

A bottom up sensor testbed

Sergio Almendros Díaz

TFG UPF / YEAR 2014

DIRECTOR/S OF THE TFG:

Jaume Barceló

DEPARTMENT:

Departament de Tecnologies de la Informació i les Comunicacions (DTIC)



Acknowledgments

Abstract

This project aims to deploy sensor node network into the guifi network, an open, free, and neutral network, and the visualization of this data in an intuitive way that helps final users to understand the meaning of these data without any previous scientific knowledge.

The main objective is to give citizens environmental conscience so that they are aware of their surroundings and could act accordingly if anything stands.

This way anybody can act consequently if the sensors detect harmful situations, such as, not going to a park where the gas sensor detects a bad air quality.

From this point, arduino Yun is the best choice to be the main component of the sensor node, and through a series of sensors attached to it and the Internet connection that provides guifi, the node can communicate with a storage platform of sensor data, in this case, opencities. Finally, an android application downloads the sensor data from the servers of opencities and display them on a map.

Resum

Cuando el Abstract esté perfecto, lo traduciré al catalan.

Contents

List of figures	xii
List of tables	xiii
1 INTRODUCTION	1
2 STATE OF THE ART	3
2.1 Introduction	3
2.2 Sensor networks and smart cities	3
2.2.1 Amsterdam smart city	3
2.2.2 Santander smart city	4
2.3 Companies	4
2.3.1 Smartcitizen	4
2.3.2 Libelium	4
2.4 Opendata services	5
2.4.1 Opencities	6
2.4.2 Xively	6
2.4.3 Sentiло	6
2.5 Sensor boards	6
2.5.1 Arduino YUN	6
2.5.2 Raspberry Pi	8
2.5.3 Picoboard	8
3 TECHNOLOGIES	11
3.1 Arduino vs Raspberry PI	11
3.2 Sensors	12
3.2.1 LM35: Temperature	12
3.2.2 Light Dependent Resistor (LDR)	12
3.2.3 Emartee Mini Sound Sensor and Analog Sound Sensor Board Microphone MIC Controller: Noise	14
3.2.4 Aosong DHT22 and DHT11: Humidity	14

3.2.5	MQ135: Gas sensor	14
3.2.6	BreadBoard with all the sensors	17
3.3	Upload sensor data	19
3.3.1	GeoJSON	19
3.4	Community network	19
3.5	Storage Resource Broker	21
3.6	Visualization platform	21
4	BOTTOM UP SENSOR TESTBED	23
4.1	Arduino Development	24
4.1.1	Collect sensor data	24
4.1.2	Communication with opencities	24
4.2	Android app	26
4.2.1	Summary	26
4.2.2	Interface	26
4.2.3	Code	26
5	TESTBED RESULTS	33
5.1	Sensor node	34
5.1.1	Connection to the Internet	34
5.1.2	Install necessary packets	35
5.1.3	Copy the scripts	36
5.1.4	Attach the sensors	36
5.1.5	Arduino Code	37
5.2	Actual Testbed	37
5.2.1	Results	38
6	CONCLUSIONS	41
7	FUTURE WORK	43
	BIBLIOGRAPHY	45
8	APPENDIXES	47
8.1	Pilot Charter	47
8.1.1	Pilot purpose or justification	47
8.1.2	Measurable pilot objectives and related success criteria	47
8.1.3	High-level requirements	47
8.1.4	High-level pilot description	48
8.1.5	High-level risks	48
8.1.6	Summary milestone schedule	48
8.1.7	Summary budget	49

8.2	Planning Report	49
8.2.1	Familiarization with the Arduino Yun	49
8.2.2	Preliminary testbed	49
8.2.3	Collect Data from sensors	49
8.2.4	Install Sentilo	49
8.2.5	Communication with Sentilo	50
8.2.6	Real deployment	50
8.2.7	Interface	50
8.2.8	Sentilo module	50
8.2.9	Final report	50
8.2.10	Gantt chart	50

List of Figures

2.1	Smart Citizen Node	4
2.2	Libelium ehealth	5
2.3	Libelium Wasp mote	5
2.4	Libelium Smart Water	6
2.5	Arduino YUN	7
2.6	Arduino YUN Bridge	7
2.7	Raspberry Pi model B	8
2.8	Picoboard	9
3.1	Sensor boards	12
3.2	LM35 sensor	13
3.3	Temperature Sensor Breadboard	13
3.4	LDR sensor	13
3.5	Light Sensor Breadboard	13
3.6	Mini Sound Sensor	14
3.7	Analog noise sensor	14
3.8	Noise Sensor Breadboard	15
3.9	DHT22 sensor	15
3.10	DHT22 Breadboard	15
3.11	DHT11 sensor	16
3.12	DHT11 Breadboard	16
3.13	MQ135 Air Quality sensor	16
3.14	Gas Sensor Breadboard	16
3.15	Sensor node Prototype with DHT22	17
3.16	Sensor node Prototype with DHT11	18
3.17	GeoJSON message	20
3.18	Guifi sensor map	20
4.1	reportGeneralView	23
4.2	Arduino sketch Flow Chart	25
4.3	Python Script Flow Chart	27
4.4	App Screenshots	28

4.5	Class Diagram of the Android App part 1	29
4.6	Class Diagram of the Android App part 2	30
4.7	Android App Flow Chart	31
5.1	TestBed Prototype	33
5.2	Yun web Password	34
5.3	Yun web Diagnostic	35
5.4	Yun web Configuration	36
5.5	Testbed Map	37
5.6	Graphic Temperature	38
5.7	Graphic Light	38
5.8	Graphic Noise	38
5.9	Graphic Humidity	38
5.10	Graphic Air Quality	39
8.1	Gantt Chart	51

List of Tables

Chapter 1

INTRODUCTION

The development of this project involves two parts, collect sensor data and show it.

A sensor testbed is a sensor network that has the goal to gather sensor data, and test the technologies used as nodes to see if they are the best option.

Bottom-up¹ is, basically, the pattern that we used to build the sensor testbed, where the end users, in this case, guifi.net users, are the ones who have to assemble the sensor nodes and attached them to their guifi nodes to create the sensor network. With the bottom-up model, the data is provide and use by the end users. This project is an easy way to understand the importance of sensor networks and how they can help us to know, for example, if there is low quality air in our city, and do something about it.

As sensor nodes we will use an Arduino[Arduino, 2014] YUN, which allows the user to obtain analog reads from a sensor very easily and, with a Power over Ethernet module, it can be attached to guifi² nodes and send the sensor data to a sensor platform, like opencities[Opencities, 2014].

The project will include an Android application to visualize this data and make it more accessible to other users not involved with guifi.net.

In the following chapters it will be explain the state of sensor networks nowadays (Chapter 2), which technologies we will use (Chapter 3), and how the project has been done (Chapter 4).

Finally there will be the results (Chapter 5) of the testbed, and conclusions (Chapter 6) and future work (Chapter 7).

¹<http://bubforeurope.net>

²<http://guifi.net/>

Chapter 2

STATE OF THE ART

2.1 Introduction

Sensor networks started as a mechanism of defense developed by the military during the Cold War, with acoustic sensors they tried to find Soviet submarines. This search continued at universities, trying to make this sensors smaller, and with the possibility of real-time data[Chong and Kumar, 2003].

Right now, sensors are small enough, and processors with network technology have low energy consumption, which allows us to deploy a test bed without people notice it.

Smart cities are the next step, a city capable of having real-time information, not only about the environment, it can go from the amount of cars that pass 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, in the case that the government build the sensor network, the data should be open to everyone could see it. There are already some sensor networks functioning, some of them are from the government, and, sometimes, there are not that open about their data, but there are also some people who have sensors nodes at home and share the information with everyone.

2.2 Sensor networks and smart cities

In this section we introduce a few projects of sensor networks deployed:

2.2.1 Amsterdam smart city

Amsterdam have a lot of 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 let drivers to know if there are free spots to park, and, in 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, or traffic intensity monitoring.

[santander, 2014]

2.3 Companies

There are some companies that are in the business of sensor networks.

2.3.1 Smartcitizen

Smart Citizen¹ is platform that allows a user to buy a node based on Arduino to monitor the environment as we can see in the figure 2.1, upload this data to their own database to anybody can see it.



Figure 2.1: Smart Citizen Node.

2.3.2 Libelium

This company is an Internet of things platform provider², which supply an open source sensor platform for the Internet of things. They have a variety of products, here are some interesting ones:

¹<http://www.smartcitizen.me/>

²<http://www.libelium.com/>

- e-Health: A sensor shield for Arduino and Raspberry Pi for body monitoring: pulse, oxygen in blood, airflow, body temperature, electrocardiogram, glucometer, galvanic skin response, blood pressure, patient position, and muscle/eletromyography sensor. We can see the shield in the figure 2.2.

