uc3m | Universidad Carlos III de Madrid

University Degree in Audiovisual System Engineering
2018-2019

Bachelor Thesis

"IOT Technologies Research and Smart Agriculture Prototype"

Jesús Gento Ribas

Supervisor: Almudena Lindoso Muñoz
Leganés, 10th June

TÍTTLE: IOT Technologies Research and Smart Agriculture Prototype

AUTHOR: Jesús Gento Ribas

SUPERVISOR: Almudena Lindoso Muñoz

The defense session of the present work was set to 01/07/2019 and was judged under the following tribunal:

PRESIDENT: Jorge Pleite Guerra

SECRETARY: Antonio Rodríguez Hidalgo

VOCAL: Javier López Santiago

The score obtained was of:

PRESIDENT: **VOCAL:** **SECRETARY:**

ABSTRACT

The present Final Project offers sensors-based solutions according to the cheapening of mini-computers and in line with the recent spreading of IOT (Internet Of Things) technologies. The main application field covered is Smart Agriculture, where sophisticated systems that sense environmental conditions and trigger output devices, usually with object on industrial control purposes, are being studied and even implanted nowadays. For this reason, consequently, a prototype has been designed. Due to a modern approach to computing and technology that demands different and more intuitive user-interactions, IOT wireless options were also considered, as well as voice compatibility. A control alternative via MQTT (Message Queuing Telemetry Transport) protocol was in addition proposed, on using the developed Android app. Finally, this whole project pretends to offer the results of a generic research of the issue, whose conclusions may be only partially exposed in the prototype. Therefore, it is open to the exploitation of further functionalities and may also be adapted to a different economical sector.

Key words:
Internet Of Things (IOT), Agriculture Engineering, Smart Devices, Control Engineering

RESUMEN

El presente Trabajo Final sugiere soluciones basadas en sensores, de acuerdo con el abaratamiento de las mini-computadoras y en línea con la reciente expansión de las tecnologías IOT (Internet De Las Cosas). Su principal campo de aplicación es la Agricultura Inteligente, en el que sistemas sofisticados que perciben las condiciones ambientales y activan equipos de salida, de modo habitual bajo el propósito de ofrecer un control industrial, están ya siendo estudiados e incluso implementados, razón por la que, consecuentemente, se ha diseñado un prototipo. Debido al enfoque moderno hacia la informática y la tecnología, el cual demanda más y diferentes interacciones con el usuario, se han considerado también opciones IOT inalámbricas y de control por voz. Se propone igualmente una alternativa de control mediante el uso del protocolo MQTT (Message Queuing Telemetry Transport) y la aplicación Android desarrollada. Finalmente, la totalidad de este proyecto pretende ofrecer los resultados de una investigación global sobre la materia, cuyas conclusiones pueden estar tan sólo parcialmente reflejadas en el prototipo. Por consiguiente, se encuentra abierto a la integración de futuras funcionalidades así como de a ser adaptado a cualquier otro sector económico.

Palabras clave:
Internet De Las Cosas (IOT), Ingeniería Agrícola, Dispositivos Inteligentes, Ingeniería de Control

ACKNOWLEDGMENT

Expressing my absolute gratitude to those who have contributed to my academic and personal training would not be enough. I would like to thank my supervisor Almudena Lindoso Muñoz, for allowing me to be independent, for her non-impositive suggestions and for understanding the scope of the present Final Project.

In second term, my whole family deserves a cheerful hug. This Project is deeply dedicated to my mother, for teaching me to learn, for her infinite support and education and for explaining me *life*; to Sara, for helping me on communicating and for providing a different view to our ideas; and to my father, for introducing me to the complex idea of optimization.

Finally, nobody has been more important to me than Sonia, for her quantum love, for understanding wave signals together and for teaching me programming.
The following Final Project is yours.

INDEX

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

IOT	Internet Of Things
MQTT	Message Queuing Telemetry Transport
LED	Light-Emitting Diode
PC	Personal Computer
WIFI	Wireless Ethernet Compatibility Alliance
RTVE	Radiotelevisión Española
RENFE	Red Nacional de los Ferrocarriles Españoles
SMS	Short Message Service
API	Application Programming Interface
GPIO	General-Purpose Input/Output
I2C	Inter-Integrated Circuit
USB	Universal Serial Bus
HDMI	High-Definition Multimedia Interface
GB	Gigabyte
RAM	Random Access Memory
OS	Operating System
IDE	Integrated Development Environment
PIR	Passive Infrared Sensor
EC	Electroconductivity
TDS	Total Dissolved Solids
VCC	Voltage Common Collector
IR	Infrared
ISR	Interrupt Service Routine
MAIN	Main function
LCD	Liquid-Crystal Display

ADC	Analog-to-Digital Converter
SSH	Secure Shell
VNC	Virtual Network Computing
IP	Internet Protocol
DNS	Domain Name System
URL	Uniform Resource Locator
SSL	Secure Socket Layer
IFTTT	If This Then That
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
LAN	Local Area Network
GPL	General Public License
SD	Secure Digital
QOS	Quality Of Service
UC3M	University Carlos III de Madrid

CHAPTER 1:

INTRODUCTION

1.1 Introduction

Internet Of Things (IOT) is a new telecommunication field that is currently growing extraordinarily fast. Its practical purpose is to have control of input devices so that output devices show a reaction, which may happen automatically. For instance, when a presence sensor is activated an IOT system may be able to lock some door or send a notification to a very far place in seconds.

For this purpose, telematics, electrical and computing engineering need to co-work in order to fulfil the requirements for a particular project. This project analyses the global situation, exposes some of the main problems encountered and proposes their IOT-based solutions.

According to tendency information found at *Statista.com* [1], it is also clear an exponential increase in the number of devices related to IOT, as *Figure 1* [1] shows, which obviously means there is a new global market generated over this idea. Furthermore, it may be inferred that investment on applications related to this field would still be very present in the near future.

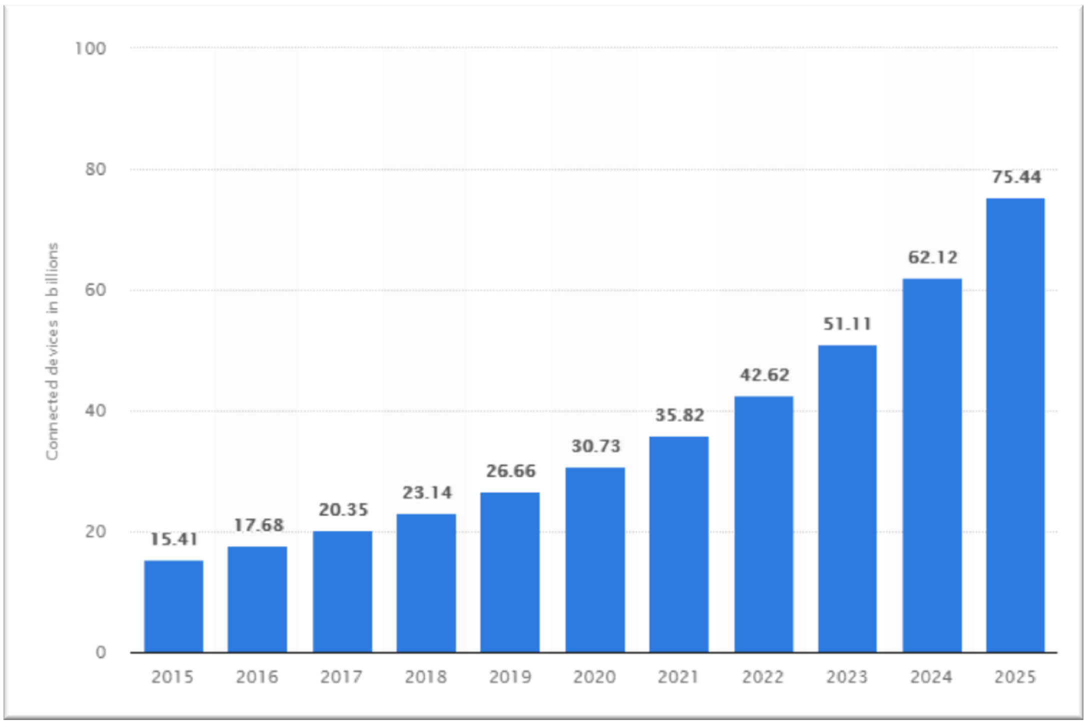


Figure 1. IOT Increasing Tendency

For an *Internet-Of-Things* system to work it is needed both the comprehension of its theoretical fundamentals and the knowledge of several techniques, all which will have a profuse impact on the following years not only for industrial interests but also for personal uses.

On this recent approach it is also essential to explain the performance of different kind of sensors and how the broadcast of their information is configured so that this travels via Internet as safe as possible.

On synthesis, this document shows the importance of the interconnection of technologies, while describing the main methods available for any IOT system. Under a deep understanding of its main functionalities and the sharing of information, new devices are each day available for our modern-society engineering.

1.2 Target Fields

IOT may even now cover many of the every-day issues and it is also increasing its presence in different types of industries. Figure 2 [2] illustrates that there is a general tendency to create connected systems, where information is shared among devices and their states are changes continuously. It is also possible that a register stores activity data in a local or cloud database.

