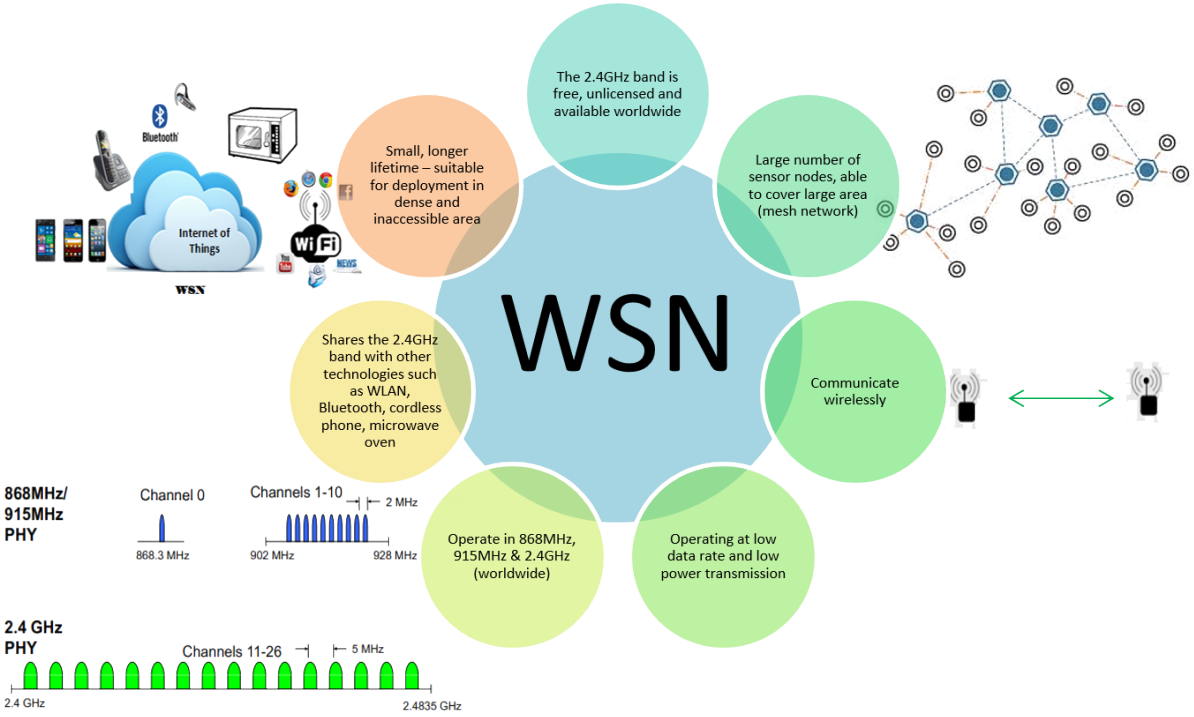
Wireless Sensor Networks



Cyril TSO
Jérémy BARGE
Elias OMRI

Table of Contents

What is a Wireless Sensor Network (WSN) ?

Wireless sensor network (WSN) refers to a group of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location. WSNs measure environmental conditions like temperature, sound, pollution levels, humidity, wind, and so on.

The WSN is built of "nodes" – from a few to several hundreds or even thousands, where each node is connected to one (or sometimes several) sensors. Each such sensor network node has typically several parts: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting. A sensor node might vary in size from that of a shoebox down to the size of a grain of dust, although functioning "motes" of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from a few to hundreds of dollars, depending on the complexity of the individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communications bandwidth. The topology of the WSNs can vary from a simple star network to an advanced multi-hop wireless mesh network. The propagation technique between the hops of the network can be routing or flooding.

Definition from Wikipedia

Introduction

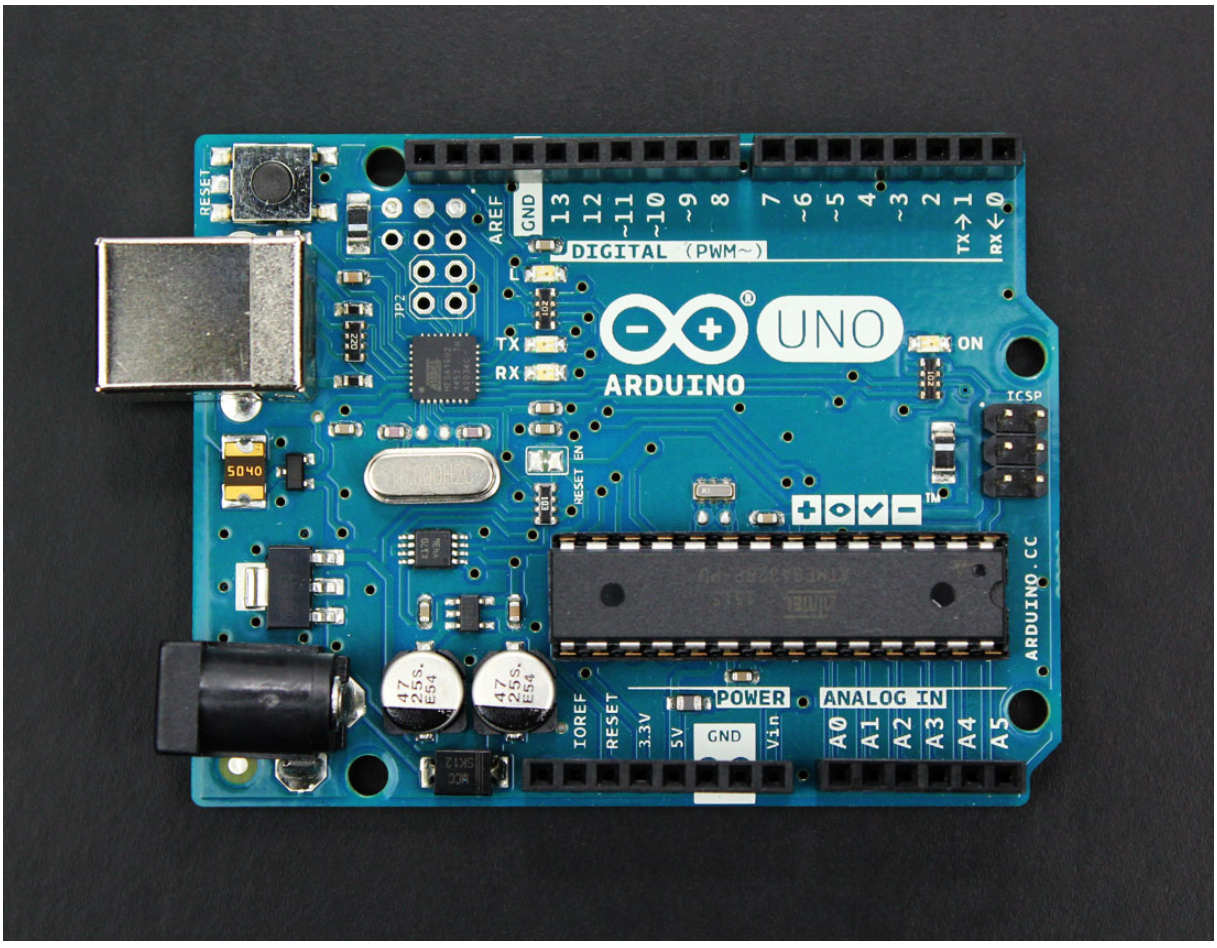
The objective of this project is to build a Wireless Sensor Network, mainly with temperature sensors, which will make us able to observe the temperature and humidity of a room in real-time or from previous days (from days to months). Those measurements can help us to look over the evolution of the temperature and humidity by days and consequently can lead us to predict (just a little) how the temperature is going to evolve in the next days.

For doing this, we had to build a system where the sensors nodes get the measurements with the brain processing them. The data collected from the nodes will be displayed in real-time in form of numbers, diagrams, graphs, dashboards, etc. but they will be also stored locally and, in a cloud, so that we can use these data later for deeper analysis.

Hardware

For this project, we are going to use many hardware to make the sensors nodes, the brain and to connect the nodes and the brain.

- **Arduino UNO**



Arduino is an open source programmable circuit board that can be integrated into a wide variety of makerspace projects both simple and complex. This board contains a microcontroller which is able to be programmed to sense and control objects in the physical world.

For the project, we are going to use the Arduino UNO as the nodes as it's one of the easiest to use and has great features.

