A bottom up sensor testbed

Sergio Almendros Díaz

TFG UPF / YEAR 2014

DIRECTOR/S OF THE TFG:
Jaume Barcelo and Davide Scaini
DEPARTMENT:
Departament de Tecnologies de la Informacio i les Comunicacions (DTIC)

Dedicated to my parents and sister for supporting and helping me at any moment,
and all my colleagues who have been with me all these years.

Acknowledgments

First and foremost I want to thank my supervisor Jaume Barcelo for guiding and helping me throughout the whole project. I would also like to thank Davide Scaini for proposing the project and help me with the final report, to Manuel Palacin for helping me with technical issues and to Albert Domingo for his constructive criticism that made me progress.

I also want to thank the other fellows: Ferran Selva, Pedro Vilchez, and Ivan Fernandez for helping in any way they could. I would like to thank Alejandro Juez, Daniel Valderas, Jacob Uribe, Carles Xavier Vilás, and Aitor Rodriguez, whose experience and knowledge have helped me finish this project. I also want to thank my family for supporting me since I started this project.

Finally, my sincere gratitude to all my friends who have accompanied me in all the endless hours in the library.

Abstract

This project aims at creating a sensor testbed that can be continued and supported by a community. The sensors gather environmental data and are connected to a community network. It gives the community a trace to improve their environmental conscience, giving them the necessary information to know the state of their surroundings and act accordingly. This way anybody can act consequently if the sensors detect harmful situations, for example, not going to a park where the gas sensor detects bad air quality.

The sensor data is uploaded to a data broker developed by the University, which makes the data available to third-party applications. A mobile application that connects to the data broker has been programmed. This application offers an intuitive and meaningful visualization of sensor's data, tailoring its presentation to people without specific scientific knowledge.

Resum

Aquest projecte té com a objectiu desplegar un banc de proves de sensors que pot ser continuat i suportat amb el suport d'una comunitat. Els sensors recullen dades medioambientals i están connectats a una xarxa de la comunitat. Li dóna a la comunitat un camir per millorar la seva consciència ambiental, donant-los la informació necessària per conèixer l'estat del seu entorn i actuar en conseqüència.

D'aquesta manera tothom pot actuar en conseqüència si els sensors detecten situacions perilloses, per exemple, no anar a un parc en el qual el sensor de gas detecta mala qualitat de l'aire. Les dades del sensors s'emmagatzema en un data broker desenvolupat per la universitat, el que fa que les dades estiguin disponibles per a aplicacions de tercers.

Una aplicació mòbil que es connecta al data broker ha estat programada. Aquesta aplicació ofereix una visualització de les dades de sensors intuïtiva i significativa, adaptant la presentació de les dades per a persones que no tenen coneixements científics específics.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Sensor networks started as a mechanism of defense developed by the United States Navy during the Cold War. With acoustic sensors they searched for Soviet submarines. This research continued at Universities, trying to make the sensors smaller, and with the possibility of real-time data adquisition and processing [Chong and Kumar, 2003].

A sensor network is composed of nodes, and nodes are composed of computers and sensors. The function of these computers are: understand the data returned by the sensors, encapsulate it, and send it to the outside.

A node is typically powered by batteries and because it would be unpractical to change batteries at short intervals, the node must require low power to perform all its functions. For this reason, these computers must be equipped with the minimum amount of hardware to perform these functions. To speed the deployment of the network, the most common are ad-hoc wireless sensor networks, which means that nodes will send data from node to node until it finds the target node. The goal is to keep all information collected by all sensors and work with it. In this case, an existing community network has been used, so the communication stage is not part of this project.

The development of this project has been divided into three phases: collect data from environmental sensors, display it to end users in an intuitive way, and make a real testbed to demonstrate its performance.

As a sensor node an Arduino[Arduino, 2014] YUN has been used, which allows to obtain easily analog reads from a sensor. With a Power over Ethernet module, it can be attached to Guifi[1] nodes and send the sensor data to a sensor platform, e.g. Opencities [Opencities, 2014].

The Bottom-up[2] pattern has been used to build the sensor testbed, where the end users, in this case, Guifi users, are the ones who has to assemble the sensor

[1] `http://guifi.net/`

[2] `http://bubforeurope.net`

nodes and connect them to their Guifi nodes to create the sensor network. With the Bottom-up model, the data is provided and used by the users.

In this project an Android application has been developed to visualize the sensor data and make it accessible to the final users. A sensor testbed has been deployed to gather sensor data, and test the technologies used and developed.

This project is an attempt to understand the structure, the technological challenges and operations of sensor networks and how they can help us to take conscience about our environment.

The thesis is structured as follows: the current state of sensor networks (Chapter 2), which technologies had been used (Chapter 3), and how the project has been developed (Chapter 4). In chapter 5 the testbed deployment is illustrated and some resulting data are shown. Finally, the conclusions (Chapter 6) and future work (Chapter 7).

Chapter 2

STATE OF THE ART

2.1 Introduction

Currently the sensors are small enough to put several into a shield of the size of a credit card.

Smart cities are the next step. A city capable of having real-time information, not only about the environment, but also from the number of cars that pass on a road, to the amount of rain water in a day. This kind of information could help to manage more efficiently the city.

It is important to share this information so everyone can know the state of their environment. There are already some sensor networks functioning, some of them are government-funded and they often apply a policy where the data is not completely public, while other sensor networks are private-owned and the data are possibly open (e.g. SmartCitizen).

2.2 Sensor networks and smart cities

In this section a few projects on sensor networks deployed are introduced, those are clear examples of the need for information of our surroundings, in our case, environmental.

2.2.1 Amsterdam smart city

Amsterdam has many projects concerning the smart city concept, like the "Flexible street lighting", which allows the government to monitor the street and switch off the lights to save energy, or the "Smart parking" which informs the drivers if there are free spots to park, and, as a consequence, reduce air pollution [smart city, 2014].

2.2.2 Santander smart city

Santander has his own sensor network testbed for environmental monitoring, outdoor parking area management, and traffic intensity monitoring. Also SmartSantander encourages the development of new applications using the data that is already available (12.000 sensors) [santander, 2014].

2.3 Companies

In the following sections some Companies that are in the business of sensor networks are described. They made a business of sensor nodes and sensor networks, and have been helpful to find a path to follow.

2.3.1 SmartCitizen

SmartCitizen[1] is a platform that offers a sensor board based on Arduino to monitor the environment (Figure 2.1), and uploads this data to SmartCitizen's public database.

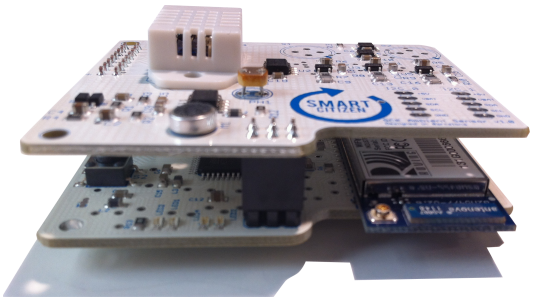


Figure 2.1: Smart Citizen Node.

2.3.2 Libelium

Libelium is an "Internet of things" platform provider[2], which supplies an open source sensor platform for the Internet of things. They have a variety of products, some interesting ones are reported here:

[1] http://www.smartcitizen.me/
[2] http://www.libelium.com/

(a) Libelium ehealth (b) Libelium Waspmote (c) Libelium Smart Water

Figure 2.2: Libellium Projects

- e-Health: A sensor shield for Arduino and Raspberry Pi for body monitoring: pulse, oxygen in blood, airflow (breathing), body temperature, electrocardiogram, glucometer, galvanic skin response, blood pressure, patient position, and muscle/electromyography sensor (Figure 2.2a).

- Waspmote: A sensor node where it is possible attach more than 60 sensors, solar powered, and though to fit onto street light poles. (Figure 2.2b).

- Smart Water: A Wireless Sensor Platform for water quality monitoring, it provides real-time data. (Figure 2.2c).

2.4 Opendata services

The sensor boards are continuously collecting data which has to be stored. The data can be stored remotely or locally, but collecting it from every sensor board it is unpractical, for this reason an online storage service, Open data, is needed. The term Open data pursues the fact that certain types of data should be available to anyone, without any control mechanism, e.g. copyright.

2.4.1 Opencities

Opencities [Opencities, 2014] is a platform to browse, visualize, and download open data from different participants. Opencities has a very simple API to upload and download data, a web page to visualize the stored data.

2.4.2 Xively

Xively [3] offers an Internet of Things platform as a service, basically it lets you store sensor data, download it, and visualize it through graphics.

2.4.3 Sentilo

Sentilo [4] is an open source platform for Smart cities, it allows the user to use their own service to store the data, but they are not looking to be a database where everyone will store their sensor data, they want that each organization has its own sentilo server. The Sentilo platform does not depend on a central server and can be installed on a private server. It also provides an interface to show the data.

2.5 Sensor boards

In this section are discussed some of the options for a sensor node.

2.5.1 Arduino YUN

The Arduino YUN is a micro controller board with two processors (Figure 2.3), an ATmega32u4 (Arduino), and an Atheros AR9331 (Which runs a Linux distribution named OpenWrt-Yun). It has an Ethernet and WiFi module, a USB-A port, a micro-SD card slot, 20 digital input/output ping, 16 MHz crystal oscillator, ICSP header, and 3 reset buttons.

Figure 2.3: Arduino YUN.

[3] https://xively.com
[4] http://www.sentilo.io

The peculiarity about the Arduino YUN is that the processor for the Arduino sketches can communicate with the Linux processor through the bridge library (Figure 2.4), which allows to write Python scripts and execute them.

A Power over Ethernet module (PoE) can be attach to the Arduino, which has a particular interest for the Guifi network, where the antennas are powered through PoE.

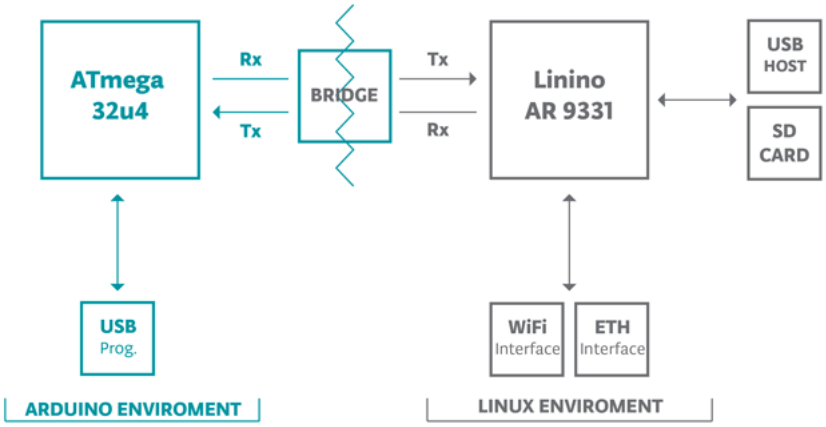


Figure 2.4: Arduino YUN Bridge.

2.5.2 Raspberry Pi

Raspberry Pi[5] is a single-board computer, produced in two models, A and B. The model B (Figure 2.5) is more appropriate for this project because it has an Ethernet controller. It is composed by an HDMI, a micro USB, an USB 2.0 connector, an SD card slot, Input/Output (GPIO) pins, an RCA connector, an audio jack, an Ethernet controller, and a Broadcom BCM2835 processor.

It is possible to attach sensors to it, and it supports Linux environment.

2.5.3 Picoboard

PicoBoard[6] (Figure 2.6) is a board to interact with the world, it can be programmed by Scratch projects. It is less flexible than the previously mentioned, it is composed by a button, a light and a sound sensor, a slider, and alligator clips which can be use to connect custom sensors.

[5] `http://www.raspberrypi.org/`
[6] `http://www.picocricket.com/picoboard.html`

Figure 2.5: Raspberry Pi model B.