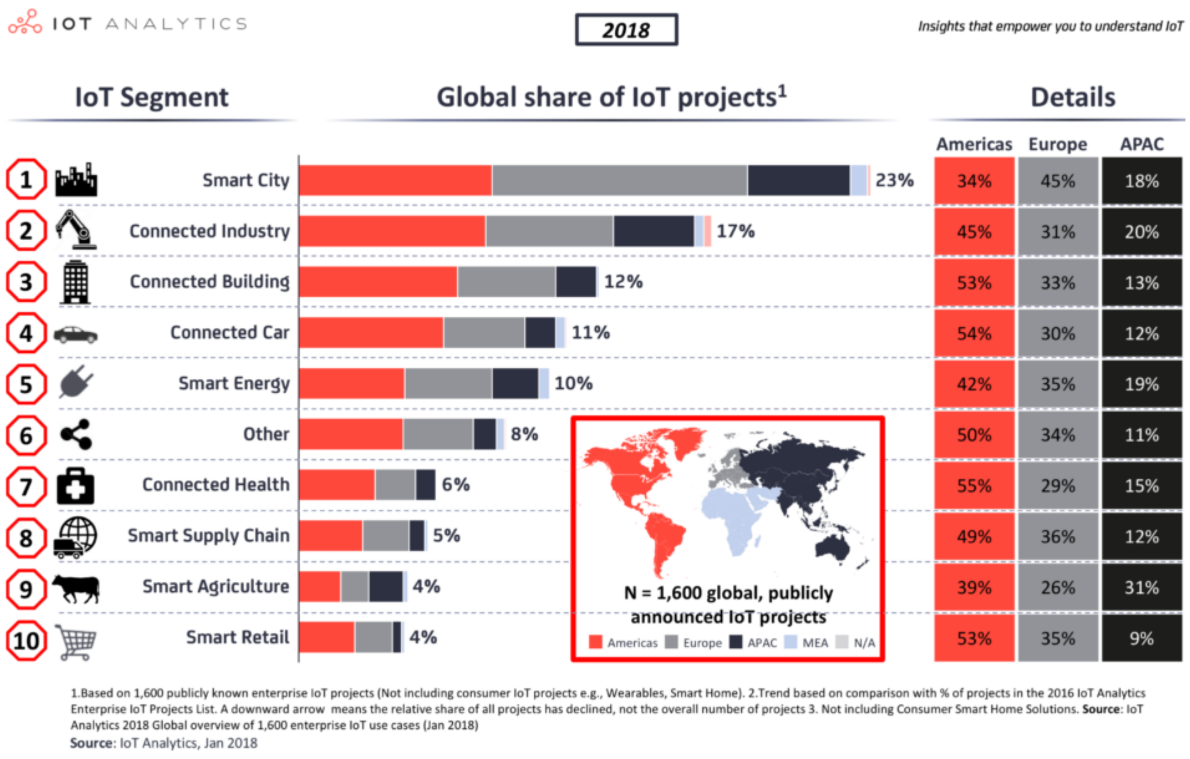


Figure 2. IOT Global Dispersion

In a general view, there is an evident dominance of this kind of industries on more-developed areas, where *APC* (Asia-Pacific), followed by *MEA* (Middle East and Africa), contributes much less to global statistics. It is also remarkable the *unbalanced distribution* of sectors over the globe, especially on focus over the differences between America and Europe.

Among the previous information it could be highlighted, for this interest, the *Connected Industry* and *Building*, where *environmental conditions* are sensor-controlled, *Connected Health*, as in the sense of automatically notifying a message in case of danger, and *Smart Agriculture*, which tries to use IOT implementations so that productivity of nourishment is improved, speed up or increased.

Finally, despite it seems to be ruling the top, *Smart City* might be understood as a generalization of some of the previous sectors' technologies. The only difference is that it targets just on the caring of a bigger installation and takes care on resources or on its self-regulation.

Therefore, a general view is offered so that the exposed prototype is correctly understood under its sector and socio-economic environment.

1.3 Motivation and Requirements of the Prototype

In short, *Smart Agriculture* refers to several procedures and techniques that check the plants' state and environmental characteristics, in order to ensure they grow and are kept in the best possible conditions -usually, also in the fewer time-, by means of sensors and automated processes.

The present work is, therefore, inspired under the conviction that industrial processes could get benefit of using more refined control systems and, particularly, that there is an upward tendency of growing plants under an enclosed climate where organic matter, chemical products and electrical resources needs to be optimized, all which requires a specific tracing based on indicators throughout the whole life of a plant.

Although detailed information about those indicators may be offered later, for now it could be assumed that soil, climate and water pH conditions will be considered. There are several indicators of plants' health to be periodically measured and, at least, some clear conditions any agriculture system should follow.

First of all, it is necessary to note that there are multiple ways to grow plants, which may be classified attending to two criteria and are, at least (see next page):

- **Type of substratum**:
  - **Soil**: the ancient technique. The soil is more tolerant to pH changes, as it acts as a low pass filter to the roots absorption
  - **Hydroponics**: the soil is replaced by water. As the roots are directly exposed to the fluid, where the nutrients are dissolved, the plant can get more benefit of them. However, any sudden pH error may lead to catastrophic results.
  - **Aeroponics**: the substratum is air. The proximity of the roots is continuously sprayed with water and nutrients and high moisture are required. As well as pH, humidity level needs to be highly regulated.
- **Protection and light source**:
  - **Exterior**: The crop is not protected to unsuitable climate or fauna threats. The Sun supplies its light needs.
  - **In-door**: A full environmental conditions' control may be set, so artificial light, usually powered by LED technology, and air circulation should also be integrated.

Each kind of cultivation, of course, will lead to a different and specific control system. However, the present prototype is more oriented to cover the care of soil and hydroponic systems, where there is a non-air substratum that needs periodical supervision.

Up to this point it is evident the need to sense the environment. The actions applied may, therefore, vary from just a simple automated irrigation to also controlling a more complex air circuit with industrial fans. Moreover, *Smart Agriculture* uses data *acquisition* and processing, so that, on the one hand, information is registered and, on the other, actions are translated to remote output devices.

In an economical view, *Smart Agriculture sector* still represents the less proportion of the IOT global for Europe, according to the reference already exposed in *Figure 2*, but "the market value of smart agriculture worldwide is forecasted to reach around 26.76 billion U.S. dollars by 2020" [2], which also justifies the interest on this particular application field.

The challenge is to create interconnected systems that take advantage of the new technologies; connected all around the globe through remote control, which may show low or middle latency in actions; and, if possible, that show graphical register and security monitoring. All of the previous requirements are fulfilled in the present proposal of the prototype, so that sensing, human supervision and network technologies could work together.

This Final Project offers a *Precision Farming* Prototype, on using different kind of sensors, Internet technologies and, finally, Alexa and/or wireless communication, depending on the desired service to cover. Lastly, an Android app was developed in order to offer an alternative way to control devices, via MQTT (Message Queuing Telemetry Transport) [3] protocol.

It is remarkable to state that part of its *Regulatory Framework* is still not clear, as it is usual on emerging markets. Nevertheless, the aspects of the using of registered devices such as the Raspberry Pi [3] and Arduino [4] with commercial interests is considered afterwards, as well as the risks and indications about transmitting private data through the Internet.

However, although the designed system is intended for a specific field, it must be noted that it can be used without modification in any other IOT field. Therefore, a *Smart Agriculture* prototype is presented, but its IOT designs and modules may apply as well to a different area, with few or none modification.

The document is organized as follows: Chapter 2 offers the global flow of any *Smart Agriculture* system and nowadays perspective, while Chapters 3 and 4 detail the sensors used in the prototype and its output system, respectively. Chapter 5 shows the IOT functionalities implemented in this prototype, while Chapter 6 would, in essence, explain the requirements for the developed Android app to work.
Chapters 7 and 8 refer to the timing, budget and legal documentation of the project. Finally, Chapter 9 summarizes the general development, proposes future developments and includes a final conclusion.

CHAPTER 2:

STATE OF ART

2.1 IOT Technologies and Applications in Autonomous Systems

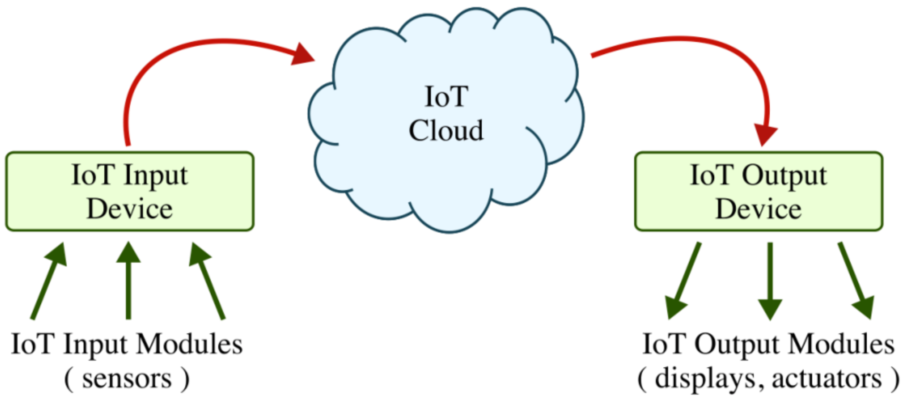

IoT Cloud
IoT Input Device
IoT Output Device
IoT Input Modules (sensors)
IoT Output Modules (displays, actuators)

Figure 3. IOT Typical Flow

The simplified diagram of an IOT system shown in *Figure 3* [3] could, in essence, be understood as any modern telecommunication system: there are input elements; a processing unit that receives information; and output devices to be triggered. However, the IOT approach focuses much more on achieving the previous configuration by means of network connectivity, whose data processing is usually coordinated under a master control and/or a big data processor.

For this particular analysis, sensors will be studied and described one by one. In general terms they are simple and inexpensive components that offer an analog or digital signal, or even both. Next, there is a processing unit, which is usually a PC or, in a more recently approach, mini-computers, such as the Raspberry Pi, used in this project. Finally, output devices may be understood both as, *stricto sensu*, **actuators** and **reactions**, which show a final result. In order to illustrate this difference, an irrigation pump may be considered as an actuator, while sending an email notification if a certain threshold is surpassed is an example of reaction.

Historically, systems may just use a microprocessor such as an *Arduino*, which present limitations such as not having Wifi connection by itself –it does have shields in case it is needed-. This disadvantage may imply a more difficult integration to an IOT network, in comparison to more recent options described on later chapters. On this aspect there is a new tendency of connecting devices together in order to create a sophisticated output system with network functionalities. For instance, it is now possible to make different elements respond to a certain event, so that a particular change results in outcomes of many types. Moreover, different correlations among ***many inputs may produce diverse results***, which is one of the latest incorporations in line of the *big data processing*.

Figure 4 shows the global configuration that sensing systems require, which could be compared to that offered by the IOT approach in the following chapters of this project.

