



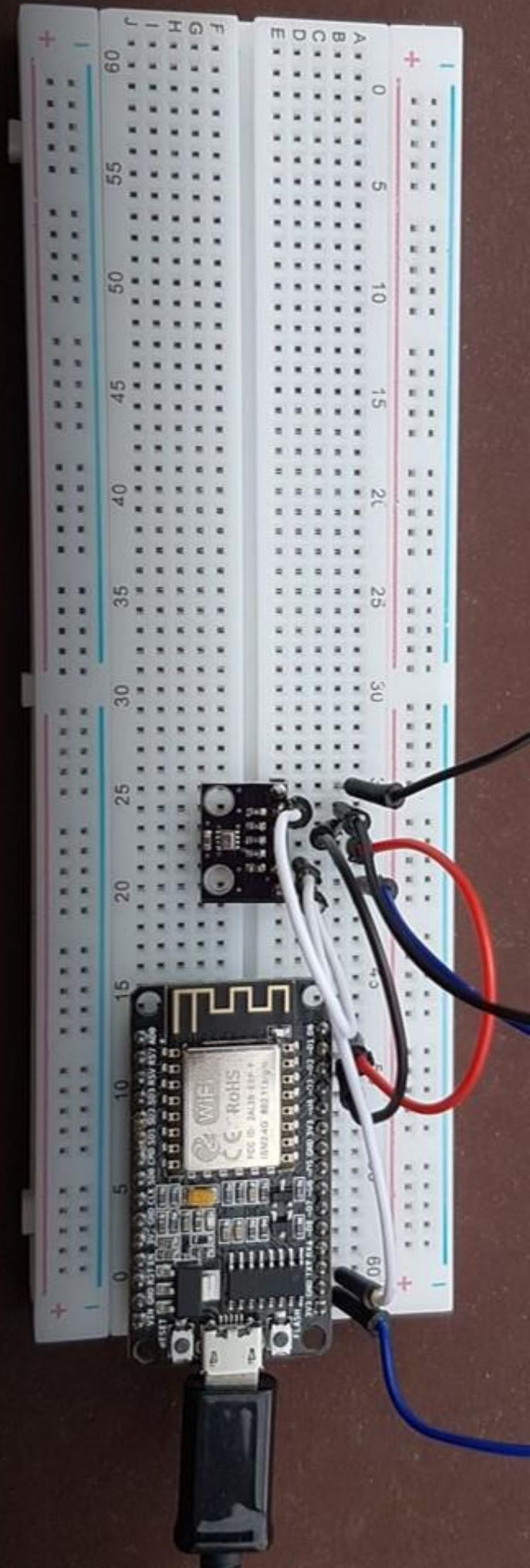
E060 Internet of Things Project Report

SEND TEMPERATURE, PRESSURE AND
ALTITUDE VALUES TO A MQTT SERVER
WITH A MICROCONTROLLER AND A
SENSOR.

DÉBUREAUX Anaïs
HENNION Anaëlle
LESOBRE Hugo

Spring Semester
Project realized from May to July 2020

Teachers:
Mr. Andrzej SMOLARZ and Mr. Tomasz ZYSKA



Contents

Introduction	5
1 Define the equipment	6
1.1 Arduino IDE	6
1.2 ESP-12E WIFI module	6
1.3 BMP280 Temperature and Pressure Sensor Module	7
2 Methodology	8
2.1 Blink Test	8
2.2 Wiring	10
2.3 Detailed Code Explanation	11
3 Results	16
Bibliography	17
Conclusion	18
Annex	19