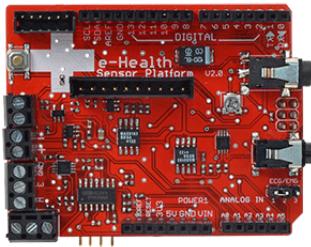


Figure 2.2: Libelium ehealth.

- Wasp mote: A sensor node where we can attach more than 60 sensors, solar powered, and though for adjust into street light posts. We can see the node in the figure 2.3.



Figure 2.3: Libelium Wasp mote.

- Smart Water: Is a wireless sensor platform for water quality monitoring, it also provides real-time data. We can see the node in the figure 2.4.

2.4 Opendata services

The sensor networks are useless if we don't store the data, although we could save it in the device, it would be too expensive to recollect it. That is why we are going to use an opendata service:



Figure 2.4: Libelium Smart Water.

2.4.1 Opencities

Opencities[Opencities, 2014] is a platform to browse, visualize, and download open data from different participants. They offer free space in their server, which is perfect, a very simple API to upload and download data, and a web page to visualize the stored data.

2.4.2 Xively

Xively³ offers an Internet of Things platform as a service, basically it let you stored sensor data, download it, and visualize it in graphics.

2.4.3 Sentilo

Sentilo⁴ is a open source platform for Smart Cities, it allows you to use their own service to store the data, but not many, because their goal is that anyone who wants the sentilo platform will have to installed it in their server, and then use it. It also provides a interface to show the data.

2.5 Sensor boards

In this section we discuss some of the options we could use as node.

2.5.1 Arduino YUN

The Arduino YUN is a microcontroller board with two processors as we can see in the figure 2.5, an ATmega32u4 (Arduino), and an Atheros AR9331 (For a Linux distribution named OpenWrt-Yun). It also have a Ethernet and WiFi module, a

³<https://xively.com>

⁴<http://www.sentilo.io>

USB-A port, a micro-SD card slot, 20 digital input/output pins, 16 MHz crystal oscillator, 16 MHz crystal oscillator, ICSP header, and 3 reset buttons.



Figure 2.5: Arduino YUN.

The great thing about the arduino YUN is that the processor for the arduino sketches can communicate with the Linux processor through the bridge library, which allows you to write python scripts and execute them. In the figure 2.6 we can see it.

A power over ethernet module (PoE) can be attached to the arduino, which is perfect for the guifi network, as their nodes are PoE.

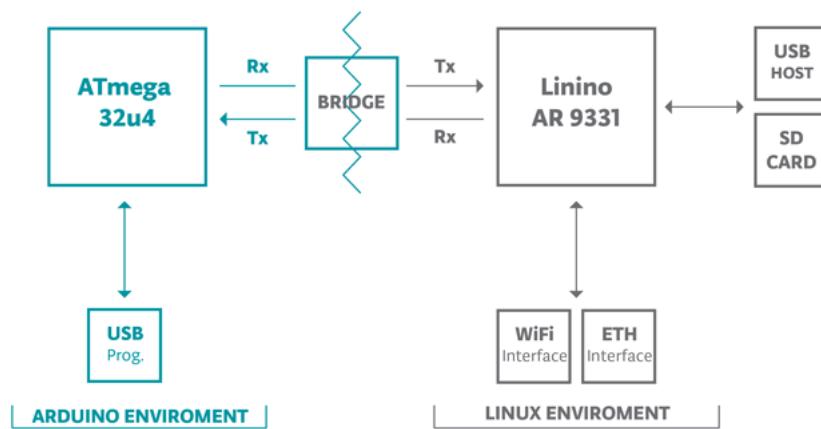


Figure 2.6: Arduino YUN Bridge.

2.5.2 Raspberry Pi

Raspberry Pi⁵ is a single-board computer, they have two models, A and B. The model B is more appropriate for this project because it has an Ethernet controller. It is composed by an HDMI Micro USB, and 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 very easy to attach sensors to it, and it supports Linux environment like raspbian. In the figure 2.7 we can see the raspberry pi B.



Figure 2.7: Raspberry Pi model B.

2.5.3 Picoboard

Puede que cambie Picoboard por BeagleBone

PicoBoard⁶ is a board to interact with the world, it can be programmed by Scratch projects. It is less flexible than the others, it is composed by a button, a light and sound sensor, a slider, and alligator clips which can be used to build custom sensors. We can see this board in the figure 2.8.

⁵<http://www.raspberrypi.org/>

⁶<http://www.picocricket.com/picoboard.html>

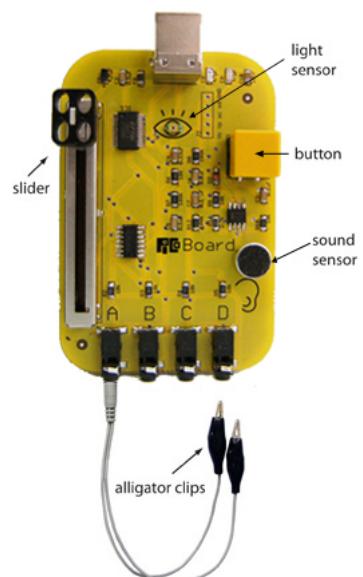


Figure 2.8: Picoboard.

Chapter 3

TECHNOLOGIES

In this chapter we focused in the technologies used to develop this project, which are divided in four parts: arduino, sensors, python, Guifi network and opencities, and Android.

3.1 Arduino vs Raspberry PI

For this project, we narrow down the sensor node possibilities to an Arduino Yun (Yun) or a Raspberry Pi model B (RPiB).

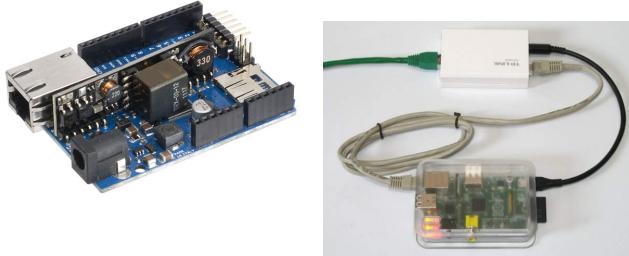
The Yun is a microcontroller while the RPiB is a full computer, which makes it more powerful, but this project does not require too much, so both Yun and RPiB serve.

The two of them had a linux environment, but with the Yun, the normal way to interact is by an arduino sketch. So RPiB will consider that you have some linux knowledge, while the Yun is better for beginners, the arduino IDE provides a variety of programs that help us to start.

The sensor board is essentially the sensor node, so the size is very important, the smaller the better. In this case, the Yun is smaller. On the other hand, the sensor node will be attach on guifi nodes, which are Power over Ethernet (PoE). For the Yun there is the possibility of a PoE module which it is not available yet, but will be. The figure 3.1a shows an arduino ethernet (similar to the Yun) with a PoE module and does not make it bigger, and with the RPiB, the module makes it bigger, as we can see in figure 3.1b.

Another advantage that YUN offers is the integrated WiFi module, since a person who does not have a guifi node can connect to one, and thus have access to opencities, and of course, to stored data.

We will use the Arduino Yun instead of the Raspberry Pi basically because of the size that will become the sensor node, but if, in the future was necessary a more



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

Figure 3.1: Sensor boards

powerful processor, it will be better use a Raspberry Pi.

3.2 Sensors

The goal is to analyze the environment around us, for that purpose, sensors are very useful. It has been decided for the use of environmental low cost 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 transforms a physical measure to an output signal that can be read by another device, such as an arduino.

In this project we will use five sensors that measured temperature, light, noise, humidity, and gas.

To show how the sensors are connected to the arduino YUN I used the program fritzing¹.

3.2.1 LM35: Temperature

LM35 [Figure 3.2] is a sensor to measure temperature, in the figure 3.3 we can see the way to connect it to the arduino. [Instruments, 2013]

3.2.2 Light Dependent Resistor (LDR)

LDR [Figure 3.4] is a light sensor, in the figure 3.5 we can see the way to connect it to the arduino.