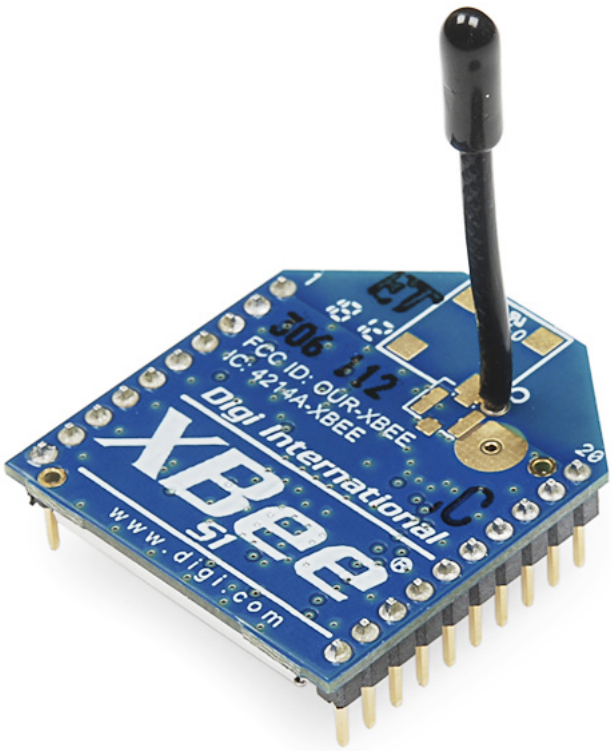
- **Raspberry PI**

The Raspberry Pi is a mini-computer which include a processor, a graphics chip, some RAM, a few USB ports, an HDMI output, an Ethernet port, and (in the latest version) integrated Wi-Fi and Bluetooth.

On the Raspberry Pi we have installed Raspbian as an operating system, equivalent to Linux.

We are going to use the Raspberry Pi as the brain who will processes all the data collected by the Arduino UNO.

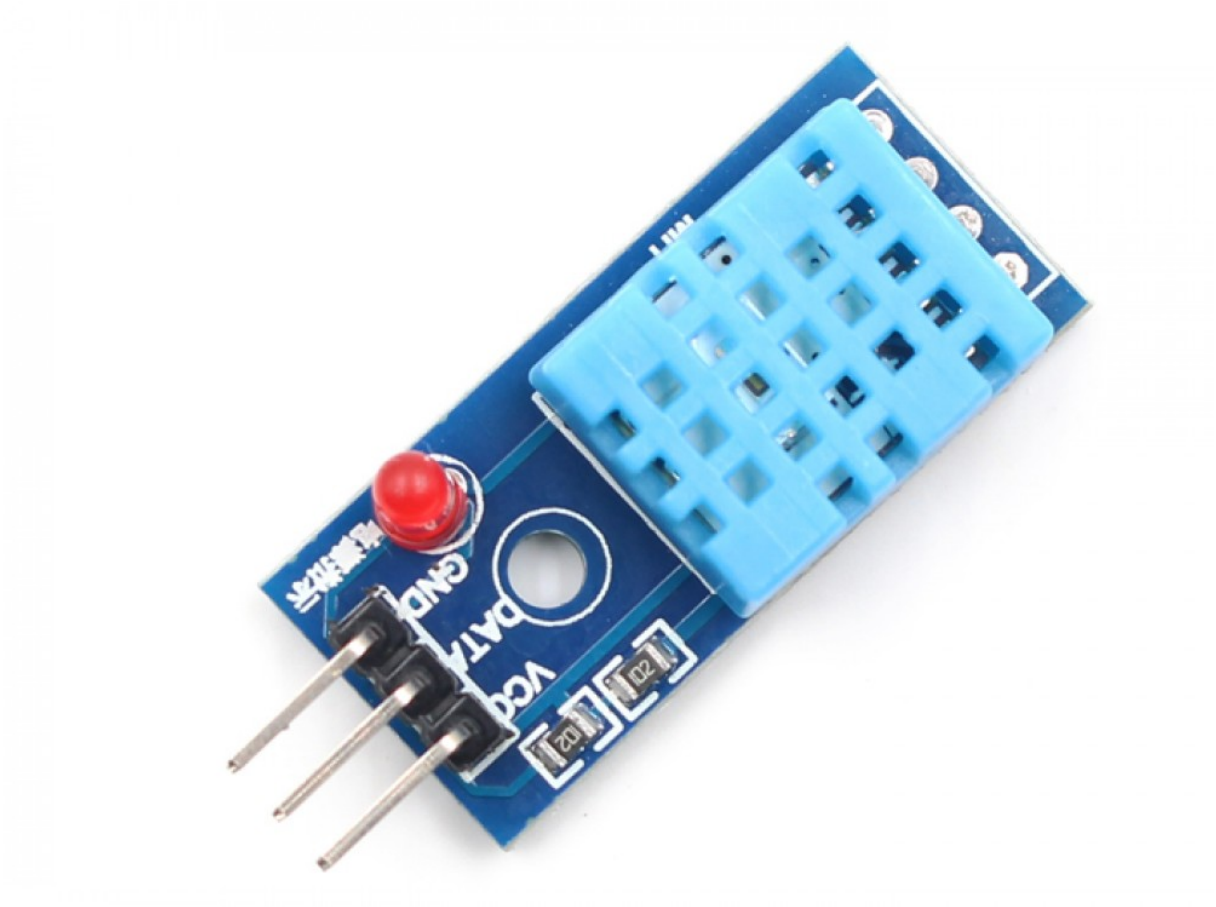
- **XBee**



Xbees are wireless transceivers which allow to connect devices. As a matter of fact, they send and receive data over a serial port, consequently, they are compatible with both computers (like Raspberry Pi) and microcontrollers (like Arduino).

Thus, we will use the module Xbee to link up the Arduino UNO and the Raspberry Pi so that it will enable the transfer of datas between those two devices.

- **Temperature and Humidity Sensor: DHT 11**



The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.

Softwares

- **Arduino IDE**

We will use the Arduino IDE to build and execute instructions on the Arduino UNO. This software will allow us to gather the data from the sensors attached on the Arduino UNO.

- **XCTU**

We will use this software to build the link between the Xbees connected to the Arduino UNO and the Raspberry Pi.

Moreover, to test if both Xbees are connected, we will send a message from XCTU to the Arduino and if the Xbees are connected, one led on the Arduino will blink, confirming that the Arduino received the message from XCTU.

- **Python**

We will use Python language to send the data from the Raspberry Pi to the cloud located on the platform Ubidots to store the data that we want to observe in real-time or for later use. But also to save locally the data collected too.

- **LibreOffice**

We will use LibreOffice to open the CSV files that we will generate from the Python script to store the data locally in CSV form on the Raspberry Pi. LibreOffice will allow us to display all the values from the sensor in a table so that we can manage them as we want, for example to create diagrams/graphs etc.

Data Management

1. Data Acquisition

To acquire the data from the sensors, attached to an Arduino UNO, we can use several methods :

a) We can use a Python code to display the values on the terminal then save them into a file

b) We can use Coolterm, a software which allows to read the data stream coming from a serial port (here, our Xbee is plugged through an USB Explorer so it's considered as a serial port), then save the data direct with the software

2. Data Communication

After we have set up the two Xbees, put the code on the Arduino (for collecting the sensors' data). We're going to write a Python code which will allow us to transfer and log the data on the Rapsberry pi in a .txt or .csv file.

Then, to transfer the data to the Cloud (Ubidots), we will use another Python code which involves the set-up of the cloud's token and api keys. With this code we will also be able to choose which data we want to transfer. It's also possible to convert the data to a .csv file directly from the cloud.

3. Data Storage

a. Which data to store locally?

As the memory of a Raspberry Pi is not very high, we will have to select only the useful data to store locally.

Thus, we will mostly use the local storage to store only the useful data, which means only the most recent data (from 0 to 2/3 days max).
The local storage will also be used as a back-up storage of the most meaningful data.

To store the data locally, we will use a Python script directly on the Python IDLE on the Raspberry Pi. This program will allow us to choose to create or modify a .txt/.csv in the logs of data from the Arduino inside.

Consequently, we will be able to observe all the data save locally in the form of graphs, diagrams etc. thanks to LibreOffice who can build such models.

b. Which data to store on the cloud?

About the cloud storage, we will use the platform Ubidots, as it's free for the use of one device (we will only use the Raspberry Pi as a device) to store the data that we want (two categories here : temperature and humidity).

Moreover, this platform will allow us to build more elaborate models such as real-time graphs, dashboards, tables of data.

