Adaptation of the sensors on a container

Figure 19 - Adaptation of the sensor's container on the constructed wetland tank

2.3.5 Water flow sensor

One of the important metrics in the system is the water flow rate both at the input and at the water output; with both values it is possible to access important information about it, e.g., the relation between the input and output and so the percentage of water that "passes" trough the tank and the percentage the that is evaporated to the atmosphere; the impact of the rain on the reservoir or a means to detect anomalies or malfunctions in the process, rather from lack of water in the main feeding container or from clogs in the tubbing.

To pump the water to the containers with a constant and adjustable rate, a peristaltic pump is used as seen is the next picture.

Figure 20 - Water pump setup

The sensor responsible to read the water flow rate for this system is an SLF3S-1300F from *Sensirion*, this model series is especially calibrated for water and IPA (Isopropanol Alcohol) up to ± 40 ml/min, and as an integrated temperature compensation and fully calibrated digital output through I^2C from a single chip.

Figure 21 - Flow rate sensor SLF3S-1300F

The connectors on both sides of the sensor are ¼″-28 flat bottom ports and although they were a standard type, they aren't easily found like common use plumbing appliances and so, because of the lack of a connector, there was the necessity to develop 3D parts able to fit the different types of tubbing used in the setup.

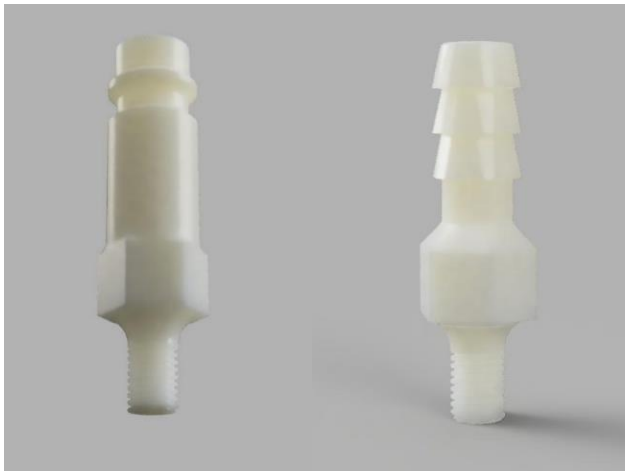

Figure 22 – Tubbing adaptations for the flow rate sensor

The tubbing adaptations were printed using a filament of PETG and the printing setting had to be specially adapted to guaranty that the parts were watertight.

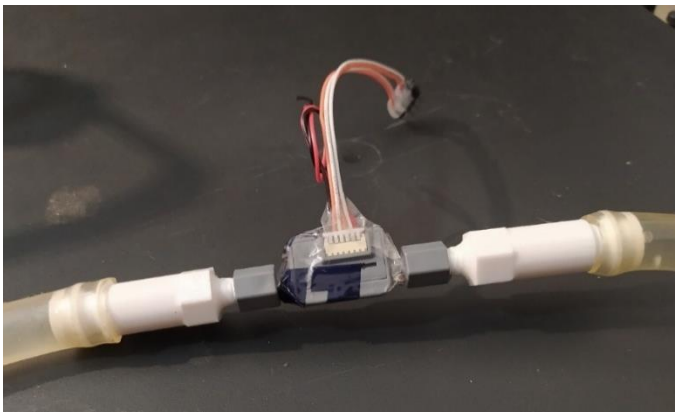

Figure 23 - Testing of the 3D pieces on the sensor

2.4 PCB design

For this project, one of the main objectives was to develop a PCB (Printed Circuit Board), as an infrastructure in which a monitoring system would be running. This system is responsible for reading the values of all sensors connected to the board with a predetermined sampling interval and send this information to the database server.

This board was design in a way that it could be applied for both projects. It operates based on a ESP32-DevkitC module which is a microcontroller development board from Espressif. It has

a 32-bit dual core processor and Bluetooth and Wi-fi integrated and is practical for IOT projects.