¹<http://fritzing.org/>

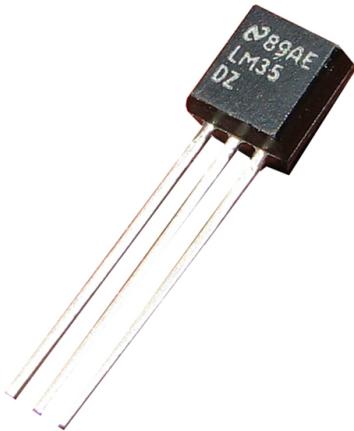


Figure 3.2: LM35 sensor

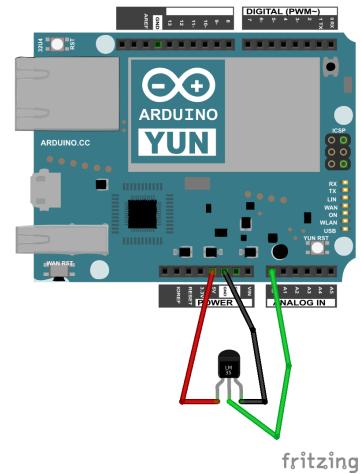


Figure 3.3: Temperature Sensor Breadboard

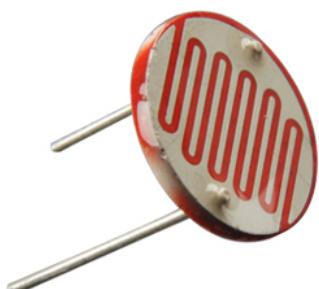


Figure 3.4: LDR sensor

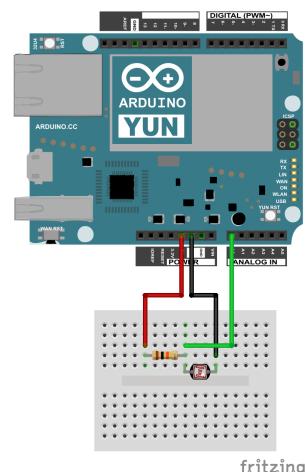


Figure 3.5: Light Sensor Breadboard

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

This two sensors are used to measured noise levels [Figure 3.6] and [Figure 3.7]. The code to read the noise values is the same for both. In the figure 3.8 we can see the way to connect them to the arduino. [Emartee, 2013]



Figure 3.6: Mini Sound Sensor

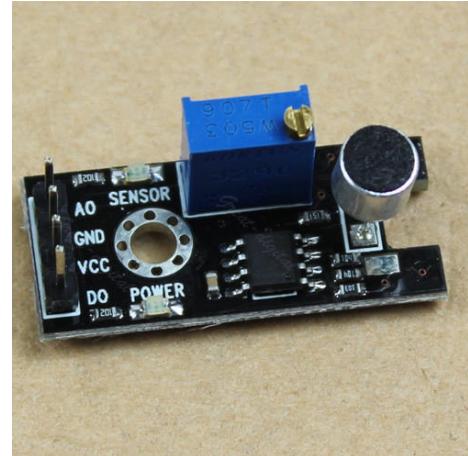


Figure 3.7: Analog noise sensor

3.2.4 Aosong DHT22 and DHT11: Humidity

DHT22 [Figure 3.9] and DHT11 [Figure 3.11] are humidity and temperature sensors, although we will only use the humidity measure. The output is digital, and to read it, we use an external library². The arduino and the humidity sensor will be connected as shown in the figure 3.10.

At the time I started the testbed, there was only 1 DHT22 sensor, so I had to use two DHT11 sensor, which change the breadboard a little bit as shown in the figure 3.12. [Aosong Electronics Co., 2013]

3.2.5 MQ135: Gas sensor

This is a gas sensor [Figure 3.13], and we will use it to measure air quality. This sensor does not have a figure in fritzing, so I used a gas sensor that has the same output, and in the figure 3.14 we can see 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.. ??.

²<https://github.com/adafruit/DHT-sensor-library>

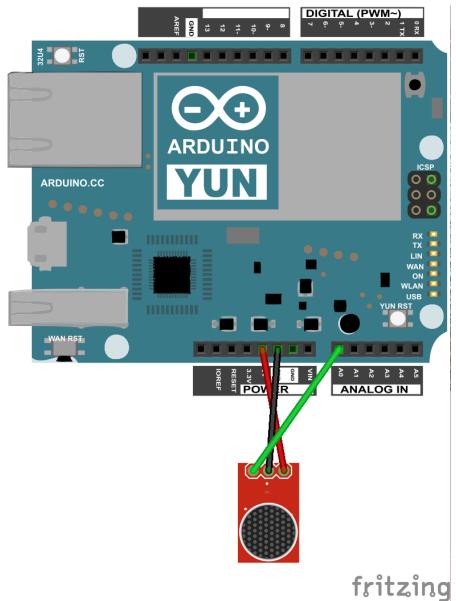


Figure 3.8: Noise Sensor Breadboard.

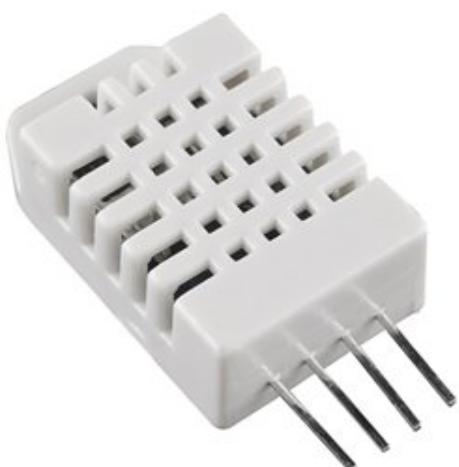


Figure 3.9: DHT22 sensor

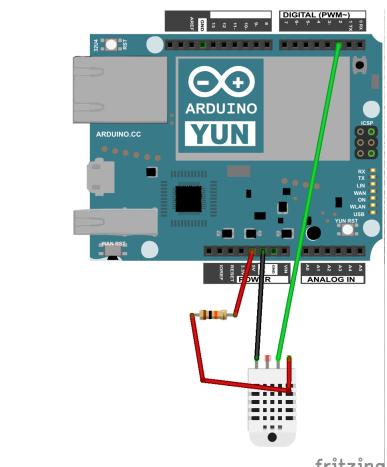


Figure 3.10: DHT22 Breadboard

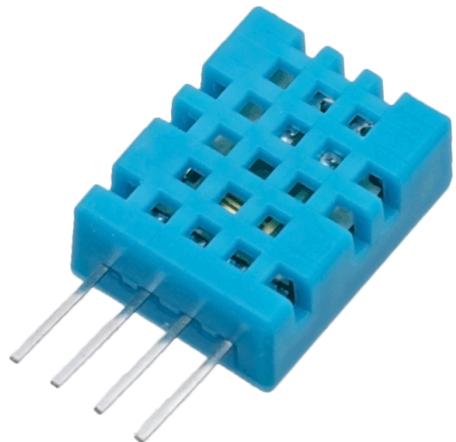


Figure 3.11: DHT11 sensor

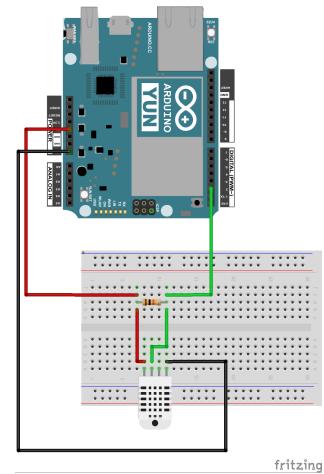


Figure 3.12: DHT11 Breadboard



Figure 3.13: MQ135 Air Quality sensor

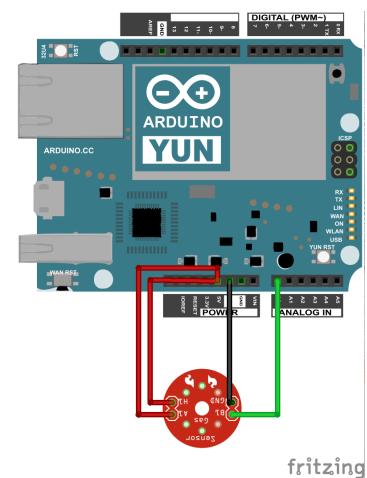


Figure 3.14: Gas Sensor Breadboard

3.2.6 BreadBoard with all the sensors

In the figure 3.15 we can see the final prototype with the DHT22 sensor, and in the figure 3.16 with the DHT11 sensor.