List of Figures

Figure 1: ESP-12E Pinout	6
Figure 2: ESP-12E.....	6
Figure 3: BMP280	7
Figure 4: Blink Test Program.....	8
Figure 5: Set the built-in led pin as an output	8
Figure 6: Form of the digitalWrite function.....	9
Figure 7: Setting the digitalWrite function to turned on the LED	9
Figure 8: Delay function	9
Figure 9: Setting the digitalWrite function to turned on the LED for 1s	9
Figure 10: Explanation of the 6 pins of the BMP280	10
Figure 11: BMP280 pins scheme	10
Figure 12: Wiring on the breadboard	10
Figure 13: Code line to connect to our WiFi router	11
Figure 14: Code lines to wait for connection to our WiFi router	11
Figure 15: Code line to print out the IP address assigned to the ESP module	11
Figure 16: Code lines to include libraries	12
Figure 17: Code lines to create an object of the sensor BMP280	12
Figure 18: Code lines to define PIN-Wiring	12
Figure 19: Code lines to define Wi-Fi and MQTT's values	12
Figure 20: Code lines to create a client instance	13
Figure 21: Code lines to connect to the WiFi network	13
Figure 22: Code line to specify the address and the port of the MQTT server	13
Figure 23: Code line to test if the sensor is found.....	13
Figure 24: Code line to verify the Wi-Fi connection and try to reconnect	14
Figure 25: Code lines to read and display the sensor values.....	14
Figure 26: Code lines to connect WiFi.....	14
Figure 27: Code line to print a status message in serial monitor about WiFi connection.....	15
Figure 28: Code lines to define a function reconnect() to the MQTT broker	15

Figure 29: Scheme of the shape of the results.....	16
Figure 30: Graphic of the shape of the results.....	16
Figure 31: Results in the serial monitor	16

Acknowledgements

We would like to thank Mr. Andrzej SMOLARZ and Mr. Tomasz ZYSKA for the material made available to us for the realization of the projects.

We would also like to thank Omar BELGHAOUTI who accompanied us throughout this project. He brought us a lot of knowledge on this field and was able to initiate us to some bases of the IoT which were unknown to us.

Introduction

The subject we undertook to study is the use of a BMP280 and an ESP12E.

The objective of the project was to collect and send temperature, pressure and altitude values to a MQTT server with a microcontroller and a sensor.

First of all, the material we used will be define, then our methodology to achieve this project and finally our results.

1 Define the equipment

1.1 Arduino IDE

We used the open-source Arduino software to write our code and upload computer code to the physical board.

The Arduino IDE (Integrated Development Environment) contains a text editor for writing code, a message area which gives feedback while saving and exporting and displays errors, a text console where programmes, called sketches, are written, a toolbar with buttons to verify and upload programs, create, open and save sketches, and open the serial monitor and a series of menus.

It uses a language similar to C or C++, making it easier to learn to program. There are many libraries available in the Library Manager (a large community of users have generated libraries) that we need to install to get our code to compile.

1.2 ESP-12E WIFI module

The ESP-12E is a WiFi module based on ESP8266 with built-in 32Mbit Flash. It allows to control inputs and outputs like an Arduino, but it comes with Wi-Fi because it offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to establish an Internet connection.

It features access to 13 GPIO pins and one analog-to-digital converter (ADC) pin with 10-bit resolution so a total of 30 pins like represented on figure 1.

The board has a built-in voltage regulator and we can power up the module using the mini USB socket or the Vin pin. It also has a built-in LED connected to GPIO2.

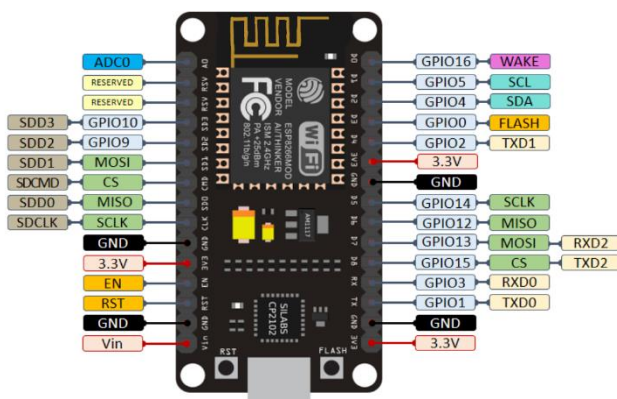


Figure 1: ESP-12E Pinout



Figure 2: ESP-12E

1.3 BMP280 Temperature and Pressure Sensor Module

The BMP280 is a combined temperature and barometric pressure sensor. It is based on Bosch's piezoresistive pressure sensor technology.

