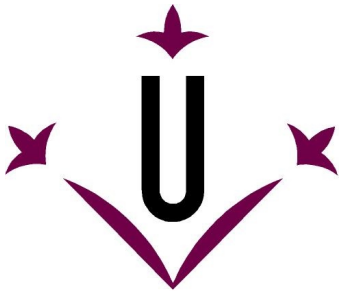

University of Lleida

Master's Degree in Informatics Engineering

Higher Polythecnic School

Supervision Station

Ubiquitous Computing and Embedded Systems

Francesc Contreras

Albert Pérez

Marc Visa

November 10, 2021

Table of contents

List of Figures

1 Introduction

The purpose of this document is to report the Supervision Station development for the third sprint of the project which corresponds to the Data Consumer. We shall introduce all the steps and software we have used when developing it and showing images as a real example for detailed explanation.

The document is structured on introducing, at first, the requirements and materials needed to accomplished the development of the Supervision Station. 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.

2 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.

Used 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.

2.1 ESP-01

1. Arduino IDE dependencies:
   - Install driver for USB adapter to ESP-01[1] 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[2]
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.

2.2 Arduino Uno

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

[1] Low-cost WiFi microchip with built-in TCP/IP networking software, and microcontroller capability
[2] MQTT Library

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.

2.3 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

2.4 Material required

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 microcontroller board based on the ATmega328P (datasheet). 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 microcontroller; 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 microcontrollers 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**

3 Development

This section explains the whole development of the Supervision Station, as well as the problems and issues encountered, thus, the precise solutions and decisions taken regards. The idea is to present the roadmap followed as being a comprehensive guide to what was our development procedure.

When you face up a project, the first step is to analyze how will the development procedure work, and even the solution or solutions to test. In this way, we though about how to implement the supervision station.

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 schema for our first proposal:

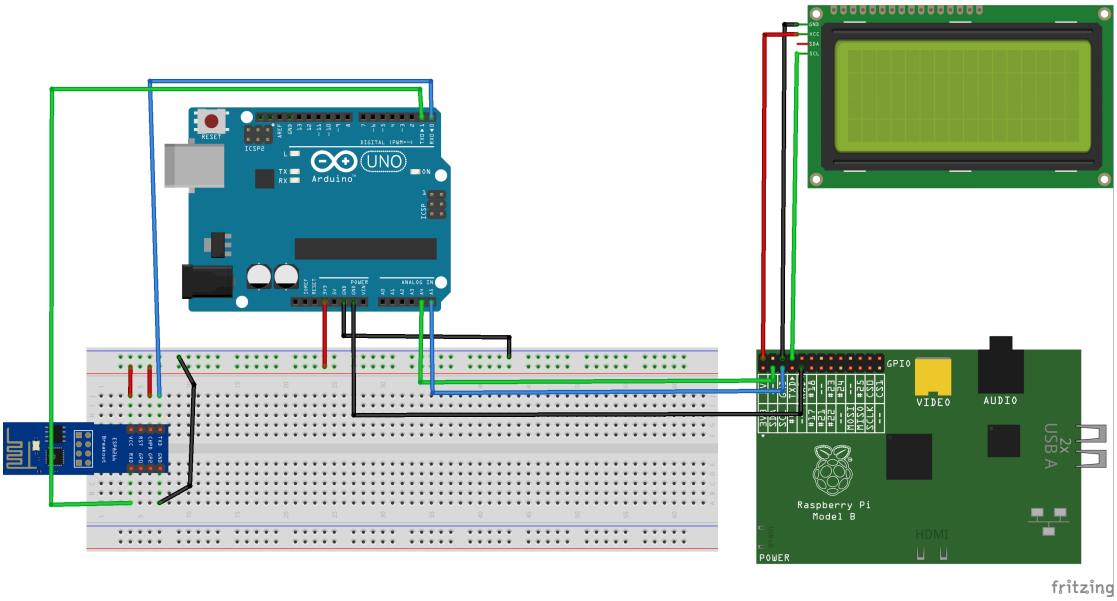

Figure 1: Solution A Schema

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, we should consider the capacity of the Arduino in the case of a scaling system. Apart from that possible constraint, we still think the Raspberry Pi should be the master module in the communication.