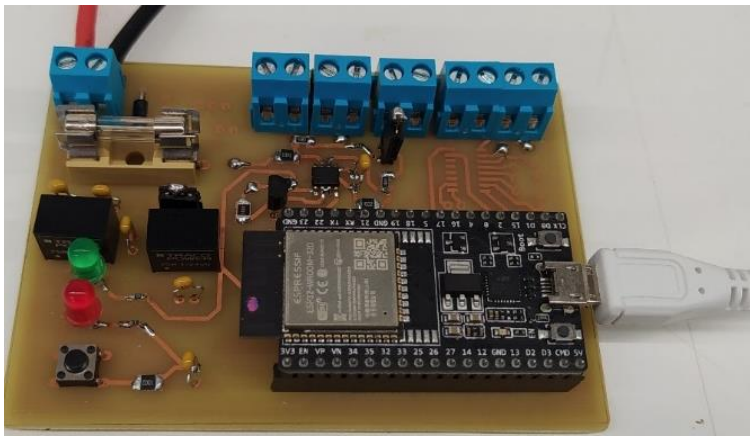

Figure 24 - Assembled PCB of the monitoring Unit

To guarantee that the main board for data acquisition could work for both projects, the key challenge was regarding the power supply of the systems because, for both cases the microcontroller could be the same, an ESP32 and since the sensors were the same, any circuitry needed to read them would also be the same.

In the case of the power supply for the Dragonfly application, since we are working with a network of autonomous vehicles, the best source of power are batteries, generally Lipo or Lithium cell batteries and for this project the ones used have a nominal DC voltage of around 14.6V (this number can be less or greater depending on the cell configuration).

On the other hand, for the SmarterCW project were considered mainly two scenarios, the usage of the mains electrical grid or the usage of solar panels. In the first case it's indispensable an AC to DC transformer and since nowadays they can be easily found for cheap for outputs of 5V or 12V, instead of designing one, is best to use a commercial alternative.

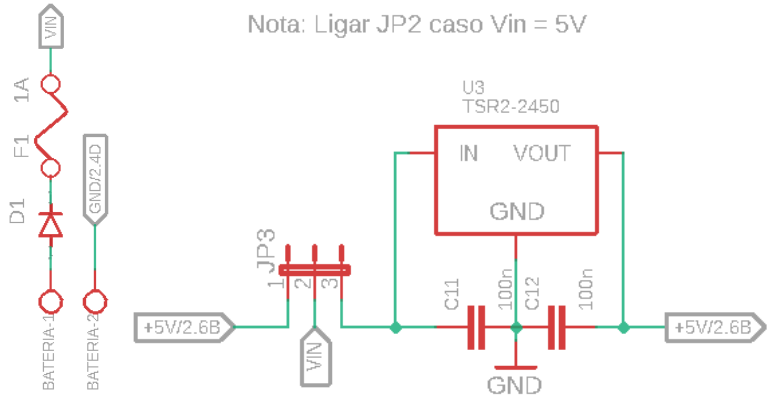

Figure 25 - 5V regulation for sensors

With these levels in mind, the PCB was design to handle input voltages between 5V and the max voltage of a fully charged 14.6V, which can be around 16V; thus, 5V DC to DC converters from *traco power* were used since they have a maximum input voltage of 36V. Besides the regulator, a 1A protection fuse was used and a diode for reverse polarity protection.

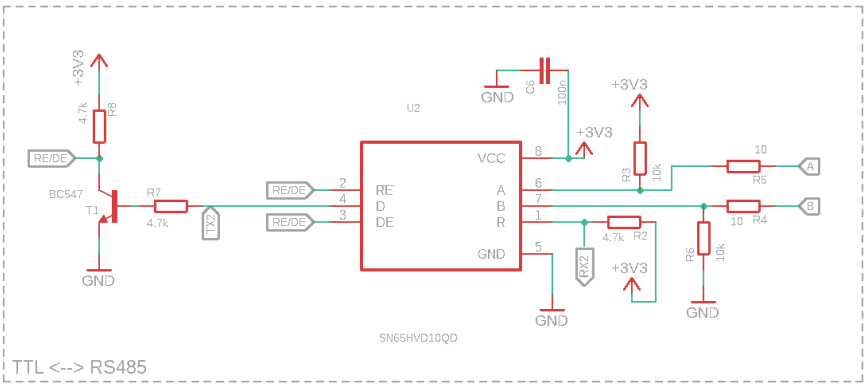

Figure 26 - Circuitry for the RS485 transceiver

Regarding the hardware for the sensor's communication protocol, the RS485, in the previous figure is shown the circuit based on a *max485* transceiver that converts the input data from the sensors RS485 bus and converts it to TTL serial communication that the ESP32 can read.