Thanks to the link between atmospheric pressure and altitude we can recover altitude data with pressure data by using the sea level pressure. The BMP280 can measure pressure between 300 and 1100 hPa and a temperature between 0 and +65°C.

The device is optimized in terms of power consumption, resolution, filter performance and has small dimensions.

The BMP280 features I2Cⁱ and SPIⁱⁱ (3-wire/4-wire) digital interfaces. Interface selection is done automatically based on CSB status. If CSB is connected to VDDIO, the I2C interface is active. If CSB is pulled down, the SPI interface is activated. In our project we connected CSB to VDDIO so the I2C interface is active. The sensor can be operated in three power modes: the sleep mode, the normal mode and the forced mode. In our project we use the normal mode.

The BMP280 has 6 pins: VCC (Higher voltage +3.3V/5V), GND (Ground, 0V), SCL (System Control Layer, serial clock input), SDA (Serial Data Input), CSB (Chip select) and SDO (Serial Data Output).

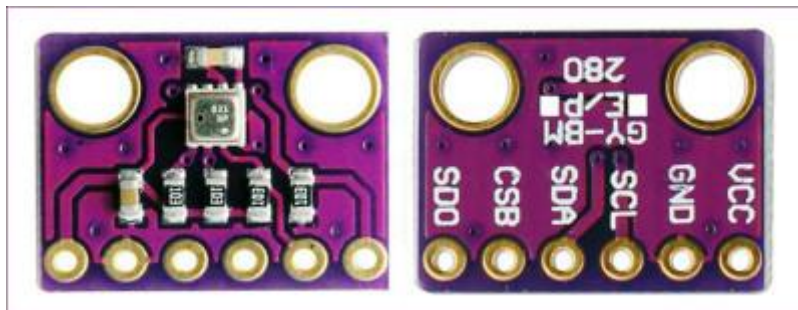


Figure 3: BMP280

2 Methodology

2.1 Blink Test

In order to get use to the given material and be sure that everything was functional, we started with a basic and simple **blink test**. A **blink test** will just make a LED blink. In our case, we'll use the integrated LED of the ESP-12E board.

Program:

```
void setup() {  
    // initialize digital pin LED_BUILTIN as an output.  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is  
    //the voltage level)  
    delay(1000);                        // wait for a second  
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making  
    //the voltage LOW  
    delay(1000);                        // wait for a second  
}
```

Figure 4: Blink Test Program

The first things that we needed to get used to were these 2 pre-built function `setup` and `loop`. These 2 functions are kind of easy to understand.

The `setup` function is made, obviously, to set up everything we will need during our program. In our blink test example, we just need to set the LED_BUILTIN as an output. More precisely, we set the associated pin of the LED_BUILTIN as an output. This is easily made with the line below:

```
pinMode(LED_BUILTIN, OUTPUT);
```

Figure 5: Set the built-in led pin as an output

Now that we set up everything we needed; we can enter the `loop` function. This function is just an infinite pre-built loop. For example, in a C, C++ or other programming languages, we could have replaced this function with a `while(1){...}` loop.

The `digitalWrite` function is used to set the voltage to a given pin. The value will be HIGH or LOW which means, in our case, respectively 3,3V or 0V.

Shape of the function:

```
digitalWrite( pin , value )
```

Figure 6: Form of the digitalWrite function

In our example it means that the pin associated to the built-in led will be set to 3,3V. Which means that the LED will be turned on.

```
digitalWrite(LED_BUILTIN, HIGH);
```

Figure 7: Setting the digitalWrite function to turned on the LED

The `delay` function is the easier one. It simply waits the given time in millisecond.

```
delay(time)
```

```
delay(1000)
```

Figure 8: Delay function

So, here, the program will stop for 1000ms (=1 second).