Obviously, all the data saved on it are detailed in function of time, date etc. Thus, we can at the same time save recent and older data (the ones who are less useful) for later use as the memory of the platform is way bigger than the Raspberry Pi one. That's why we will mostly use the local storage as a "back-up" storage and the cloud storage as the main storage.

4. Data Processing

a. Refreshing the data

We have planned to refresh the data from sensors every 10 minutes because it will allow us to obtain enough values quickly to observe how the temperature is evolving.

We chose to not set up the refresh on seconds or hours because we will get too much values (useless because the temperature is unlikely to vary that much over seconds) or we will wait too much to get enough values to establish a meaningful analysis of the evolution/variation of the temperature.

In fact, unless there are a huge concentration of people inside the same room or the AC is turned on/off, the temperature and the humidity of the room is unlikely to vary a lot or change every second, minute, hour.

So, to set up the refresh every 10 minutes seems to be a good compromise.

b. Which data to remove?

We have chosen to remove the data which are older than 2-3 years as they will no more be useful to observe, analyze or use.

For doing this, we can use different ways.

Locally, we can simply suppress manually all the old CSV files by ourselves with using commands on Linux (rm *.csv). It's also possible to directly update an outdated data file by just re-writing the new data over it.

On the cloud, it's also possible to suppress the data that we don't need by removing it directly on the Ubidots platform as its interface is very versatile and easy to use.

5. Data Analysis

To analyze the data, we can use Python libraries (Scipy, Scikit-Learn, Pandas ..) to make some statistical analysis (mean, 1st and 3rd streets, min, max values) .

We can also use some Machine Learning algorithms to make prediction about the evolution of the temperature few days after the day of the analysis with for example a Linear Regression analysis.

6. Goals of using data

The data collected will be used to do plenty stuffs.

For short term use (less than 1 month):

a) As a matter of fact, we can use those data to observe the actual temperature and humidity in the room so that we can adjust it by turning on/off the AC, opening/closing the window or the door.

b) Other utility is to use the data collected to know if the room have been used a lot during the day or not by observing the evolution of the temperature inside the room during the day (by setting up the refresh to 4 hours instead of 10 minutes).
Indeed, a human body emits heat, so if there a lot people inside a room, this room will naturally heat up so by observing the data in this way it's possible to make this conclusion.

c) Another utility of those data is to use them to try to make small prediction (not precise but still useful!) about how the weather is gonna be in the next days by observing the evolution of the temperature on few days.
For example, we take a sample of temperature data on 7 days (the values are: 22-24-25-27-24-20-18), we can slightly deduce that the temperature is gonna be around 20 and 30 degrees in next days or is likely to fall around decade degrees.

For long-term use (from months to years):

a) For some chemical experience, we have to control and adjust the temperature of the content (liquid, gas) to observe the behavior and evolution of the whole experience (which can go from days to months and even years!).

A simple example is the analyze of the three states of the water, we know that below 0 degree the water becomes solid (ice form, process called solidification); above 100 degrees the water becomes gas (vapor form, process called Evaporation); between 0 and 100 degrees the water stays liquid (Ice to Water is called Fusion, Gas to Water is called Condensation).

So, the use of temperature sensor can allow us to do more advanced research and analyze of those different states of water. In fact, sometimes we will want to analyze the water when it's completely froze or vapored or return to the liquid form, so we will have to wait quite a lot of times before it reached this state, and the analyze from the sensors' data will allow to us to observe how the temperature has fluctuated following this water transformation.

b) Another example is using the temperature sensor for the solar applications.

It's possible to install those sensors on solar panel to control if the temperature is too high for the panels to handle (indeed, extremely high temperature can damage the solar panel and overheat the components inside which can be hazardous for the surroundings).

The data collected by the sensors dated by months/years to see how much the panels can handle (the damages will not be displayed directly, so we will have to extend the analysis on months/years).

c) In the pharmacy domain, controlling the temperature is very important too.

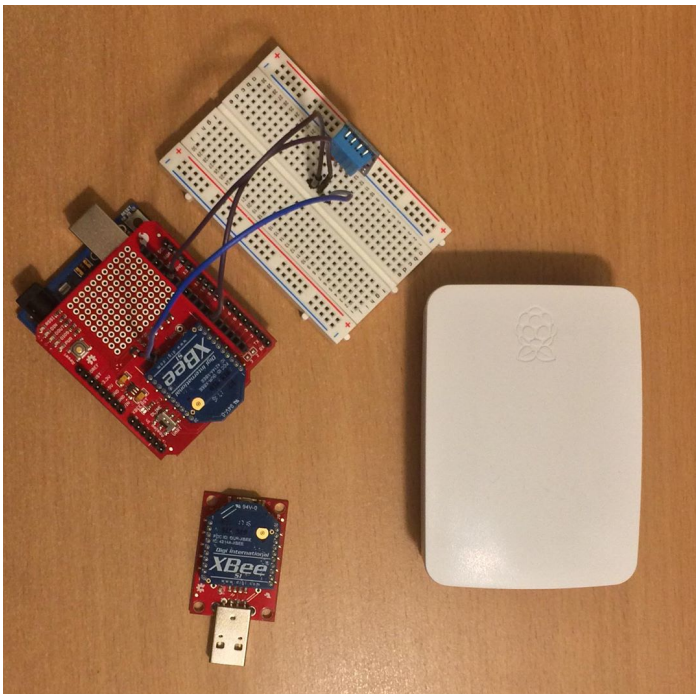
As a matter of fact, the fabrication of active substances (the chemical substances which contain therapeutic effect) but also

the tools needed in this domain like incubator, mixers, ovens need a strict requirement in terms of traceability and security.

How will we make this work?

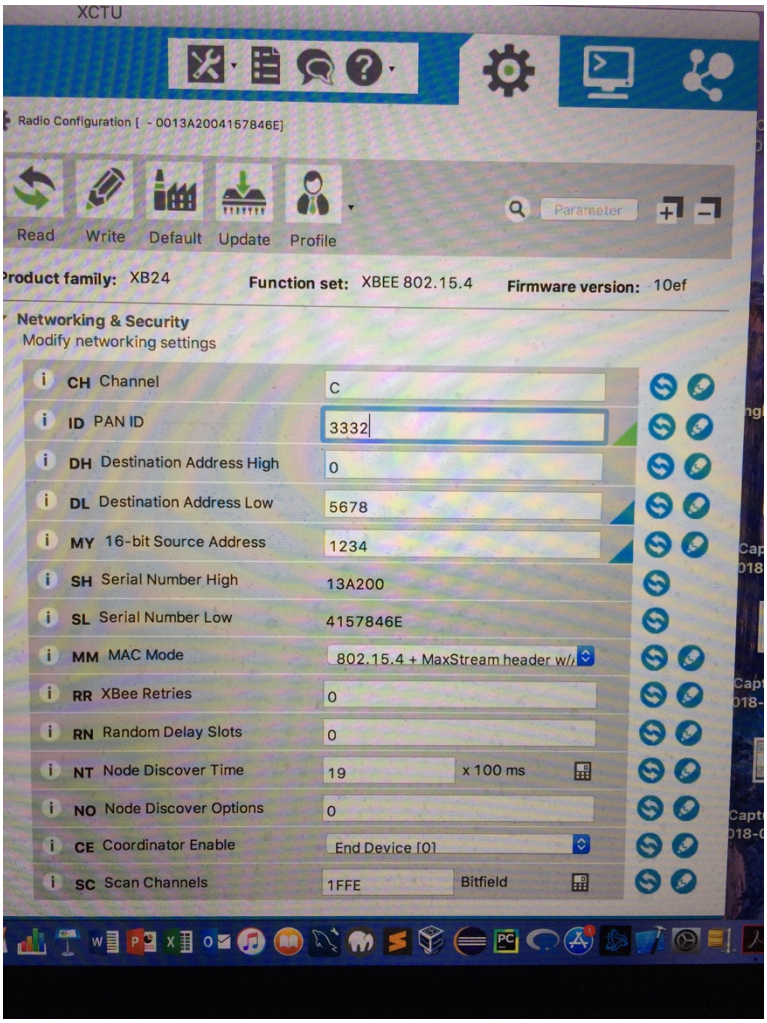
To make the Wireless Sensor Network up, we will:

1) Plug the sensor on the breadboard and then connect it to the Arduino's pins (a DHT11 got 3 pins: Vcc, Data and GND, here we will plug Vcc on +5V, Data on digital pin 7 and GND on gnd).