To configure the ESP-01 as a WIFI 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 the original firmware was flashed as we previously uploaded some own sketches. So, we tried to reset the firmware to make the ESP-01 answer the *AT Commands*. Sadly, we couldn't.

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 you to take it into account, but we had to keep going because of the deadline, in this manner, we focused 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, 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'.

It's obvious we are adding another layer to the communication, which is unnecessary as adding new memory and time synchronization constraints. But, due to the problems encountered and the deadline as if we were working in real for clients, we had to continue developing among the problems. Even though, documenting all the processes in case we have time to remodel.

At the moment, we 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 the use of this solution, although it would be better to use the ESP-01 as a WIFI interface, since it is more simple and more scalable because we just need to use the MQTT Explorer for visualizing. Even using VPNs we can connect to the several brokers in scaling terms, directly from our home. And, on the spot, 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 schema for our solution:

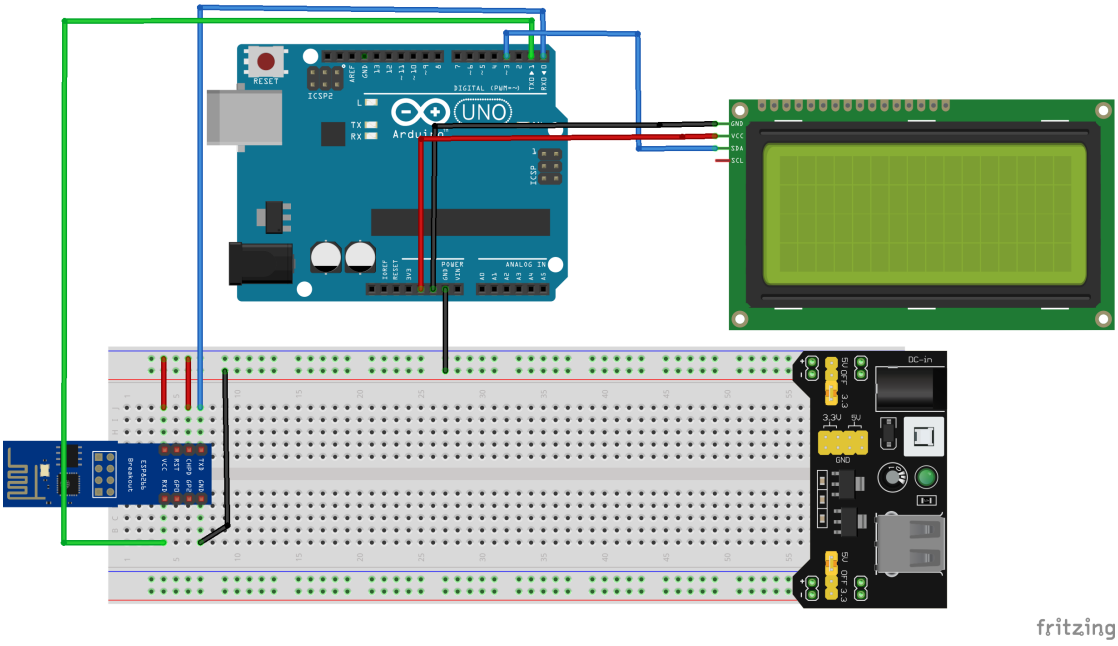

Figure 2: Solution B Schema

3.1 Software

3.1.1 ESP-01

```
//Libraries
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Constants
WiFiClient espClient;
PubSubClient client(espClient);

// MQTT
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
const char* mqtt_server = "192.168.1.163"; // todo: change ip

// Wi-Fi
const char* ssid = "-";
const char* password = "-";

// Data
char temp[32];
char hum[32];
char acc[32];

void callback(char* topic, byte* payload, unsigned int length) {

  // Temperature /temperature
  if((char)topic[1] == 't'){
           for (int i = 0; i < 5; i++) {
          temp[i] = (char)payload[i];
      }
      temp[5]='t';
  }

  // Humidity
   if((char)topic[1] == 'h'){
           for (int i = 0; i < 5; i++) {
          hum[i] = (char)payload[i];
      }
      hum[5]='h';
   }
  // Acceleration
  if((char)topic[1] == 'a'){
           for (int i = 0; i < 6; i++) {
        acc[i] = (char)payload[i];
    }
    acc[6]='a';
  }

}
```

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "AT";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "connected");
      // ... and resubscribe
      client.subscribe("#");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {

  Serial.begin(9600);