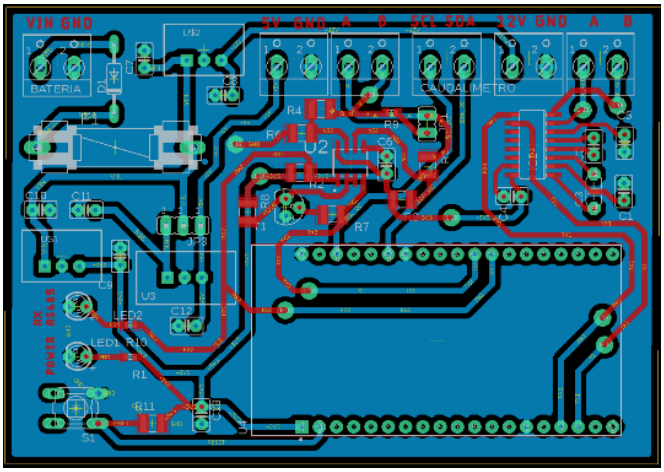

Figure 27 - Board view on the Eagle software

2.5 Acquisition Unit / ESP32 code

For this application the code was implemented based on the FreeRTOS, Free Real Time Operating System, architecture while programming the ESP32. The FreeRTOS library basically allows for a more efficient use of the microcontroller resources, instead of the traditional cyclic execution of the different tasks in a loop that generally only utilises one core (next figure on the left), the FreeRTOS permits a more flexible execution of tasks, allowing to attribute a specific task to a core, in a multicore system, having more than one task running at a time in parallel (next figure on the right), organize tasks based on priority levels and precisely control the execution time of tasks.

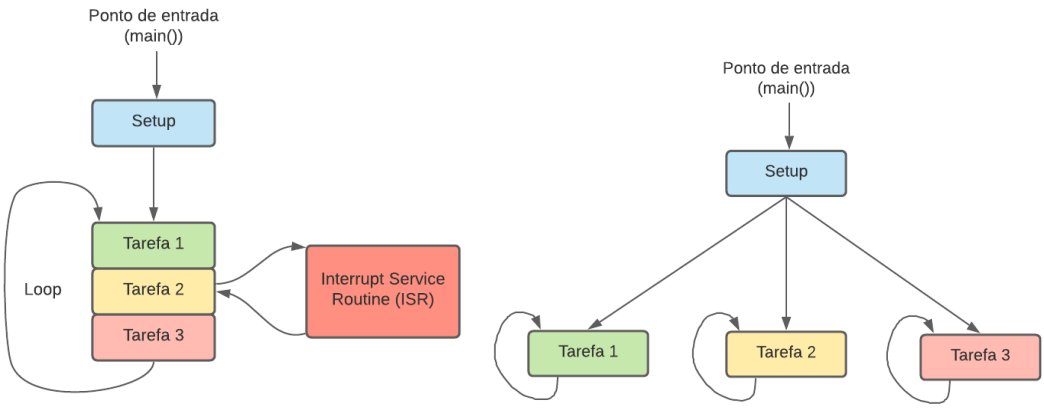

Figure 28 - Loop implementation VS FreeRTOS example

Overall, the FreeRTOS architecture allows more control over the microcontroller for more complex applications, for this one, although it has added benefits and was also an interesting concept to explore, it's use was mainly to control both cores of the ESP32 and priority management and so was not necessary especially regarding the complexity of the system as it is at the moment.

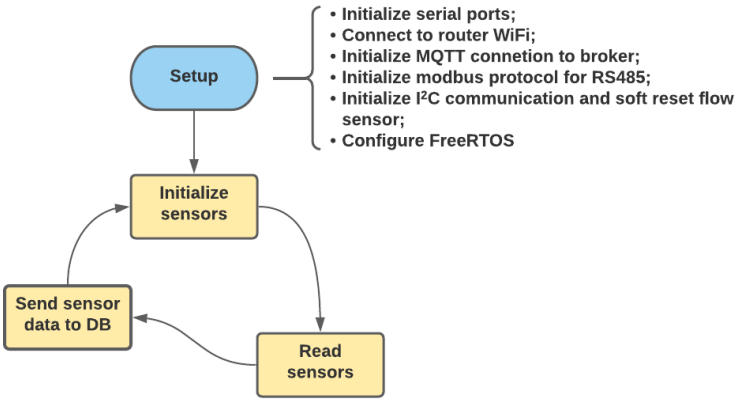

Figure 29 - General code diagram

As seen in the previous image, after the initial setup where the initial configurations are made, the code mainly initializes the sensors to prompt them to acquire a new value, after a delay that value is read by the ESP32 and then is sent to the Raspberry pi via MQTT, after which the cycle repeats.