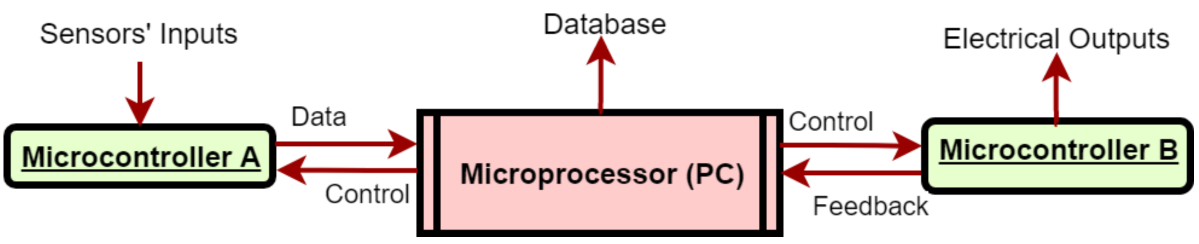


Figure 4. Block Diagram

According to the configuration proposed on *Figure 4*, a PC is usually at the core of the sensing flow, which processes instructions according to the sensors' values, evaluates whether an action should be applied and stores the essential information to a database register. IOT systems may operate many information, which needs to be synthesized in a few indicators, which are precisely those to be stored or transmitted to other sub-systems. Therefore, information processing should be carefully designed.

It is necessary to state the **difference between autonomous and smart devices** and the scope of the prototype. To begin, ***autonomous*** systems usually refer to a program flow where the path is beforehand designed, by means of a closed loop control. This means the system will work exactly the same on the beginning and on the end of its life, so the behaviour will not adjust its approach (*model*).

In other matters, ***smart*** is an ambiguous word. It can be related to taking a decision via *machine learning*, so that available data refreshes a prediction model and further operations are set dynamically. Or it could reference ***the fact of connecting multiple devices so that they interact with each other or share information***. This definition is the one that explains better the performance of the exposed Smart Agriculture Prototype.

2.2 Analysis to the State of Art

After the previous information, it should be clear that ***IOT technologies may in essence share the main purpose of control systems' technologies***, but change drastically on the user experience, offering new command ways such as remote or voice control and also an elegant information presentation. Therefore, providing an easy user interaction and clear output results are indispensable requirements for IOT future systems.

On one side, it is also important to remark that one of the main features of this new sector is that information is greatly shared among sub-systems; and online communication plays a very important role too. Devices such as ESP32 [31], described in *Chapter 4*, which works similarly to an Arduino but has a Wifi module integrated, are perfect for this purpose.

Wireless devices needs to be coordinated and open to Internet. For this reason, MQTT (Message Queue Telemetry Transport) rises as the main protocol in IOT and domotics. MQTT is a protocol that usually transports light messages. The most common MQTT configuration is that where a receptor device listens to a (***MQTT).topic*** and, when information is *published* there, it shows an action (for example, it turns a LED On). Also, feedback could be received from those wireless devices, so that they can also *publish* if, for example, an attached sensor's value changes drastically. The Internet security required for connecting to these devices over the Internet is studied in following chapters.

Besides the MQTT network set in this project, IOT has also changed a lot in respect to new **control technologies**, as, for instance, with the appearance of Alexa [6] and Google Home [7] **voice-assistants**. It is true that, for a user, it may be more comfortable to activate devices by means of an always portable instrument, which happens to be the voice.

In Spain the low-cost models of both alternatives, which correspond to *Alexa Echo Dot* (third generation) [8] and *Google Home Mini* [9], were sold for the same price at the moment of studying the economical options of incorporating a voice-assistant, **59.99€**, according to the online market references at June 2019.

However, the choice of using Alexa in this project is not only a personal preference of the author, but has also a justification based on the extra functionalities and possibilities Alexa could provide, which are called ***Skills***. These allow Alexa to respond to voice requests in order to run certain specific services, such as starting a RTVE newscast, providing RENFE information or sending an SMS.

The equivalent word for Google is called ***Action****.* There is enough evidence that Alexa provides much more options, with 4,253 *actions* in the U.S. compared to approximately 60,000 non-duplicated Alexa *skills*, at January 2019. [10]

Nowadays, there is also a very recent IOT control method which is precisely by means of an Android or iOS app. The preferred mobile operating system in Spain is widely Android, with about 80% of the whole market in 2019 [11]. Under this country's predominance, consequently, an Android app was developed, in order to control MQTT messages.

As for monitoring and data visualization of the system activity, a user-friendly answer is provided in this project via *Thingspeak* [12], a service which requires a write API and is updated each time a group of sensors' measurement is checked *and* also if a user presses any ON/OFF button in the developed Android app explained in *Chapter 6*.

This service is essential, as *Thingspeak* is used in this project not only for graphical purposes, but also as a cloud database and as a means of triggering actions, after a whole process detailed. ***The main reason for using this service is to graph in a same service the information of the Smart Agriculture Prototype and that of the developed Android app.***

Alternatively, a **database** could be proposed in order to store each of the activity values and a ***Matlab*** or similar script to create very comparable graphs to those given by Thingspeak. However, it would still be needed a mechanism in order to periodically upload these graphs so that the user could access to them, which, globally, seems a much more complex solution than the explained service, with same results.

Moreover, it is important to notice that the Raspberry Pi used in this project is a mini-computer and, therefore, has limited computational resources and would already be running several services. Even from this point of view it seems better to send the *raw* information to an external service such as Thingspeak, in order to store it there and, if needed, even process it further, as on applying big data processing for which the same platform offers solutions.

In brief, the general tendency in IOT is to sense environmental characteristics by means of connected sensors that sample them periodically. Wireless devices are about to rule this sector, which need to be small and be connected, as in a local MQTT network where they can both *listen* to a specific *topic* to which react and *publish* information.

In summary, the prototype presented in this Final Project could be defined as an *autonomous* system with networks functionalities that *senses* the environment and *applies actions* under a low-cost but perfectly functional perspective. The system also provides graphical output to the user and satisfies modern IOT lines on introducing wireless devices. It was also included an alternative control option via Android app or Alexa voice-assistant.

In conclusion, the state of art in this matter has the challenge of computing the incoming information from sensors so that a complete automated decision could be taken. The cheapening and expansion of mini-computer with excellent technical features, whose legal framework should also be studied in detail, allow the integration of complex systems for a low price and even on a big data approach.

CHAPTER 3:

SELECTION OF COMPONENTS AND SENSORS

3.1 Raspberry Pi Board and Computational Resources

In order to understand how the prototype is proposed, it should be first analysed the general block diagram of *Figure 5*.

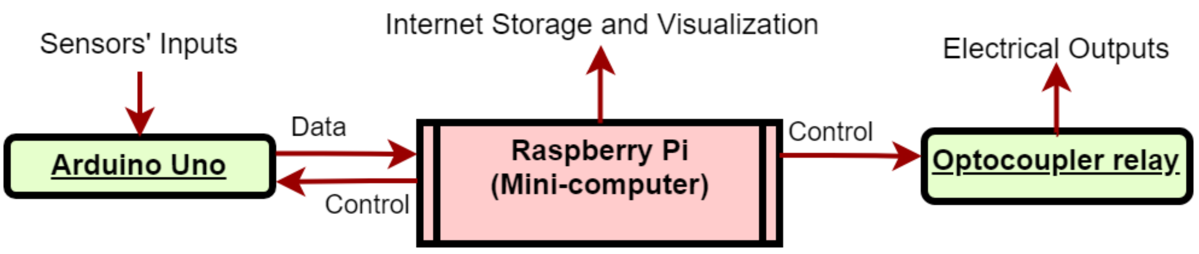


Figure 5. Prototype Block Diagram

This project uses both digital and analog sensors, which will be connected to the Raspberry Pi mini-computer and to the Arduino Uno microcontroller, respectively. It is also remarkable to understand that in the present design Arduino transmits its analog sensors' information via serial communication to the Raspberry Pi, so that it processes all the data and switches the output devices consequently. Therefore, it could be understood that other different configurations could also lead to similar performance. The design decision of using Arduino Uno as the analog sensor microcontroller is justified in *5.2 Arduino Sensing*.

On another perspective, Raspberry Pi also runs the following network services:

1. **MQTT server (broker)**, to forward data to wireless devices
2. **Surveillance camera server**, as a security functionality
3. **Remote connection**, to control Raspberry Pi over the Internet

Raspberry Pi (Model: 3 B +) [15]:
The main element in this prototype is the Raspberry Pi, which is the head that processes all the information, connects to the Internet and controls both the MQTT and security camera servers. Its operating system has been chosen to be *Raspbian* [13], a free operating system very popular and optimized for the **Raspberry Pi**, which is based on **Debian** [14].

The main code was written in ***Python*** and executes the system's actions *automatically*. This script is able to run automatically on the Raspberry Pi boot, option considered in section *5.7 Autoboot*.

The Raspberry Pi [15] is also not only responsible of receiving the data from digital sensors, but even triggers the optocoupler relay that open/closes the different output elements by means of its output pins. On the other hand, it connects to the Arduino UNO via USB, to receive, on command, its analog readings via serial.

Raspberry Pi 3 B + [16] is the latest one at June 2019 and shows the *pinout* map of *Figure 6* [4].

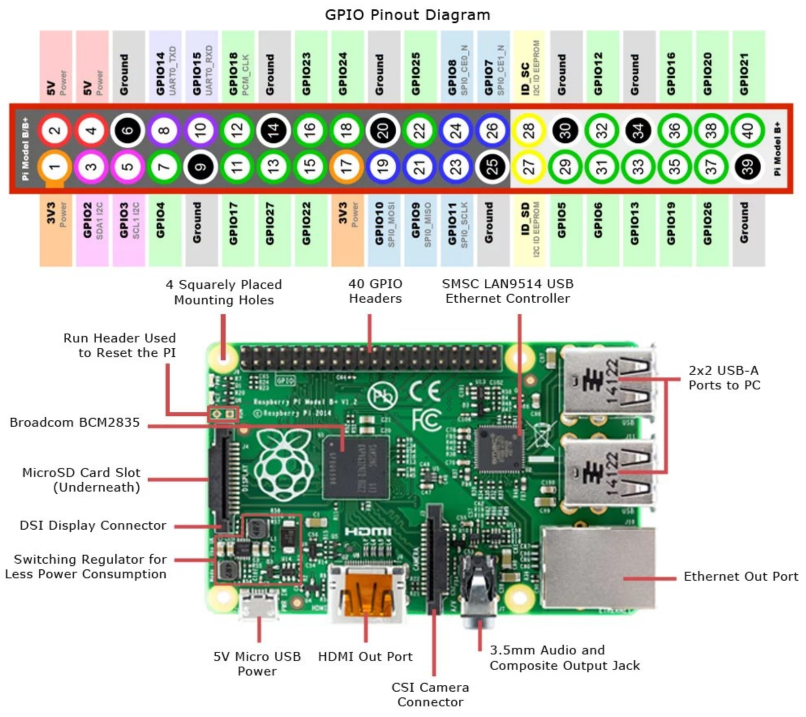


Figure 6. Raspberry Pi 3 B+ - Pinout

The most important characteristic of the Raspberry Pi board of *Figure 6* is that it offers many GPIO in order to connect sensors in this project; it feeds with 3.3V and 5V power supplies (with maximum current restrictions); and own I2C connections. Obviously, it can also access to Wifi, owns 4 USB connections and outputs HDMI signal in order to connect to a screen in case remote access is not preferred.