Components that we are going to use

2) We will set up both xbee to communicate to each other



As seen in this screenshot, we have to set both of the Xbee configuration like this way:

- Channel: C
- Pan ID: 3332
- Destination Address High: 0
- Destination Address Low: 5678 (Xbee 1), 1234 (Xbee 2)
- 16-bit Source Address: 1234 (Xbee 1), 5678 (Xbee 2)

3) Then we will upload two Arduinos' codes in the Arduino Uno

a) The Temperature Code

```
#include <DHT.h>
#include <SoftwareSerial.h>

//Constants
#define DHTPIN 7     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE); //// Initialize DHT sensor for normal 16mhz Arduino
SoftwareSerial XBee(2, 3);

int chk;
float temp; //Stores temperature value

void setup()
{
  XBee.begin(9600);
  Serial.begin(9600);
  dht.begin();
}

void loop()
{
    delay(2000);
    //Read data and store it to variables temp
    temp= dht.readTemperature();

    //Print temp values to serial monitor
    Serial.println(temp);
    delay(2000); //Delay 2 sec.

    XBee.println(temp);
    delay(2000); //Delay 2 sec.
}
```

b) The Humidity Code

```
//Libraries
#include <DHT.h>
#include <SoftwareSerial.h>

//Constants
#define DHTPIN 7     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE); //// Initialize DHT sensor for normal 16mhz Arduino
SoftwareSerial XBee(2, 3);

int chk;
float hum; //Stores humidity value

void setup()
{
  XBee.begin(9600); //setting the baud rate
  Serial.begin(9600);
  dht.begin();
}

void loop()
{
    delay(2000);
    //Read data and store it to variables hum
    hum = dht.readHumidity();

    //Print temp and humidity values to serial monitor
    Serial.println(hum);
    delay(2000); //Delay 2 sec.

    XBee.println(hum);
    delay(2000); //Delay 2 sec.
}
```

4) One those 3 steps are done, normally the Arduino and the Raspberry Pi is able to communicate wirelessly, so now we want to store the data locally.

```
#Program that allows to save locally the humidity data, here in a csv file

import serial #library that allows to read the data stream from the arduino

serial_port = '/dev/cu.usbmodem1421'; #name of the serial port (its name vary depending on the location of the usb port)
baud_rate = 9600; #we chose to set the baud rate here to 9600
write_to_file_path = "Humidity logs.csv"; #the name and the type of file in which we want to write the data in
output_file = open(write_to_file_path, "w+"); #writing the data
print("Humidity (in %) :")
ser = serial.Serial(serial_port, baud_rate) #reading the data stream
while True: #to allow a permanent stream and not a limited stream
    line = ser.readline(); #reading the data line by line
    line = line.decode("utf-8") #ser.readline returns a binary, convert to string
    print(line); #print the data
    output_file.write(line); #write the data coming in the file
```

Local storage code for the humidity in Python

```
#Program that allows to save locally the temperature data, here in a csv file
#It functions in the same way of humidity local save, so the comments I have to made are the same

import serial

serial_port = '/dev/cu.usbmodem1421';
baud_rate = 9600;
write_to_file_path = "Temperature save logs.csv";
output_file = open(write_to_file_path, "w+");
print("Temperature (in C°) :")
ser = serial.Serial(serial_port, baud_rate)
while True:
    line = ser.readline();
    line = line.decode("utf-8")
    print(line);
    output_file.write(line);
```

Local storage code for the temperature in Python

5) Then finally we want to transfer those data to Ubidots Cloud, so that we can have a detailed following of the logs of both temperature and humidity sensors.

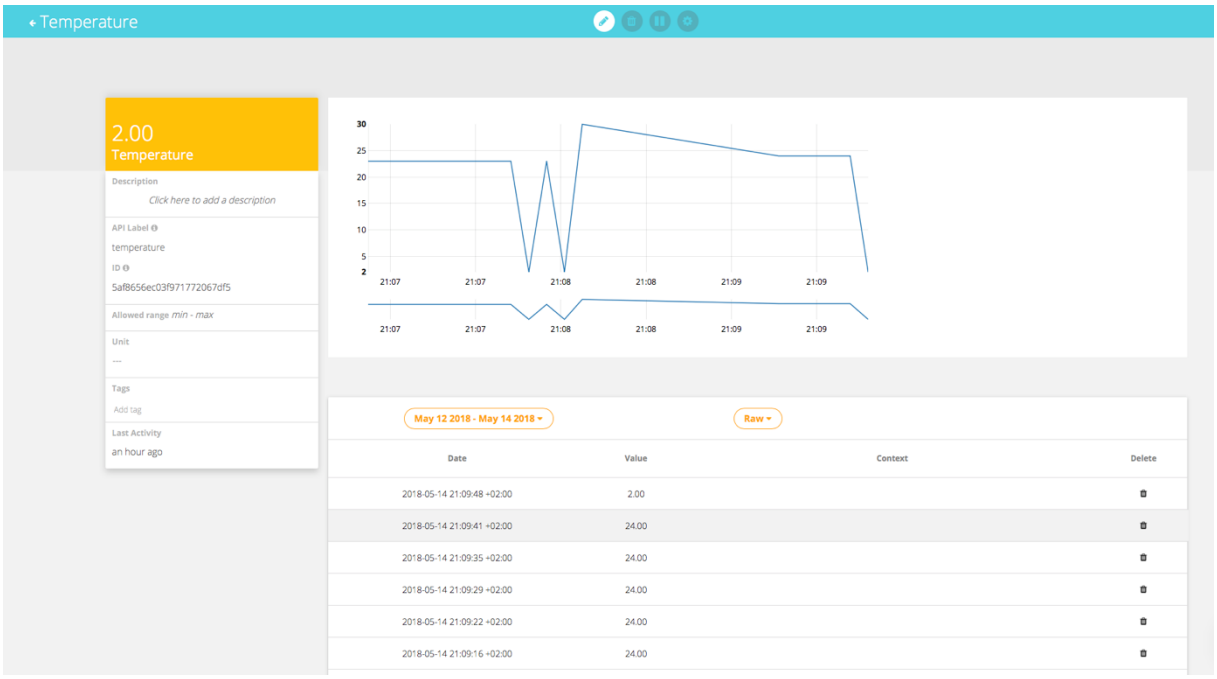
```python
#Program to send the data of the Humidity read to Ubidots Cloud

#Librairies
import serial #allows to read the data coming from the Arduino
from ubidots import ApiClient #allows to transfer the data to Ubidots Cloud

print("Sending Data to the Cloud ...")

api  = ApiClient(token = 'A1E-4XbpDAYDVUDPiYaC3I3F6ukMHIrP3r') #updating the API token

my_hum = api.get_variable('5af86576c03f97171dc84bb4') #updating the Humidity ID

ser = serial.Serial('/dev/ttyUSB3', 9600) #update with port with Arduino

while True:
    read_serial = ser.readline() #read the data income line by line
    humReading = read_serial.decode("utf-8") #convert from binary to string
    new_value2 = my_hum.save_value({'value': humReading}) #save the value to the cloud
    print(read_serial) #prints serial reading to python
```