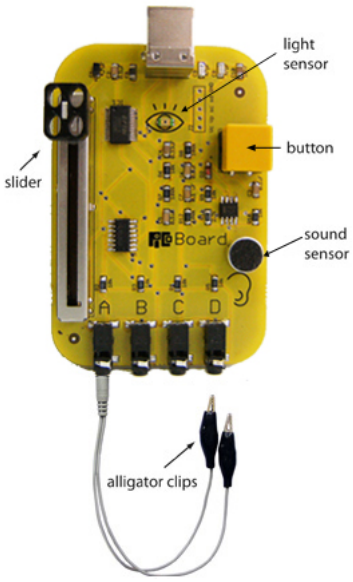


Figure 2.6: Picoboard.

Chapter 3

TECHNOLOGIES

This chapter is focused on the technologies used to develop this project.

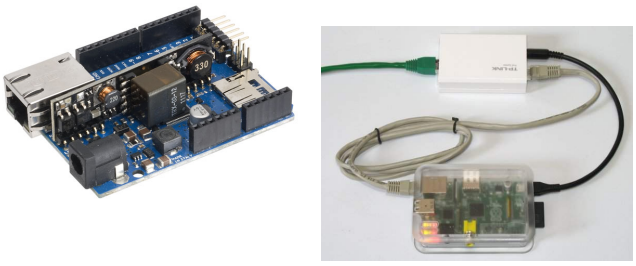
3.1 Arduino-Raspberry PI comparison

Sensor networks should not affect the environment, in our case, visually. For this purpose we must ensure that the nodes are relatively small, so the board and sensors had been selected accordingly.

To choose the processor board, have been considered the following characteristics:

- **Power:** A sensor node can obtain energy in several ways: batteries, solar energy, connecting to an already deployed network, etc. The important thing is not to increase the size of the sensor node, in our case, the power is given by an already deployed network.
- **CPU:** A powerful CPU is not necessary to process sensor data, so this helps to decrease the size of the board and reduce power consumption.
- **Communication:** The collected data has to be sent to the broker, usually through a wireless or wired network. If the communication is wireless, the topology and location of the sensor nodes has to be chosen properly. But if the connection is through a wired network, it should take advantage of something already deployed, in our case, the Guifi network.

For this project, two possible boards have been considered: the Arduino Yun (Yun) and the Raspberry Pi model B (RPiB). The Yun is a microcontroller while the RPiB is a full computer, which makes it more powerful, even more than needed.



(a) Arduino Ethernet PoE (b) Raspberry Pi (B) PoE

Figure 3.1: Sensor boards

The two of them have a Linux environment, but with the Yun, the normal way to interact is by an Arduino sketch. RPiB considers that the developer has some prior Linux knowledge, while the Yun is better for beginners. Also, the Arduino IDE provides a variety of programs that help the learning process.

Another difference is that Arduino is open hardware, which might be helpful if it has to be improved.

The sensor board is the biggest element of the sensor node, so the size is very important, the smaller the better. In this case, the Yun is smaller than the RPiB. On the other hand, the sensor node will be attach to Guifi nodes, which are powered by Power over Ethernet (PoE). For the Yun there is the possibility to connect a PoE module (not available yet), which will power the board. The Figure 3.1a and 3.1b show the comparison between the size of an Arduino Ethernet (similar to the YUN) and the RPiB both powered by PoE. It is clearly visible that the Arduino is more compact.

Another advantage that YUN offers is the integrated WiFi module, which offers more flexibility when there is no ethernet available.

Finally, the YUN has been used instead of the RPiB basically because of the size that will become the sensor node, but if in the future will be necessary a more powerful processor, the RPiB is better suited.

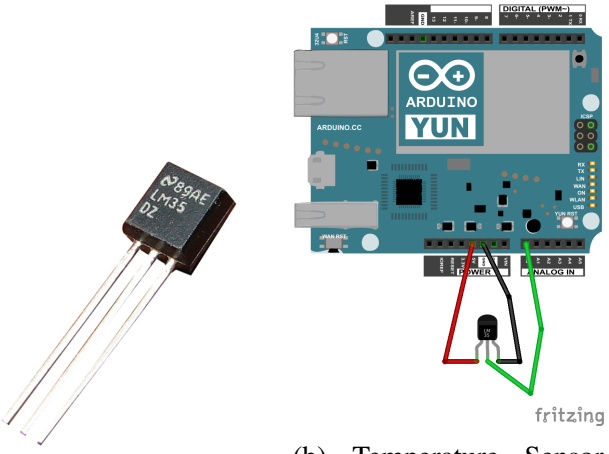
The Table 3.1 shows the differences between the RPiB and the YUN explained before.

3.2 Sensors

The goal is to analyze the environment around us, and sensors are a meant to detect physical or chemical variables. It has been decided to use low cost environmental

Functionalities	YUN	RPiB
Way of interaction	Arduino Sketch and Linux	Linux
Development skills required	Low skills with Arduino and Linux	Medium skills
Size of the board	6.58 x 5.33 cm	8.55 x 5.41 cm
Computational power	Yun:16MHz - Linux:400MHz	700MHz
Available memory	Yun: 32 KB - Linux: 16 MB	512MB

Table 3.1: YUN-RPiB comparison



(a) LM35 sensor

(b) Temperature Sensor Breadboard

Figure 3.2: Temperature sensor

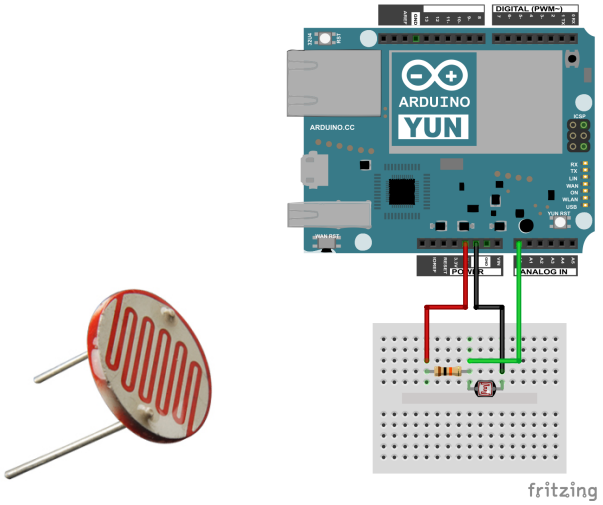
sensors that are easily accessible and usable. These sensors measure the aspects that may be more useful for citizens: temperature, light, noise, humidity, and air quality.

A sensor is a device which transform a physical measure to an output signal (Voltage or Digits) that can be read by another device, such as an Arduino. In this project five sensors has been used to measure temperature, light, noise, humidity, and gas. To show how the sensors are connected to the Arduino YUN the program Fritzing[1] has been used.

3.2.1 LM35: Temperature

LM35 [Figure 3.2a] is a sensor to measure temperature, in the Figure 3.2b is shown how to connect it to the Arduino. [Instruments, 2013]

[1] `http://fritzing.org/`



(a) LDR sensor

(b) Light Sensor Breadboard

Figure 3.3: Light sensor

3.2.2 Light Dependent Resistor (LDR)

LDR [Figure 3.3a] is a light sensor, in the Figure 3.3b is depicted how to connect it to the Arduino.

3.2.3 Emartee Mini Sound Sensor and Analog Sound Sensor Board Microphone MIC Controller: Noise

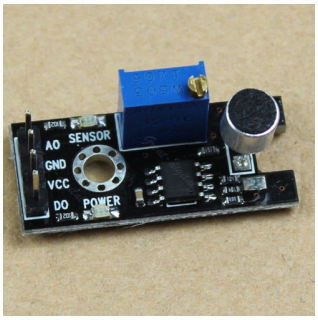
This two sensors [Emartee, 2013] are used to measured noise levels [Figure 3.4a] and [Figure 3.4b]. The code to read the noise values is the same for both. In the Figure 3.4c is shown how to connect them to the Arduino.

3.2.4 Aosong DHT22 and DHT11: Humidity

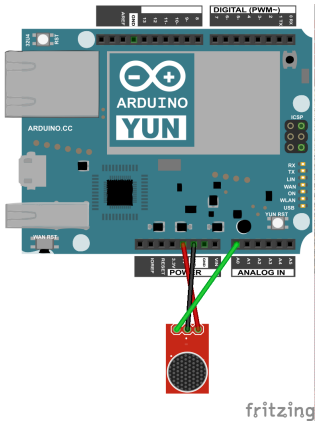
DHT22 [Figure 3.5a] and DHT11 [Figure 3.6a] are humidity and temperature sensors, although the humidity measure is the only one used. The output is digital, and to read it, an external library[2] is needed. The Arduino and the humidity sensor have to be connected as shown in the Figure 3.5b for the DHT22 and as shown in the Figure 3.6b for the DHT11 [Aosong Electronics Co., 2013].

[2] `https://github.com/adafruit/DHT-sensor-library`

(a) Mini Sound Sensor

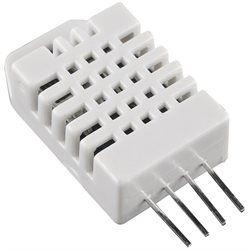


(b) Analog noise sensor

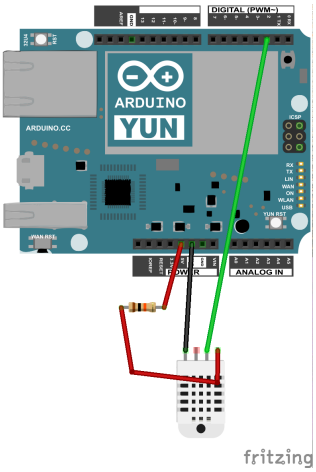


(c) Noise Sensor Breadboard

Figure 3.4: Noise sensor

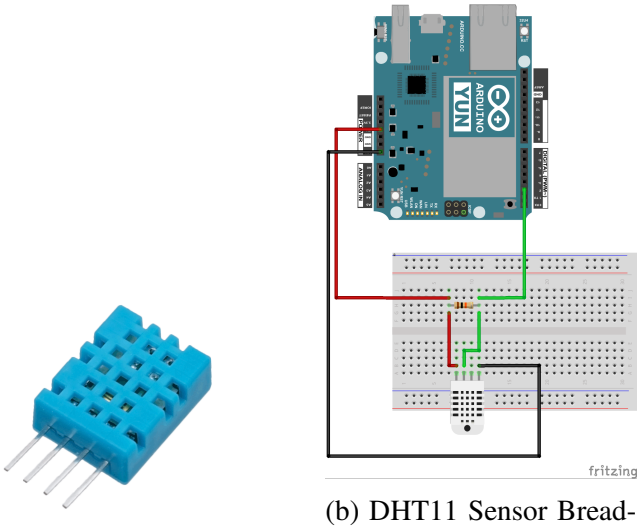


(a) DHT22 sensor



(b) DHT22 Sensor Breadboard

Figure 3.5: DHT22 sensor



(a) DHT11 sensor

(b) DHT11 Sensor Breadboard

Figure 3.6: DHT11 sensor

3.2.5 MQ135: Gas sensor

MQ135 is a gas sensor [Figure 3.7a], and is used to measure air quality. Figure 3.7b illustrates how to connect it to the Arduino.

The Mq135 sensor reacts to the concentration of the following gases: NH3, NOx, alcohol, Benzene, smoke, CO2, etc. [Futurlec, 2013].

3.2.6 BreadBoard with all the sensors

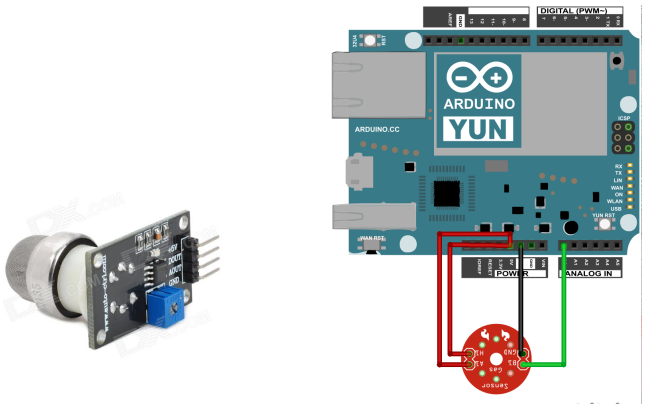
In the Figure 3.8 is shown the final prototype with the DHT22 sensor, and in the Figure 3.9 with the DHT11 sensor.