The Raspberry Pi 3B+ is very affordable, as it only cost about 35€ at the usual online markets (June 2019). Nevertheless, it offers a 1GB RAM system and is able to run at a nominal frequency of 1.4 MHz. All these features justify using this mini-computer as the main processor of the whole system.

3.2 Main System Electrical Characteristics

This section tries to sum up the considerations that have influenced design decisions as well as give a very brief understanding of Raspberry Pi GPIO structure. The main target is to define how to prevent overvoltages and keep the working under nominal values. It could be summarized as:

- If a GPIO is configured as input, it should receive always <0.5 mA.
- If a GPIO is configured as output, it will never demand >16mA. The reason for this is that the current is drawn from the 3.3.V supply, whose limitation is 50mA, shared for all 3.3 connections. In other words, it may draw no more (>50 mA) from the +3.3 V supply.
- Pins 3 and 5 use pull-ups (1.8 KΩ). These affect the system, as they drive *I2C. communication* that will be used in the pressure sensor and in the *LCD*

These characteristics limit the using of Raspberry Pi as a power source for too much sensors –however, more than enough for this and bigger projects-. On the other hand, if externally supplied, it is very hard that the Raspberry suffers problems for having many ***input*** sensors connected. The principal inconvenient is *only* not to exceed 0.5mA per input channel.

The general configuration of a GPIO pin is shown in *Figure 7* [5]:

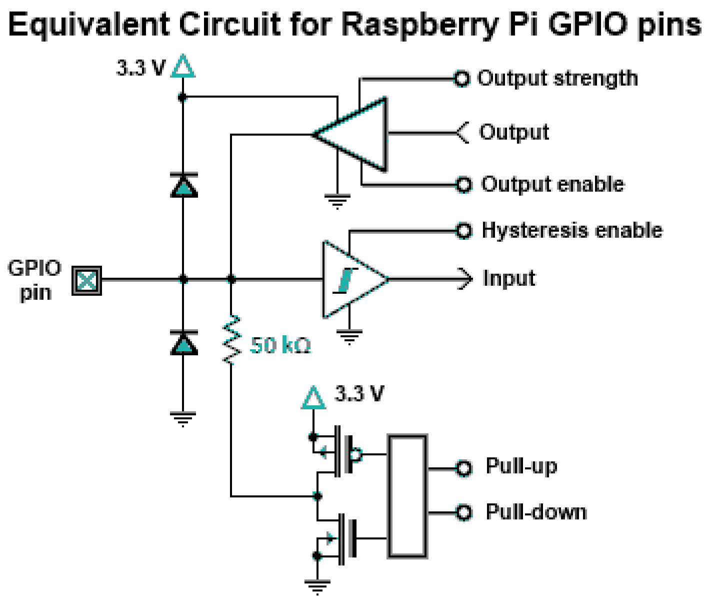


Figure 7. GPIO Pin Schematics

Of course, in the implemented system it is important to note that Raspberry uses mainly 3.3V for its inputs and outputs, while the Arduino will work on 5V. This means that, if connected one to each other, a conflict may appear. A solution for this problem could be the using of an intermediate circuit called ***level shifter***, which converts voltages (for instance 3.3V signals to 5V). However, as stated before, Raspberry Pi connects to Arduino only by USB in the present design, which also feeds its power source and, therefore, this effect is not produced.

3.3 Software (Python 3, Arduino IDE)

Python 3:
The software of this system has been developed with ***Python 3*** [17]. Using other Python versions is possible but may require the adaptation of the code.

The project is oriented to create a whole unit of different sensors working together. Each one should deliver information to the Raspberry Pi. The different modules are stored in independent scripts (*see Figure 8*).

Afterwards, they all have been integrated in the main script, which is called ***tfg.py***. Also note that this kind of architecture makes easier the integration to a *different* system, which can be designed to need all, modify them or use none of the modules.

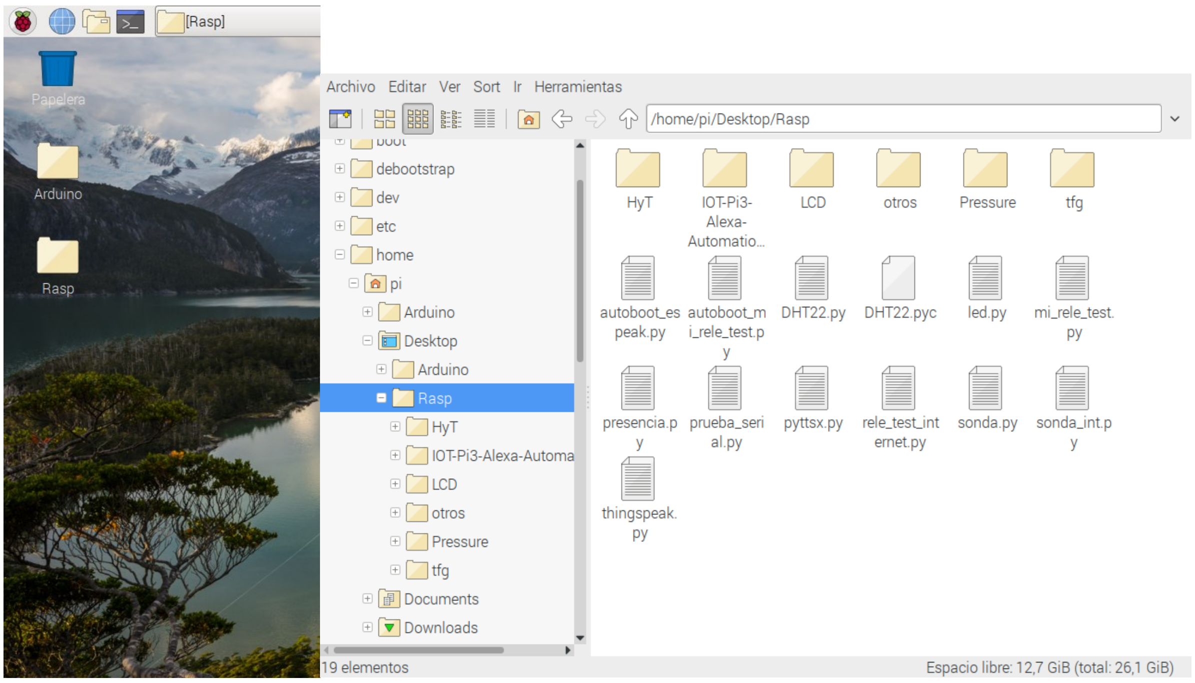
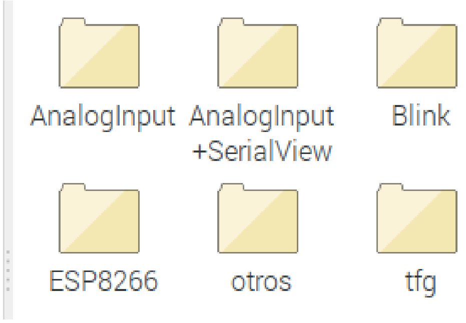


Figure 8. Modular Disposal in Desktop (left)
Figure 9. Modular Disposal in Raspberry Pi Files(right)
Figure 10. Modular Disposal in Arduino Files (down)



Moreover, in order to use GPIO, the following packet needs to be installed on using the following commands:

Example of packet installation in <pi> directory:

- ✓ *wget abyz.me.uk/rpi/pigpio/pigpio.zip*
- ✓ *unzip pigpio.zip*
- ✓ *cd PIGPIO #changes directory to the created folder*
- ✓ *make*
- ✓ *sudo make install*
- ✓ *sudo python3 setup.py install #Allows to use the packet in Python*

It is singular to require the installation of a specific packet even in the Raspberry Pi's dedicated OS (Raspbian) in order to use GPIO functionalities. Nevertheless, this situation may be understood under the consideration that many Raspberry Pi projects may precisely not require GPIO. In fact, an extended use of this mini-computer requires only its network functionalities, as an Internet server. To conclude, once the previous installation is done, this configuration ends.

Arduino IDE:
In order to program the Arduino UNO and ESP32 devices the Arduino IDE [18] was used. The installation is simple and the correct version for Raspberry Pi is called "*Linux ARM 32 bits"*. The link to the official IDE download is provided in the external references.

Arduino IDE was used to program in C language both the Arduino UNO and also the ESP family wireless devices. On the one hand, Arduino needs to execute a code that responds to the Raspberry Pi commands and returns, via serial, the value for the required analog sensor. On the other, Arduino IDE also allows to upload code to the ESP01S (discarded option commented on *Chapter 9*) and to the ESP32 that finally compose the wireless alternative of the prototype.

From the software description explained in this section it could be deduced that the chosen design aims to implement simple code in the Arduino part, which only reads the required sensor's value, maps it in a 0-100 range and returns it via USB. On the contrary, the Raspberry Pi receives this value and stores it on an array of sensors. Therefore, the logic of the whole system and the output activation is produced thanks to the processing in the Raspberry Pi.

3.4 Sensors

This project proposes many devices in order to sample enviromental characteristics. However, not all of them are implemented in the Smart Agriculture Prototype. A summary of the sensors studied is included.