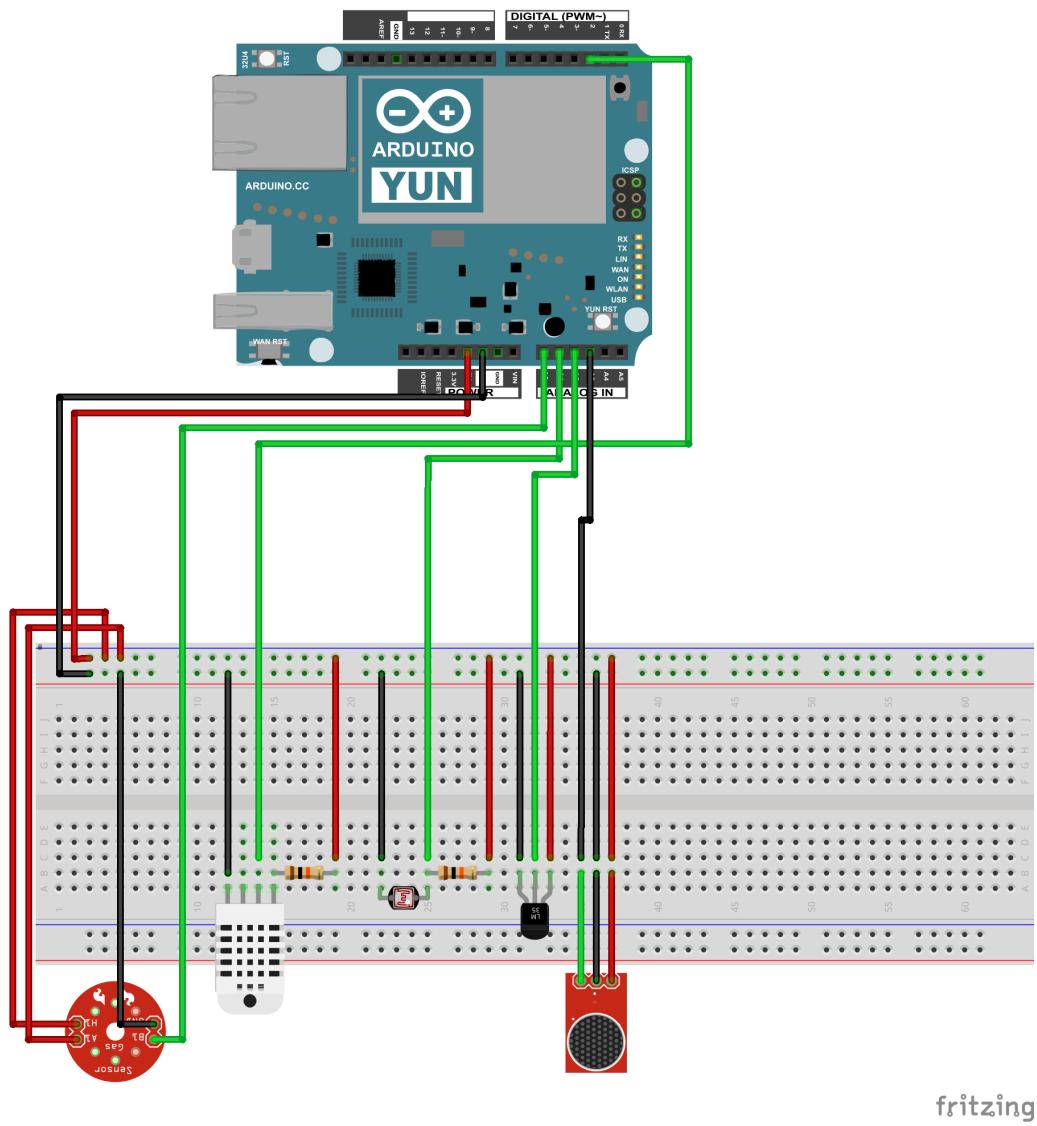


Figure 3.15: Sensor node Prototype with DHT22.

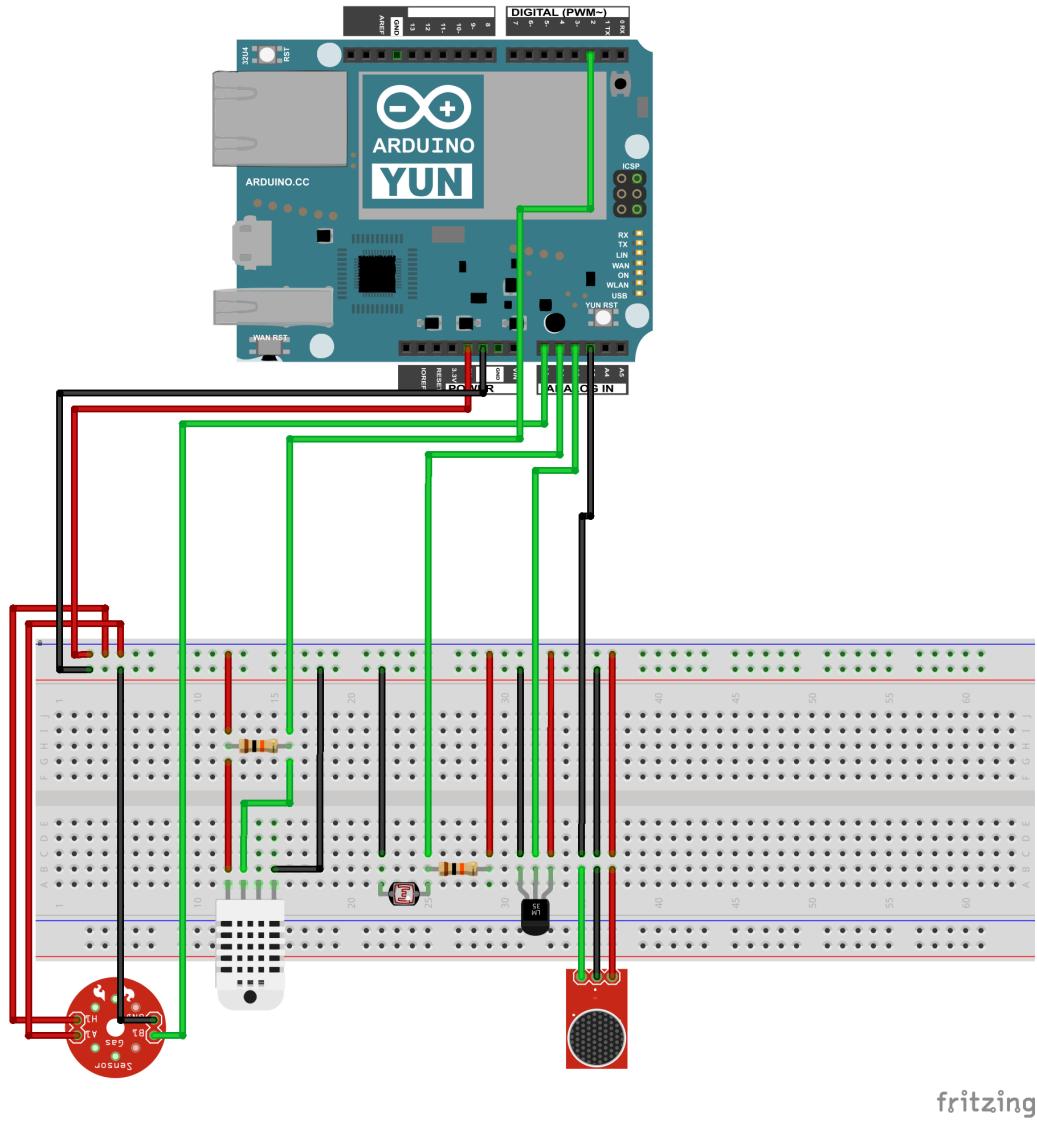


Figure 3.16: Sensor node Prototype with DHT11.

3.3 Upload sensor data

A main part of the project is upload the data from the sensors to a platform so that everyone can access them.

The problem is that is needed to upload five sensor values, consequently the GeoJSON message (Explained in section 3.3.1) takes too much memory, so the Arduino sketch can not handle it.

For this type of operations is possible to run a script in the linux environment through the bridge library^{2.6}.

This being the path that is followed, there are several options for developing this script, for example, programmed in Java, C / C ++, python, etc..

We preferred to use python because of the following reasons: fast and perfect for prototyping programming, the code is shorter and therefore, easier to understand, especially if you want to improve in the future.

3.3.1 GeoJSON

A GeoJSON³ is a format for encoding a variety of geographic data structures. The GeoJSON that we use is a collection of features, every feature contains a geometry object, in our case, a “point” with the longitud and latitud of the sensor node, and some properties required: ID, name, datasetID, datasetName, address, description, timestamp, value of the sensor, and unit.

In the figure 3.3.1 it can be see an example of a GeoJSON message that is used in this project.

3.4 Community network

Guifi⁴ is the network where the arduino's will be deployed, and the one providing the access to Opencities through the Internet.

Guifi is a network where the people that are interested put their effort in creating an infrastructure which provides acces to the Internet at a fair price.

Because of their philosophy of participation, Guifi is the perfect network for the deployment of this sensor nodes. Also their networks is large enough to become a useful sensor network, as we can see in the figure 3.18.