Next lines are similar, we turn off the light, wait 1 second and start again the `loop` function loop.

```
digitalWrite(LED_BUILTIN, LOW);  
delay(1000);
```

Figure 9: Setting the digitalWrite function to turned on the LED for 1s

So, because the lines above are in the `loop` function, the LED will be turned on for 1 second, then turned off for 1 second and so on.

2.2 Wiring

One of the most important part of this project is the wiring, a bad wiring will result in a failure whatever happen. This is why we should be sure about our wiring before trying to launch any program on our board.

We did not have any knowledge about wiring but we had only one module and our board which make the task way easier. Moreover, we chose to use the I2C interface.

As presented before, our card is a BMP280 and contains 6 pins shown below.

PIN	EXPLANATION	WIRED TO
SDO	Serial D ata O ut / Master In Slave Out pin. Sending data from BMP280 to processor	3,3V
CSB	Chip select	3,3V
SDA	Serial Data Signal	D1
SCL	System control layer	D2
VCC	Power pin - Higher voltage	3,3V
GND	Ground. Common for power and logic.	GND

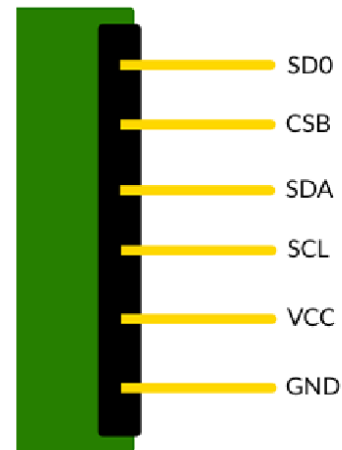


Figure 10: Explanation of the 6 pins of the BMP280

Figure 11: BMP280 pins scheme

For the wiring we have used a breadboard for many reasons. The main reason is that our wiring is not permanent, so the breadboard allows us to wire everything without welding anything. Then, the breadboard is also easy to use and allows us multiple attempts.

Here is how we wired everything on the breadboard. The VCC, CSB and SD0 pins of the BMP280 are wired in serial to the 3,3V. While the GND pin is wired to the GND pin of our ESP-12E board. The SCL and SDA pins are simply wired, respectively to D1 and D2 pin of our microcontroller.

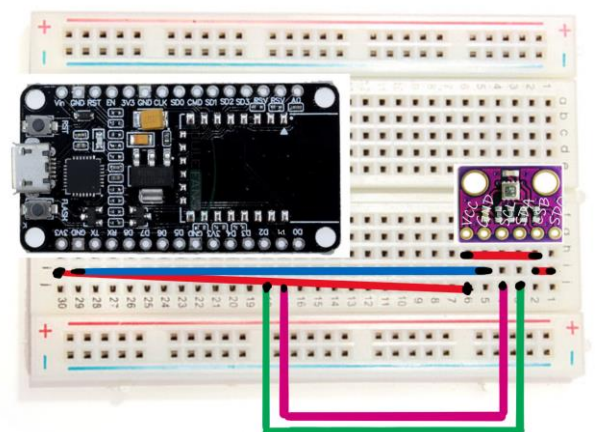


Figure 12: Wiring on the breadboard

2.3 Detailed Code Explanation

The schema sketch by including following libraries:

- **ESP8266WiFi.h** library provides ESP8266 specific WiFi routines we are calling to connect to network.

The actual connection to Wi-Fi is initialized by calling:

```
WiFi.begin(ssid, password);
```

Figure 13: Code line to connect to our WiFi router

The connection process can take couple of seconds and we are checking for whether this has completed in the following loop:

```
while (WiFi.status() != WL_CONNECTED) {  
    delay(500);  
    Serial.print(".");  
}
```

Figure 14: Code lines to wait for connection to our WiFi router

The `while()` loop will keep looping as long as `WiFi.status()` is other than `WL_CONNECTED`. The loop will exit only if the status changes to `WL_CONNECTED`.