3.3 Upload sensor data

A main part of the project is uploading the data from the sensors to a platform to make the data publicly available.

The GeoJSON message (explained in section 3.3.1) includes data from 5 sensors, which makes its size too large for the memory of the Arduino. For this type of operations is possible to run a script in the Linux environment through the Bridge library (Figure 2.4).

This being the path that is followed, there are several options for developing this script, for example Java, C / C + +, Python, etc.. Python has been used because



(a) MQ135 Air Quality sensor

(b) Gas Sensor Breadboard

Figure 3.7: Gas sensor

of the following reasons: fast and perfect for prototyping programming, the code is shorter and therefore, easier to understand, and modular.

3.3.1 GeoJSON

A GeoJSON[3] is a format for encoding a variety of geographic data structures. The GeoJSON that has been used is a collection of features, every feature contains a geometry object, in our case, a "point" with the longitude and latitude of the sensor node, and some properties required: ID, name, datasetID, datasetName, address, description, timestamp, value of the sensor, and unit.

The Figure 3.3.1 shows an example of a GeoJSON message that is used in this project.

3.4 Community network

Guifi[4] is the network where the Arduino nodes will be deployed, and the one providing the access to Opencities through the Internet. Guifi is a network created by people interested in building an open, free and neutral network infrastructure.

Because of their philosophy of participation, Guifi is the perfect network for the deployment of this sensor nodes. Also their network is large enough to become a useful sensor network (Figure 3.11).

[3] http://geojson.org/
[4] https://www.guifi.net/

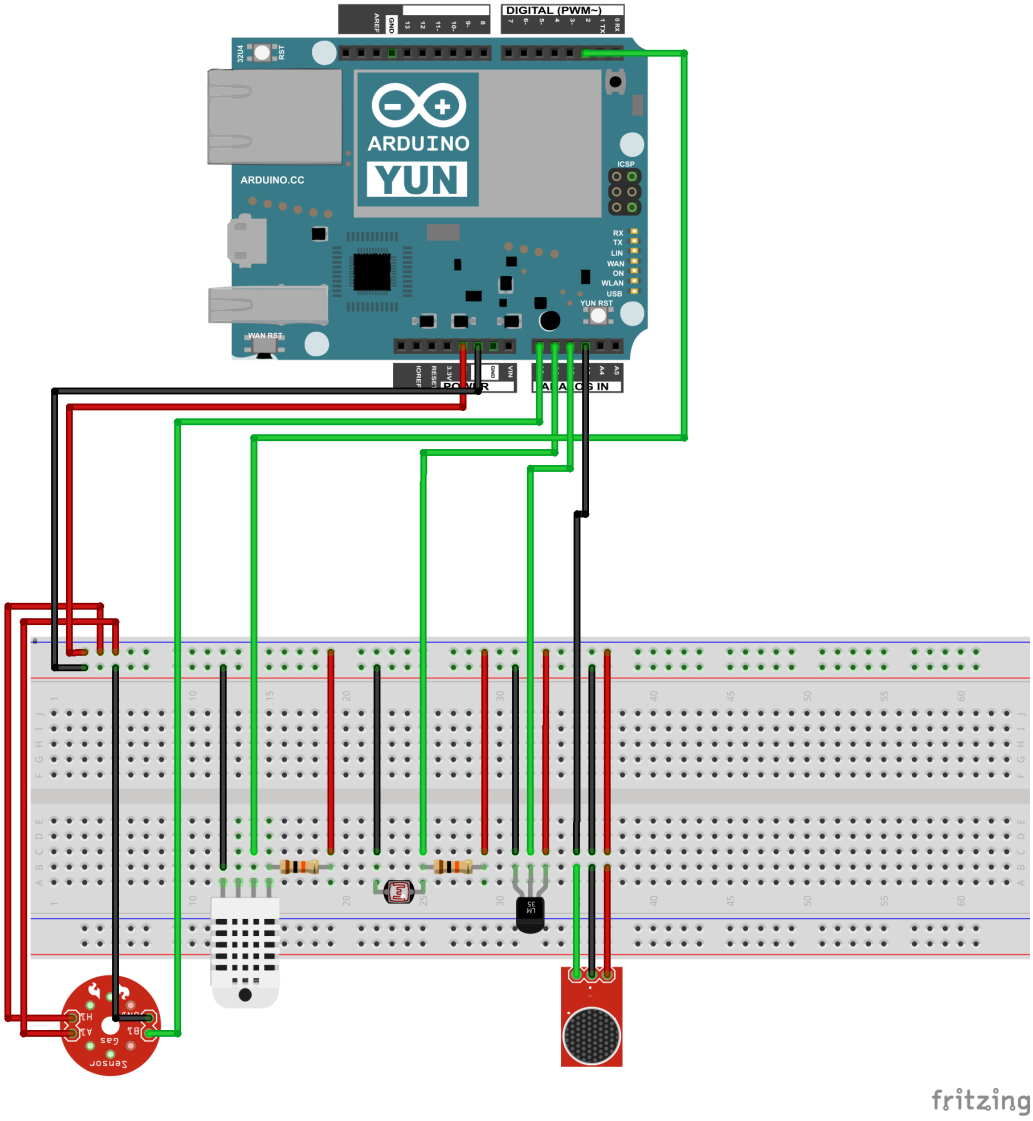


Figure 3.8: Sensor node Prototype with DHT22.

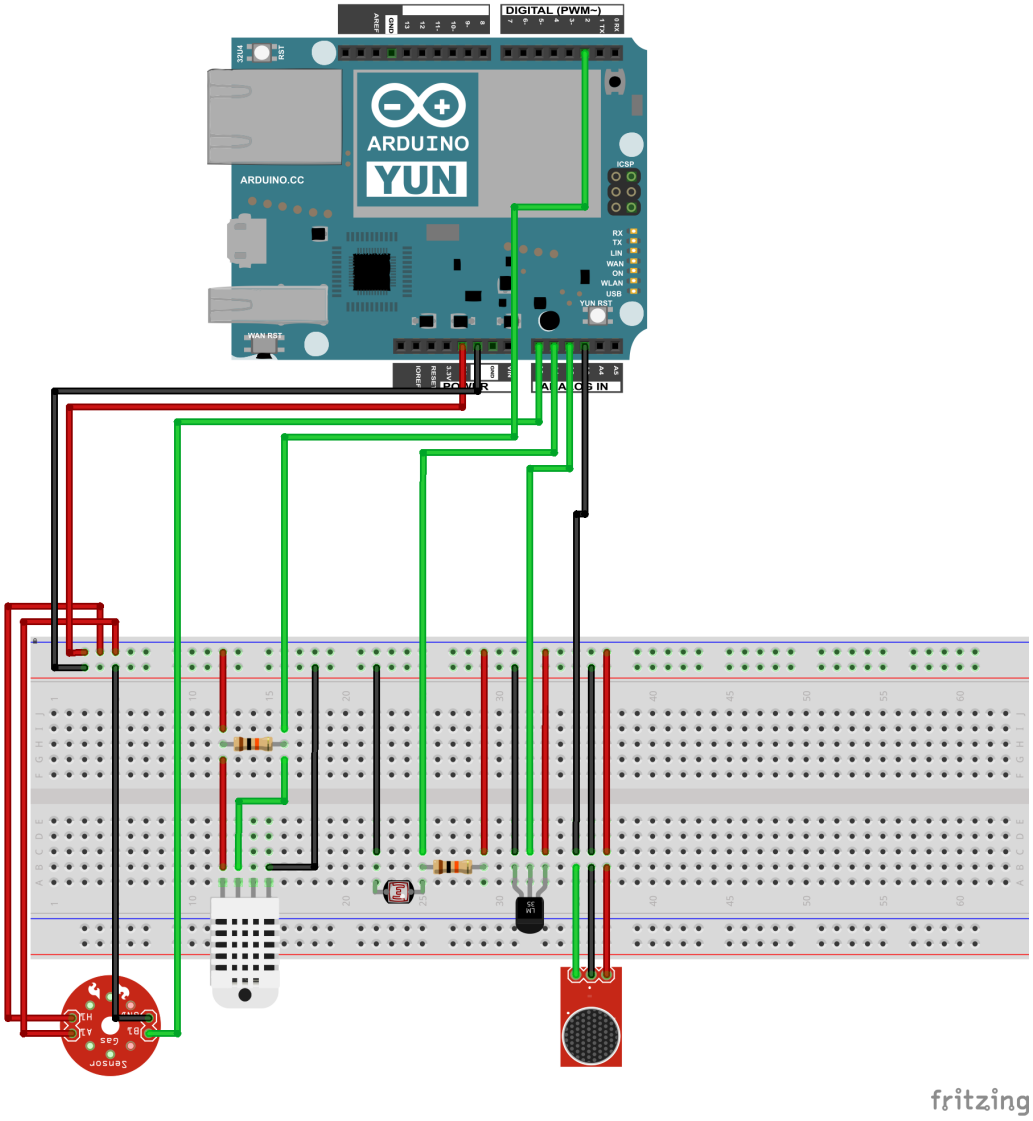


Figure 3.9: Sensor node Prototype with DHT11.

```
{
    "type": "FeatureCollection",
    "name": "dummy",
    "timeStamp": "2014-06-12T08:54:59.424Z",
    "features": [
        {
            "type": "Feature",
            "tags": [
                "red",
                "tall",
                "cheap",
                "upf"
            ],
            "geometry": {
                "type": "Point",
                "coordinates": [
                    2.18946,
                    41.403809
                ]
            }
        }]
}
```


Figure 3.10: GeoJSON message.

Figure 3.11: Guifi nodes.

3.5 Storage Resource Broker

Opencities [Opencities, 2014] is the opendata service that has been chosen, for these reasons:

- The developers are at UPF, so the process of improving both projects (feedback, bug fixing, etc) can be fast and effective.
- Easy API to upload and download the data.

The role of Opencities is to be the intermediary between data creators and data users. In Opencities, the sensor data is uploaded to a dataset. Every user has an unique API key, and can create one or more datasets with a unique ID for each one.

3.6 Visualization platform

For the purpose of the sensor data visualization there are several options, in plain text, in a map, in graphics, etc. In this project a map is used to display the data. This can be done on a web page, in a mobile application, tablet application, etc.

Since the goal is that a user checks it for a small period of time, the best option is a mobile application. Additionally, almost everybody has a smartphone.

There are three mobile operating system, iOS, Android, or Windows Phone. There is another option that is to developed the application in Html, Javascript, and CSS, and then compile it with Phonegap[5], which builds the application for the three operating systems. This system ends up being much more difficult than to create an application directly to an operating system. Of the three operating systems, the one with more market share has been chosen to reach as many people as possible, so the Android operating system has been chosen.

Android is a mobile operating system developed by Google, it runs on smartphones, and applications are programmed in Java. Java is a computer programming language that is object-oriented. This application will be tested on a Sony Xperia Z1, with an Android 4.4.2.

[5] `http://phonegap.com/`

Chapter 4

BOTTOM UP SENSOR TESTBED

This chapter is focused on the process followed during the project, which has two main parts, the software to collect and send the data and the Android application to show it. A scheme of the project is shown in Figure 4.1.