SENSORS		
RASPBERRY PI	Temperature, Humidity Sensor (DHT22)	Accuracy: Temperature +/-0.5°C, Humidity +/-2RH; Minimum Detection Time: 2 seconds
	Atmospheric Pressure	Acurracy: +/- 1 hPa
	Presence Sensor (PIR)	Detection Range (adjustable): 3-7 m; Detection angle: 120°; Minimum Detection Time: 3 seconds
	USB Camera (Security Camera)	This project runs on *gardenhost666.ddns.net:8081*. Requires Motion and Server Configuration
ARDUINO UNO	Soil Moisture Sensor	4 Capacitive sensors, which are much stronger to corrosion processes than resistance-based
	Water Level Sensor	May suffer corrosion on long term
	Light Sensor (Top and Roots)	On having two light sensors it could be deduced the light dispersion in our room
	Air Quality Sensor (MQ135)	Detection of: NH3, NOx, Alcohol, Benzene, Smoke, CO2
MANUAL	PH Sensor	Manual sensor Adwa AD11. Accuracy: +/-0,1 pH, +/-0.3°C
	EC/TDS Sensor	Manual sensor Adwa AD31. Accuracy: +/-0,2, +/-0.3°C

Table 1. Sensors Studied in the Research

The following paragraphs will describe each sensor used. They will justify why each of them has been selected over the rest of options for a ***particular type of measurement*** and, finally, they will cover their global role on the design too.

3.4.1 SOIL SENSING

The importance of ***soil sensors in Smart Agriculture*** is such that, even though there is just one type of sensor of this type, which may work under the two different physical principles exposed afterwards, it needed a specific section for its consideration. Soil sensors suppose *one of the first sensing in the humanity history*.

Although new technology in this field proposes more elaborated techniques, the most simple and effective solution for a low cost system consists still on a simple sensor that needs to be put into the water, fluid or soil to be measured and a sample of its voltage change.

Soil Moisture

During the design period an analog soil sensor based on resistance principle was used (see *Figure 11* [6]). It adjusted well to the global expected results. However, it is true that the metal presented in those sensors is very sensitive to water interaction and it may quickly develop into corrosion. It is remarkable also that this model is not a very stable sensor along time. Though it is very inexpensive at usual online shops (~1.5€, June 2019), it remains short in perfect conditions; its output may not be very stable in time and has memory problems –the output at a moment is influenced on previous states-.

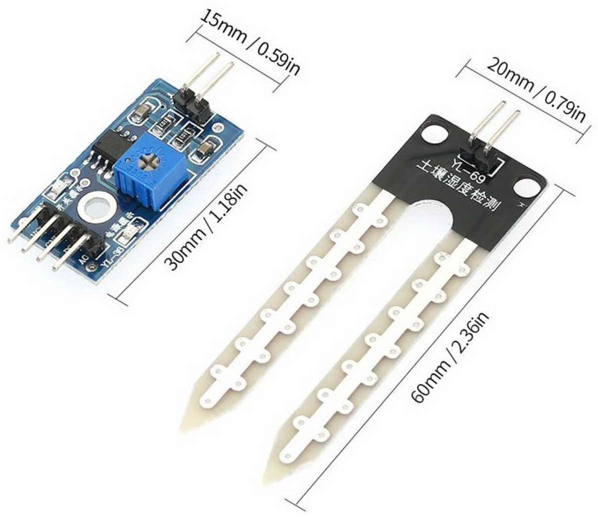


Figure 11. Resistance Soil Moisture Sensor

This device is usually sold with a probe (right element in the *Figure 11*), which is the sensor itself, and a small circuit (left element in *Figure 11*) that feeds the probe with current and provides an analog output as well as a digital one, whose value is compared to a threshold controlled by the blue potentiometer which, in detail, could be seen on *Figure 12* [7].

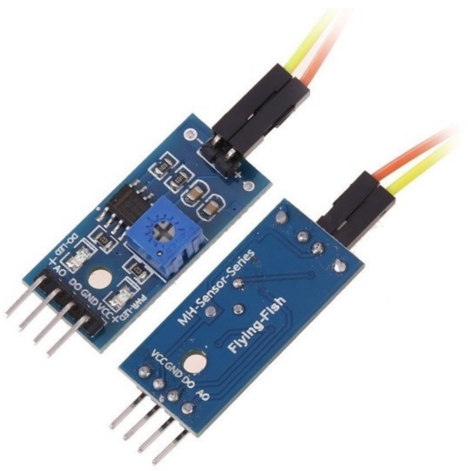


Figure 12. Complementary Circuit

Figure 13 [8] shows the connection of the resistance sensor to the Arduino Uno by using the complementary unit described before. It is important to note the preference for the analog signal output of this sensor in the present design.

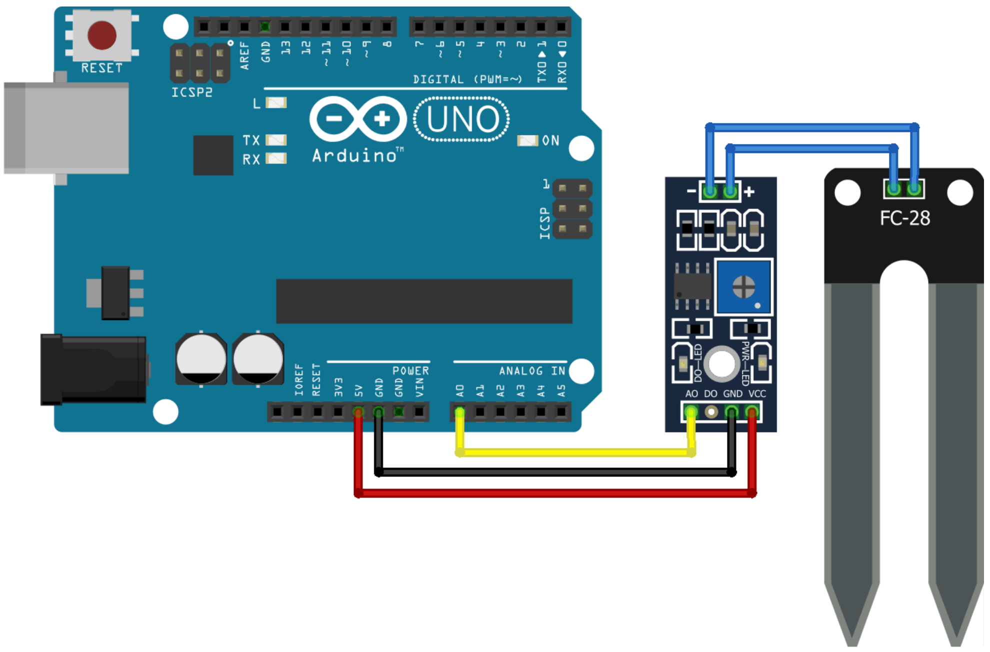


Figure 13. Resistance Soil Moisture Connection

On the other hand, the final design uses *capacitance soil sensors (see Figure 14* [9]*)*, which work under a different principle, as it measures the dissolved ions of its proximity and does not expose its metallic surface directly to the exterior, so that corrosion is not a problem. They are just a little bit more expensive but last longer. As an analog device, it is connected to Arduino UNO. Moreover, its better appearance coincides to being a more professional and accurate option than the previous one, for about 3€ at electronics shops (June 2019).

Figure 14. Capacitive Soil Moisture Sensor

3.4.2 CLIMATE SENSING

The next category includes all the sensors that may be used in order to register *environmental conditions*. Though the prototype does not need all of them, it is important to mention they all have been studied for future development or in order to allow an easier integration of the prototype to a different control field. They could perfectly be used in an industrial control system.

Environmental Humidity and Temperature

These two parameters are obtained thanks to a small sensor named DHT22 [19] (see *Figure 15* [10]). It is a digital sensor, so will communicate to the *Raspberry Pi* GPIO, and shows high accuracy at a low price (Temperature +/-0.5°C, Humidity +/-2RH). It is very similar to DHT11 [20] (see *Figure 16* [11])], which also outputs these same measurements, but with a lower accuracy (Temperature +/-1°C, Humidity +/-5RH). Consequently, DHT22 was chosen because its uncertainty range is narrower.

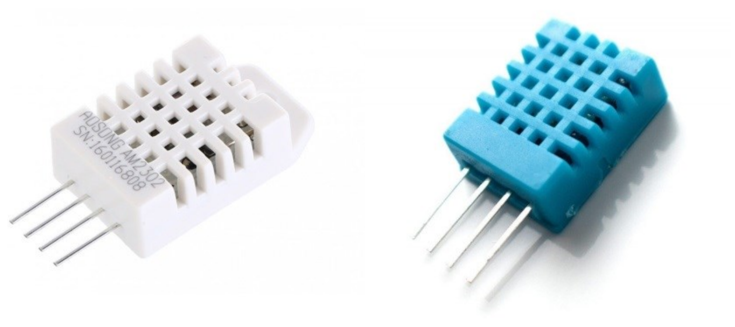


Figure 15. DHT22 Sensor (left)
Figure 16. DHT11 Sensor (right)

Figure 17 [12] shows the dimensions of the product in order to perceive it is a small device to be easily hidden or protected, if needed. Nevertheless, its frontal face should not be covered if accurate measures are required. On the other hand, the electrical connection is simple, as it only requires a 3.3-5.5 V power feed and a bidirectional port is offered to trigger a new sample and receive its value (see *Figure 18* [12]).