The last line will then print out the IP address assigned to the ESP module by [DHCP](#)¹

```
Serial.println(WiFi.localIP());
```

Figure 15: Code line to print out the IP address assigned to the ESP module

Note: if connection is established, and then lost for some reason, ESP will automatically reconnect to the last used access point once it is again back on-line. This will be done automatically by Wi-Fi library, without any user intervention.

¹ DHCP - Dynamic Host Configuration Protocol: The Networking Protocol that gives you an IP address.

- **Wire.h** library communicates with any Inter Integrated Circuit (I2C) device not just BMP280.
- **SPI.h** library communicates with Serial Peripheral Interface (SPI) devices, with the Arduino as the master device.
- **Adafruit_BMP280.h** library is a hardware-specific library which handles lower-level functions.
- **PubSubClient.h** library allows to send and receive MQTT² messages. It supports all Arduino Ethernet Client compatible hardware.

```
#include <ESP8266WiFi.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>
#include <PubSubClient.h>
```

Figure 16: Code lines to include libraries

Next, we create an object of the sensor.

```
Adafruit_BMP280 bmp; // I2C
```

Figure 17: Code lines to create an object of the sensor BMP280

We define PIN-Wiring for I2C connection:

```
#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS   (10)
```

Figure 18: Code lines to define PIN-Wiring

We define the values for the Wi-Fi network, SSID, and password, along with the MQTT served used. We use a local one.

```
const char *ssid = "mqtt"; //WIFI ssid
const char *password = "password"; //WIFI password
const char *mqtt_server = "192.168.137.118"; // MQTT server
```

Figure 19: Code lines to define Wi-Fi and MQTT's values

² MQTT is a lightweight messaging protocol ideal for small devices.

After that, we declare an object of class **WiFiClient**, which allows to establish a connection to a specific internet IP address and port. We also declare an object of class **PubSubClient**, which receives as input of the constructor the previously defined WiFiClient to create a partially initialised client instance.

```
WiFiClient espClient;  
PubSubClient client(espClient);
```

Figure 20: Code lines to create a client instance

Now, moving for the setup function, we open a serial connection, so we can output the result of our program. We also connect to the WiFi network.

```
void setup() {  
  Serial.begin(115200);  
  connectWifi();  
}
```

Figure 21: Code lines to connect to the WiFi network

Next, we need to specify the address and the port of the MQTT server. To do so, we call the `setServer` method on the `PubSubClient` object, passing as first argument the address and as second the port.

`mqtt_server` variable was defined before, in constant string.

```
client.setServer(mqtt_server, 1883);
```

Figure 22: Code line to specify the address and the port of the MQTT server

In this same setup function, we initialize the sensor with `begin()` which will return `True` if the sensor was found, and `False` if not. When we get a `False` value back, we check our wiring.

```
if (!bmp.begin()) {  
  Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));  
  while (1);  
}
```

Figure 23: Code line to test if the sensor is found

Next, if the connection with the broker is lost, we want to try to reconnect but we want to make this more robust. So, we verify the Wi-Fi connection and try to reconnect to the Wi-Fi router and then to the MQTT broker.

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

Figure 24: Code line to verify the Wi-Fi connection and try to reconnect

In this loop, after checking the connection, we read and display the sensor values.

```
    snprintf (msg, MSG_BUFFER_SIZE, "%d,%f", i, bmp.readTemperature());
    Serial.print("Publish message temp: ");
    Serial.println(msg);
    client.publish("room/temperature", msg);
    snprintf (msg, MSG_BUFFER_SIZE, "%d,%f", i, bmp.readPressure());
    Serial.print("Publish message pression: ");
    Serial.println(msg);
    client.publish("room/pressure", msg);
    snprintf (msg, MSG_BUFFER_SIZE, "%d,%f", i, bmp.readAltitude(1013.25));
    Serial.print("Publish message altitude: ");
    Serial.println(msg);
    client.publish("room/altitude", msg);
    delay(1000);
    i++;
}
```

Figure 25: Code lines to read and display the sensor values

After reading the sensor values, we want to connect WiFi with a loop function.