³<http://geojson.org/>

⁴<https://www.guifi.net/>

```
{  
  "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 3.17: GeoJSON message.



Figure 3.18: Guifi sensor map.

3.5 Storage Resource Broker

Opencities[Opencities, 2014] is the opendata services that we have chosen, the reasons opencities has been chosen are:

- The developers are at UPF, this helps because the platform can be tested, and because they can help solve problems and make these part of the project more easy.
- Easy API to upload and download the data.

The role of Opencities is to be the intermediary between data creators and those who want to see them. In Opencities, the sensor data is uploaded into a dataset. Every user have an unique API key, and can have one or more datasets. To distinguish, the datasets have a unique ID.

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 is used a map to display the data. This can be done on a web page, in a mobile application, tablet application, etc.

Since the goal of the application is that a user consult it for an small period of time, the best option is a mobile application. Additionally, almost everybody has an smart phone.

From this step forward it has to be chosen in which mobile operationg system it should be developed the application. There are three options, iOS, Android, or Windows Phone. There is another option that is to developed the application in html5, javascript, and CSS, and then compiled it with phonegap, and that will return the application for the three operating system. But that was too much of a challenge, so it was decided to code an application directly to one operating system.

Having decided that the application it has to be for only one operating system, it is more logical to chose the operating system with more market share, to reach as many people as possible.

Android is a mobile operating system from Google, it runs on smartphones, and applications are programmed in Java. Java is a computer programming language that is object-oriented.

This application it will be tested on a Sony Xperia Z1, with an Android 4.4.2.

Chapter 4

BOTTOM UP SENSOR TESTBED

This chapter focused in the process that has been followed to complete the project, which has two main parts, the software to recollect and send the data and the Android application to show it.

To make the project easier to understand an image has been made (Figure 4.1).

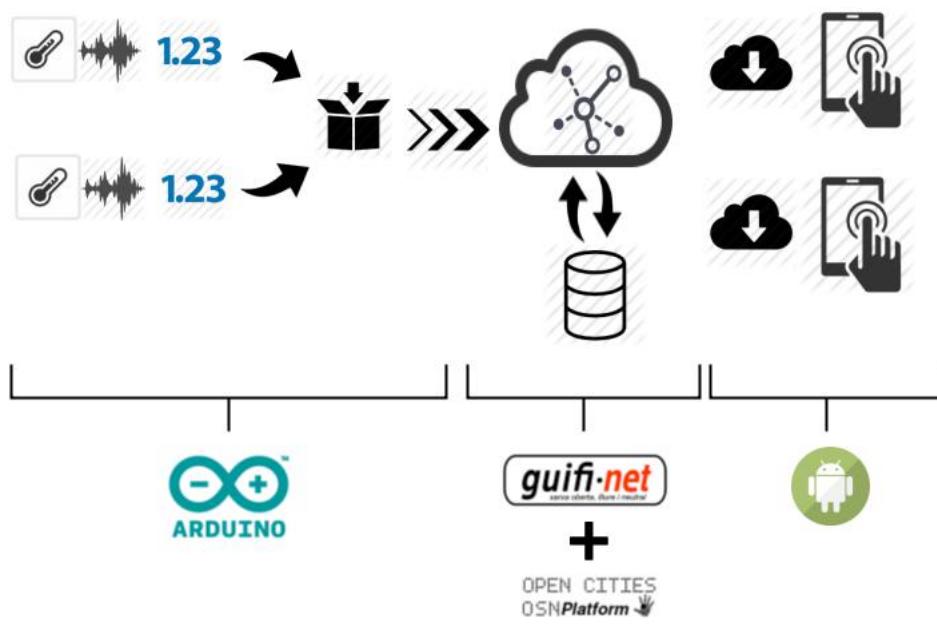


Figure 4.1: reportGeneralView.

4.1 Arduino Development

We will need two scripts, 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 to opencities is too big. That is why we use a python script called by the arduino sketch.

The arduino sketch is responsible for collecting the data, write it down in a logData file, and call the python script with the collected values and an unique ID. Finally the python script have to create a GeoJSON and send it to opencities.