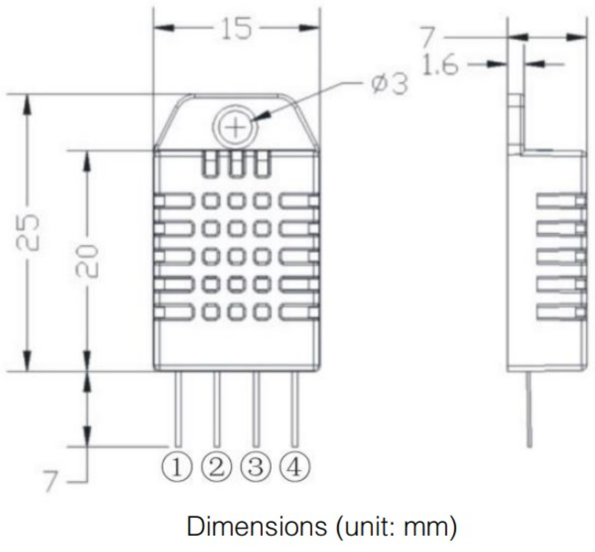


Figure 17. DHT22 Sensor Dimensions

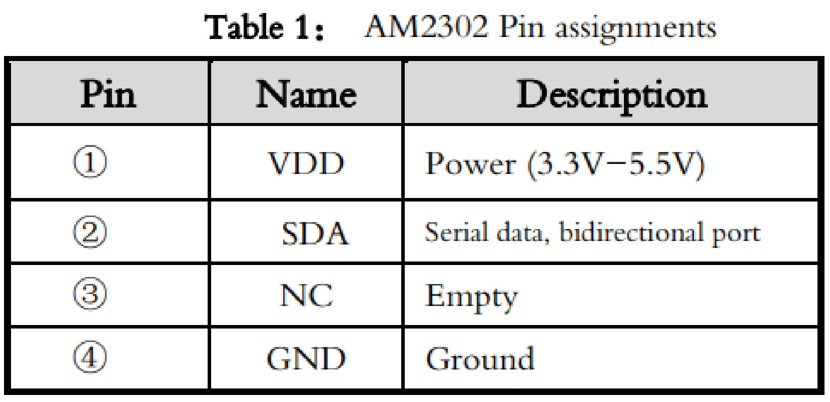
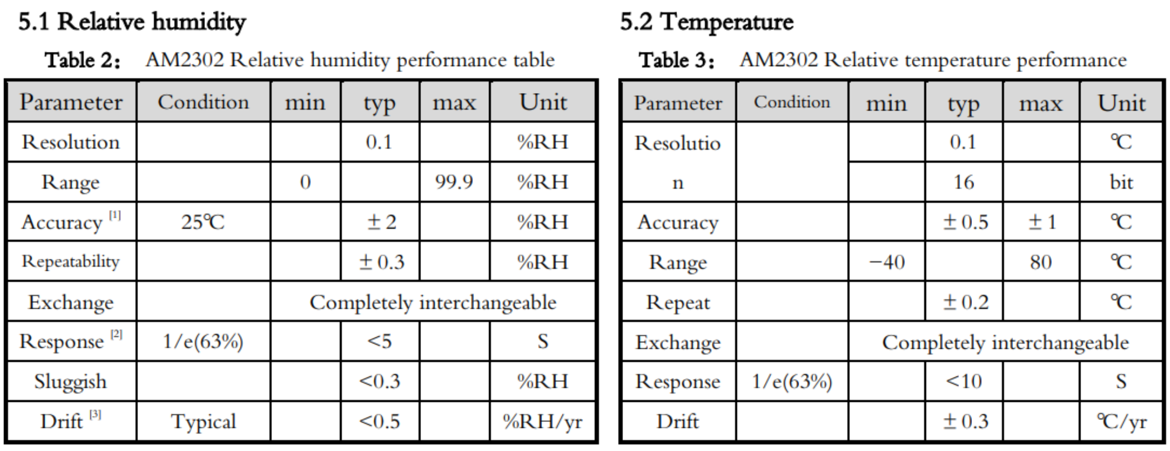


Table 1: AM2302 Pin assignments

Pin	Name	Description
①	VDD	Power (3.3V−5.5V)
②	SDA	Serial data, bidirectional port
③	NC	Empty
④	GND	Ground

Figure 18. DHT22 Sensor Electrical Connections

Although the main DHT22 features were already exposed in the beginning of this section, *Figure 19* [12] shows other which may also be of interest:



5.1 Relative humidity

Table 2: AM2302 Relative humidity performance table

Parameter	Condition	min	typ	max	Unit
Resolution			0.1		%RH
Range		0		99.9	%RH
Accuracy [1]	25℃		±2		%RH
Repeatability			±0.3		%RH
Exchange		Completely interchangeable			
Response [2]	1/e(63%)		<5		S
Sluggish			<0.3		%RH
Drift [3]	Typical		<0.5		%RH/yr

5.2 Temperature

Table 3: AM2302 Relative temperature performance

Parameter	Condition	min	typ	max	Unit
Resolutio			0.1		℃
n			16		bit
Accuracy			±0.5	±1	℃
Range		−40		80	℃
Repeat			±0.2		℃
Exchange		Completely interchangeable			
Response	1/e(63%)		<10		S
Drift			±0.3		℃/yr

Figure 19. DHT22 Sensor Main Characteristics

In order to program this sensor, an Adafruit library ([21], [22]) was required. In the system its values trigger the ventilation circuit, if temperature >24ºC and/or humidity >60%, which are the growing conditions required for most of the plants.

Air interchange is really important for plants, as their main communication to this world is by photosynthesis, which obviously produces gas interchanges. In this way, it is also necessary to note that a controlled climate system should periodically test the different gas concentrations.

Gas

Although this sensor has no implementation in the present prototype, it needed a place in this exposition. Contrary to the previous types, gas sensors usually perceive not only the main element, but also, in a smaller proportion, other compounds. For simplicity, it could be assumed there is only one main component to be detected by these devices.

Initially some gas sensors, all of the MQ family [23] in order to satisfy the low-price requirement, were studied. For instance, MQ2 was sensitive to LP gas and propane, while MQ3 responds to alcohol and MQ7 to carbon dioxide.

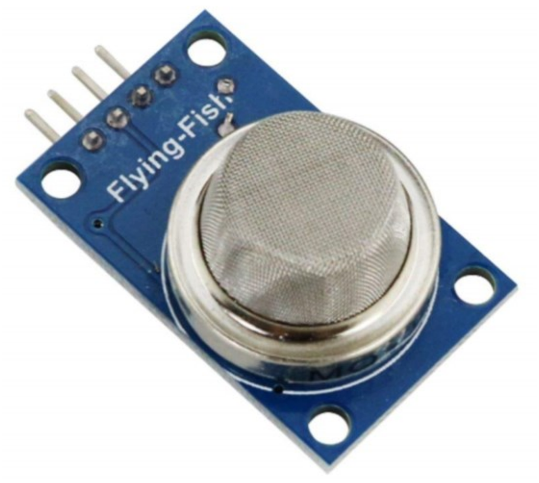


Figure 20 – Gas Sensor MQ135

The ***MQ135*** of *Figure 20* [13] was finally selected because, on the one hand, it reacted to different gases (NH3, NOx, Alcohol, Benzene, Smoke, CO2) in an adequate proportion (see *Figure 21* [14]), all of which compose the main pollution to avoid in buildings or offices. Nevertheless, the value of each gas is not stated independently, but it is still a good indicator for air pollution for most application.

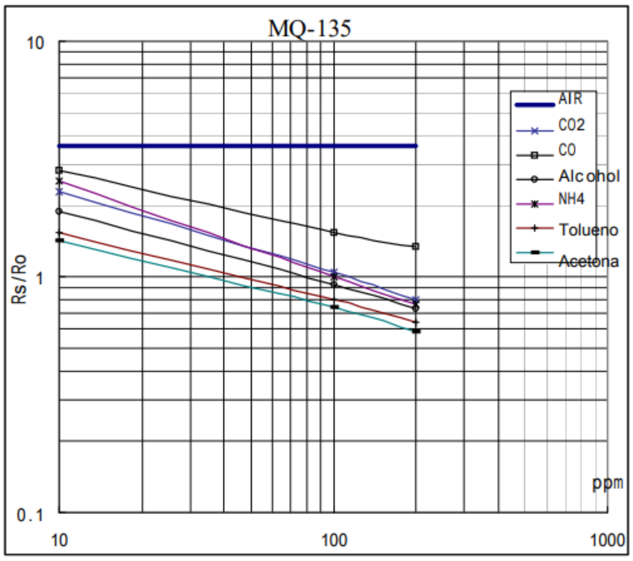


Fig.3 is shows the typical sensitivity characteristics of the MQ-135 for several gases.
in their: Temp: 20℃、
Humidity: 65%、
O_2 concentration 21%
RL=20k Ω
Ro: sensor resistance at 100ppm of NH_3 in the clean air.
Rs:sensor resistance at various concentrations of gases.

Figure 21. Gas Sensor Sensitivity to Compounds

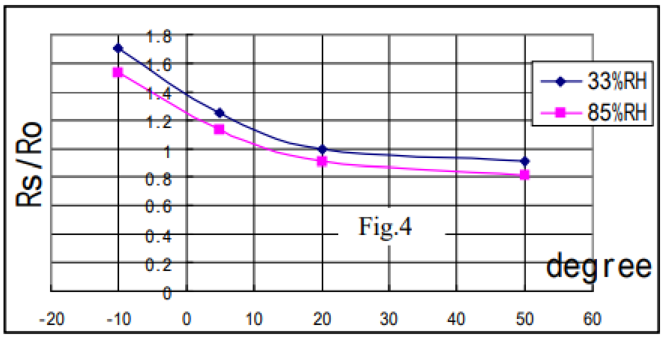


Fig.4 is shows the typical dependence of the MQ-135 on temperature and humidity.
Ro: sensor resistance at 100ppm of NH_3 in air at 33%RH and 20 degree.
Rs: sensor resistance at 100ppm of NH_3 at different temperatures and humidities.

Figure 22. Gas Sensor Environmental Dependency

In fact, this device is a small heater, inside an electro-chemical sensor. In *Figure 22* [14] it could be seen that there is a relation between temperature and humidity that affects MQ135 measures in a non-linear way. Therefore, a complete system that uses this sensor may consider some *in situ* correlation (regression) study between humidity, temperature and MQ135 outputs, so the appropriate threshold to trigger an alarm is chosen.

Light

An analog candidate was proposed for the luminic sensor. The option proposed was selected for two main reasons: its behaviour is simple, as it uses mainly a photoresistor component and it responds well to the low-cost purpose. In this measurement, there were no other sensors considered, both because they all were based on this same photoreaction, under a similar range of price, and also due to the personal possibility to find the selected type for the author.

The light sensor described (see *Figure 23* [15]) has a little circuit regulated by potentiometer incorporated, as also explained in the soil moisture section. In this way this device could deliver the analog reading on one pin or, on another, the digital pin would return a high-level value if the received light is over a certain threshold. It is important to note that, on having at least two light sensors (ie. one on top of the plant; other at the lowest leaves), the ***light dispersion*** in a certain room could be deduced, so that the light system could be optimized on its orientation and power required. This sensor is low cost (~2€, June 2019), very linear, compatible with 3.3 – 5V feeding and, in a general view, it is easy to understand and be integrated.