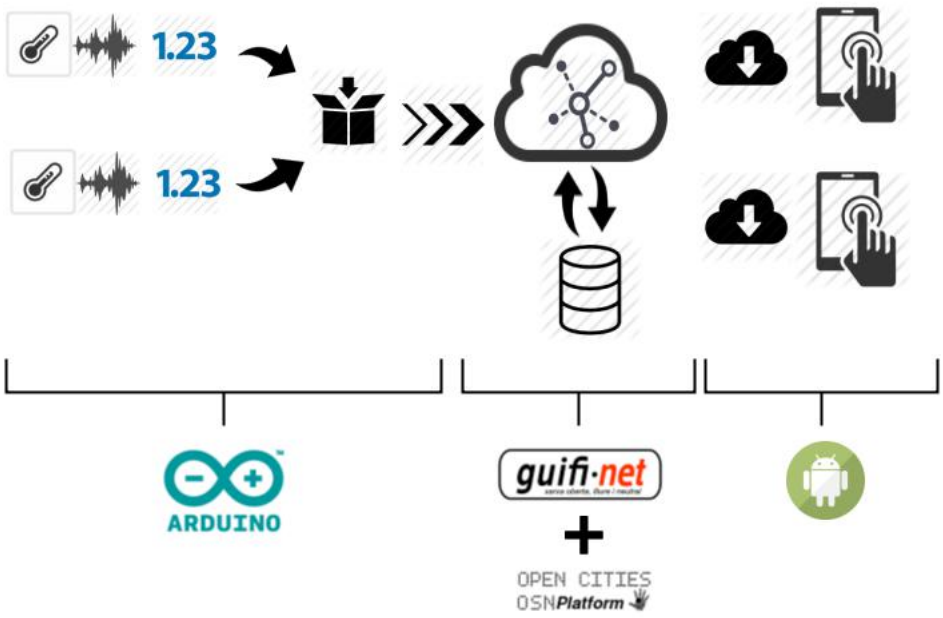


Figure 4.1: General View.

4.1 Arduino Development

Two scripts are needed, one to collect the data, and other to send it. This is because the memory to run an Arduino sketch is very low, and the creation of the GeoJSON message is too big with respect to the available memory. That is why a Python script has been used, called first by the Arduino sketch.

The Arduino sketch is responsible of collecting the data, write it in a logData file, and call the Python script with the collected values and an unique ID. Finally the Python script creates a GeoJSON message and sends it to Opencities.

4.1.1 Collect sensor data

To collect almost all the data the sketch does not need to include external libraries, except for the humidity sensors (DHT22 and DHT11)[1].

The following libraries were needed to develop the code: FileIO, Process library, and Bridge.

The Arduino sketch is coded in a very simple way, the setup function will initialize the Bridge library to communicate with the Linux environment, the Serial library for debugging purposes, and the FileSystem to log the process. The other function by default is the "loop", which calls three functions: readSensors, readFile, and executePythonScript.

- **readSensors:** It calls 5 different functions to read every sensor, the reason for doing a separate function is to make the code more clear.
- **readFile:** This function logs the process, saves the ID of the message, the sensor values, and a timestamp.
- **executePythonScript:** The script in Python located in the SD card is in charge to create the GeoJSON message with all the sensor data and upload it to Opencities. This script is called by the Arduino sketch.

The Figure 4.2 explain how the Arduino sketch works.

4.1.2 Communication with Opencities

To communicate with Opencities it is necessary to create an account, and a dataset to store the sensor data. At the end, the user has an API key and a dataset ID, the required information to upload and download data.

[1] `https://github.com/adafruit/DHT-sensor-library`

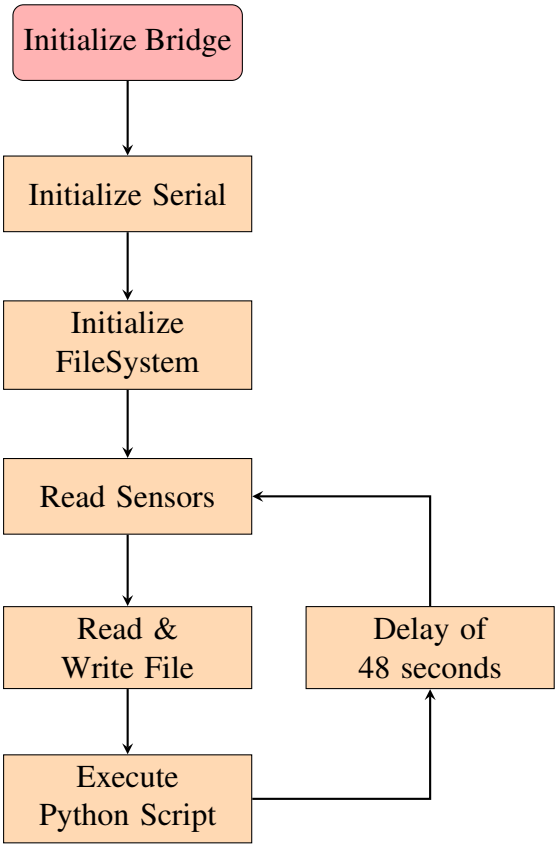


Figure 4.2: Arduino sketch Flow Chart.

The communication with Opencities is done by means of a Python script. This are the packages needed into the Linux environment of the Arduino: distribute, python-openssl, geojson, geopy, and httplib2.

The Python script has three processes. First it stores all the sensor data collected by the Arduino sketch into a class. Then it creates a GeoJSON message with all the data, and all the parameters needed to upload it into Opencities. Finally uploads the GeoJSON message.

With all these libraries the script can communicate with Opencities and store the sensor data collected by the Arduino. The Figure 4.3 explains how the Python script works.

4.2 Android app

4.2.1 Summary

To easily visualize the results of the testbed, an Android application has been developed which shows the sensor data in a map. The application shows the data in two ways, with markers that show the actual value in that point, and also with heatmap points, the larger the value, the more intense the red will be (Figure 4.4).

4.2.2 Interface

The application interface is a map view where the user can zoom, go to his location, and use the top buttons to change the sensor read. From left to right, the first button is the Marker button, the user decides whether the markers are displayed or not, and the next buttons refer to the type of sensor data to show as markers and/or as heatmap points (Temperature, Humidity, Noise, Light, and Air Quality). If the Marker button is checked, the user can click on the marker in the map and it will show the value and the unit of the temperature, humidity, noise, light, and air quality (Figure 4.4).

4.2.3 Code

First of all, to create this application the Google Maps Android API v2[2] for the map view, and the Google Maps Android API utility library[3] for the heatmaps have been used.

[2] `https://developers.google.com/maps/documentation/android/`
[3] `http://googlemaps.github.io/android-maps-utils/`

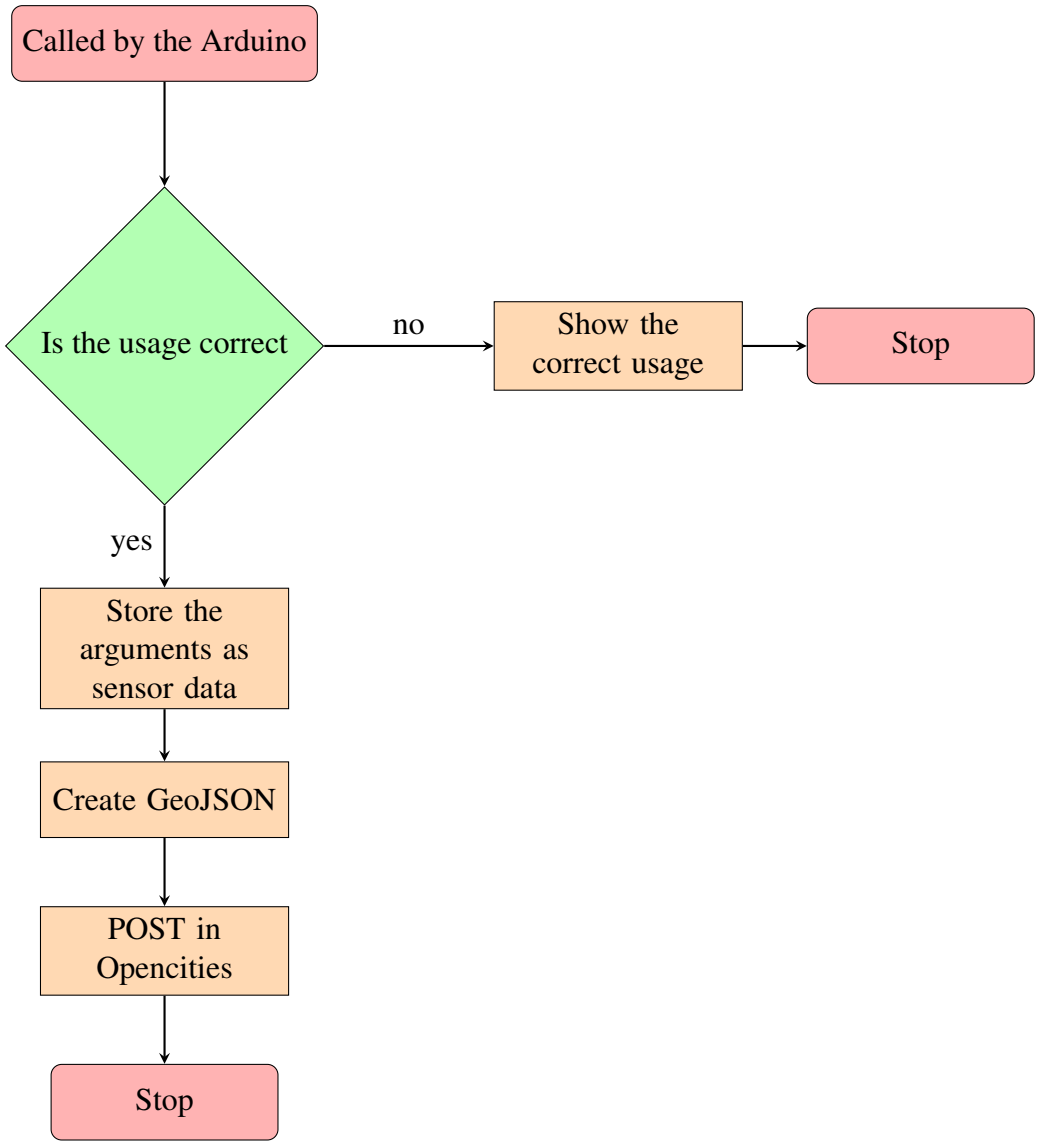


Figure 4.3: Python Script Flow Chart.

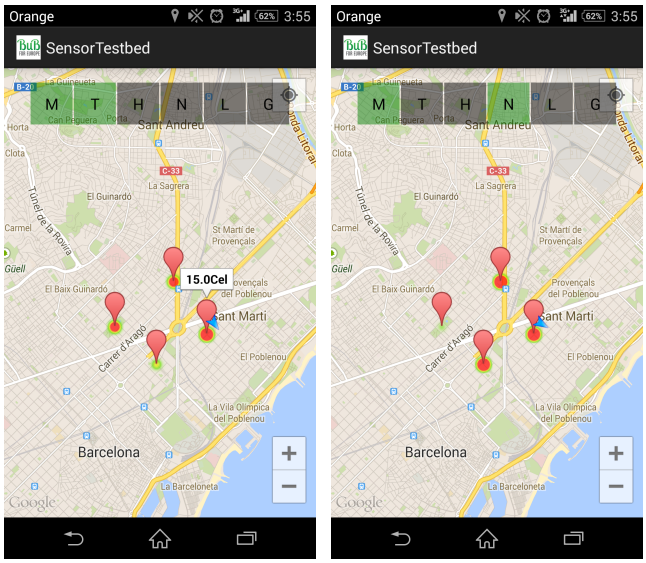


Figure 4.4: App Screenshots

In this application each class has a defined function, starting with MainActivity, which is the one that initialize the other classes, calls all the functions needed in the other classes to get the application started. It also has all the code to interact with the interface. For example, if the user clicks on the marker button, this class has to call all the necessary functions and display all the markers into the map.

Next comes the dataBase class, which is the class that stores and processes the data downloaded from Opencities. This is a Singleton class, which means that there will only be one instance of this class and any instance of any class could access to the data stored in it.