4.1.1 Collect sensor data

To collect almost all the data the sketch does not need to include any library, except for the humidity sensors (DHT22 and DHT11) which need an external library¹.

To read and write into the logData file we need the FileIO library², and to call the python script we need the Process library³.

This sketch is code 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, and the loop function. The loop will call three functions: readSensors, readFile, and executePythonScript.

- **readSensors:** It call 5 different functions to read every sensor, the goal of doing a separate function is to make the code more clear.
- **readFile:** This function log 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 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

First of all, it is needed to create an account in opencities, and a dataset to store the sensor data. This process gives all the information needed to upload and download data.

¹<https://github.com/adafruit/DHT-sensor-library>

²<http://arduino.cc/en/Reference/YunFileIOConstructor>

³<http://arduino.cc/en/Reference/YunProcessConstructor>

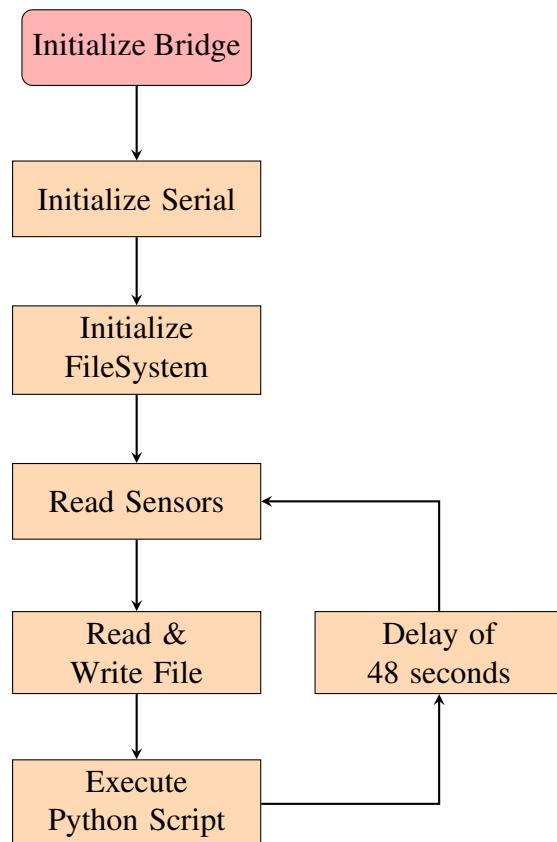


Figure 4.2: Arduino sketch Flow Chart.

The communication with opencities is done by a python script. This are the packages needed into the linux environment of the arduino: distribute, python-openssl, geojson, geopy, and httplib2.

Code explanation

With all this libraries we can communicate with opencities and store the sensor data recollected by the arduino. The 4.3 figure explain how the python script works.

4.2 Android app

4.2.1 Summary

To make easy to see the results of the testbed, I developed an android application. The application shows the data of the sensors in two ways, with markers that show the actual environmental value in that point, and also with heatmap points, the larger the value, the more intense the red will be. This can be seen in figure ??.

4.2.2 Interface

The application interface is a unique map view where the user can zoom to a limit, go to their location, and use the top buttons. 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 of the temperature, humidity,... and the unit. In the figure ?? we can see the app with the Marker button checked, and in the figure ?? without.

4.2.3 Code

First of all, to create this application I have used the Google Maps Android API v2⁴ for the map view, and the Google Maps Android API utility library⁵ for the heatmaps.

This application has the next classes:

- MainActivity: Is the controller of the whole application.

⁴<https://developers.google.com/maps/documentation/android/>

⁵<http://googlemaps.github.io/android-maps-utils/>