- **WiFi.mode(WIFI_OFF)** prevents reconnection issue (taking too long to connect).
- **WiFi.mode(WIFI_STA)** this line hides the viewing of ESP as WiFi hotspot.
- **WiFi.begin(ssid, password)** connect to our WiFi router

```
void connectWifi(){
  delay(1000);
  WiFi.mode(WIFI_OFF);
  delay(1000);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");
  Serial.print("Connecting");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
}
```

Figure 26: Code lines to connect WiFi

If connection is successful, we want to print a status message in serial monitor declaring which network we are connecting to. Then, we want to print a confirmation message to show we are connected and to print the connected IP address.

```
Serial.print("Connected to ");  
Serial.println(ssid);  
Serial.print("IP address: ");  
Serial.println(WiFi.localIP()); //IP address assigned to our ESP  
}
```

Figure 27: Code line to print a status message in serial monitor about WiFi connection

Finally, we define a function **reconnect()** to keep trying to connect to the MQTT broker until it is successful. Here the client id can be random because the MQTT broker we use does not require authentication.

```
void reconnect() {  
    // Loop until we're reconnected  
    while (!client.connected()) {  
        Serial.print("Attempting MQTT connection...");  
        // Create a random client ID  
        String clientId = "ESP8266Client-";  
        clientId += String(random(0xffff), HEX);  
        // Attempt to connect  
        if (client.connect(clientId.c_str())) {  
            Serial.println("connected");  
            // Once connected, publish an announcement...  
        } else {  
            Serial.print("failed, rc=");  
            Serial.print(client.state());  
            Serial.println(" try again in 5 seconds");  
            // Wait 5 seconds before retrying  
            delay(5000);  
        }  
    }  
}
```

Figure 28: Code lines to define a function reconnect() to the MQTT broker

3 Results

The results are going to be sent to the MQTT server in 3 different topics:

- room/temperature
- room/pressure
- room/altitude

Results will be sent in this shape:

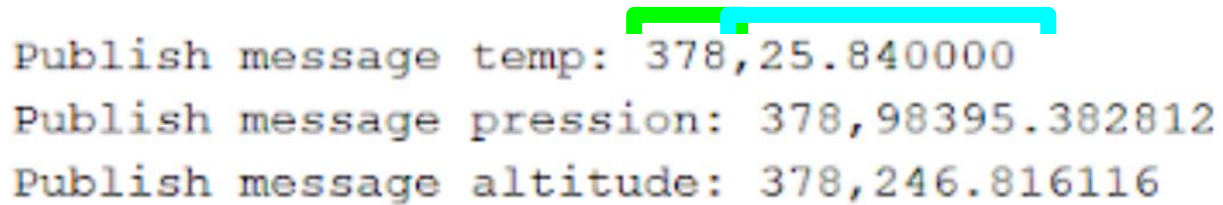


The diagram shows a rectangular box with a black border. Inside the box, the letter 'i' is highlighted in green, followed by a comma and the word 'value' which is highlighted in cyan.

Figure 29: Scheme of the shape of the results

i starts from 0 and will be incremented by 1 with each set of new sent value. This *i* simplify the work of the MQTT server for reading values and display them in a shape of a graphic.

value simply return the float value “read” by the BMP280 module.



The graphic displays three lines of text in a monospaced font. The first line is 'Publish message temp: 378,25.840000', the second is 'Publish message pression: 378,98395.382812', and the third is 'Publish message altitude: 378,246.816116'. A green and cyan bar is positioned above the text.

