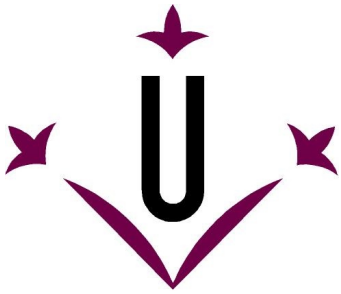

University of Lleida

Master's Degree in Informatics Engineering

Higher Polythecnic School

Technical Documentation

Ubiquitous Computing and Embedded Systems

Francesc Contreras

Albert Pérez

Marc Visa

January 3, 2022

Table of contents

List of Figures

1 Introduction

The purpose of this document is to show the technical documentation of the "Wind Turbine Generator Farm" (WTGF) project which is about designing, testing, and working with micro-controllers simulating.

It will consist of, for each module of the project, its wiring schema, its environment preparation and its development explanation. Furthermore, it will contain the data transmision protocol explained for our project case and some relevant content just to improve the module development explanation.

The motivation lies on providing a full guideline to whoever is interested in the project, and its solution proposal.

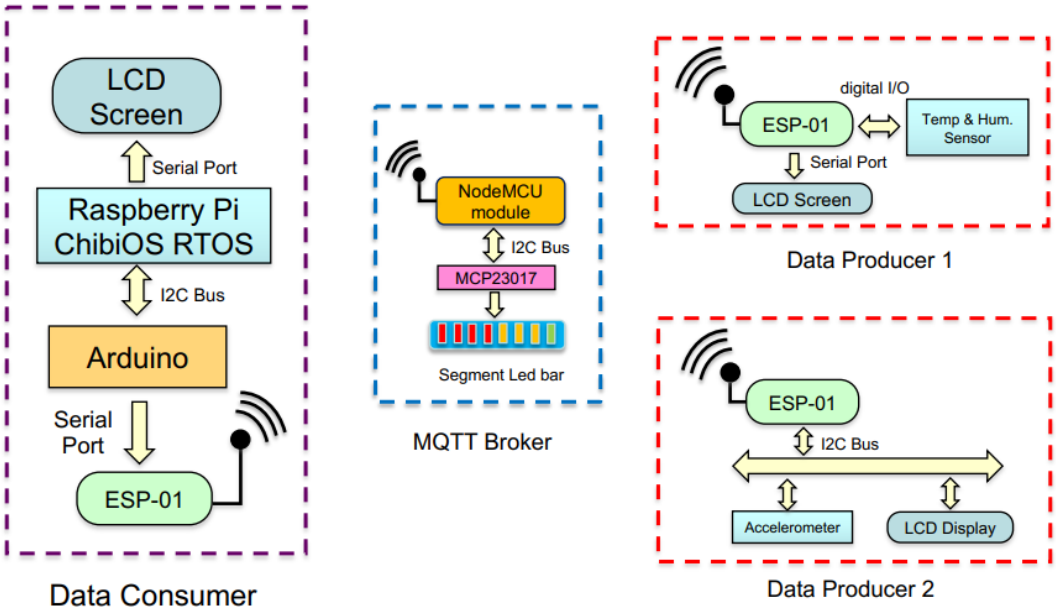

Figure 1: Wind turbine generator prototype schema

2 Data Producer 1

The objective of this module is to get the temperature and the humitity at the top of the wind turbines. Moreover, displaying those values into a LCD Screen in case of in-situ visualization, but also, provide them through a WI-FI interface enabling the collection of this data.

2.1 Environment

At first, these are the steps for installing the libraries and setting up the Arduino IDE to get started for the Data Producer 1 development:

1. Install Arduino IDE v.1.8.16.
2. Install driver for USB adapter to ESP-01 and switch the interrupter to PROG mode.
3. Install ESP8266 Board from Boards Manager from Arduino IDE.
4. Install the following libraries from the Manage Libraries of Arduino IDE:
   (a) DHT [1] library by Adafruit.
   (b) Adafruit Unified Sensor by Adafruit.
5. Install the following libraries from GitHub: (These libraries should be added to the Arduino/libraries folder):
   (a) PubSubClient[2]
6. Connect the ESP-01[3] along with the USB adapter to your machine.
7. Code the program for the ESP-01 and DHT11.

2.2 Material

Subsequently, you will find all the required components for the Data Producer 1, and a short explanatory description for those not so common.

- **ESP-01 (3.3 V)**: it is a Wi-Fi module that allows microcontrollers access to a Wi-Fi network.
- **Breadboard**: A board, having a matrix of small holes to which components may be attached without solder.
- **LCD Screen (5V)**: it is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers.
- **DHT11 (3.3V)**: it is a basic, ultra low-cost digital temperature and humidity sensor.
- **VCC Protoboard adapter (3.3V, 5V)**
- **ESP Programmer module**
- **Wires**

[1] Temperature and humidity sensor
[2] MQTT Library
[3] Low-cost WiFi microchip with built-in TCP/IP networking software, and micro-controller capability