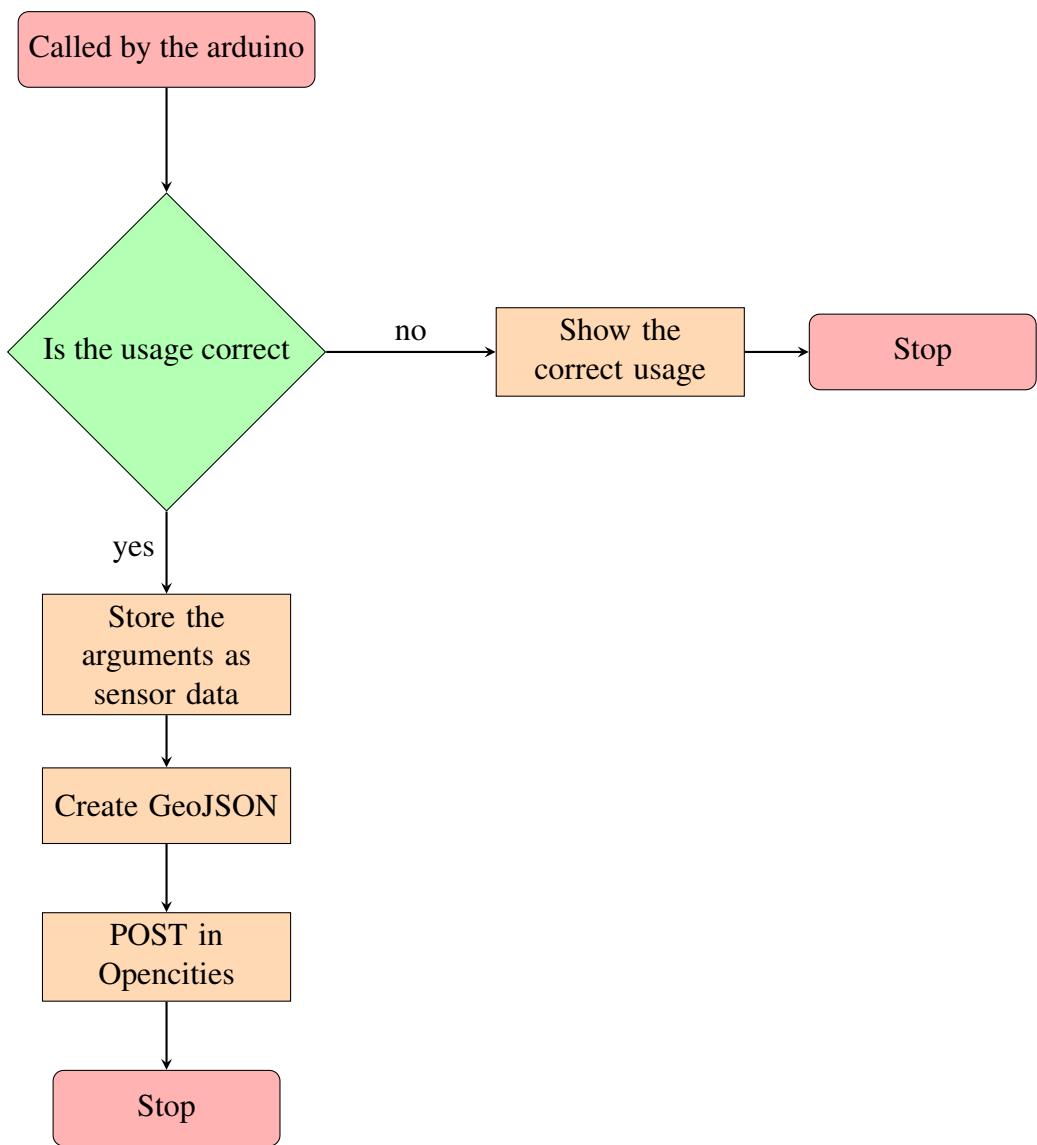


Figure 4.3: Python Script Flow Chart.

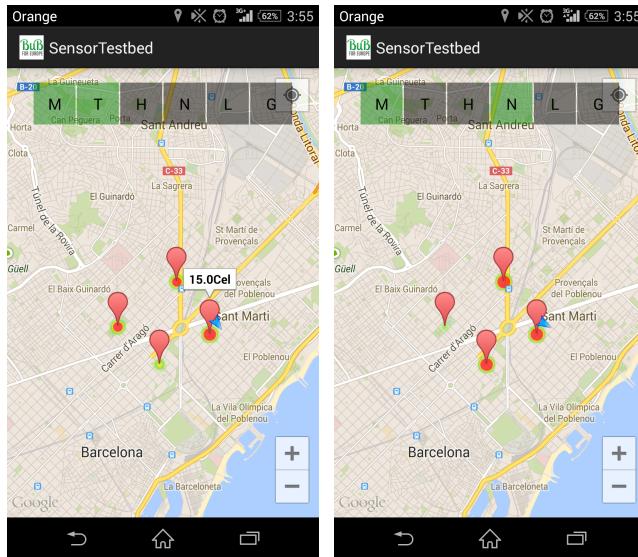


Figure 4.4: App Screenshots

- GPSTracker: Is a class to get the current location of the user.
- Feature, Geometry, and Properties: This are the classes where it will stored the data from the parsed JSON from opendata.
- DataBase: This is a singleton class where all the variables are stored, because is more easy to access from different classes.

In the figure 4.5 and 4.6 are the class diagram:

The explanation of the code is in the following flow chart 4.7:

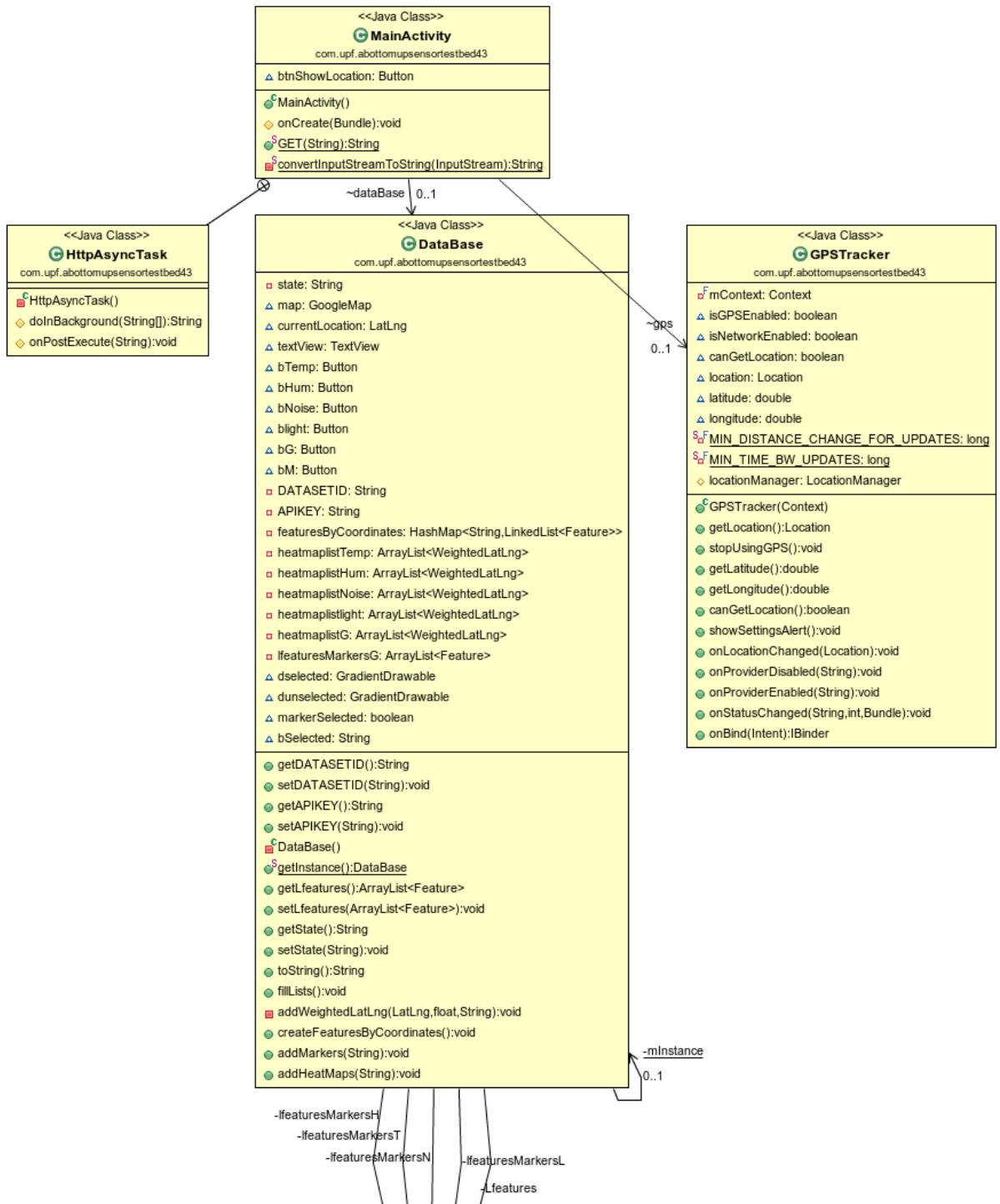
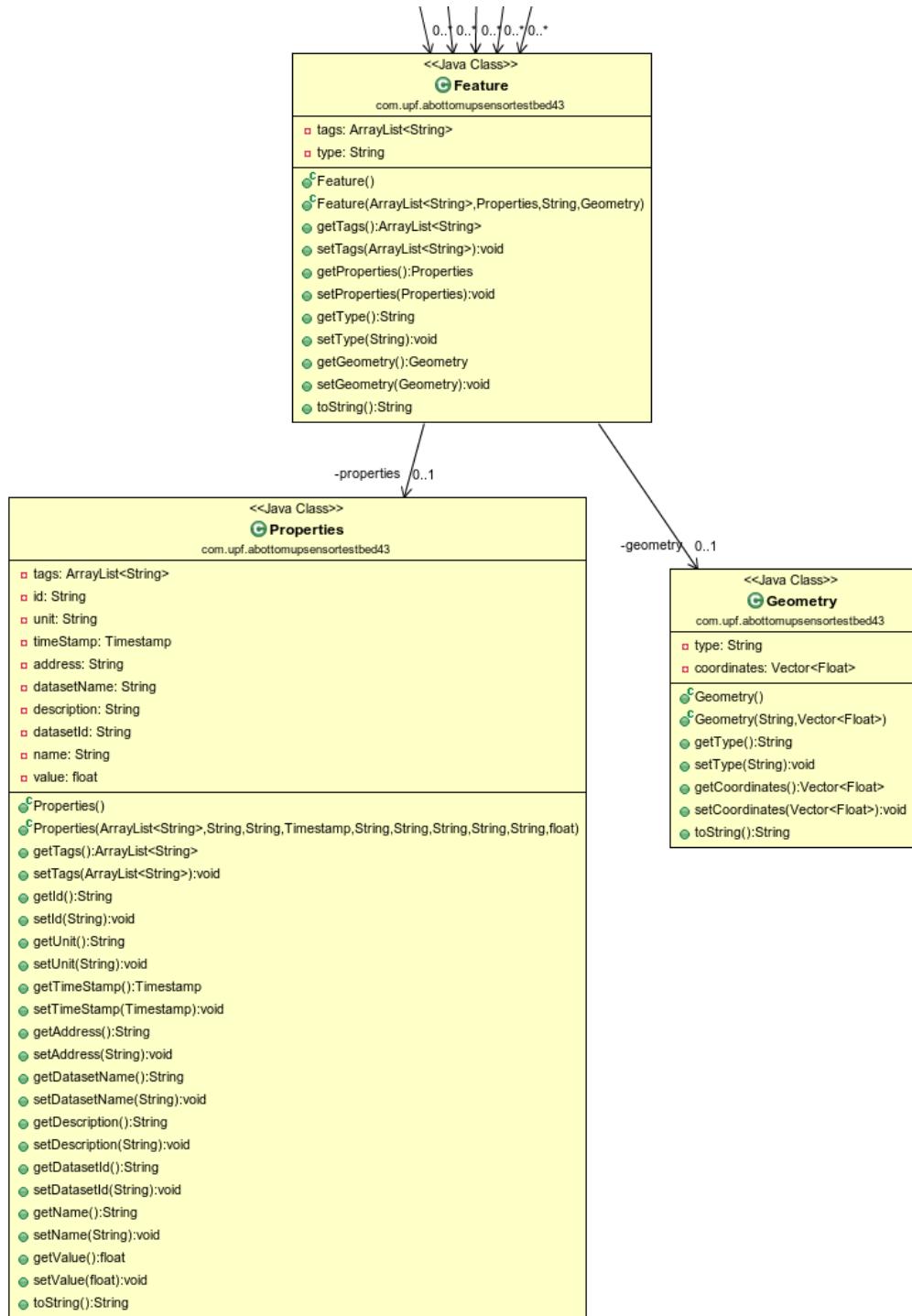


Figure 4.5: Class Diagram of the Android App part 1.



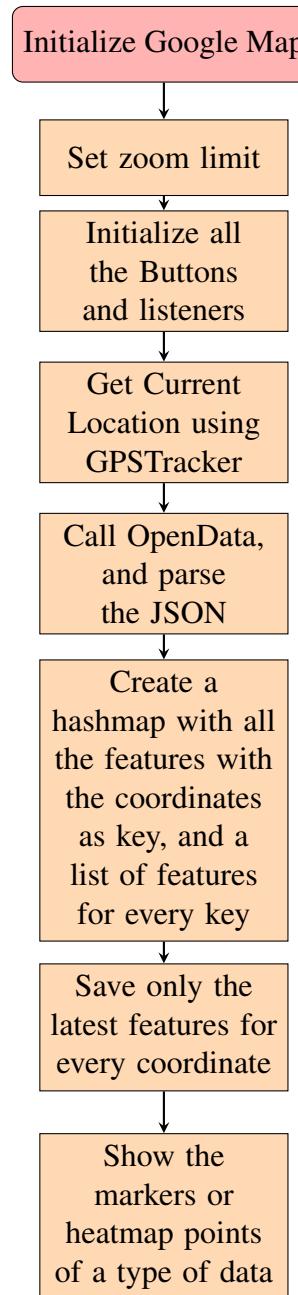


Figure 4.7: Android App Flow Chart.

Chapter 5

TESTBED RESULTS

In this chapter I will explained the procedure I followed to make this testbed, and an explanation of the results.

In the figure 5.1 there is a photograph of the prototype I will use 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).

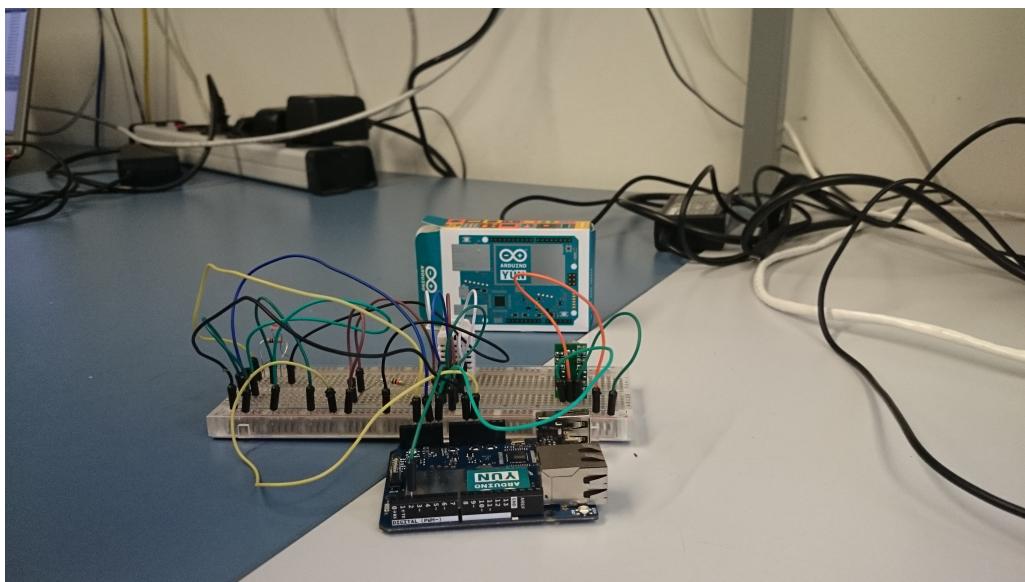


Figure 5.1: TestBed Prototype.

5.1 Sensor node

Now, I will show the process to configure the node, part of this process is taken of the website of arduino¹:

5.1.1 Connection to the Internet

First of all we have to provide of Internet connection to the arduino.

5.1.1.1 Through Ethernet

This is the fastest way to provide of Internet connection, the arduino will behave the same way as a computer, automatically will have an IP address.

5.1.1.2 Through WiFi

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

1. First we power the arduino YUN.
2. The arduino will create his own WiFi network (ArduinoYun-XXXXXXXXXXXX), and with a computer we connect to it.
3. When we are connected to the YUN network, we go to a web browser and go to `http://arduino.local` or `192.168.240.1`. We have to put the password, which is “arduino”. As we can see in the figure 5.2.

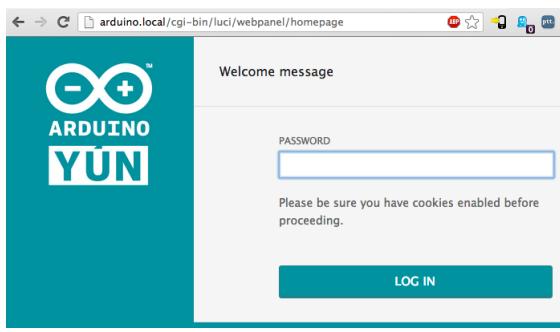


Figure 5.2: Yun web Password.

4. The next page will be an information page, click on the configure button 5.3.

¹<http://arduino.cc/en/Guide/ArduinoYun>

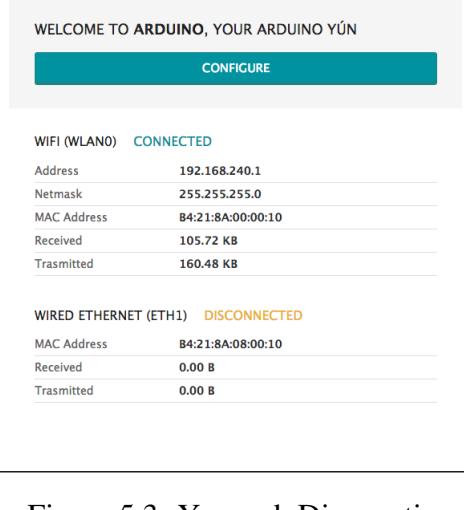


Figure 5.3: Yun web Diagnostic.

5. We will give a unique name to the Yun, and the network we want to connect 5.4.
6. We press the configure & restart button.
7. Finally we have to connect to same network as the Yun is connected.

5.1.2 Install necessary packets

The arduino will run an arduino script and a python script, for the first one we do not have to install anything, but for python we will have to install some packets.

To install any packet we first have to connect to the arduino (The password will be the one we entered when we configured the onboard WiFi or “arduino”):