In parallel to these main tasks, the ESP32 will also maintain connection to the WiFi network and MQTT broker and if any of both connections is lost it will try to reconnect with a defined time interval of 2 seconds sing timers, (code in the next image)

```
608  void initMQTT(){
609    mqttReconnectTimer = xTimerCreate("mqttTimer",
610                                      pdMS_TO_TICKS(2000), pdFALSE, (void*)0,
611                                      reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
612    wifiReconnectTimer = xTimerCreate("wifiTimer",
613                                      pdMS_TO_TICKS(2000), pdFALSE, (void*)0,
614                                      reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));
615
```

Figure 30 - Implementation of the reconnect timers

In re rest of this chapter the more important functions are going to generically explained.

2.5.1 Flow rate sensor initialization

The liquid flow rate sensor uses the I^2C protocol to communicate and so in the code setup a initialization is required both for the protocol communication setup and to do a soft reset on the sensor which is recommended by the product datasheet.

In this code snippet the ESP32 tries to do the soft reset a set number of o times.

```
void initCaudalimetro(const int ADDRESS){
  int ret;
  int i = 0;

  Wire.begin();       // join i2c bus (address optional for master)
  vTaskDelay(5);
  do {
    i++;
    // Soft reset the sensor
    Wire.beginTransmission(ADDRESS);
    Wire.write(0x06);
    ret = Wire.endTransmission();
    if (ret != 0) {
      Serial.println("\nError while sending soft reset command, retrying...");
      delay(500); // wait long enough for chip reset to complete
    }
    else{
      Serial.println("soft reset done!");
      delay(50); // wait long enough for chip reset to complete
    }
  } while (ret != 0 && i < 5);
}
```

Figure 31 – Code snippet of the flow rate sensor initialization

2.5.2 Serial ports initialization

In this function the serial communication ports are configured, in this application two ports were configured, the first as seen in the next image is used mainly for debug using the PC as it's the serial port by default for the USB, and the second serial configuration is used for the RS485 protocol used to read the Aqualabo sensors.

```
102  void initSerial()
103  {
104    // use Serial (port 0); initialize Modbus communication baud rate
105    Serial.begin(115200);
106    while (!Serial) {;} // wait for serial com
107
108    Serial.println("\nSetup");
109    Serial1.begin(9600, SERIAL_8N2, RX2, TX2); // Serial port for Aqualabo sensors
110  }
```

Figure 32 - Code snippet for the serial initialization

2.5.3 Create FreeRTOS task

In the next figure we can see the code referent to the configuration of one of the tasks, this function allows to create a task and pin it to a specific core, in this case the core 0, one of the two ESP32 cores; the amount of memory allocated for the task, stack size and the priority level of the task.

```
xTaskCreatePinnedToCore(
  SensorInitReadTask,
  "InitRead", // A name just for humans
  1500,     /* Stack size of task */
  NULL,      /* parameter of the task */
  5,         /* priority of the task */
  NULL,    /* Task handle to keep track of created task */
  //tskNO_AFFINITY);
  0);
```