2.3 Wiring Schematics

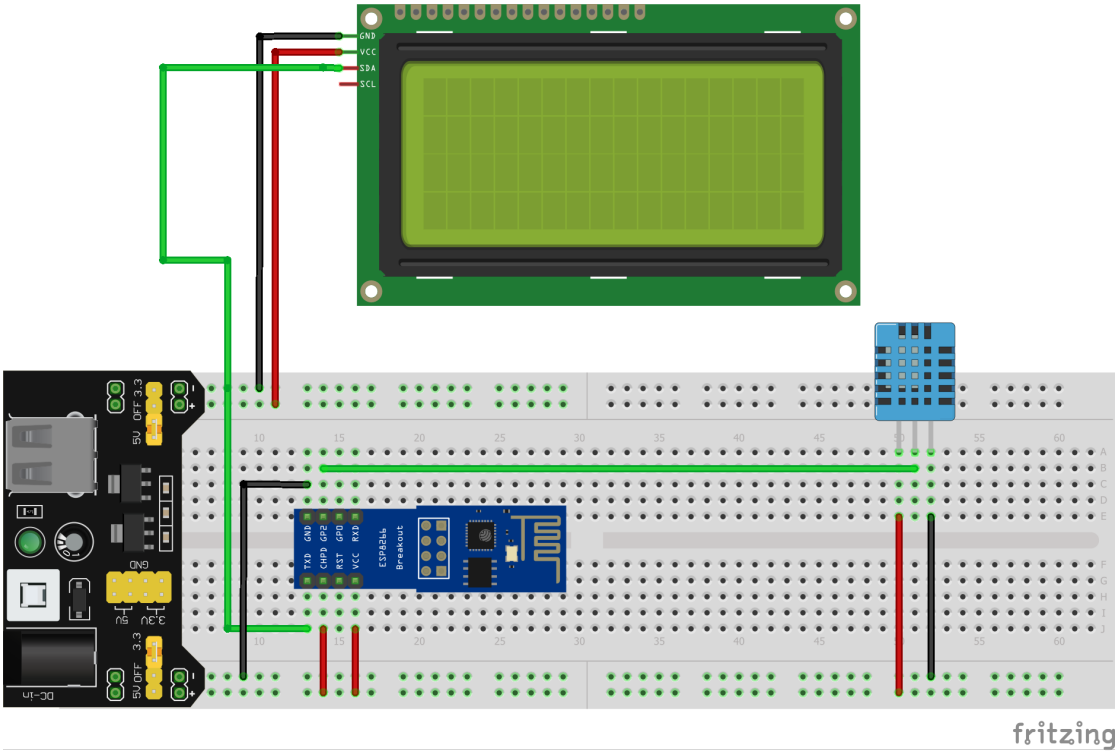

Figure 2: Data Producer 1 Wiring Schema

Pay Attention!
In our LCD Screen we don't have SDA and SCL connections, we have RX and TX, then our RX is SDA and TX is SCL in Figure 11.

3 Data Producer 2

The objective of this module is to register the movement of the wind turbines. In this way, it will get the space components (x, y, z) from the accelerometer sensor, thus, enabling to find out the total acceleration by time unit of the tower. Also, it will display the values into a LCD Display, and provide them via a WI-FI interface.

3.1 Environment

At first, these are the steps for installing the libraries and setting up the Arduino IDE to get started for the Data Producer 2 development:

1. Install Arduino IDE.
2. Install driver for USB adapter to ESP-01 in PROG mode.
3. Install ESP8266 Board from Boards Manager from Arduino IDE.
4. Install the following libraries from the Manage Libraries of Arduino IDE:
   (a) SparkFun ADXL345
   (b) Adafruit Unified Sensor by Adafruit.
5. Install the following libraries from GitHub: (These libraries should be added to the Arduino/libraries folder):
   (a) LiquidCrystal I2C[4]
   (b) PubSubClient[5]
6. Connect the ESP-01[6] along with the USB adapter to your machine.
7. Code the program for the ESP-01.

3.2 Material

Subsequently, you will find all the required components for the Data Producer 2, and a short description for those not so common.

- **ESP-01 (3.3 V)**: is a Wi-Fi module that allows micro-controllers access to a Wi-Fi network.
- **Breadboard**: A board, having a matrix of small holes to which components may be attached without solder.
- **LCD Display (5V)**: is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers.
- **ADXL345 (3.3V)**: is an component that measures the acceleratometer, it allows us to use I2C or SPI .

[4]We use this custom library, because it correct the bugs related to ESP-01 using Arduino IDE v 1.8.16
[5]MQTT Library
[6]Low-cost WiFi microchip with built-in TCP/IP networking software, and micro-controller capability

- **VCC Protoboard adapter (3.3V, 5V)**
- **ESP Programmer module**
- **I2C Bus**: is an component that extends I2C Bus and allows us to use with another components such as LCD Display.
- **Wires**

3.3 Wiring Schematics

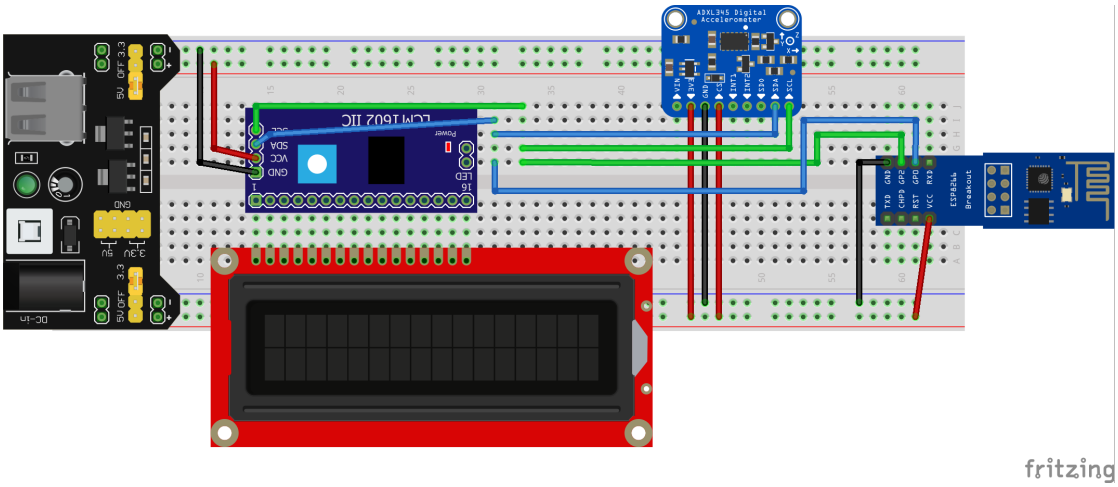

Figure 3: Data Producer 2 Wiring Schema

4 MQTT Broker

MQTT broker provides fast, efficient, and reliable movement of data to and from connected IoT devices. It uses MQTT protocol to transmit and recive the data from the devices connected. Then, it will receive and make availble the data from the Data producers at the wind turbines to whom is subscribe to the corresponding topics.

4.1 Environment

Firstly, these are the steps for installing the libraries and setting up the Arduino IDE to get started for the MQTT Broker development:

1. Install ESP8266 Board from Boards Manager from Arduino IDE.
2. Install the following libraries from the Manager Libraries of Arduino IDE:
   (a) Adafruit MCP23017
   (b) Adafruit Unified Sensor by Adafruit.
3. Install the following libraries from GitHub: (These libraries should be added to the Arduino/libraries folder):
   (a) uMQTTBroker
4. Select NodeMCU v1.0 as compilation environment.
5. Connect the NodeMCU v1.0 along with the micro-USB adapter to your machine.
6. Code the program for NodeMCU v1.0.

4.2 Material

Subsequently, you will find all the required components for the Data Producer 2, and a short description for those uncommon ones.