```
ssh root@X.Y.Z.W
```

Now we can install all the necessary packets:

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

Figure 5.4: Yun web Configuration.

5.1.3 Copy the scripts

First of all we have to create the some directories, so once we make the ssh command, go to “/mnt/sda1”, make the following comands:

```
mkdir arduino
cd arduino
mkdir www
```

We have to copy the python script “main.py” into the SD-Card. We have two ways to do this, put the SD-Card into a computer an save the files in there, or we can copy the files into the arduino through the network with the following comand:

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

5.1.4 Attach the sensors

Now that we have the python step done, we have to attach the sensors to the arduino Yun, to do that look at figure 3.15 or figure 3.16.

5.1.4.1 Prototype

At the end of this project I used a proto shield instead of a breadboard to create a real prototype.

5.1.5 Arduino Code

To upload an arduino sketch to the yun we have to use the following IDE: arduino 1.5.5. There are two ways to upload an sketch, through a USB cable connected to the arduino, or through the Internet, if we are in the same network as the arduino, it will appear in the IDE.

5.2 Actual Testbed

We have three arduino, so we have to mount the sensor node and put them in three different places in free space but being sure it will not get wet.

When we have decided when we are going to put them, we have to introduce manually the location and the unique ID into the python script. We can do that by entering into the arduino by secure shell as we did earlier, go to the folder where the “main.py” is, and modified the following line by using “nano”:

```
self.address = 'Sagrada Familia, Carrer de Mallorca, Barcelona'
```

After a day of collecting data, we will get the arduino’s back, take the three logData files, and show the data into some graphics.

In the figure 5.5 there are the nodes deployed, the location try to be a little bit different to get distinc values.

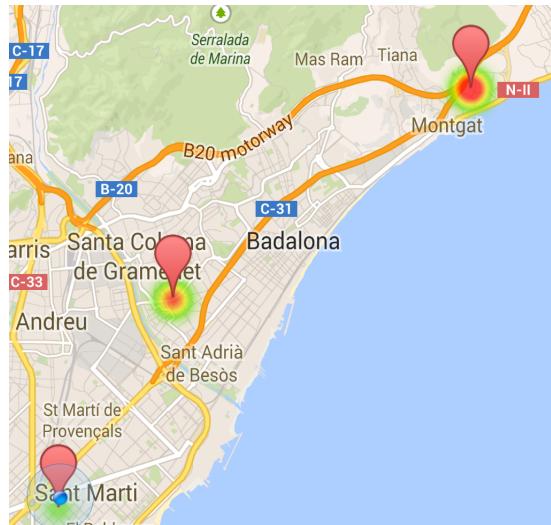


Figure 5.5: Testbed Map.

5.2.1 Results

In the next figures: 5.6, 5.7, 5.8, 5.9, 5.10, there are some graphics of the testbed. It can be seen that the node in Montgat, near the sea, the humidity is bigger, on the other hand, the noise is lower than the one in Glorias.

This graphics had been done with an octave script that analyze the logData file in every sensor node.

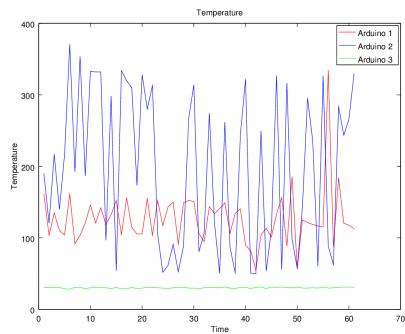


Figure 5.6: Graphic Temperature

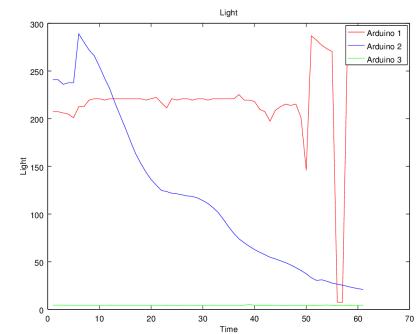


Figure 5.7: Graphic Light

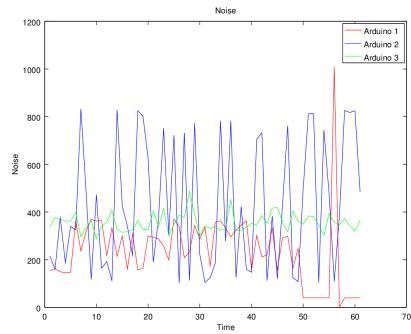


Figure 5.8: Graphic Noise

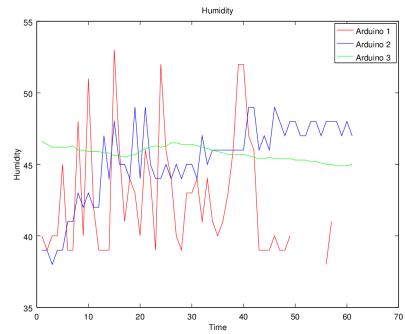


Figure 5.9: Graphic Humidity

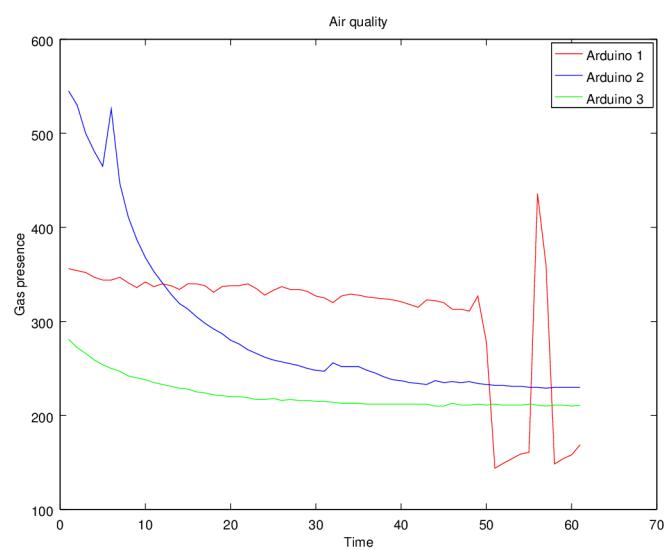


Figure 5.10: Graphic Air Quality.

Chapter 6

CONCLUSIONS

During this project we analyzed, developed, and deployed sensor nodes into an open network, the initials goals of the pilot.

It is clear, therefore, that to deploy a sensor network, like the one explained in the previous chapter, is necessary a bit of knowledge about sensors, arduino, and linux.

The testbed shows that the nodes worked perfectly, so, in terms of scalability, the problem could be in opencities or in the Android App. Opencities proved work perfectly, but the Android App came out with a problem, it downloads all the data from opencities, so it takes a little bit of time to be ready.

In conclusion, the project satisfies the goals presented at the start, which are share sensor data on an open network, and let the users visualize it.

Chapter 7

FUTURE WORK

This project have three ways of improvement:

1. **Sensor node:** Make a final prototype trying to reduce the size to make it the smaller the better, and attach the power over ethernet module.
2. **Opencities:** It is a platform that can improved in many ways, starting with the way to create a dataset, which can be confusing, to the presentation of the data.
3. **Android App:** It showed some problems when there is a lot of data to download, so it should be improved to download only the data that is going to show. Also it could show the data of the whole day and let the user see the changes.

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].
- [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].

Chapter 8

APPENDIXES

8.1 Pilot Charter

Fellow: Sergio Almendros Diaz

Mentor:

Advisor: Jaume Barcelo

8.1.1 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.

8.1.2 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.

8.1.3 High-level requirements

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

8.1.4 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.

8.1.5 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.

8.1.6 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.

8.1.7 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.

8.2 Planning Report

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

8.2.1 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.

8.2.2 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 reduce 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.

8.2.3 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.

8.2.4 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.

8.2.5 Communication with Sentilo

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

8.2.6 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.

8.2.7 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.

8.2.8 Sentilo module

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

8.2.9 Final report

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

8.2.10 Gantt chart

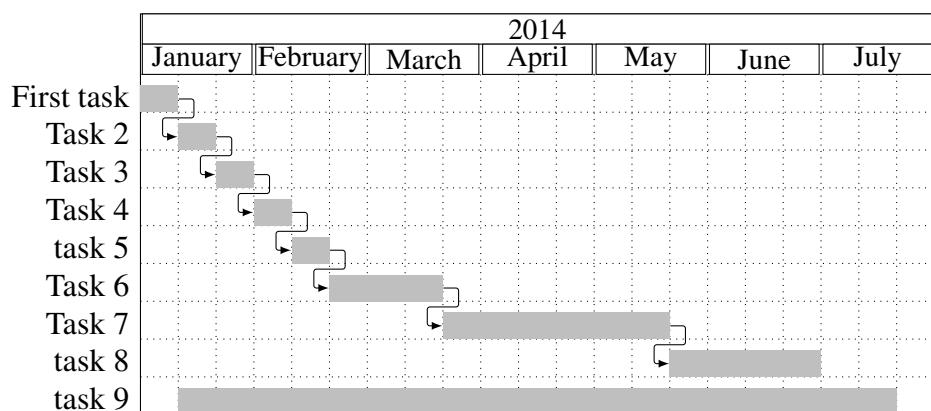


Figure 8.1: Gantt Chart