Figure 33 - Code snippet to create one task using FreeRTOS

2.5.4 Read liquid flow rate sensor

To read from the flow rate sensor is necessary to write four specific bytes and specify the address of the sensor, 8 by default, after that the code awaits the response from the sensor and the different values are separated, e.g., the flow value and temperature read.

```
Serial.println("\n~~~~ Caudalímetro ~~~~");
Wire.beginTransmission(ADDRESS);
Wire.write(0x36);
Wire.write(0x08);
ret = Wire.endTransmission();
vTaskDelay(pdMS_TO_TICKS(50));

if (ret != 0) {
  Serial.println("Error during write measurement mode command");
  return true;
} else {
  Wire.requestFrom(ADDRESS, 9);

  if (Wire.available() < 9)
    Serial.println("Error while reading flow measurement");
  else{
    sensor_flow_value  = Wire.read() << 8; // read the MSB from the sensor
    sensor_flow_value |= Wire.read();      // read the LSB from the sensor
    sensor_flow_crc    = Wire.read();
    sensor_temp_value  = Wire.read() << 8; // read the MSB from the sensor
    sensor_temp_value |= Wire.read();      // read the LSB from the sensor
    sensor_temp_crc    = Wire.read();
    aux_value          = Wire.read() << 8; // read the MSB from the sensor
    aux_value         |= Wire.read();      // read the LSB from the sensor
    aux_crc            = Wire.read();
```

Figure 34 - Code snippet to read the flow rate sensor

2.6 InfluxDB

The influxDB software is an open-source time series database developed by *InfluxData* and is targeted for storage and retrieval of time-based data in fields such as operations monitoring, application metrics, IOT sensor data and real-time analytics.

Figure 35 - InfluxDB logo

In the next image it can be seen the list of measurements accessed through the influx terminal on the Raspberry pi.

```
> show measurements
name: measurements
name
----
sensor_c4e
sensor_optod
sensor_ph
sensor_slf3_1
>
```

Figure 36 - List of InfluxDB measurements from the raspberry

In the Figure 37 we can see a couple of data points read from the liquid flow sensor. From left to right, the first column represents the timestamp at which the data was received, on the second we find the value of the flow measured, in *ml/min*, the third is the reference of the sensor, just an informational parameter and finally, the last value is the temperature read in ºC.

```
2021-10-26T15:16:28.623081549Z 17.308          slf3_1 25.255
2021-10-26T15:16:28.736625948Z 65              slf3_1 25.23
2021-10-26T15:16:28.852209543Z 65              slf3_1 25.225
2021-10-26T15:16:28.965154522Z 64.838          slf3_1 25.125
2021-10-26T15:16:29.07823457   65              slf3_1 25.095
2021-10-26T15:16:29.19098344Z  65              slf3_1 25.065
2021-10-26T15:16:29.309059349Z 64.354          slf3_1 25.12
2021-10-26T15:16:29.412841362Z 64.966          slf3_1 25.09
2021-10-26T15:16:29.520841087Z 45.286          slf3_1 25.11
2021-10-26T15:16:29.633078575Z 31.572          slf3_1 25.245
2021-10-26T15:16:29.742247817Z 29.518          slf3_1 25.195
```

Figure 37 - Data points measured from the liquid flow sensor

2.7 MQTT and mosquitto software

In this application, the MQTT communication protocol was used to allow communication over the WiFi connection between the different sensor nodes, in this case represented as the monitor unit with the ESP32 microcontroller, and the Raspberry pi running the database server and the *Grafana* platform.

The MQTT protocol is based on a publish and subscribe approach, for this example, the different sensor nodes responsible to read the sensors will publish data on so-called topics, e.g., "sensor_ph", "sensor_optod", etc are topics referenced to the pH sensor and the conductivity sensor respectively. The topic can be interpreted as a label which classifies to what the data refers to. Different nodes/monitor units can publish on the same topic and one node can also publish on multiple topics, as an instance, the monitor unit developed for this project can read data from three different water quality sensors from *Aqualabo* and another water flow rate sensor, ending publishing data to four different topics.

To receive the information from the different sensors, the Raspberry pi will subscribe to all the topics.

All the MQTT communication is managed by a MQQT broker, it's a key element to implement this protocol, and for that the *mosquitto* software was installed in the Raspberry pi to act as a broker.

2.8 Node-red

The Node-Red platform is a Node.Js based software that allows to connect hardware devices, APIs and online services using a graphical interface on a browser-based editor. This interface is much more user friendly, and its functions can be easily implemented without the necessity of learning Node.js and writing multiple lines of code.

The initial implementation on the project was using the Node-Red platform as seen in the figure bellow, the basic function of it was (from left to right), to receive data from the different topics, extract the information from the json format and send the data to the database.