- **NodeMCU**: open source firmware for which open source prototyping board designs are available. The firmware is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. Initially included firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which was based on the ESP-12 module.[6][7] Later, support for the ESP32 32-bit MCU was added.
- **Breadboard**: A board, having a matrix of small holes to which components may be attached without solder.
- **MCP23017**: Component that measures the accelerometer sensor, it allows us to use I2C or SPI. It is a port expander that gives you virtually identical PORTS compared to standard micro-controllers e.g. Arduino or PIC devices and it even includes interrupts. It gives you an extra 16 I/O pins using an I2C interface as well as comprehensive interrupt control.
- **Wires**
- **LED**

4.3 Wiring Schematics

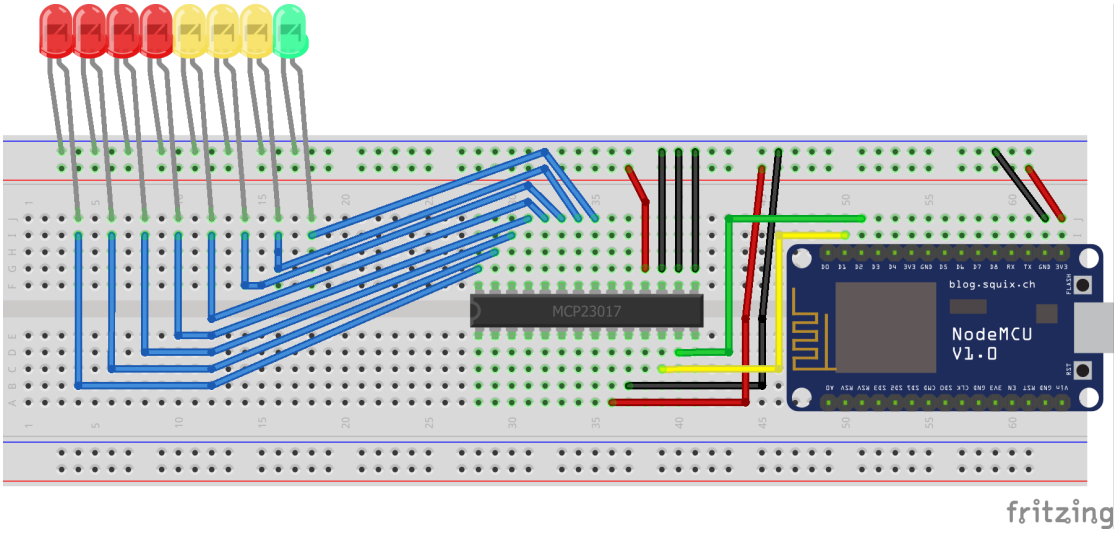

Figure 4: MQTT Broker Diagram

4.4 Connections Visualization

To control the connections to the MQTT Broker we decide to use LEDs that represents the number of subscribers that are connected. Then, we make use of the MCP component to enable far multiple pins for the simulated LED bar.

The functioning of the LED bar is, basically, one led activated for active connection.

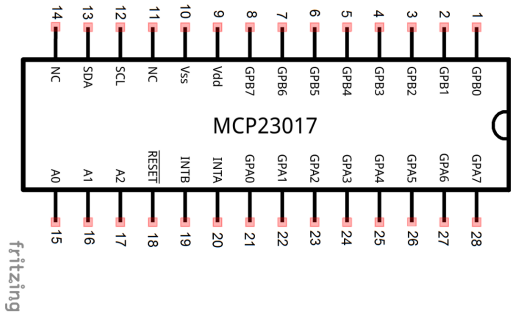

Figure 5: MCP23017 Schema

Adafruit PIN	**Pin Name**	**Physical Pin**
0	GPA0	21
1	GPA1	22
2	GPA2	23
3	GPA3	24
4	GPA4	25
5	GPA5	26
6	GPA6	27
7	GPA7	28
8	GPB0	1
9	GPB1	2
10	GPB2	3
11	GPB3	4
12	GPB4	5
13	GPB5	6
14	GPB6	7
15	GPB7	8

Table 1: Adafruit MCP23017 pin mapping

MCU Pin	MCU Pin Name	MCP23017 Pin	MCP23017 Pin Name
17	D1	12	SCL
18	D2	13	SDA
21	3V3	18	Reset
21	3V3	9	VDD
22	GND	10	VSS
22	GND	15	A0
22	GND	16	A1
22	GND	17	A2

Table 2: Our schema

For the testing process of the development, the team used the MQTT Explorer in order to check if there were real connections with the broker and also check if the Data Producers sent the data correctly.

Additionally, when testing, if a client was connected, that was coded as the green LED lighted, if only one client was connected. If two clients were connected, the green one and the first yellow one were lighted and so in succession.

5 Data transmission

This section reports how the data transmission works between the MQTT Broker and the other devices subscribed.

The transmission of the data uses MQTT Protocol to transfer data between devices and the MQTT Broker. In our case, the MQTT Broker and the devices are connected to the same network to transmit and receive data. However, in the real world, most cases use a VPN[7] to connect the devices to MQTT Broker and access remotely from anywhere which is an awesome solution.

The following are detailed for each data producer device how its data is received and represented to whoever is interested in registering their values.

Note, that we are using the MQTT Explorer software to access the values of the broker.

5.1 Data Producer 1

In the first picture below, it can be seen the number of connections that the MQTT Broker has, along with the retrieved data from the temperature and humidity sensor.

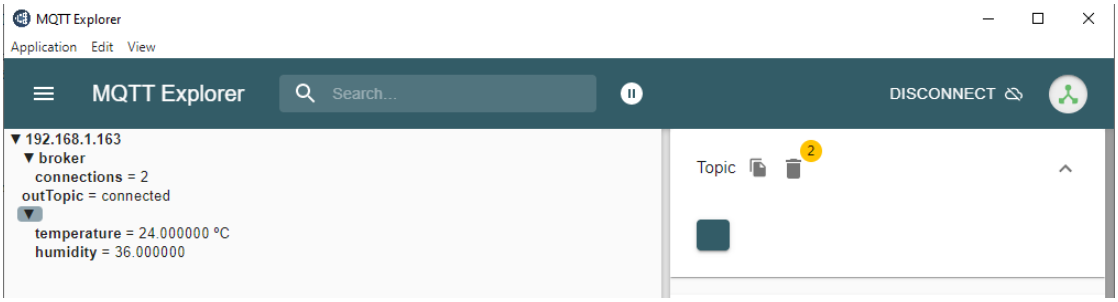

Figure 6: MQTT Explorer for Data Producer 1

For the second picture, it can be seen in the right side of the Explorer some information regarding the topics, values of the MQTT Broker. In this case, we have two topics, the humidity topic and the temperature topic. As for the values it can be observed a history of the temperature and humidity sensor values, which the broker has retrieved.