To store the data from Opencities, a set of classes have been created. These classes are equivalent to the GeoJSON message. At the end, the data is stored in features, which is composed primarily of Geometry and Properties. Then a class called HttpAsyncTask, that handles the communication with Opencities, stores the downloaded data using an instance of dataBase.

Finally, a GPSTracker class is responsible for obtaining user location to place the map on the user location.

The Figure 4.5 shows a simple class diagram where it only shows the class name and the connections between them.

The explanation of the code is in the flow chart 4.6.

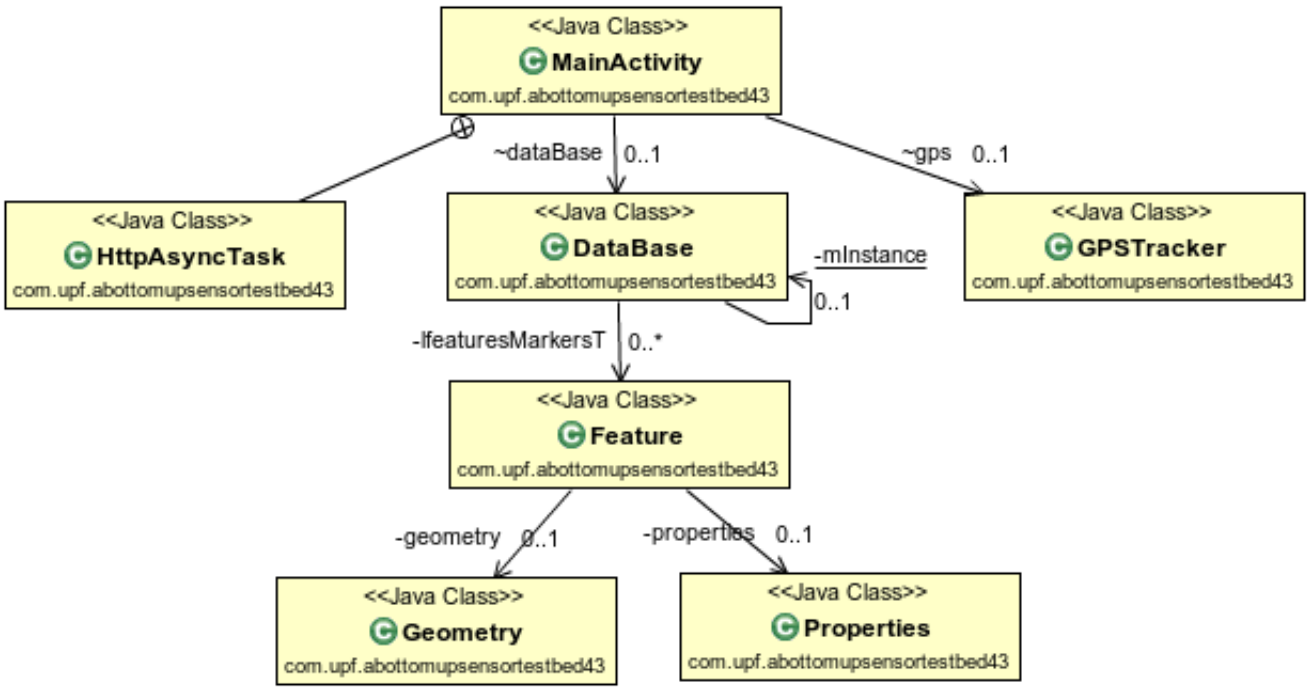


Figure 4.5: Simple Class Diagram of the Android App.

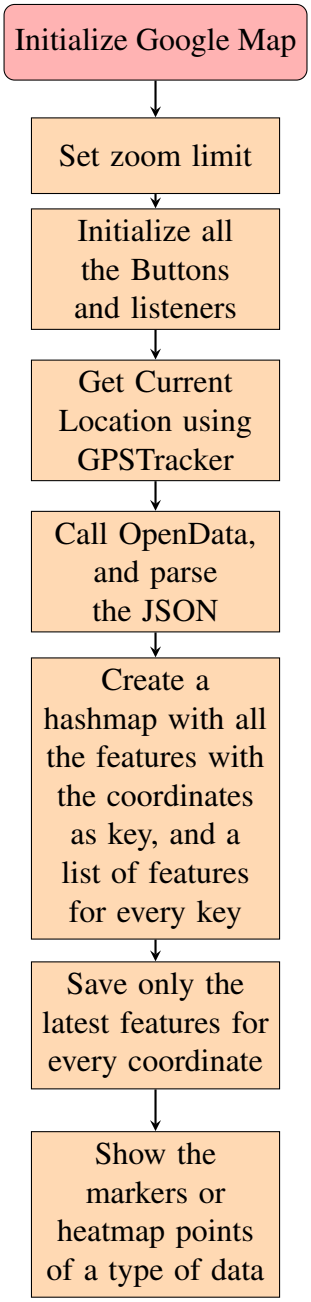


Figure 4.6: Android App Flow Chart.

4.3 Repository

All the code, report, figures, etc has been stored in a public repository[4], so anyone could see it, download a copy, and change it if necessary.

The Figure 4.7 illustrates how the repository is organized, each folder represents part of the process that has been done in this project.

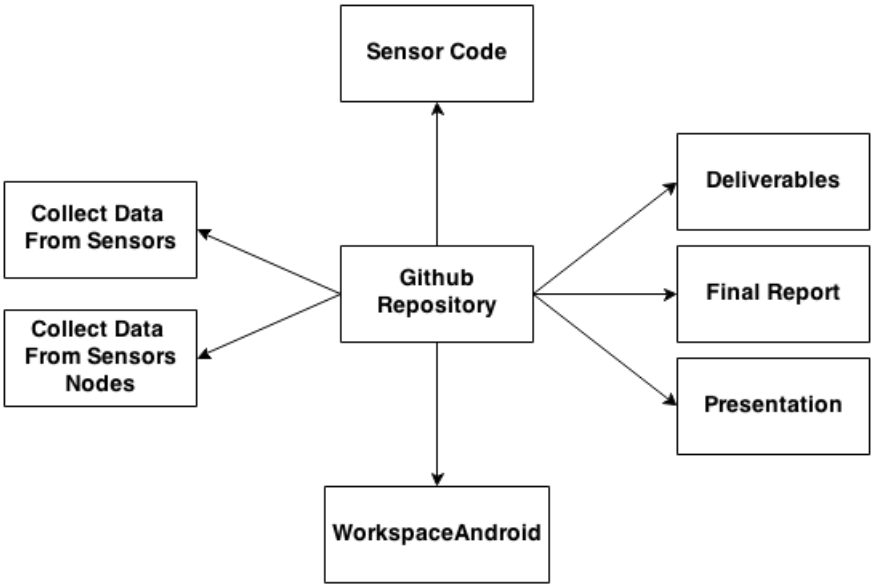


Figure 4.7: Github Repository.

The most important folders are:

- **Sensor Code:** Where the code has been stored that runs into the Arduino (Arduino sketch and Python script).
- **WorkspaceAndroid:** This folder has been used as workspace for the developing of the mobile application using the Eclipse IDE (Integrated Development Environment).
- **Final Report:** Is where the project it has been documented. The report has been written in Latex, a system for the presentation of technical and scientific documentation.

The other folders are:

[4] `https://github.com/SergioAlmendros/A-bottom-up-sensor-testbed`

- **Collect Data From Sensors:** Here it has been stored the code to get the value of each sensor separately, for easier reusability of the code.
- **Collect Data From Sensors Nodes:** Using the logData files that have been created during the testbed, an octave script reads this data and displays it through graphs.
- **Deliverables:** In this folder the initial documentation of the project has been stored, that are reported in the Appendices A and B.
- **Presentation:** Finally, here is the slide presentation that has been made to explain the project.

During all the project there have been constant uploads of new information to the repository, as it was evolving.

Chapter 5

TESTBED DEPLOYMENT AND RESULTS

This chapter explains the procedure to deploy a testbed step by step.

In the Figure 5.1 there is a photograph of the prototype used in this testbed. It is composed of an Arduino YUN,- a microSD card, a breadboard, and all the sensor connected (temperature, humidity, noise, light, and gas) to the Arduino YUN.

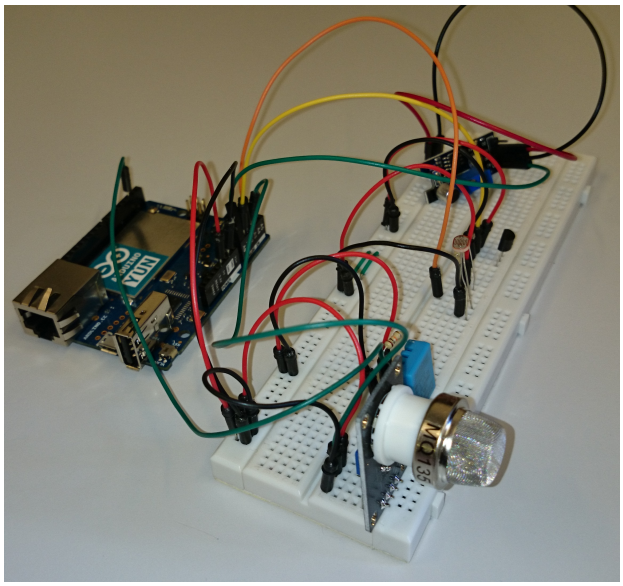


Figure 5.1: TestBed Prototype.

5.1 Sensor node

This section shows the process to configure the sensor node [Guide, 2014]:

5.1.1 Connection to the Internet

First of all an Internet connection has to be provided to the Arduino.

5.1.1.1 Through Ethernet

This is the fastest way to provide Internet connection, the Arduino automatically obtains an IP address.

5.1.1.2 Through WiFi

This is the slowest way. The process is the following:

1. After powering the Arduino YUN, it creates his own WiFi network (ArduinoYun-XXXXXXXXXXXX).
2. After connecting to the YUN network, the address `http://arduino.local` or `192.168.240.1` can be accessed and the password should be provided. The default password is "arduino". The procedure is shown in the Figure 5.2.

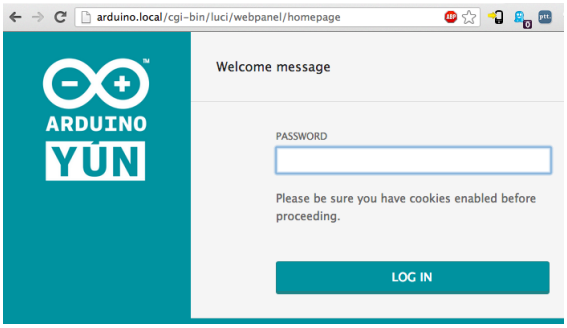


Figure 5.2: Yun web Password.

3. After providing the password, an information page is displayed, as shown in Figure 5.3.
4. It is necessary to tell the Yun what network to connect to, and after pressing the the configure & restart button, the computer can connect to the Yun through the wifi network (Figure 5.4).

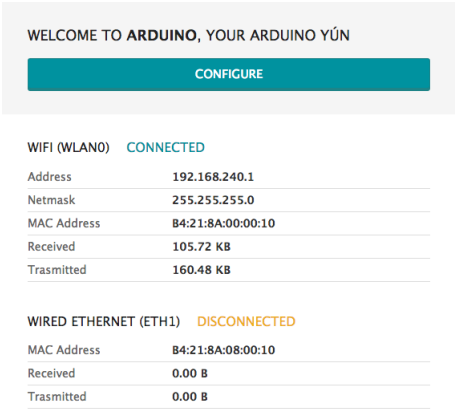


Figure 5.3: Yun web Diagnostic.