  // WI-FI
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println(WiFi.localIP());

  // MQTT Subscribe
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  client.subscribe("/temperature");
  client.subscribe("/humidity");
  client.subscribe("/acceleration");

}

void loop() {

  // MQTT Connection
   if (!client.connected()) {
    reconnect();
```

```
  }

  client.loop();

  // Response request to Arduino
   if(Serial.available() > 0){
    Serial.write(temp);
    Serial.write(hum);
    Serial.write(acc);
   }

  delay(4000);
}
```

3.1.2 Arduino UNO

```
#include <SoftwareSerial.h>

#define rxPin 2 // receive
#define txPin 3 // transmit

SoftwareSerial lcdSerial = SoftwareSerial(rxPin, txPin);
String msg;
String temp[10];
String hum[10];
String acc[10];

void clean_screen()
{
  lcdSerial.write((byte)0x7C);
  lcdSerial.write((byte)0x00);
}
void coordenades(int x, int y)
{
  lcdSerial.write((byte)0x7C);
  lcdSerial.write((byte)0x18);
  lcdSerial.write((byte)x);
  lcdSerial.write((byte)0x7C);
  lcdSerial.write((byte)0x19);
  lcdSerial.write((byte)y);
  delay(20);
}

void pixel (int x, int y, int on_off)
{
  lcdSerial.write((byte)0x7C);
  lcdSerial.write((byte)0x10);
  lcdSerial.write((byte)x);
  lcdSerial.write((byte)y);
  lcdSerial.write((byte)on_off);
}
```

```
void line (int des_de_x, int des_de_y, int fins_a_x, int fins_a_y, int on_off)
{
  if (des_de_x<fins_a_x)
  {
    lcdSerial.write((byte)0x7C);
    lcdSerial.write((byte)0x0C);
    lcdSerial.write((byte)des_de_x); // Primer punt
    lcdSerial.write((byte)des_de_y);
    lcdSerial.write((byte)fins_a_x); // Segon punt
    lcdSerial.write((byte)fins_a_y);
    lcdSerial.write((byte)on_off);
  }
  else
  {
    lcdSerial.write((byte)0x7C);
    lcdSerial.write((byte)0x0C);
    lcdSerial.write((byte)fins_a_x); // Segon punt
    lcdSerial.write((byte)fins_a_y);
    lcdSerial.write((byte)des_de_x); // Primer punt
    lcdSerial.write((byte)des_de_y);
    lcdSerial.write((byte)on_off);
  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  lcdSerial.begin(115200);

  // define pin modes for tx, rx, led pins:
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  Serial.println("Start");
  lcdSerial.print("Start");
  delay(1000);
}

void loop() {
  clean_screen();
  // Request Data To ESP-01
  Serial.write("[Req]");
  Serial.println("");

  // Recive Data From ESP-01
  if(Serial.available() > 0){
    Serial.write("[Res]: ");
    msg = Serial.readString();
    Serial.println(msg);
    Serial.println(";");
    }
  lcdSerial.print(msg);
  delay(5000);
}
```

4 Code repository

Wind Turbine Generator Github repository

References