Humidity transfer code in Python

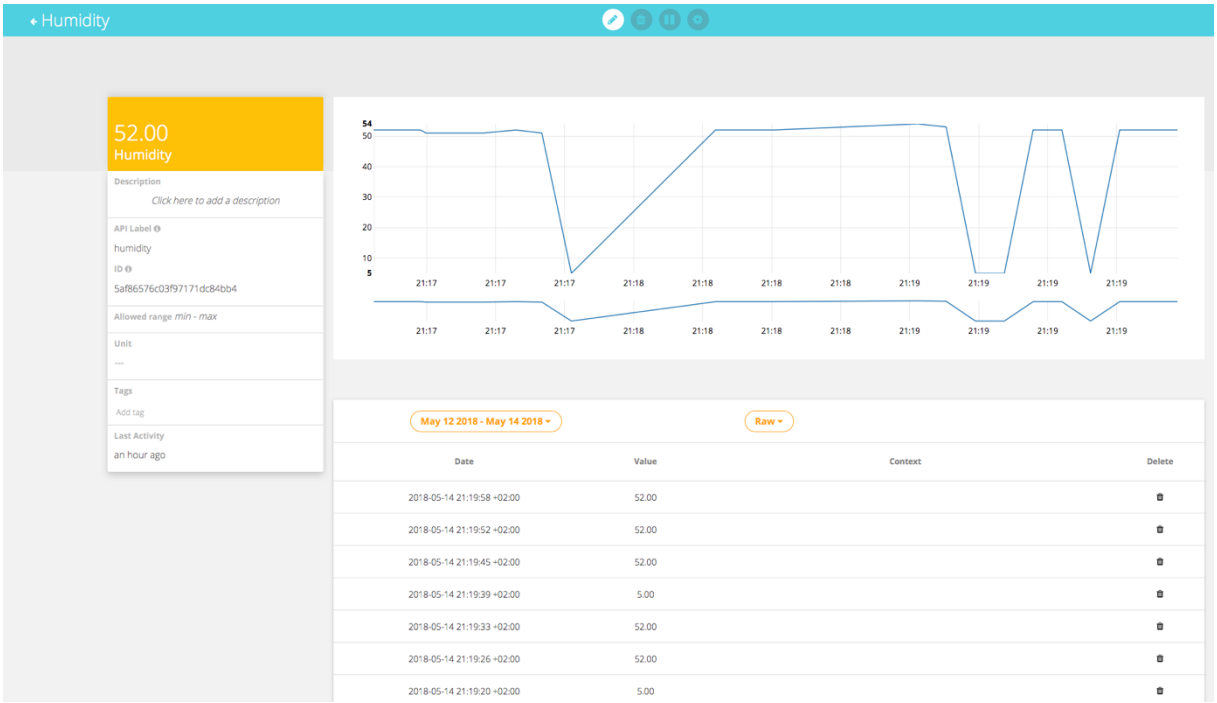
```python
#Program to send the data of the Temperature read to Ubidots Cloud
#Works in the same way than the Humidity one so comments are the same

import serial
from ubidots import ApiClient

print("Sending Data to the Cloud ...")

api  = ApiClient(token = 'A1E-4XbpDAYDVUDPiYaC3I3F6ukMHIrP3r') #update token

my_temp = api.get_variable('5af8656ec03f971772067df5') #update Temperature ID

ser = serial.Serial('/dev/ttyUSB3', 9600) #update with port with Arduino

while True:
    read_serial = ser.readline()
    tempReading = read_serial.decode("utf-8") #read the temperature value
    new_value = my_temp.save_value({'value': tempReading})
    print(read_serial) #prints serial reading to python
```

Temperature transfer code in Python

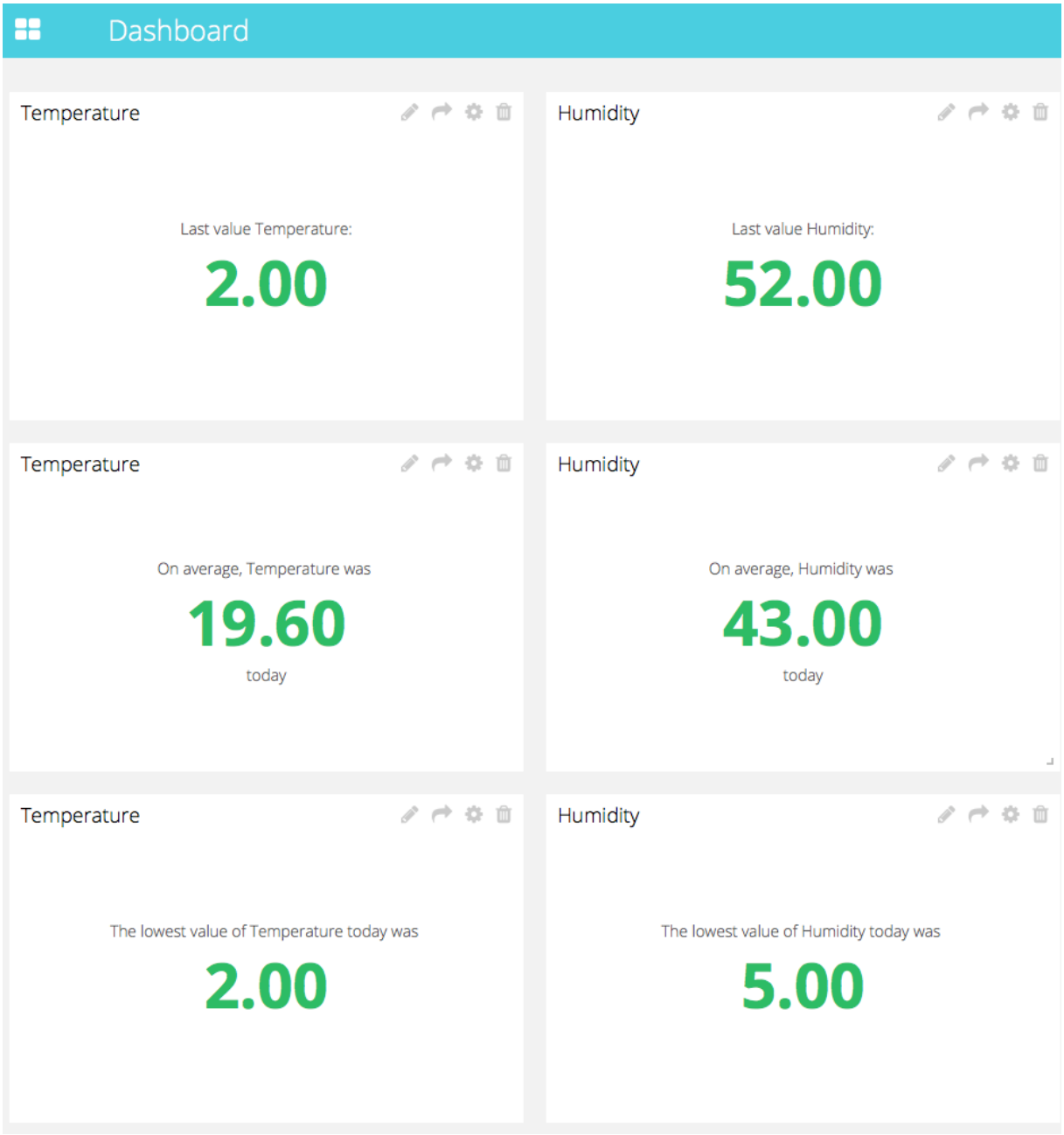
6) Therefore, we got those results on Ubidots Cloud