[7] Virtual Pricate Network

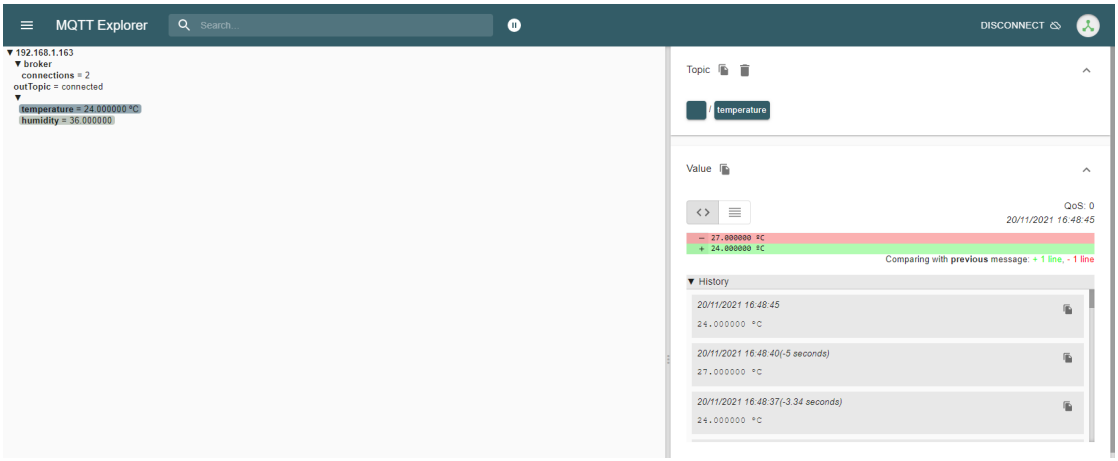

Figure 7: MQTT Explorer for Data Producer 1 History and values

For this third picture as regards Data Producer 1, it can be observed that the Explorer shows some chart that renders the high and low peaks, which represents the data retrieving period as a time-series.

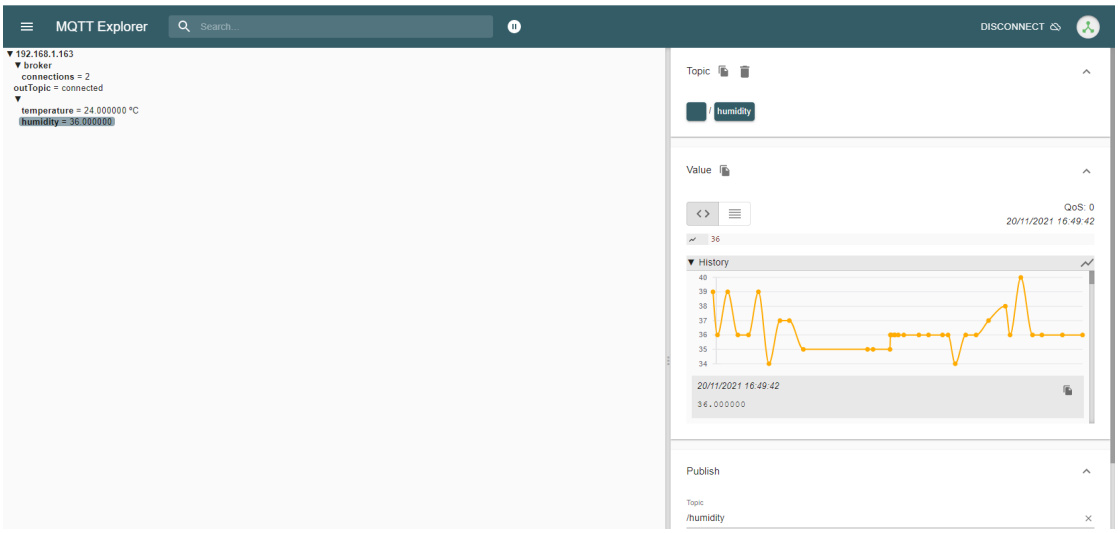

Figure 8: MQTT Explorer History chart

5.2 Data Producer 2

For the Data Producer 2 in the MQTT Broker, it is defined the acceleration topic, in which the accelerometer publishes the data and the Broker retrieves it.

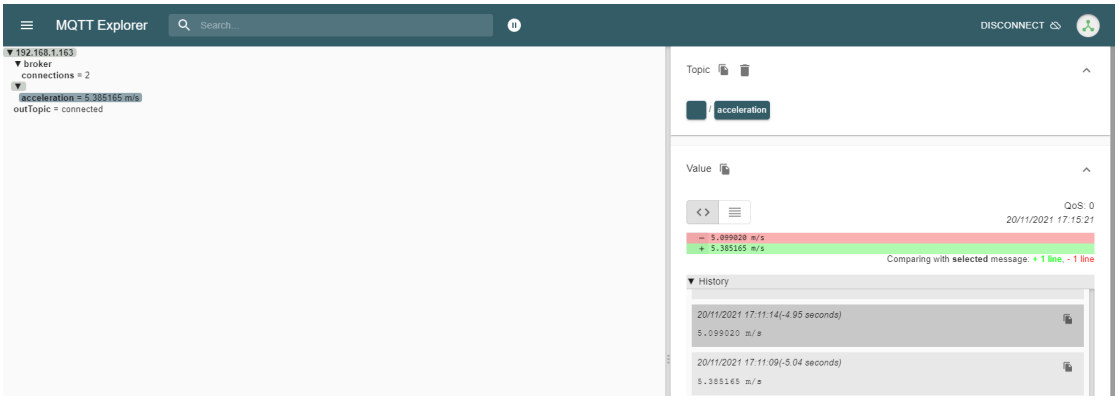

Figure 9: MQTT Explorer for Data Producer 2

5.3 Both Data Producers

For the last picture, it can be observed both Data Producers that are publishing data in the three mentioned topics previously (humidity, temperature, acceleration) and the MQTT is retrieving all the data by the topics and shows the values of the respective topics on the left part of the screen of the MQTT Explorer.

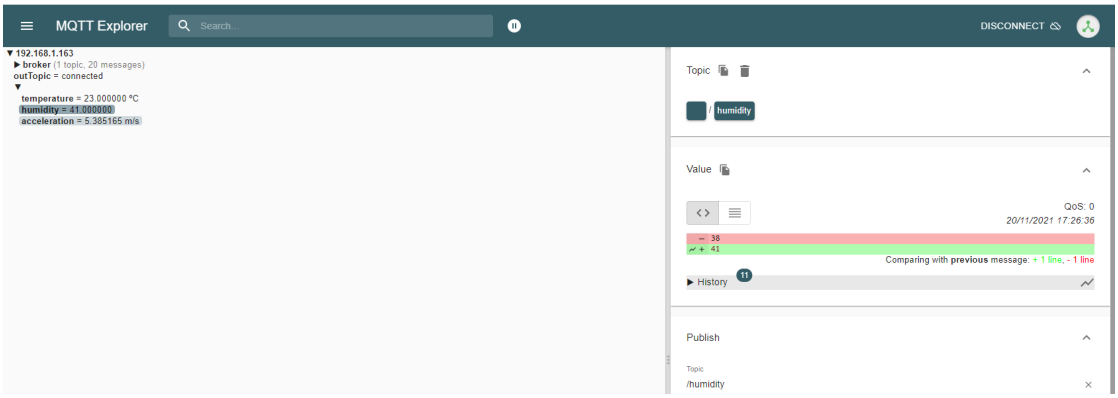

Figure 10: MQTT Explorer for both Data Producers

6 Supervision Station

The objective of this module is to collect the data from the MQTT Broker subscribing to its service, and make them accessible fo historical registering purposes.