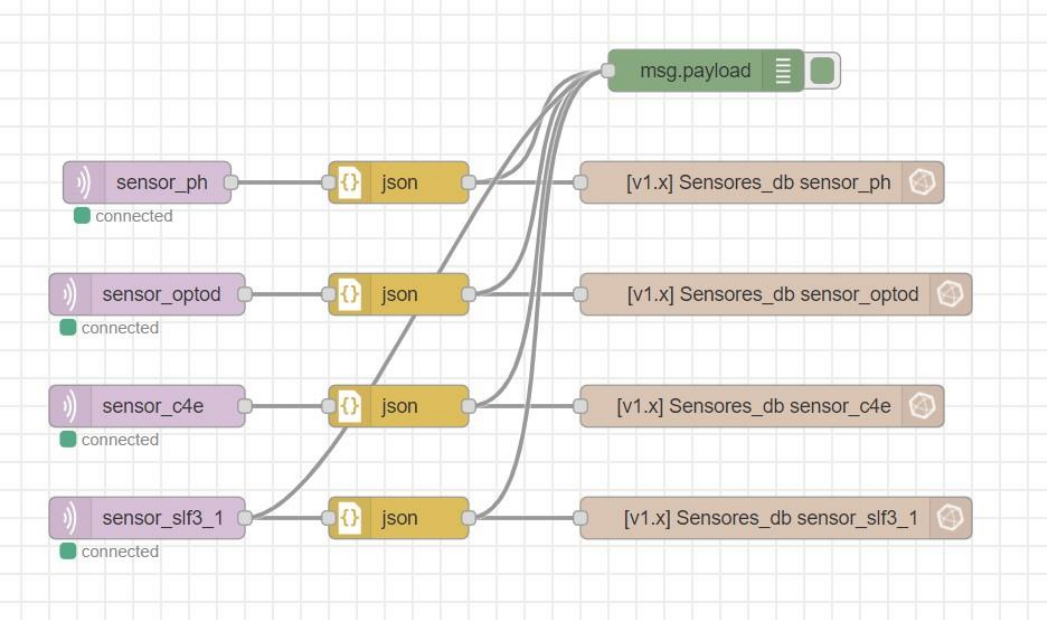

Figure 38 - Node-Red diagram used

2.9 Node-js

In a later implementation, the Node-red was substituted directly by a script of Node-js that has the same function although has the vantage, assuming the code is optimised and does not have a worst performance, of not depend on a graphical interface like Node-red that occupies more space.

```
const options = { // MQTT connection
  // Clean session
  clean: true,
  connectTimeout: 4000,
  // Authentication
  username: 'admin',
  password: 'dragonfly'
}

// var client  = mqtt.connect('mqtt://192.168.1.101:1883', options) //home setup
var client  = mqtt.connect('mqtt://localhost:1883', options)
```

Figure 39 - Code for configuration a connection to the MQTT broker

For the Node-js script that will run on the raspberry pi firstly it was to connect to the mosquitto MQTT broker using the right username and password for authentication, and to the influxDB, then subscribe to all the topic sensors to receive the data transmitted.

```
influx.getDatabaseNames()
  .then(names => {
    if (!names.includes('sensores'))
      console.log('DB nao encontrada!');
    else
      console.log('Sucesso, DB encontrada!');
  })

client.on('connect', function () {
  console.log("Connected to MQTT broker!");

  client.subscribe(topico1, function (err) {
    if (!err)
      console.log(`Subscribed to \'${topico1}\'.`);
  })
  client.subscribe(topico2, function (err) {
    if (!err)
      console.log(`Subscribed to \'${topico2}\'.`);
  })
  client.subscribe(topico3, function (err) {
    if (!err)
      console.log(`Subscribed to \'${topico3}\'.`);
  })
  client.subscribe(topico4, function (err) {
    if (!err)
      console.log(`Subscribed to \'${topico4}\'.`);
  })
});
```

Figure 40 - Subscription to the different MQTT topics

After subscribing to the sensor topics all is left is to save the received data correctly to da database. As seen in the next image, for every received message, the data is parsed to the Json format, converting a string to a object in the Json format, and finally sends the data to the influxDB.

```
client.on('message', function (topic, message) {
  // message is Buffer
  console.log(`Received from topic \'${topic}\': ${message}`)
  var obj = JSON.parse(message)

  if (topic == topico1){
    influx.writePoints([
    {
      measurement: 'sensor_ph',
      tags: {ref: obj.ref},
      fields: { temp: obj.temp, PH: obj.pH, redox: obj.redox},
    }
  ],{
    database: 'sensores',
  })

  .catch(err => {
    console.error(`Error saving data to InfluxDB! ${err.stack}`)
  });
```