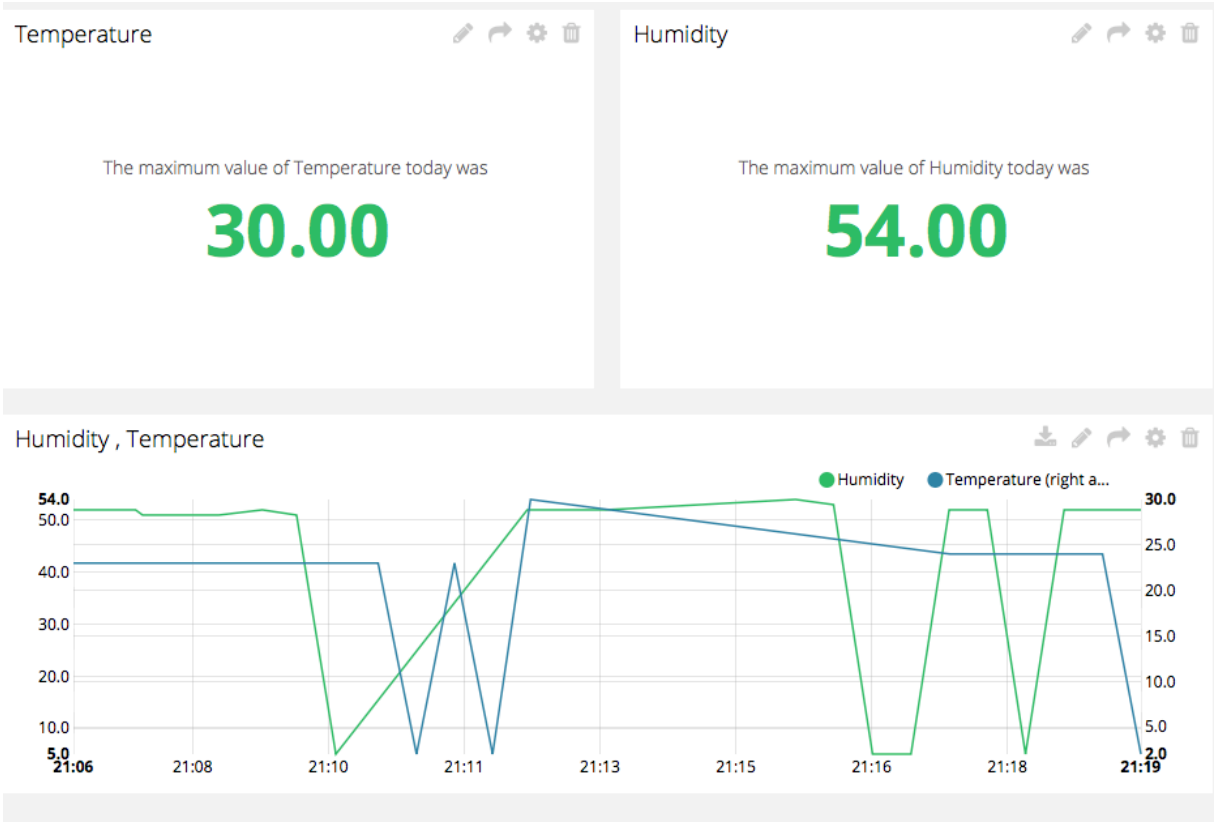
Logs of the temperature data



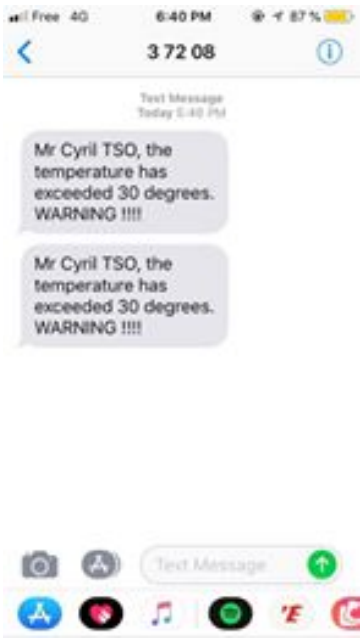
Logs of the humidity data



Ubidots Cloud Dashboard 1



Ubidots Cloud Dashboard 2



Ubidots will send a message if the temperature exceeds 30 degrees

Criticism about the project

Even though the project was very interesting and make me progressed a lot in a variety of fields (telecommunication, embedded systems, programming), some stuffs can be improved.

For example, about the materials, the Xbee shield for the Arduino was not exactly at the same size than the Arduino's size, so we had to incline a little the shield, which sometimes create misreading of the data as the shield is not exactly superposed on the board.

Moreover, the Xbee shield should have been shipped with pins already soldered on it. As a matter of fact, I had lot of troubles soldering every pin one by one on the shield with a lot of connections problems …

Furthermore, we should have been assigned to use rather Windows or Mac OS as an operating system to process the data. As a matter of fact, by using Raspian (Linux), we got troubles about the local storage python code. Because the code works on both Windows and Mac OS but only partially on Linux (sometimes it works and sometimes it doesn't, and I didn't find a solution to make it work at 100% on Linux, it's not about an access or authorization problem on the file neither ….

Conclusion

This project helped me to grow my technical skills in the IoT & Big Data field. As the matter of fact, I feel like that I have improved a lot in the hardware stuffs (Arduino, Sensors..), in programming (by coding stuffs in Python and making predictions with Machine Learning), in telecommunication (Xbee part) but also, the fact that we had to use a Raspberry Pi as a computer makes us improve a lot our skills in Linux OS by using Raspian.

Overall, I found this project quite interesting because it's a very all-round project as it touches a lot of field at the same time, so there were a lot of different tasks to execute and found it very pertinent to do a lot of different things at the same time.

The most important thing that this project makes me realize is that we always have to do deeper research on internet to find what are searching for and that everything can be found on internet. By doing this, you are learning a lot more, very quickly and efficiently rather than if someone has helped and advised you about how to do those stuffs.

Project Progress

From February to March:

- Brainstorming
- Test of Arduino UNO and Raspberry Pi
- Test of Xbee

Week of 09 April:

- Able to collect data from the Ultrasonic Sensor
- Able to store data collected on CSV, Text file (with Python or CoolTerm), possible to build graphs with excel from the CSV's file, we can create a new file for each measure taking by changing the filename on the python code, so that it will automatically create a new file on a classic laptop with usb serial port (not too different from wireless connexion so the Data Transfer from Arduino to Raspberry Pi part shouldn't take too much time).
- Optional: can display the evolution of the values collected in a real-time graph (some dysfunctionnements)
- Update of the report (added "How to send data from Arduino to the Rpi?" part, added long-term goal in "Goals of using data" part, modified the "local data storage" part, with cutecom replaced by python, added screenshots)

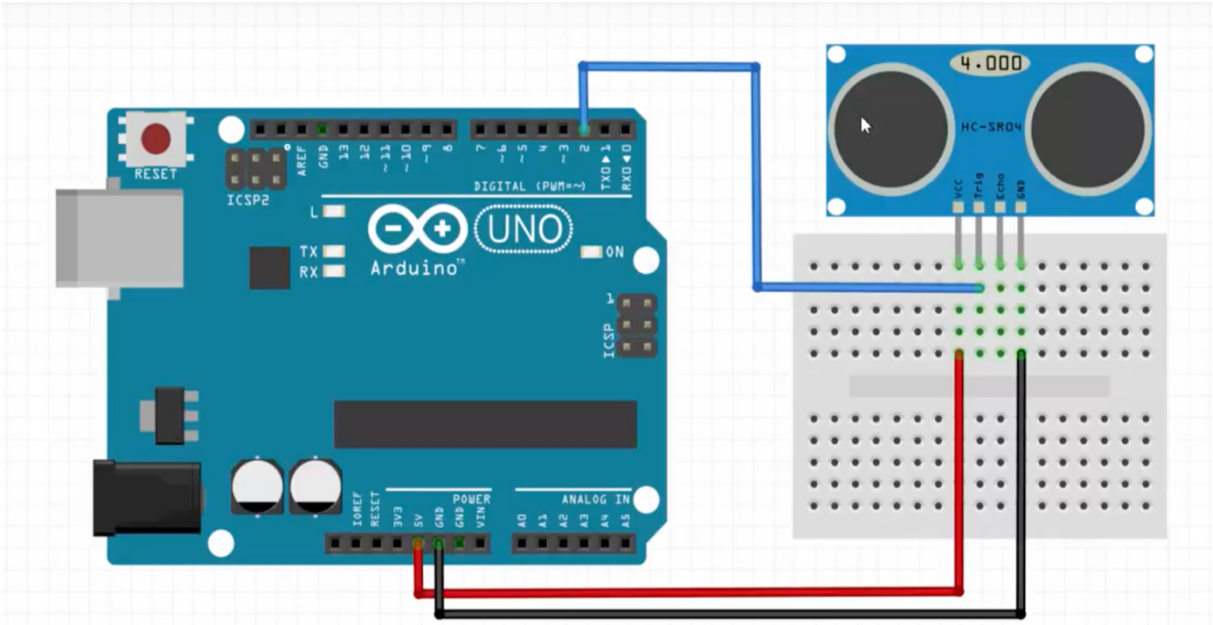
Week of 23 April:

- Able to collect data from the Temperature Sensor

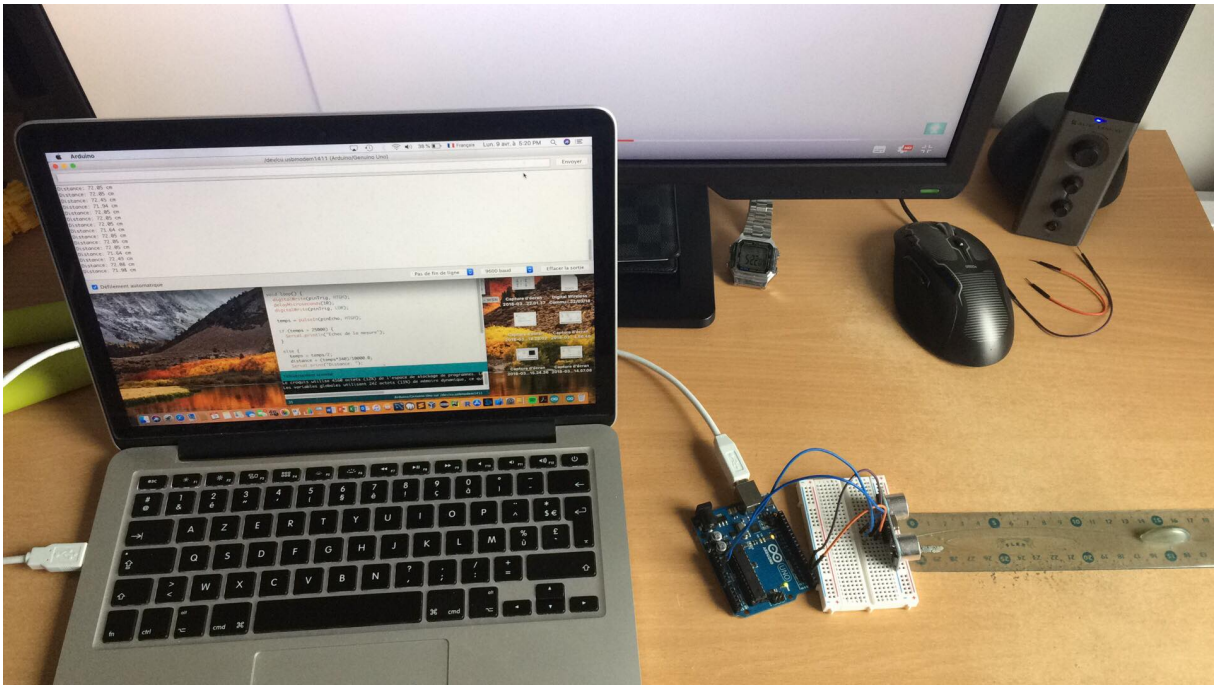
Week of 30 April – 14 May:

- Soldering the Xbee shield's pins to connect it to the Arduino UNO
- Xbee connectivity is set-up
- Can collect and send data from the sensor to the computer/Rpi wirelessly thanks to the Xbees with Python or Coolterm or XCTU, also able to save the data in a CSV, Text file
- Temperature Sensor is set up (can collect and measure temperature and humidity)
- Try to send data from Arduino to Raspberry Pi (same as we did with the combo Python/Arduino, use the same program to call the Arduino from Python IDLE, it will generate the values on the RPi Python Terminal)
- Store Data on the RPi with Python on CSV Format (same than what we did with the previous step with the combo Arduino/Mac/Python, so it will be quick and easy to do)
- Can put Data on Ubidots
- Have to make the prediction of the temperature (/w Machine Learning algorithm)
- Finish the report
- Little incoherence to correct (temperature values, pb Mac OS/Linux), convert temperature values from Farenheit to Celsius
- Change the transfer from Rpi to Arduino for Ubidots

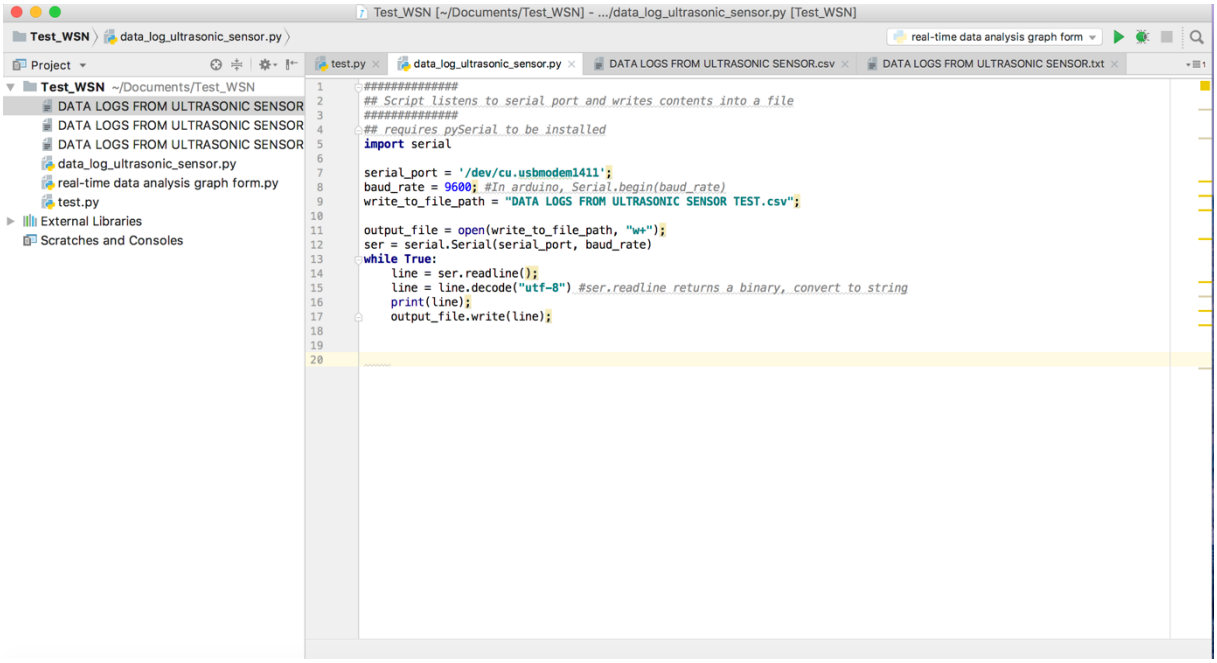
Additional Screenshots (added during ultrasonic sensor testing !)