This section will detail all to care about the Supervision station. At first, the environment and materials needed to accomplished the development are introduced. But the key section remarks on explaining the criteria and decisions taken while developing.

So, in the development section, the idea is to focused not only on detailing the final solution but also to care about all the steps and decisions taken during the process. In this manner, it makes sense the final solution when specifying its underground.

6.1 Environment

In this section, we will introduce you to preparing the environment to work and program the supervision station, and also for each of the main components involved is specified specific criteria to develop their functionalities.

Firstly, to work with the *Arduino UNO* and *ESP-01* components we should,

1. Install Arduino IDE.

It is used in order to program the sketches, from which the components are based on to develop the supervision station functionality.

Subsequently, there are specific remarks for each of these components regarding the Arduino IDE dependencies such as libraries used, and some criteria about working with them.

As for the ESP-01:

1. Arduino IDE dependencies.
   - Install driver for USB adapter to ESP-01[8] in PROG mode.
   - Install *ESP8266 Board* from the Arduino IDE Boards Manager.
   - Install the following libraries from GitHub: (These libraries should be added to the Arduino/libraries folder):
     (a) PubSubClient[9]
2. Use a *USB adapter* to connect the ESP-01 to your machine in order to set up the sketch programmed.
3. Code the program/sketch.

[8] Low-cost WiFi microchip with built-in TCP/IP networking software, and micro-controller capability
[9] MQTT Library

As for the Arduino UNO:

1. Connect the *Arduino Uno* to your machine via USB to set up the sketch.
2. Code the program/sketch.

Then, the raspberry Pi is quite different as actually we were using a *Linux distribution* (Ubuntu), not an IDE, to develop its functionality. In this way, you will need a machine running Ubuntu if following our own way to work. The following are specified the dependencies and criteria to be taken into account.

As for the Raspberry Pi:

1. Install Developer ARM Toolchain
   (a) Ubuntu:

   ```
   sudo apt-get install gcc-arm-none-eabi
   ```

   (b) Windows 10: gcc-arm-none-eabi
2. Download ChibiOS-RPi for Raspberry Pi
3. Prepare SD-Card
   (a) bootcode.bin
   (b) start.elf
4. Check arm-none-eabi-gcc

   ```
   arm-none-eabi-gcc --version
   ```

5. Compile your code using "make" command.
6. Rename the file build/ch.bin to kernel.img
7. Load kernel.img to SD-Card

6.2 Material

Next, you will find all the required components for the Supervision Station, and a brief description for those uncommon ones.

- **Raspberry Pi**: a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.
- **ChibiOS/RT**: a compact and fast real-time operating system supporting multiple architectures and released under a mix of the GNU General Public License version 3 (GPL3) and the Apache License 2.0 (depending on module).
- **Arduino UNO**: a micro-controller board based on the ATmega328P (data-sheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the micro-controller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.. You can tinker with your Uno without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.
- **ESP-01**: is a Wi-Fi module that allows micro-controllers access to a Wi-Fi network.
- **LCD Serial Screen**: is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers.
- **Breadboard**: a board having a matrix of small holes to which components may be attached without solder.
- **ESP Programmer module**
- **Wires**

6.3 Development

The requirements were to use a Raspberry Pi to show historical data through an LCD Screen, from the data received from the Arduino via an I2C Bus communication. Moreover, the Arduino should get the data from the MQTT Broker from the topics set up, such as temperature, acceleration, and humidity.

In this way, the main idea was to apply a master-slave schema among the Raspberry PI and the Arduino respectively, and use an ESP-01 as a WIFI interface to enable the Arduino to subscribe to the MQTT Broker topics. The following is the wiring schema for our first proposal:

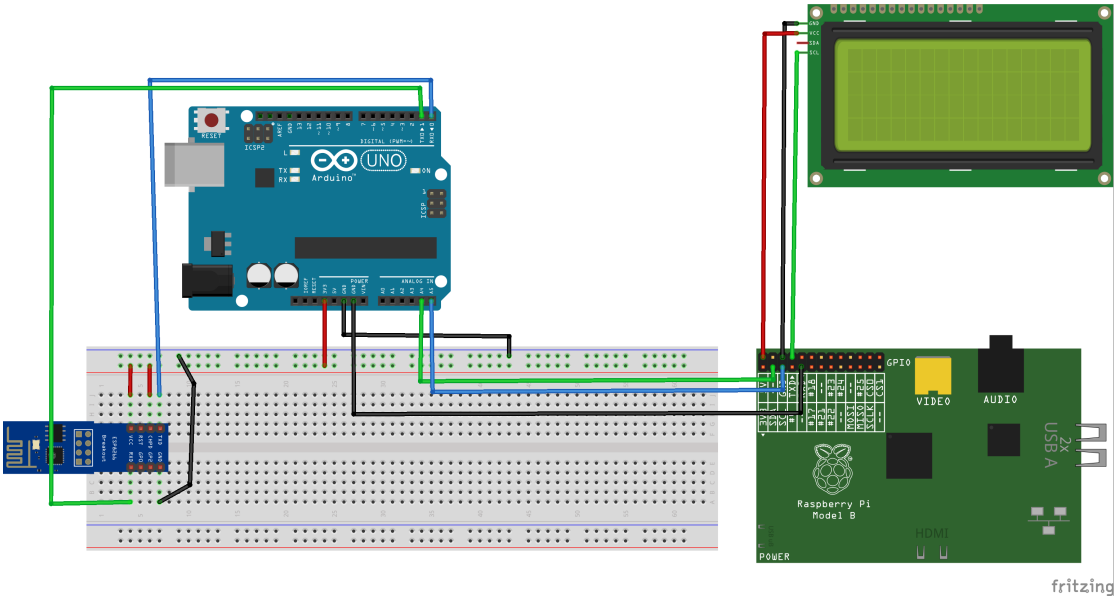

Figure 11: Solution A Schema

The good point of this solution is, basically, that the ESP-01 doesn't have to be programmed, just configured through the Arduino sketch to work for it. Also, the master-slave schema will enable the Raspberry Pi to access the data Arduino is saving at the time required. But, it should be considered the capacity of the Arduino in the case of a scaling system. Apart from that possible constraint, we still think that the Raspberry Pi should be the master module in the communication.