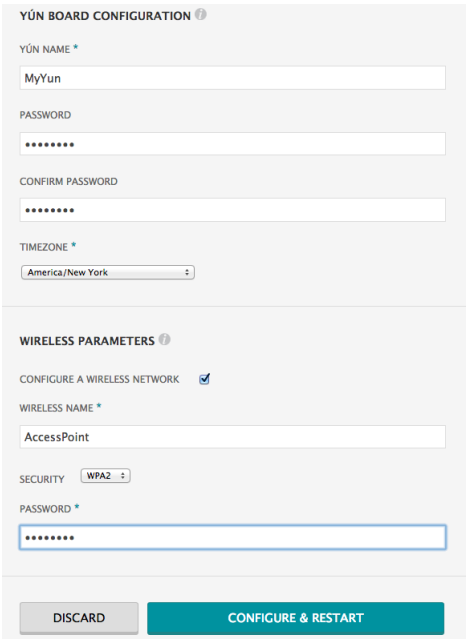


Figure 5.4: Yun web Configuration.

5.1.2 Install necessary packets

The Arduino runs an Arduino sketch and a Python script, for the Python script some packets have to be installed.

To install any packet, a ssh connection to the arduino has to be created (as shown before):

ssh root@X.Y.Z.W

The required commands are the following:

opkg update
opkg install distribute
opkg install python-openssl
easy_install install geojson
easy_install install geopy
easy_install install httplib2

5.1.3 Copy the scripts

First of all, it is necessary to create some directories, the following commands have to be executed:

cd /mnt/sda1
mkdir arduino
cd arduino
mkdir www

After connecting to the YUN network, the address http://arduino.local can be accessed and the password should be provided. The default password is "arduino". The procedure is shown in the Figure 5.2.

After these processes have finished, the Python script "main.py" should be copied into the SD-Card. There are two ways to do this, putting the SDCard into a computer an saving the file directly, or copying the file into the Arduino through the network with the following command:

scp main.py root@192.168.2.149:/mnt/sda1/arduino/www/main.py

5.1.4 Attach the sensors

Now that the Python step is done, the sensors have to be attached to the Arduino Yun (Figure 3.8 or Figure 3.9).

5.1.5 Arduino Code

To upload an Arduino sketch to the Yun the IDE (Arduino 1.5.5) has to be used. There are two ways to upload a sketch, through an USB cable connected to the

Arduino, or through the Internet. If the Arduino and the computer are in the same network, the Arduino will appear in the IDE as a possible device to connect with.

5.2 Actual Testbed

Three Arduino nodes were built and placed in three different locations.

The unique ID has to be introduced manually into the Arduino sketch with the Arduino IDE and the location into the Python script by logging into the Arduino by secure shell and modifying the following line in “main.py”:

self.address = ’Sagrada Familia, Carrer de Mallorca, Barcelona’

After a day of collecting data, the three logData files has been analyzed, and some graphs to show the data had been made.

The Figure 5.5 shows the nodes deployed.

Figure 5.5: Testbed Map.

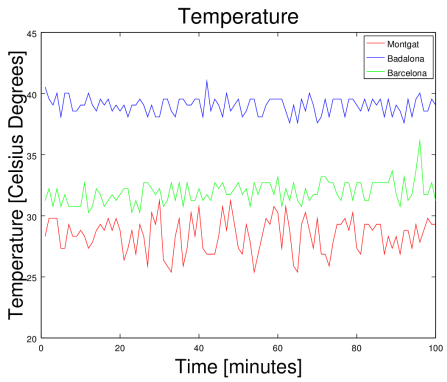
5.2.1 Results

Thanks to the testbed a sensor network has been deployed, with three nodes, for economical reasons, without any problems.

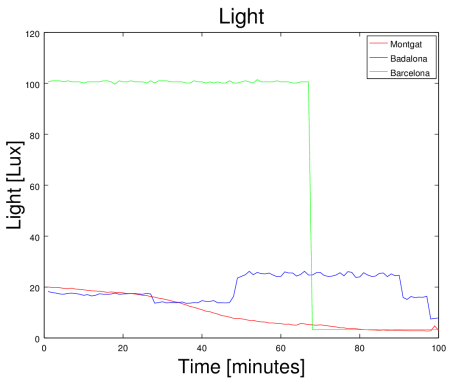
A mobile application has been made, and shows an example that could led to more people to create other applications that work with the sensor data stored in Opencities. The data has been stored on a platform and the mobile application has

been able to access them. A problem happened, the application was not designed to cope with big amount of data, so it needs improvement.

All the data produced during this testbed has been stored in the SD card of the Arduino nodes, and graphically displayed in figures: 5.6a, 5.6b, 5.7a, 5.7b, 5.8.

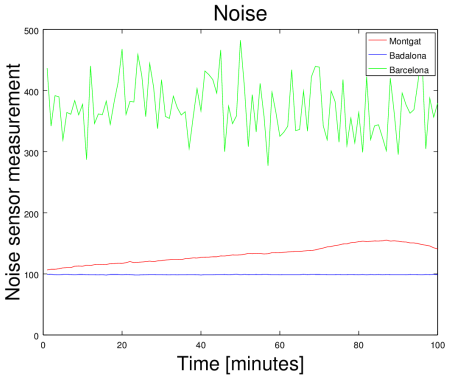


(a) Graphic Temperature

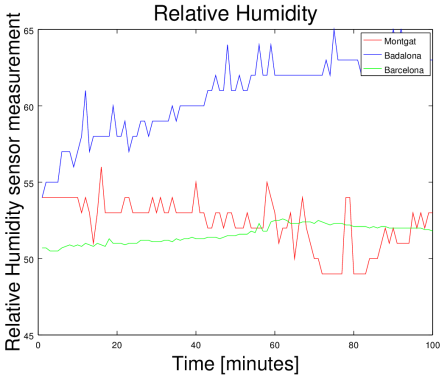


(b) Graphic Light

Figure 5.6: Temperature and Light Graphics, for 100 minutes acquisition window



(a) Graphic Noise



(b) Graphic Humidity

Figure 5.7: Noise and Humidity Graphics, for 100 minutes acquisition window

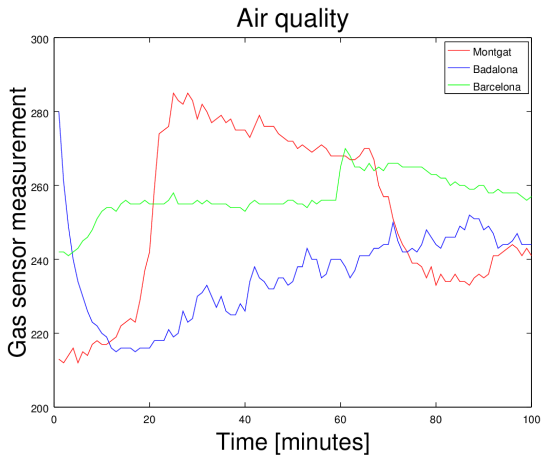


Figure 5.8: Graphic Air Quality, for 100 minutes acquisition window.

From the acquired data it is possible to draw some conclusions:

The temperature is clearly higher in Badalona, but the three locations are not far apart, so this value should not vary much. It is important to notice that this is a low quality sensor so it is possible that it may have failed during the testbed since the graphs of Montgat and Barcelona are similar between them, but very different in comparison to Badalona.

In the case of light, the sensor node in Barcelona receives direct sunlight during most of the day, but the node in Montgat and Badalona does not. Consequently, it is normal that the Barcelona node had a much higher light value.

The Barcelona node collects noise values much higher than the other two nodes. The Badalona and Montgat nodes are in areas with punctual car traffic, while the Barcelona node is in a busy area, causing more noise.

In the case of the relative humidity, the Badalona node is near a river and the Montgat node is near the sea, so it is logical that the Barcelona node measures less relative humidity.

The Badalona node captures better air quality than the Barcelona and Montgat nodes. The Barcelona node collects constant values, more or less, which is normal because there is in an area with high car traffic. The Badalona node collects a decrease in air quality as time passes, which coincides with the arrival of workers who park in the area. In the case of Montgat, there is a big difference in a time slot. This node is located near a highway, so it is possible that this time slot coincide when there is more traffic on the highway.

Chapter 6

CONCLUSIONS

The deployment of a sensor network made during this project has been successful, this has proved that the sensor nodes have functioned properly. Therefore, it has been shown that anyone can deploy its own network in an inexpensive way, with open source hardware and software, and easy to use.

A mobile application has been developed to serve as an example, so the citizens who want to create their own mobile application find the process easier. All the code that has been created during this project is open source and is available online, in a Github repository.

In conclusion, the project had satisfied the goals presented in the project proposal, sharing sensor data on an open network, and giving the users the tools to visualize it.

Chapter 7

FUTURE WORK

This project can be improved in two ways, the sensor node and the Android application.

The sensor node could be ameliorated building a more compact and solid prototype, removing the breadboard. This prototype should be built bearing in mind that the node will be mounted outdoor. To be fully compatible with Guifi network, this prototype should be powered using PoE technology.

The Android application showed some issues when requested to handle a lot of data. This issue can be solved downloading a limited set of data (e.g. the most recent). Moreover, the application can be improved adding the possibility to draw the history data.

Finally the project could be disseminated by a web page or conference on sensors.

Bibliography

[Aosong Electronics Co., 2013] Aosong Electronics Co., L. (2013). Digital-output relative humidity & temperature sensor/module dht22. https://cdn.shopify.com/s/files/1/0045/8932/files/DHT22.pdf?100745. [Online; accessed 10-February-2014].

[Arduino, 2014] Arduino (2014). Arduino. http://arduino.cc/. [Online; accessed 8-January-2014].

[Chong and Kumar, 2003] Chong, C.-Y. and Kumar, S. P. (2003). Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256.

[Emartee, 2013] Emartee (2013). Mini sound sensor - emartee.com. http://www.emartee.com/product/42148/Mini%20Sound%20Sensor%20%20Arduino%20Compatible. [Online; accessed 10-February-2014].

[Futurlec, 2013] Futurlec (2013). Mq-135 gas sensor. https://www.futurlec.com/Datasheet/Sensor/MQ-135.pdf. [Online; accessed 12-June-2014].

[Guide, 2014] Guide, A. (2014). Arduino guide. http://arduino.cc/en/Guide/ArduinoYun. [Online; accessed 8-January-2014].

[Instruments, 2013] Instruments, T. (2013). Lm35. www.ti.com/lit/ds/symlink/lm35.pdf. [Online; accessed 8-January-2014].

[Opencities, 2014] Opencities (2014). Opencities. http://opendata.nets.upf.edu. [Online; accessed March-2014].

[santander, 2014] santander, S. (2014). Smart city in santander. http://www.smartsantander.eu/. [Online; accessed 10/02/14].

[smart city, 2014] smart city, A. (2014). Smart city in amsterdam. http://amsterdamsmartcity.com/. [Online; accessed 07/02/14].

Appendix A

PILOT CHARTER

A.1 Pilot Charter

Fellow: Sergio Almendros Diaz
Mentor:
Advisor: Jaume Barcelo

A.1.1 Pilot purpose or justification

The purpose of this pilot is to build a sensor platform that can be attached to guifi nodes to gather and share sensory data.

A.1.2 Measurable pilot objectives and related success criteria

- Gather data about temperature, humidity, light, and noise.
- Share the data as open data.
- Deploy at least two nodes and gather data for at least two weeks.

A.1.3 High-level requirements

- Outdoor enclosure.
- Use open hardware and open software to the possible extent.
- Use standardized interfaces to integrate with other projects.

A.1.4 High-level pilot description

The goal is to use an arduino platform to create a bottom-up broadband wireless sensor networks. As guifi.net has already over 20,000 nodes, the idea is to co-locate the sensory platforms together with the guifi.net nodes and use the guifi.net network to transmit the data. This data should be gathered and shared. Ideally, the pilot should include a presentation interface for the users to visualize the data.

A.1.5 High-level risks

A possible risk is that the prototypes are not rugged enough for outdoor environments. It is also a risk that the prototype is not stable and needs to be reset very often.