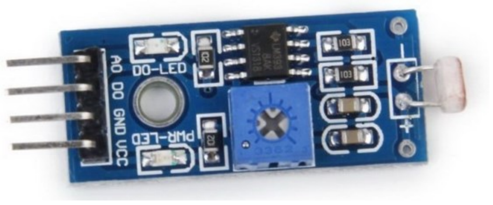


Figure 23. Light Sensor

Barometric Pressure

For this magnitude two very similar options were studied: BMP 280 [24] and BME 280 [25], as they represent low-cost solutions and, nevertheless, accuracy values are enough for many industrial purposes. They both offer at least a barometric pressure indicator and a temperature reading via I2C protocol, whose details can be found at the description of this connection (*Chapter 5*). The main difference is that the BME 280 device also delivers humidity readings.

According to the previous explanations, the environmental conditions were already covered by means of the DHT22 sensor. Therefore, more humidity readings were not necessary in this project and, finally, the BMP 280 model (*Figure 24* [16]) was chosen. Its accuracy is of ±1 hPa (pressure) and ±1.0°C (temperature).

Although more electrical specifications could be find afterwards [26], which are also referenced at the end of this Final Project, it is important to note this sensor only works under a maximum of 3.3V. This characteristic explains why BMP 280 barometric pressure sensor is connected to the Raspberry Pi and not to Arduino Uno –which works easily under 5V- in the prototype explained in this project.

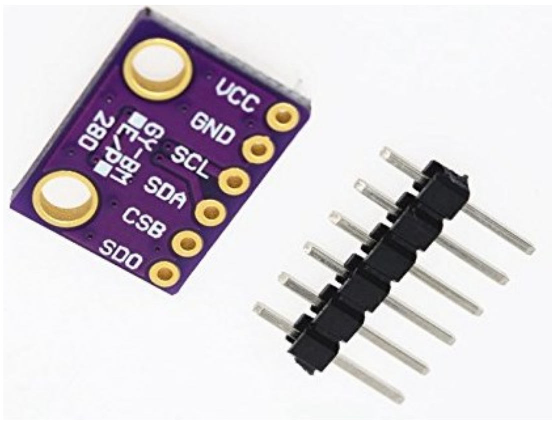


Figure 24. Barometric Pressure Sensor

Electrical specifications of GY-BMP280 Module [26]

- **Model**: GY-BMP280-3.3
- **Chip**: BMP280
- **Power supply**: 3V/3.3V DC
- **Peak current**: 1.12mA
- **Air pressure range**: 300-1100hPa (equi. to +9000…-500m above sea level)
- **Temperature range**: -40 … +85 °C
- **Digital interfaces**: I²C (up to 3.4 MHz) and SPI (3 and 4 wire, up to 10 MHz)
- **Current consumption of sensor BMP280**: 2.7µA @ 1 Hz sampling rate

This device is sold unwelded, so its pins have been soldered. It is a small piece, so the task needs some soldering skills. Even though this element offers both temperature and pressure data, in this project the temperature measurement was not used. Nevertheless, altitude readings could also be easily computed, if the *Mean Sea Level Pressure* is known, which, in Leganes, is of 1026-1030 mbar. In this way, it could be compared the pressure at a location with that of the sea and, thus, also obtain an altitude reading for other purposes with this sensor.

3.4.3 OTHER

Water Level Sensor

The purpose of considering this element is to have a track of a fluid level in a tank and set a warning in case it is empty. As soil sensors may trigger the action to water, if there is no fluid obviously this activity cannot be done, and pumps could even break in this situation, some of which may be really expensive in industrial sectors. Two different water level sensors were studied: The most typical resistance sensor of this kind (see *Figure 25* [17]) and an alternative by using the HC-SR04 [27] (see *Figure 26* [18]).

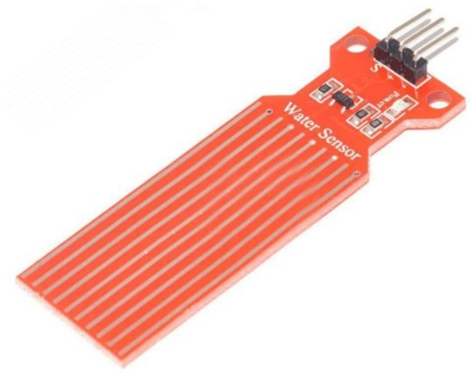


Figure 25. Resistive Water Level Sensor

To begin, an analog resistance sensor is proposed. Once again, it simply works on comparing the voltage of its metallic plates, which are directly exposed to the fluid. In this sense, it is clear corrosion may appear, leading to inaccurate readings. However, for most of the cases it is enough to know in general terms the water level and, with a simple threshold, it may adapt well to any system.

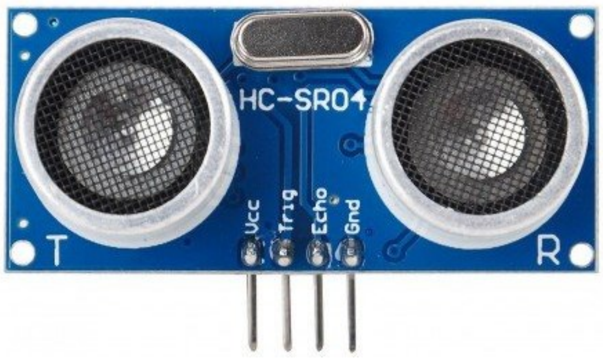


Figure 26. Ultrasonic Distance Sensor

On the other hand, the ultrasonic sensor HC-SR04 [27] (*Figure 26*) was also studied. It is a very accurate device that measures distance. Its working is as follows: it emits an *acoustic* high frequency burst (40KHz), waits until the echo is bounced back, it transduces that return signal and ***estimates the distance to the first object*** according to the following:

i. This acoustic signal (*1octave further than heardable by a baby*) is propagated in air at about **343m/s** (0.0343 cm/ µs) (acoustic wave)
ii. It supposes that the first bounced ray to reach is that of the ***shortest path*** (the straight one, according to geometric theory of acoustics after a few particularizations not related here)
iii. Therefore,

$$\mathbf{RayDistance} = \mathbf{0.0343} \cdot \mathbf{Time} \qquad [\mathbf{1}]$$

iv. Notice that the Time is measured since the burst is triggered so, in fact, this data stores both the going and returning of the ray.
v. Finally,

$$\mathbf{DistanceToObject} = \mathbf{0.0343}\ x\ \frac{cm}{\mu s} \cdot \frac{Time}{2} \qquad [\mathbf{2}]$$

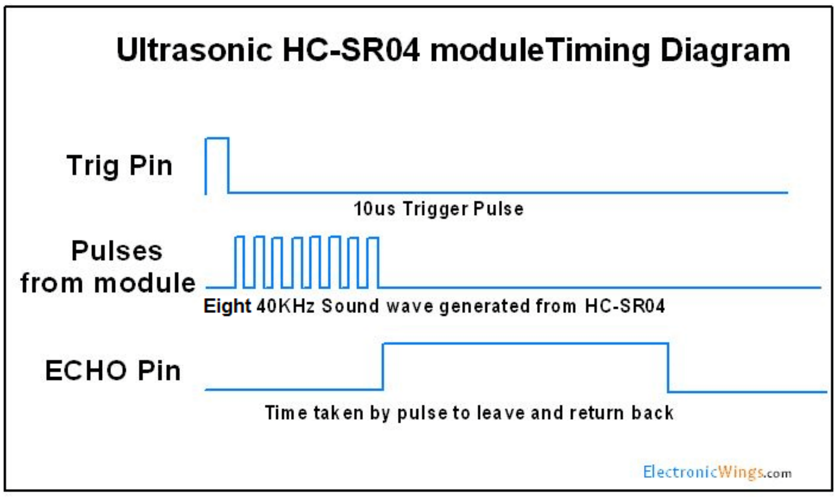

Ultrasonic HC-SR04 moduleTiming Diagram
Trig Pin
10us Trigger Pulse
Pulses from module
Eight 40KHz Sound wave generated from HC-SR04
ECHO Pin
Time taken by pulse to leave and return back
ElectronicWings.com

Figure 27. Ultrasonic Distance Sensor Procedure

In order to sum up the previous procedure, it could be said that *this sensor releases an ultrasonic ray when its trigger pin is set to 1* and it waits until the bounce comes back. The return information is translated as a voltage to the *echo* pin for the next system to process –Raspberry Pi. *Figure 27* [18] illustrates the trigger-echo procedure already described, on adding value information.

If the HC-SRO4 is located at the top of a given tank, pointing down to the fluid, it could be measured the distance to an obstacle and, therefore, the Water Level of a Tank. This sensor is very accurate and delivers and extraordinary valuable data even on having *some degree of reflections*.

PH and EC/TDS Manual Sensors

For the measure of this two characteristics Adwa manual sensors were selected: **AD11** [28] and **AD31** [29] (pH, EC/TDS, respectively). In many industrial processes both PH and TDS are manually controlled, because, due to their importance in nutrients' absorption, a human supervision or even a specialist's work is usually required. PH decreases with an increase in temperature too, which makes it difficult to estimate automatically the perfect pH target. There are concentrated products that modify the pH of a fluid but it is very complex to design a system that automatically pumps the correct amount of pH modifier so that the dissolution reaches a certain point.

Figure 29 [20] shows a table that shows how easily a plant is able to absorbed each nutrient, depending on the soil pH. Therefore, most of the plants would absorb the majority of nutrients under 5-7 pH, but the optimum value may vary according to variety and growth phase.