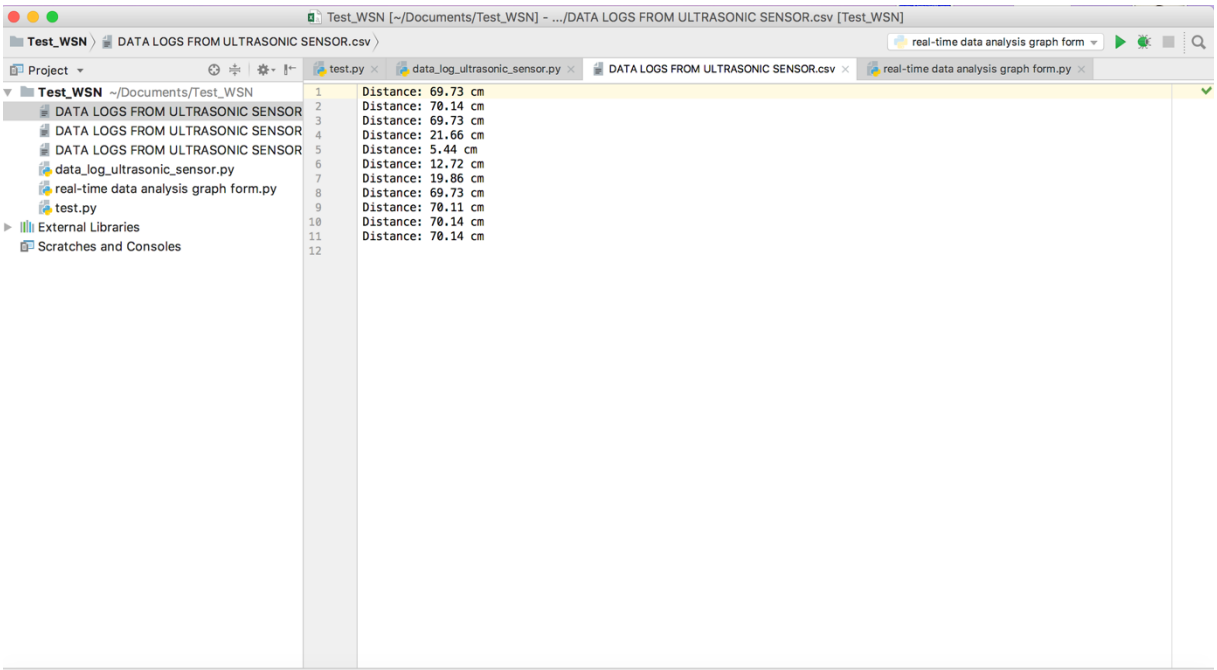
Ultrasonic Sensor - Arduino



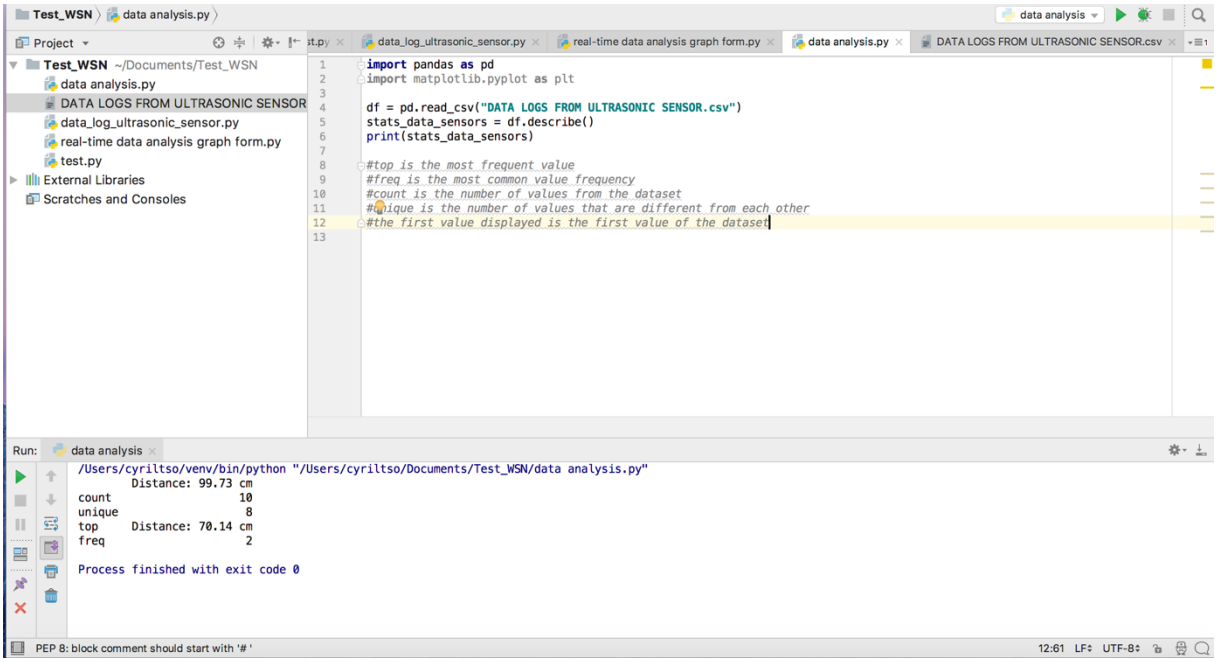
Montage of the Ultrasonic Sensor on Arduino and the results from the sketch on Arduino IDE displayed on the laptop by using a USB serial port connexion



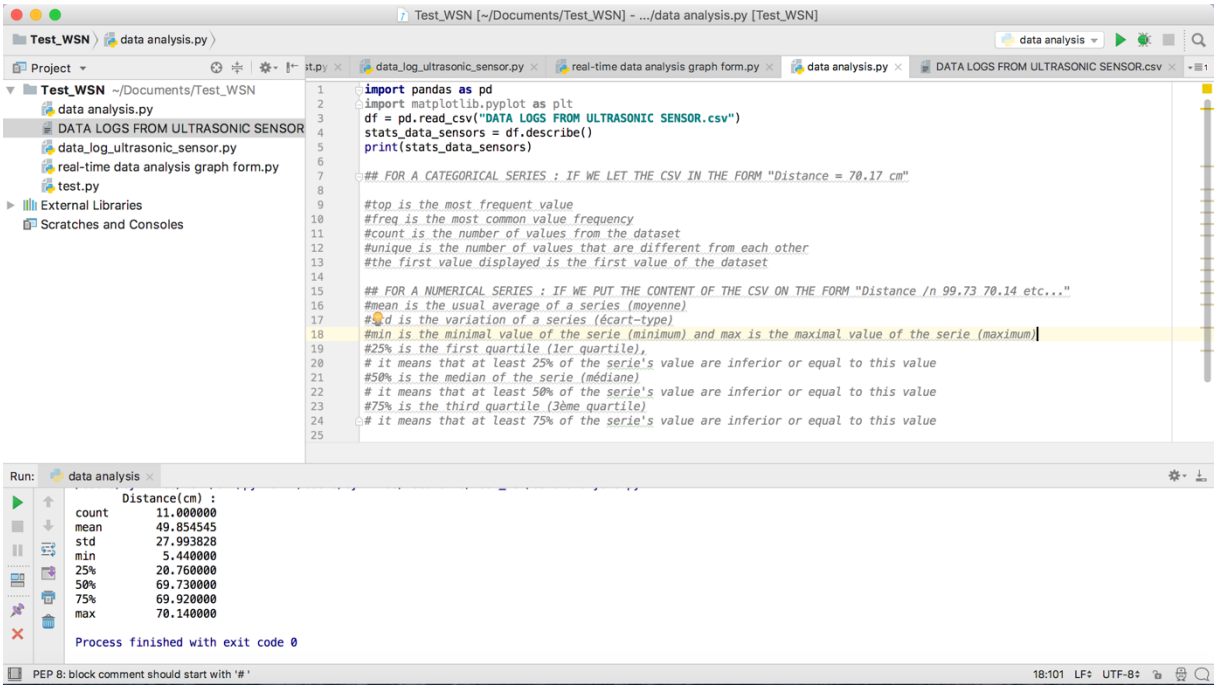
Code that allow to create a .csv or a .txt file with the data collected from Arduino logged in



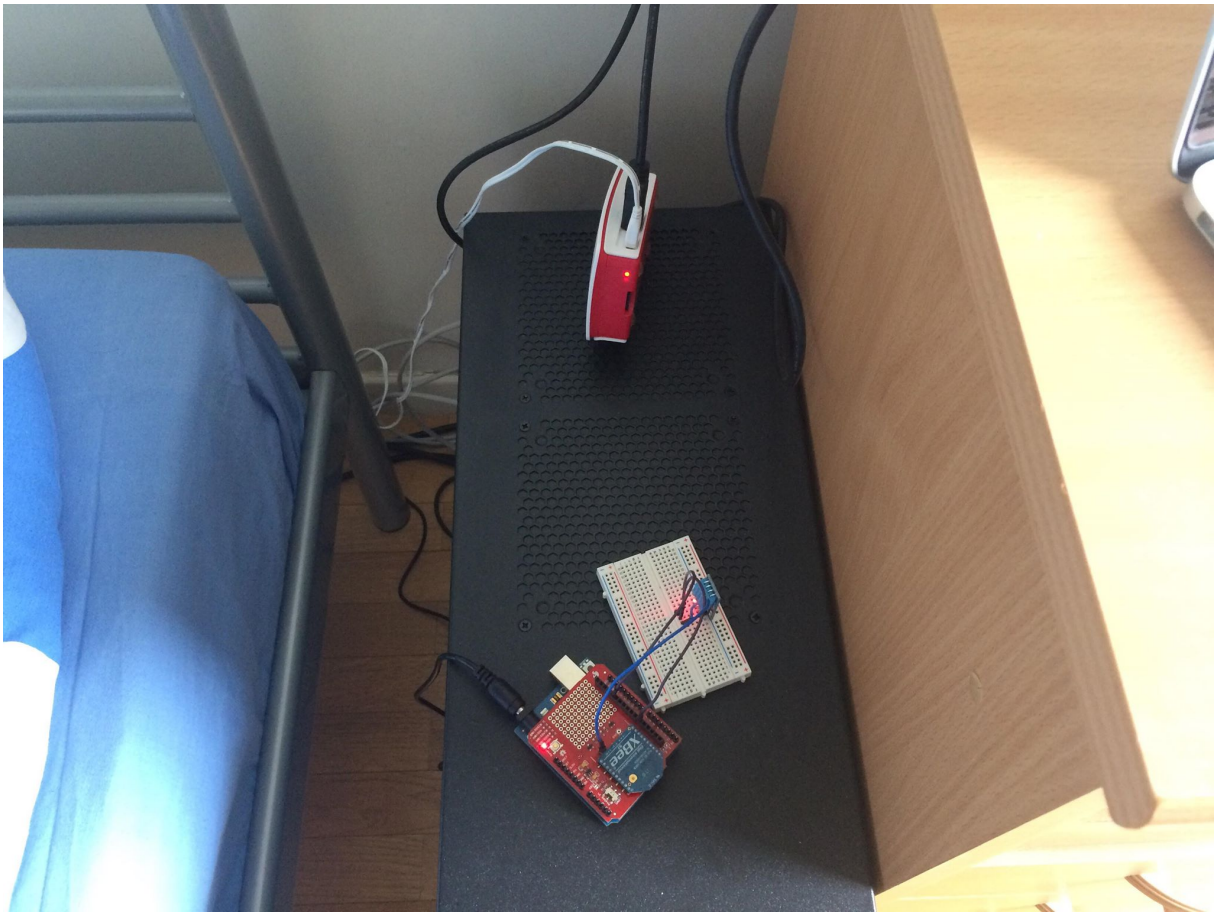
The .csv file generated with all the measures



Some statistical analysis of the values from the .csv file (categorical series mode here)



Some statistical analysis for numerical series mode from the .csv file



Hardware setup

Transfer the data to the Cloud

Sources & Links

- makerspaces
- google images
- Arduino
- Raspberry Pi
- Ubidots tutorial
- Wsn pdf
- StackOverflow
- GOOGLE !!!!