A.1.6 Summary milestone schedule

- From 20/09/2013 to 23/09/2013
  - Establish the general idea of the TFG and specifics goals.
- From 23/09/2013 to 11/10/2013
  - Specify the tasks to do and make a planning.
- From 11/10/2013 to 30/10/2013
  - Connect first sensors to the Arduino.
- From 31/10/2013 to 10/01/2014
  - Connect to guifi network and upload data to an open data platform.
- From 10/01/2014 to 01/06/2014
  - Integration of sensors and communication aspects.
  - Install prototypes.
  - Data sharing and visualization.
  - Data analysis and evaluation of the testbed.
- From 02/06/2014 to 30/06/20014
  - Preparation of the final memory.
- From 01/07/2014 to the date of the presentation
  - Make the presentation.

A.1.7 Summary budget

The cost of this pilot will be approximately 4000 €. This quantity is for the scholarship to the student that will develop this pilot, budget for attending a conference or visiting collaborators, and the purchase of the necessary hardware.

Appendix B

PLANNING REPORT

B.1 Planning Report

The following sections explain the tasks that I will do in the course of this project.

B.1.1 Familiarization with the Arduino Yun

In this project I will be working with an arduino Yun, but I never worked before with any type of arduino, so the first task is to start coding different kind of programs. Then I will have to learn how to interact with the Linux in the arduino Yun.

B.1.2 Preliminary testbed

I want to do an easy example to how to connect an arduino with a server running in my computer, what I want to do is establish a bridge between an arduino program and the Linux within the arduino to be able to communicate with a server in my laptop, and send a string with the value returned by a sensor. This is a reduced problem of the real ”bottom-up sensor testbed” because, at the end, in every arduino will be a program that will have to send a message to a server with the data of the sensors attached to it.

B.1.3 Collect Data from sensors

First I will connect a temperature sensor to the arduino YUN, then, I will develop a program to collect the information from it, and send it to a server. When the temperature sensor works, I will do the same process with a humidity, light, and noise sensor.

B.1.4 Install Sentilo

Sentilo (www.sentilo.io) is an open source sensor and actuator platform that I will install in my laptop to act as the server between the sensor network and the interface for the users to visualize the data.

B.1.5 Communication with Sentilo

I will adapt the messages that the arduino send to fit with the Sentilo.

B.1.6 Real deployment

At this moment, the part of the arduino and the server will be done, so I will test the server installing the arduino in real nodes of the guifi network, for example, the node in the Universitat Pompeu Fabra, and any other node that allow me to install it. The arduino will have a temperature, humidity, light, and noise sensor.

B.1.7 Interface

I want to do an interface for any user to understand the meaning of the temperature, humidity, light, and noise values. This interface will be develop for an android mobile application.

B.1.8 Sentilo module

I will contribute to Sentilo and other sensor data brokerage platforms accommodating the sensor testbed deployed in the previous tasks.

B.1.9 Final report

This task has to be done in parallel with all the other ones, and its purpose is document all the work that I will do.

B.1.10 Gantt chart

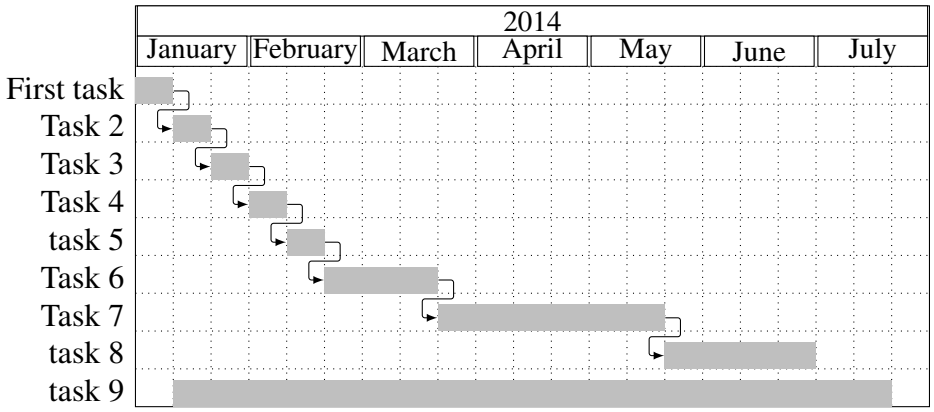


Figure B.1: Gantt Chart

Appendix C

CLASS DIAGRAM

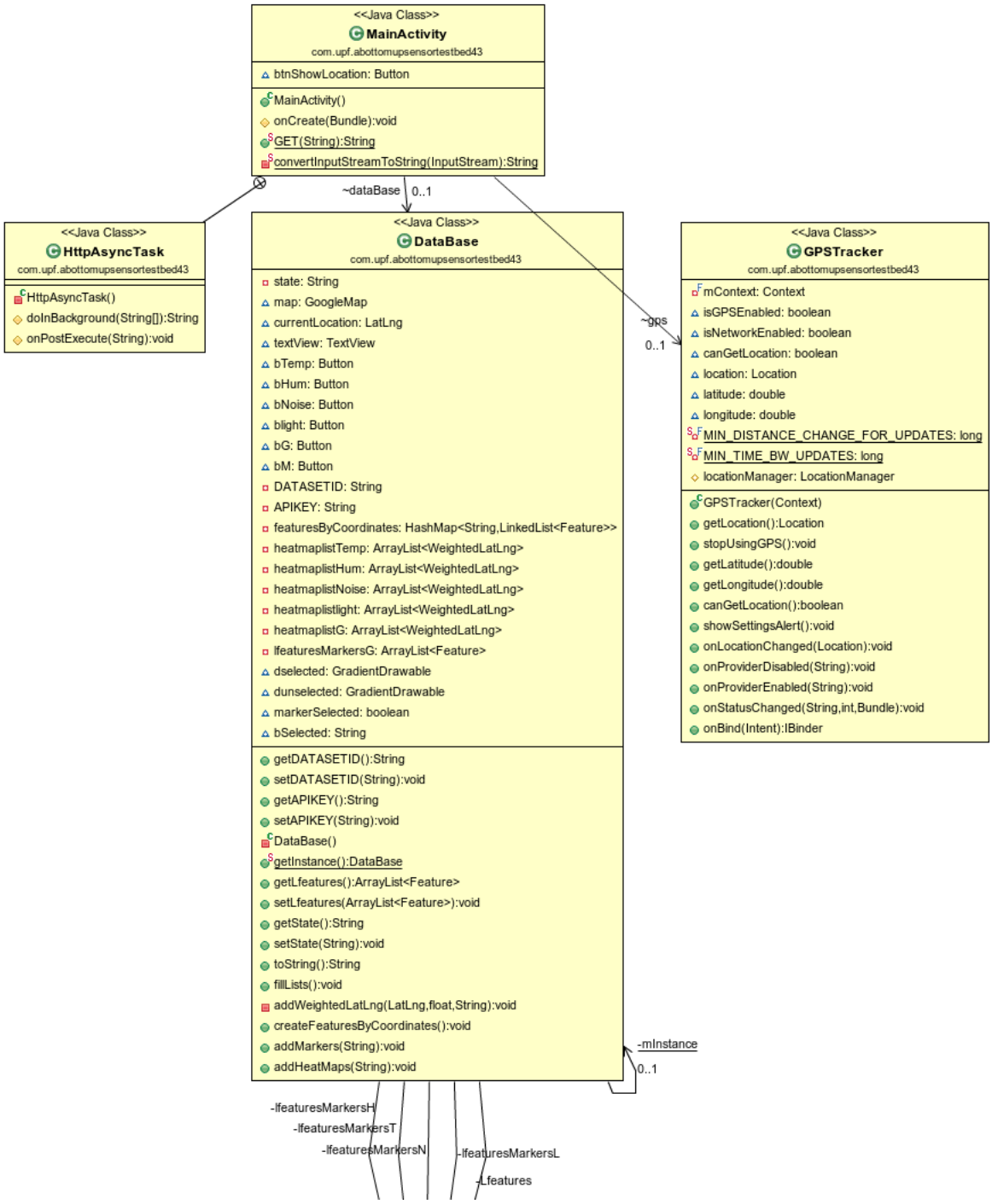


Figure C.1: Class Diagram of the Android App part 1.

Figure C.2: Class Diagram of the Android App part 2.

Appendix D

TESTBED GRAPHICS

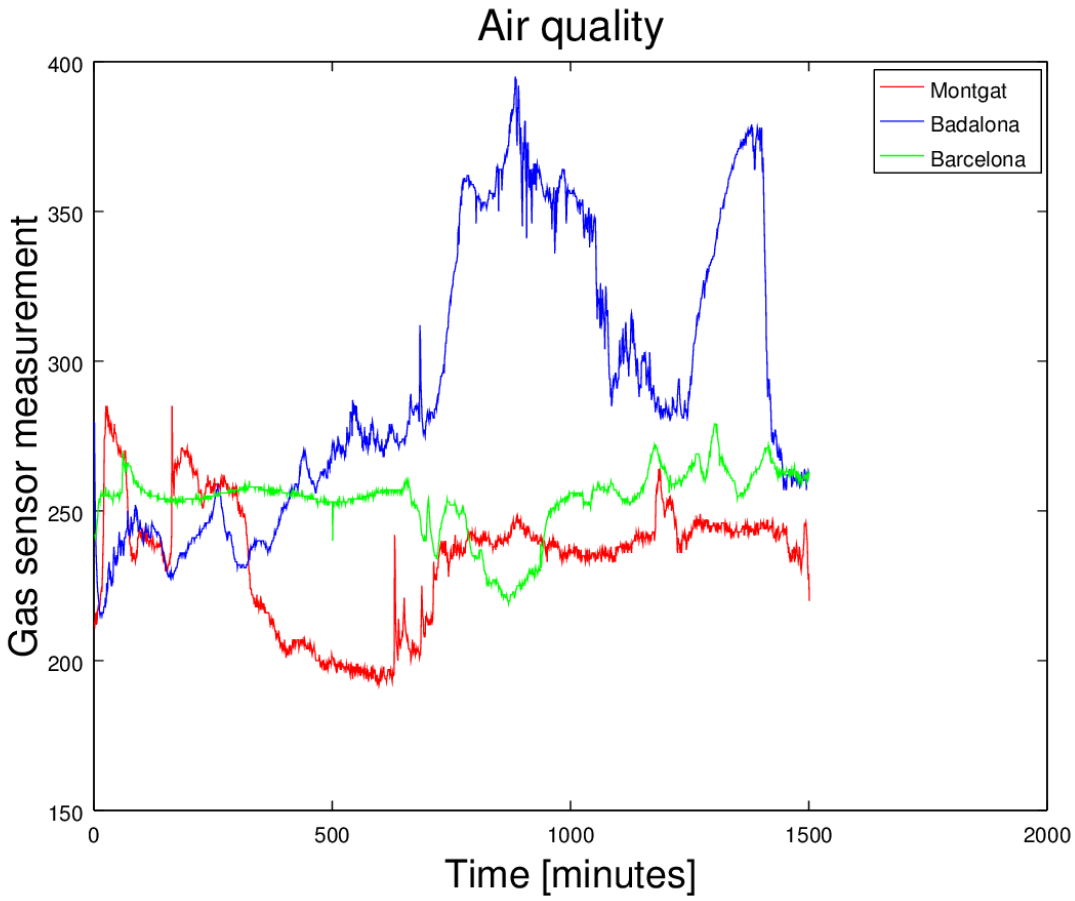


Figure D.1: Complete Graphic Air Quality.