To configure the ESP-01 as a WI-FI interface from the Arduino sketch you should use the *AT Commands*. But, in our case, the ESP-01 was not answering the orders. We found out that the original firmware was flashed as we previously uploaded some of our own sketches. So, we tried to reset the firmware to make the ESP-01 answer the *AT Commands*. Sadly, we couldn't accomplished this approach.

We used multiple tools to upload some firmware but there was no way the ESP-01 answered the commands. We still think this is the best option and we encourage other teams to take it into account, but we needed to focus on not blocking the development and getting another solution.

Then, a second idea came up to encourage the project. Simply program the ESP-01 and use a client-server schema to pass the data to the Arduino throughout a custom protocol, not too complex, just to approximate the best efficient communication. As for the assembly scheme, it is the same as shown in the previous image.

The Arduino would request to the ESP-01, and this would send the subscribed data from the MQTT Broker. In our case, the temperature, humidity, and acceleration.

The data will be sent as a particular format, which is a string of 6 or 7 characters, of which the *n0* and *ni - 1* will define the size and the type of size respectively, For instance, 26.00t, this value denies we have 26.00 degrees and the character 't' means 'temperature. While 'a' stands for 'acceleration' and 'h' for humidity'.

Listing 1: Custom Protocol Trace Example

```
26.00t38.00h5.12a
22.00t
25.00t39.00h
9.81a
```

It is obvious we are adding another layer to the communication, which is unnecessary as adding new memory and time synchronization constraints.

At the moment, the team got the first part of the data transfer solved. Now, let's talk about the I2C Bus communication between the Raspberry PI running ChibiOS, and the Arduino. As we just commented, we proposed to use a master-slave schema and when the Raspberry PI requires new data just it will need to request to the Arduino through the I2C Bus.

We programmed some sketches to test the communication, but the data transfer was failing. First, we thought about the code. Maybe there's an error. So, we tried some unitary communication tests. But again, they didn't work properly. Only at the beginning we were receiving data, but till that, no more. Then we started checking the assembly, even presenting it all to the professor in case there is something we couldn't see.

This was a big mishap. We got all the parts working but not the communication between the Raspberry PI and the Arduino. Also, we were using the ChibiOS Raspberry PI to represent the historical data into the LCD Screen correctly, but using static values though.

Moreover, we commented in class the problem to share it with our partners and the professor. They told us a possible solution, to reset the I2C Bus communication when the action code received were RESET. But, even trying this, it didn't work. So, we redefined the schema to find out a solution in order to accomplished the user stories, what the client would expect.

Finally, we are using the Arduino for representing the data into the LCD screen. Hence, the Raspberry PI is disabled for non-working purposes. Furthermore, the main idea is to use the

MQTT Explorer which is an open-source and free software to allow us to connect to the MQTT Broker and get the data as a detailed and graphical way. It also provides us the fluctuation in the data, connection problems, registration, etc.

Why we bet for this solution? First because of the problems encountered. But also, as it is a cheaper, more vialbe and more versatile way to obtain a Supervision station seeing in real time the values but also using a software to enable historical representation. We strongly recommend to make use of this solution, although it would be better to use the ESP-01 as a WI-FI interface, since it is more simple and more scalable because we just need to use the MQTT Explorer for visualizing. Even using VPN's we can connect to the several brokers in scaling terms, directly from our home. Nonetheless, there will be always an Arduino board displaying the values at each time, just similar as the Raspberry Pi would do.

The following would be the final wiring schema for our solution:

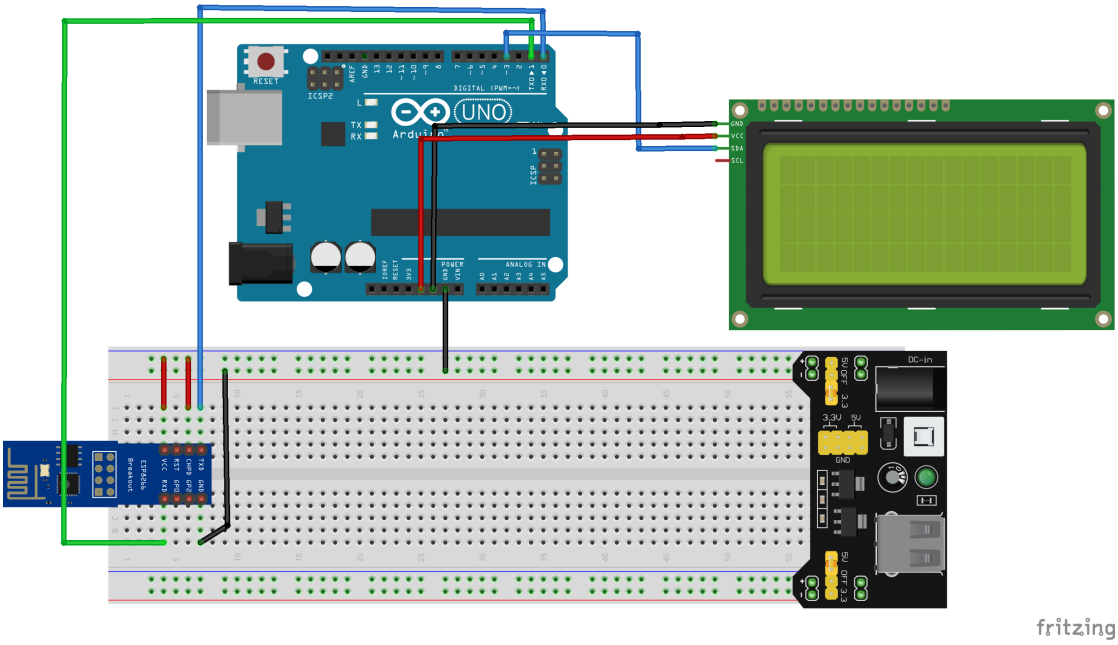

Figure 12: Solution B Schema

7 Project Source Code

Our git repository: Wind Turbine Generator Github repository

Appendices

A Data Producer 1

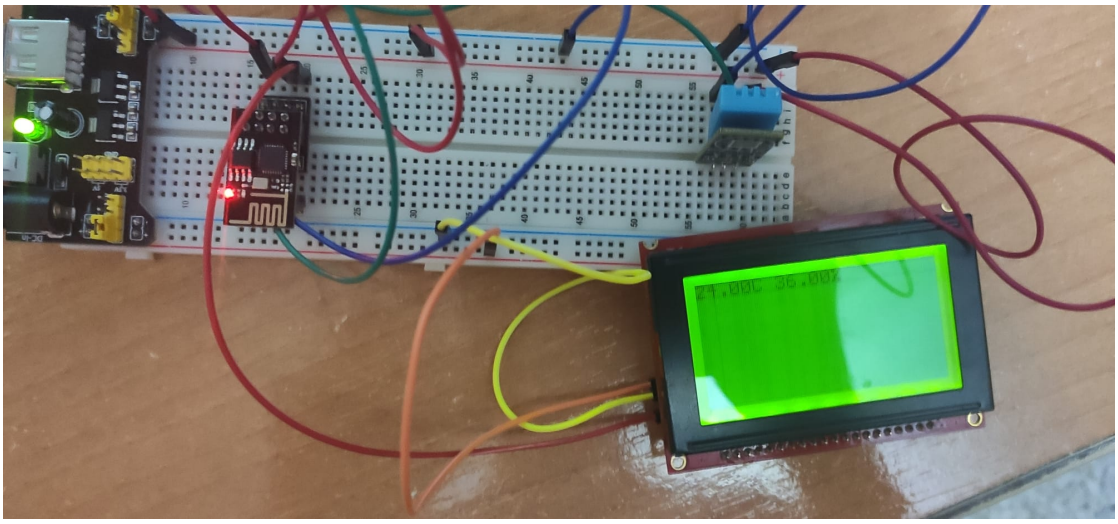

Figure 13: Data Producer 1 Assembly

B Data Producer 2

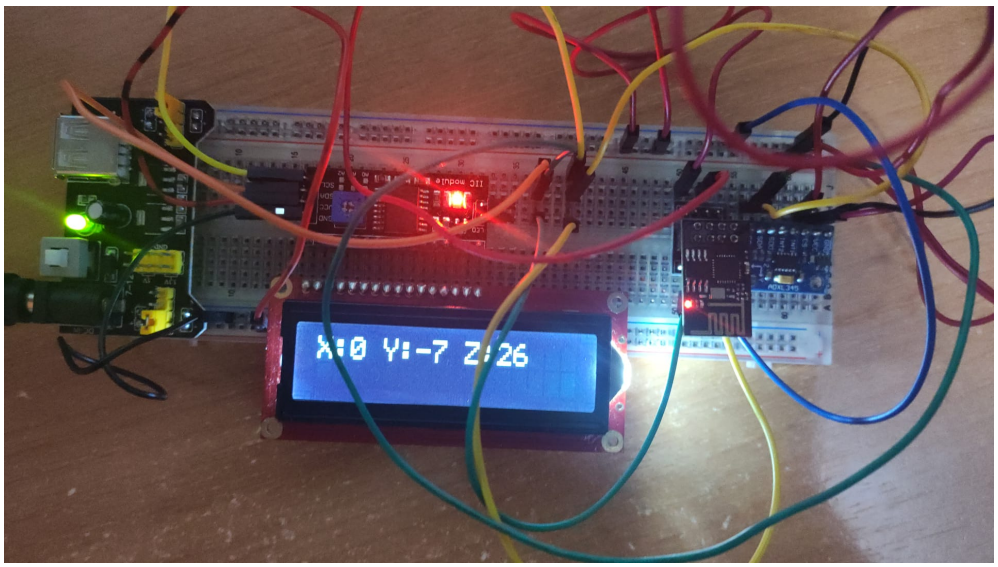

Figure 14: Data Producer 2 Assembly

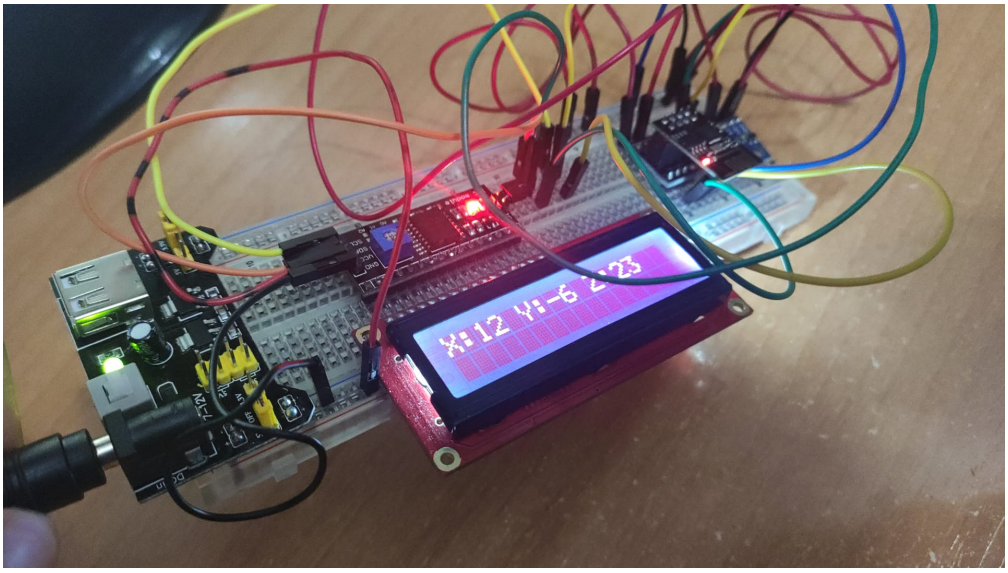

Figure 15: Data Producer 2 in movement