Figure 30: Graphic of the shape of the results

Below are shown the results in the serial monitor provided by the Arduino software.

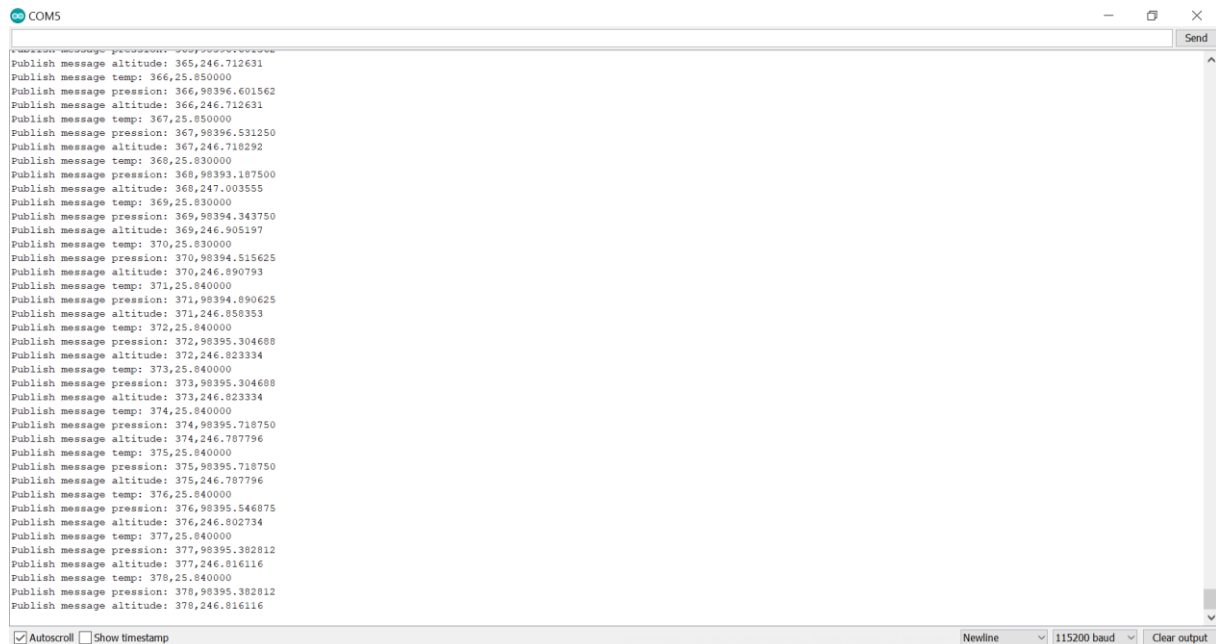


Figure 31: Results in the serial monitor

Bibliography

CONSULTED WEBSITES

- Blogs:

<https://lastminuteengineers.com>

<https://techtutorialsx.com/2017/04/09/esp8266-connecting-to-mqtt-broker/>

- Official websites:

<https://www.arduino.cc/>

<https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/pressure-sensors-bmp280-1.html>

- PDF files:

<https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf>

<https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bmp280-barometric-pressure-plus-temperature-sensor-breakout.pdf>

- Dictionary:

<https://www.wikipedia.org/>

Conclusion

This subject allowed us to acquire a lot of knowledge, especially in electronics and in programming with Arduino, both on a practical and theoretical level.

It was very interesting to assemble by ourselves a data transmission that we are used to obtain by means of assembly already done in altimeter, thermometers or in barometers.

This project also helped us to be autonomous to set up electronic connections with simple internet access as a support.

We are satisfied with ourselves and the achievement of our project which we defined as "Send temperature, pressure and altitude values to a MQTT server with a microcontroller and a sensor."

Annex

ⁱ **I²C** (Inter-Integrated Circuit), is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

ⁱⁱ **SPI** is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.