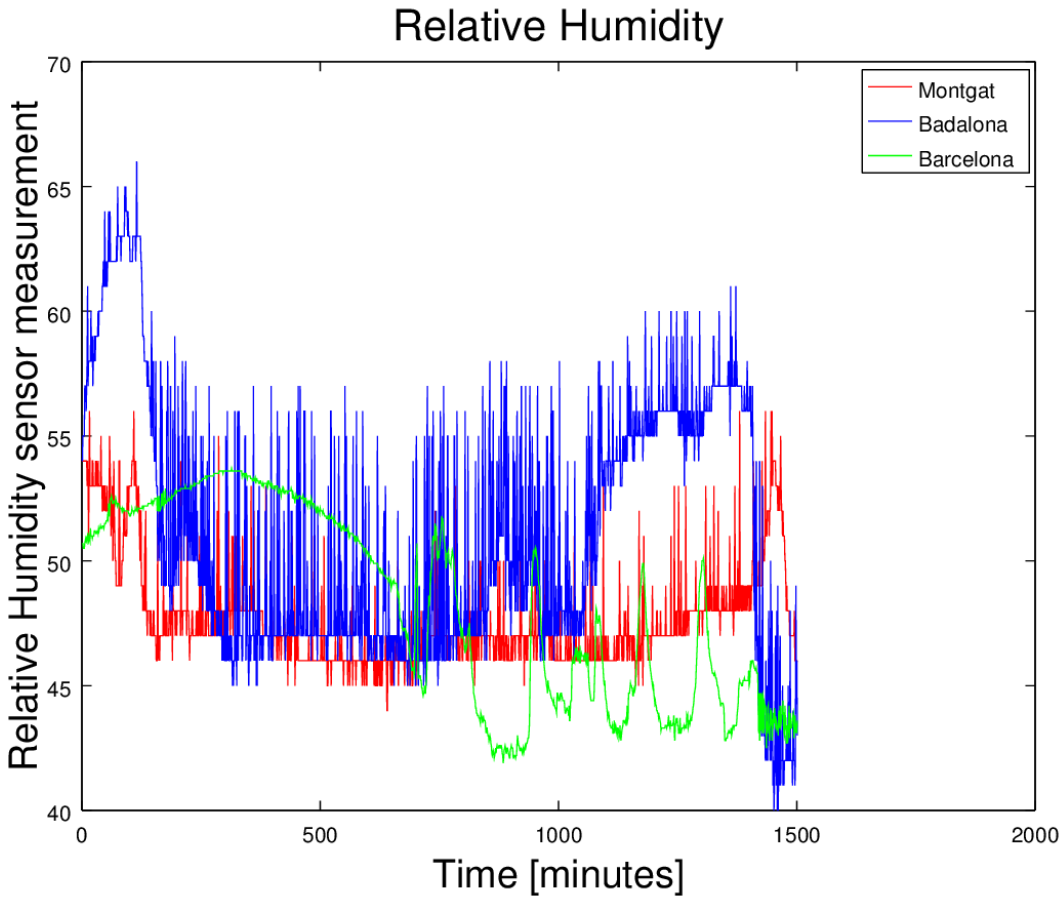


Figure D.2: Complete Graphic Humidity.

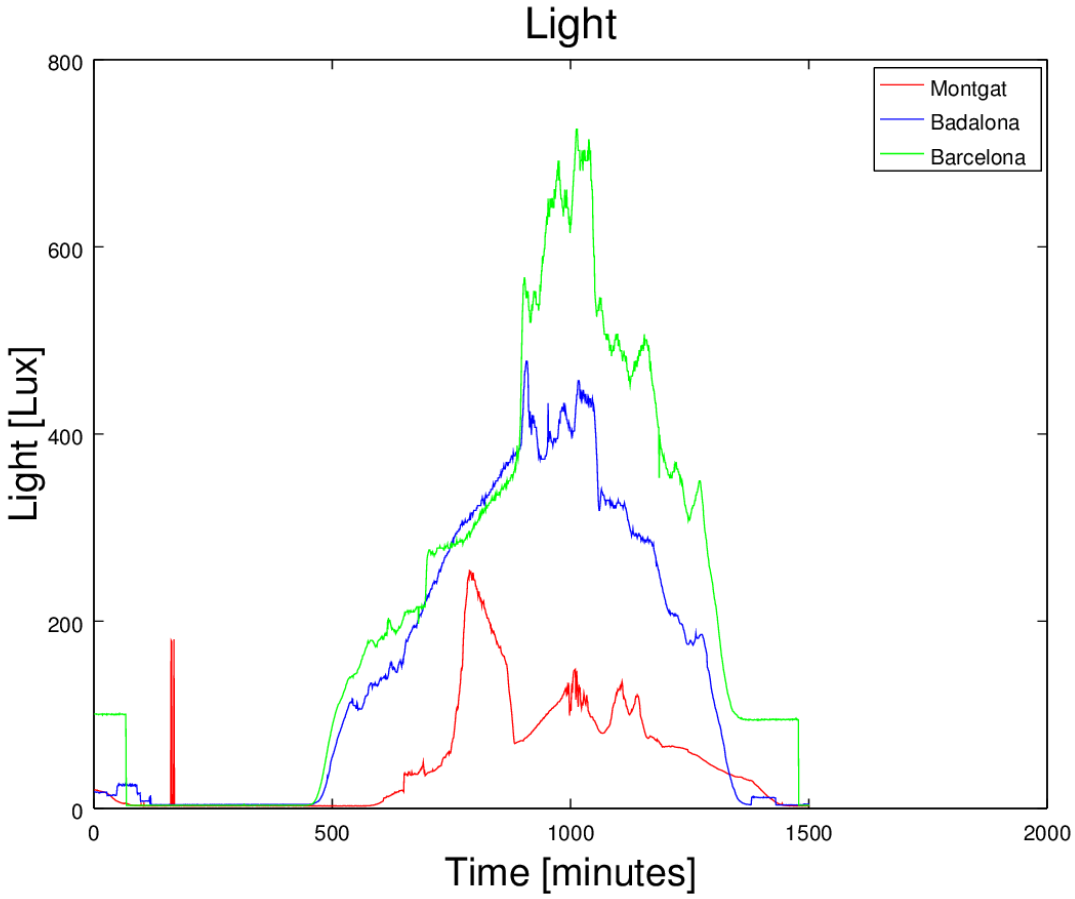


Figure D.3: Complete Graphic Light.

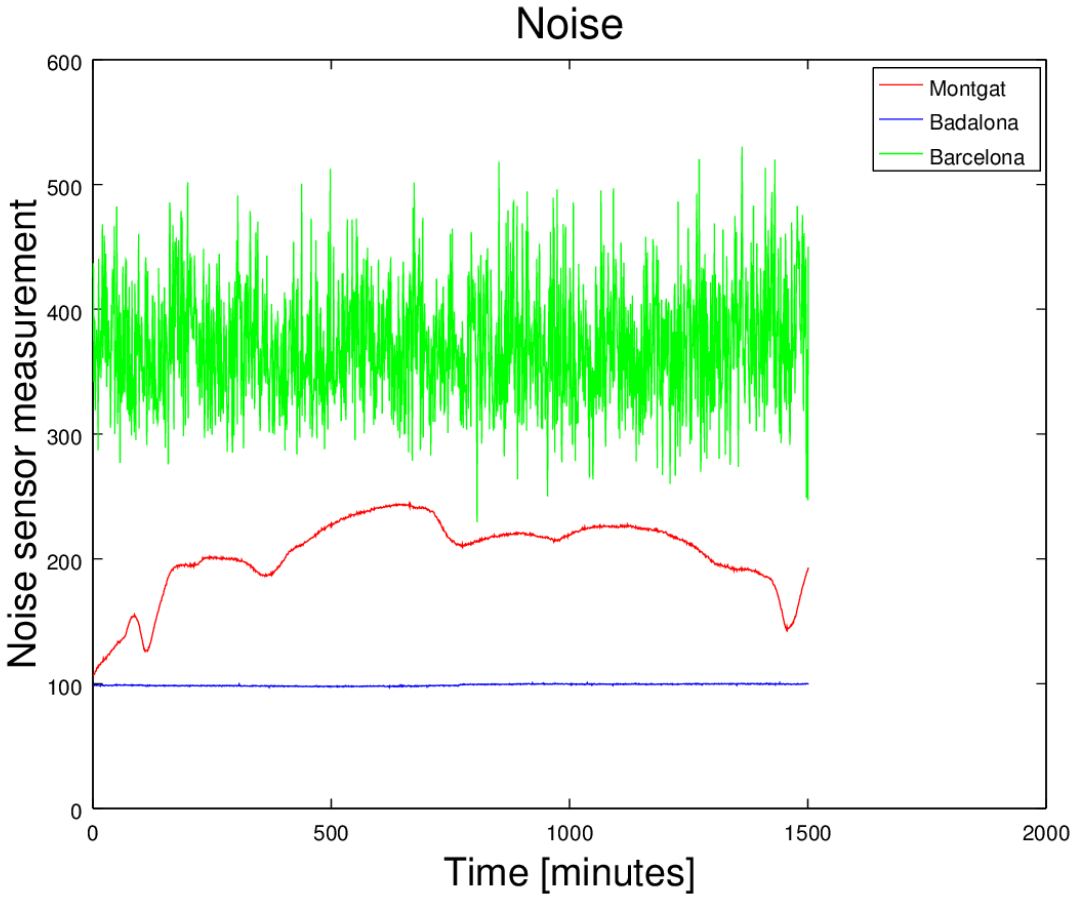


Figure D.4: Complete Graphic Noise.

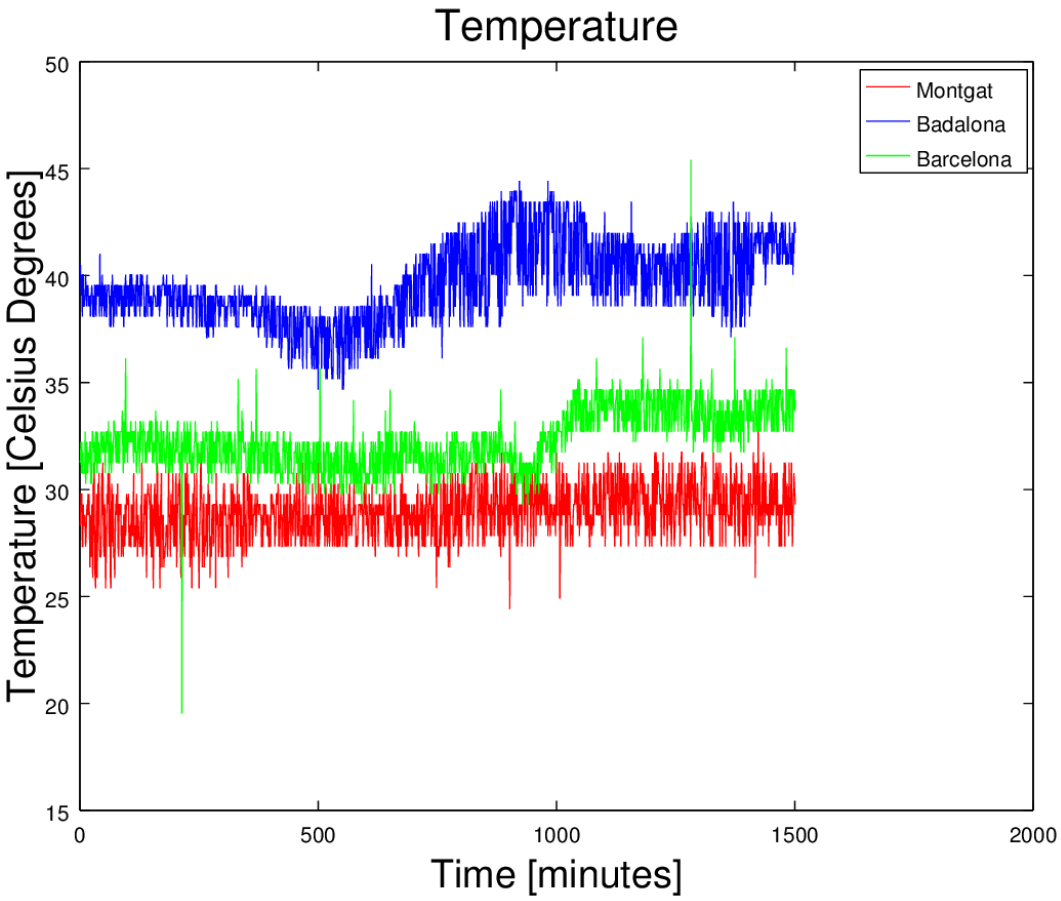


Figure D.5: Complete Graphic Temperature.