C MQTT Broker

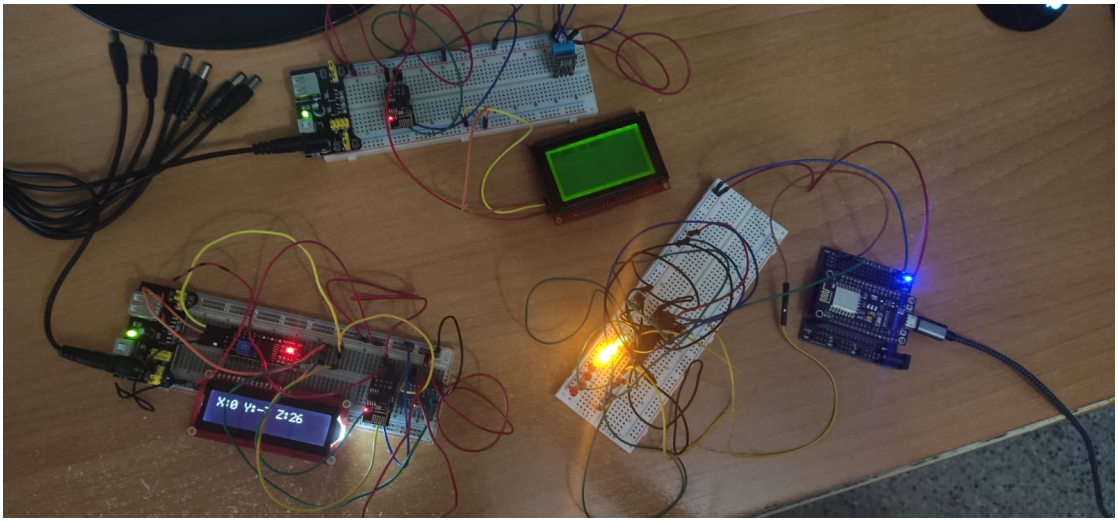

Figure 16: MQTT Broker Assembly

D MQTT Explorer

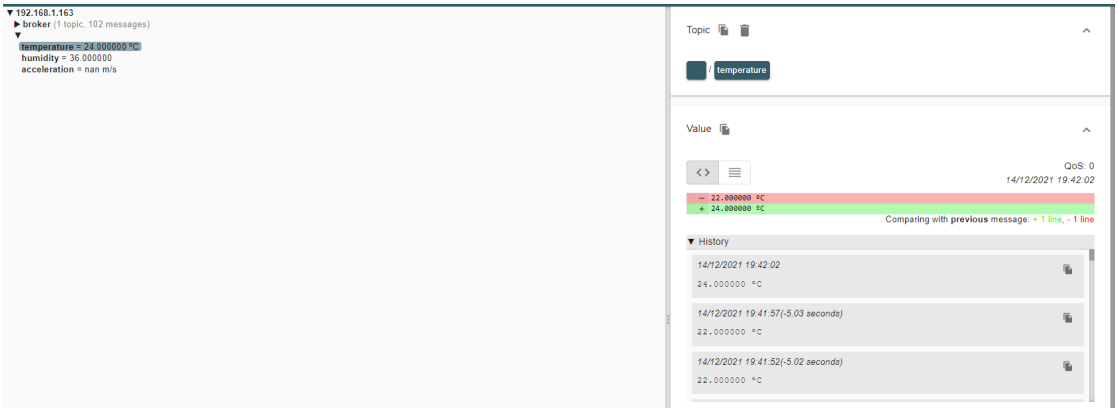

Figure 17: Temperature Changes

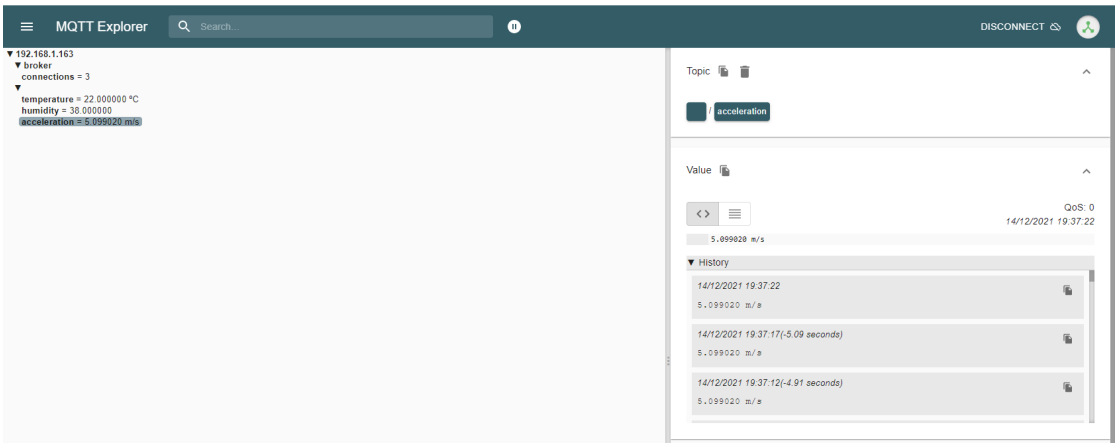

Figure 18: Acceleration Changes

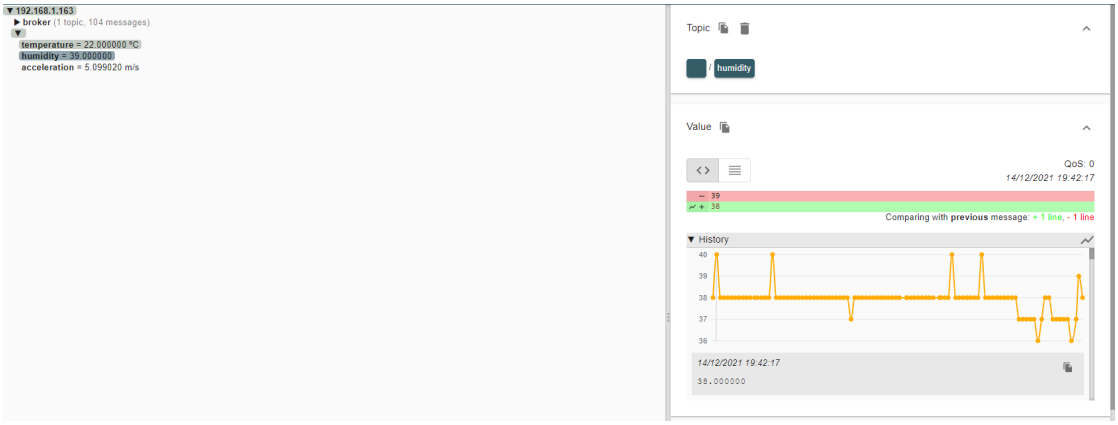

Figure 19: Humidity History

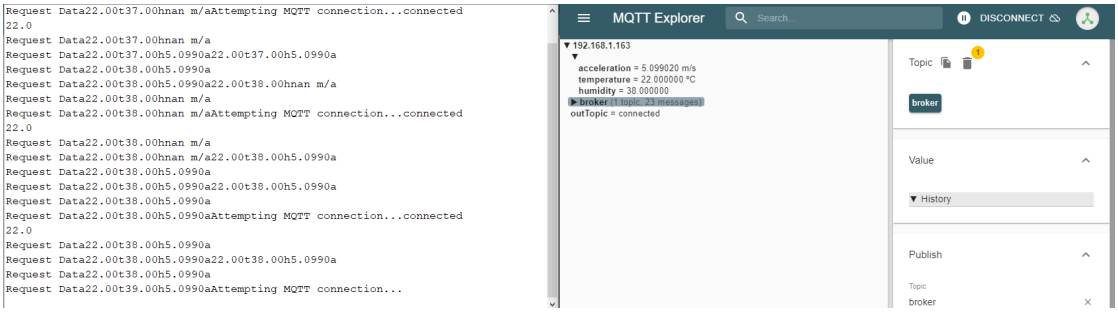

Figure 20: Comparison between Solution B and C