Figure 41 - Code snippet that reads from a received topic and saves on DB

2.10 Grafana

The *Garafana* API is an open-source multi-platform toll for analytics and data visualization. It provides editable dashboards with charts, graphs, and alerts. By integrating this tool, it is possible to access the stored data in the InfluxDB database on a computer or a mobile device as long as it's connected to the same network in the case of a local installation, e.g., on a Raspberry Pi or other computer; on the other end, Grafana also has a cloud service.

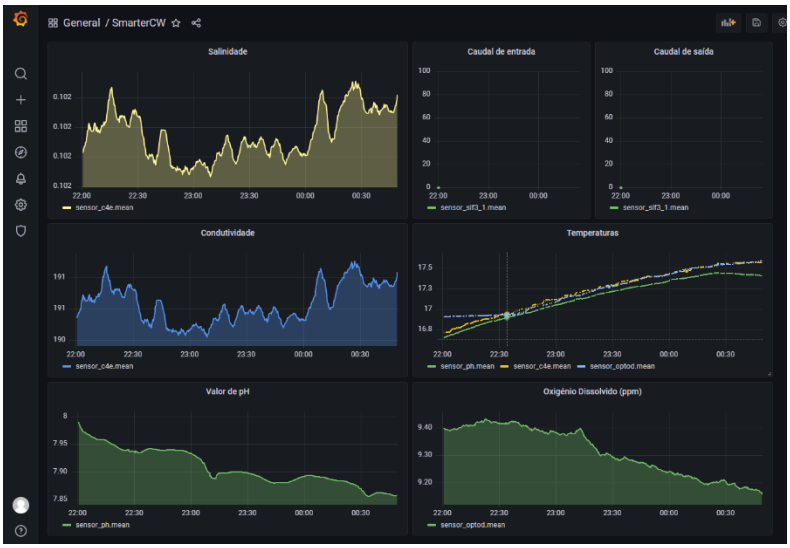

Figure 42 – SmarterCW dashboard on grafana

3 General Conclusion

Through this project was possible to get closer look at the water treatment technology and able to understand the amount of effort, time, and money invested into such technology as well as the fact of how important it is to keep developing our technologies and methods of water treatment since it plays an important role in our life whether the water is directly consumed by humans or used for industrial and/or agricultural propose. The concept of reusing wastewater itself is very important and hopefully more technologies will be developed in the future.

With this was also possible to learn about sensor technology applied both in constructed wetlands for water treatment and for autonomous data acquisition on the level of the Dragonfly project greatly showing the possibilities of data acquisition and data analysis for these types of applications.

Overall, the main objectives of this project were achieved with the development of the PCB capable of acquire sensor data and the integration of the database and software interface to access the time-based data that can be easily implemented on both projects, SmarterCW and Dragonfly.

4 Bibliographic references

[1] PINHO, HENRIQUE; MATEUS, DINA - Projeto VALORBIO - Tratamento de Águas Residuais por Zonas Húmidas Construídas Modulares. Uma contribuição para a economia circular. 2019

[2] https://en.wikipedia.org/wiki/Constructed_wetland

[3] Guide for installation of the influxDB and Grafana on Raspberry Pi - https://notenoughtech.com/raspberry-pi/grafana-influxdb/

[4] Guide for installation of the influxDB and Grafana on Raspberry Pi - https://simonhearne.com/2020/pi-influx-grafana/

[5] Github repository of influxDB - https://gist.github.com/boseji/bb71910d43283a1b84ab200bcce43c26

[6] Documentation on the InfluxDB platform relative to query language - https://docs.influxdata.com/influxdb/v1.8/query_language/spec/

[7] Documentation on Node-red for Raspberry Pi - https://nodered.org/docs/getting-started/raspberrypi

[8] RS485 interface for ESP32 based on the Arduino platform - https://www.mischianti.org/2020/05/11/interface-arduino-esp8266-esp32-rs-485/

[9] https://drones.stackexchange.com/questions/1505/lipo-4s-batterys-output-voltage-higher-than-14-8v