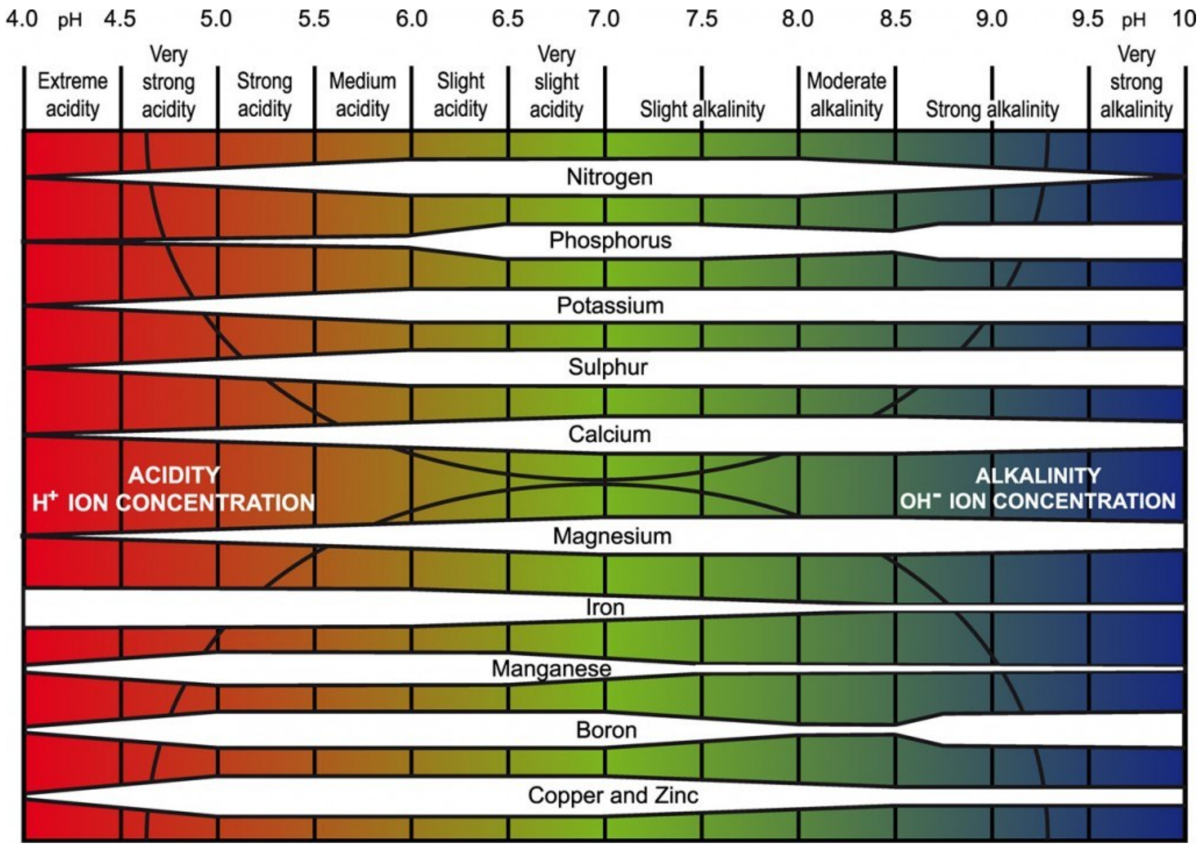


Figure 28. Nutrients Absortion According to pH

EC and TDS are both indicators of the level of nutrients and salts present in a fluid. The first one stands for *Electrical Conductivity*.
Therefore, **if many salts are dissolved the *E*lectro*C*onductivity raises**.

Similarly, TDS stands for ***Total Dissolved Solids*** and is an indicator of the global concentration of substances dissolved in water. TDS includes both salts and organic matter. These two parameters are important, because they affect as well in the nutrients absorption in great level.

If certain raw water presents a high EC/TDS, *whose salts are usually not good nutrients*, it means that just a small quantity of industrial or biological fertilizers could be added to this fluid. Plants simply will not take advantage of higher levels.

For this reason low EC/TDS water (distilled water) is sometimes used, so that the level of salts is that provided by the desired fertilizers.

The problem of not taking them into account could, thus, be ***sub*** and ***over-fertilisation***. If a plant receives more nutrients than its saturation tolerance, they are wasted and, even, the plan itself may suffer. First, it could produce some grow disorders, as leaves extremely green or anomalous, but also some nutrients may not be correctly absorbed because the receptors are saturated. In absence of those nutrients (sub-fertilisation), the behaviour seems evident.

In conclusion, both pH and EC/TDS are important indicators of the soil or water to which plants are exposed and their tolerance may vary according to variety, growing state and even the temperature.

Finally, those manual sensors need a *test calibration*, which is an automatic process where probes are exposed to reference liquids. The following lines refer to the ***calibration*** **and the** ***accuracy characteristics***:

- **Adwa AD11**: 1 or 2 points calibration (pH 7, or 7 and 4, respectively). Accuracy: +/-0,1 pH, +/-0.3°C,

- **Adwa AD31**: 1 point calibration by 1413 μS/cm solution. Accuracy: +/-0,2, +/-0.3°C

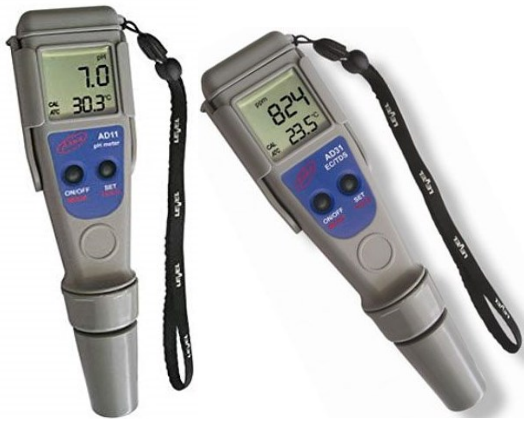


Figure 29. PH Manual Sensor (left)
Figure 30. EC/TD Manual Sensor (right)

Presence (PIR)

This, the last sensor proposed is also part of the main system and tries to identify whether a movement is produced on its proximity. For this purpose it uses an *infrared-sensitive system* that notices those radiation changes, as when a body starts blocking a lens [30].

The manufacturer designs independently the lens and the sensor itself, which is actually made up of ***two IR sensors***. For the optics it uses a *multiple Fresnel lens configuration* that allows rays coming from its 120º detection angle to concentrate to the sensors' chamber.

To simplify the functioning, if an object intercepts, the exposition changes between the two IR sensors located inside the chamber, which is registered as a positive/negative electrical peak. A movement *moves* electrical currents. All this composes the PIR Sensor summarized in *Figure 31* [23].

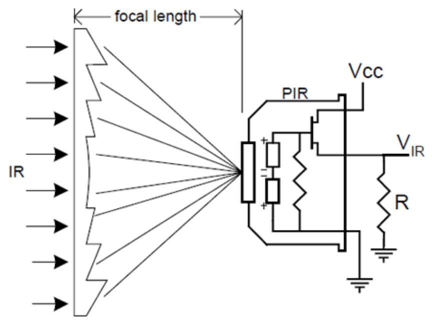


Figure 31. Presence Sensor Incidence

It is remarkable to note that this full sensor has two potentiometers, which regulate the *Detection Range* from [3-7] meters and the *Detection Time*, which minimum is 3 seconds. In this way, the device is able to focus more on sensing the near/far field and the output, once triggered, may not vary in 3 seconds or more (see *Figure 32* [24]).

As it delivers a digital signal, it is connected to the Raspberry Pi. This particular device is important in the program flow, as, when triggered, an interruption is set and the security protocol is activated. This process is explained in detail later.

Some of the information provided in this chapter may have been summarized, globally coherent on light propagation and paraphrased, ordered and presented for the particular interest to this project. A very descriptive reference is that included in [30].

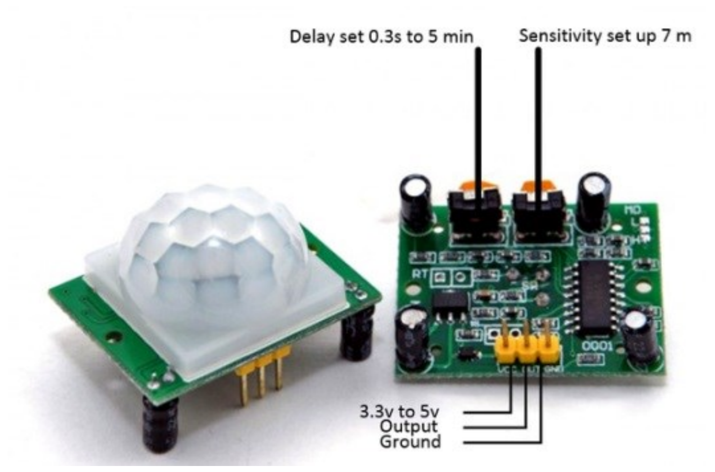


Figure 32. Presence Sensor

CHAPTER 4:

ELECTRICAL CONNECTIONS AND OUTPUTS

The present chapter includes all the information that refers to the activation of output devices in the prototype. It also required a dedicated place according to its place in the flow. According to a general abstraction, it represents all the terminal devices to be activated according to the sensors' information and the Python processing.

4.1 Optocoupler Relay

This section shows the electrical devices that are needed in order to make the Prototype system work. As a prototype, two main wired outputs are set: the irrigation system (water pump) and the air system circuit (extractor and intractor). The present Final Project provides the option to connect devices by means of a *wired connection*, which is the main proposal, as well as an alternative by using ESP32 *wireless devices*.

In either case, the central element that allows to control the output devices is the optocoupler relay. It has two separates circuits: control and relay commutation. If the jumper is set as in the image, both will share VCC voltage, which is more handable and works perfectly if enough source current is supplied.

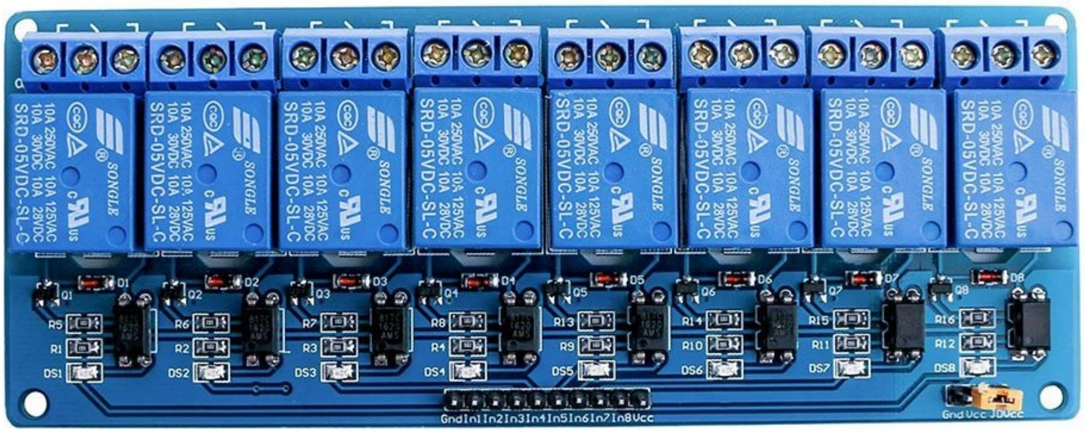


Figure 33. Optocoupler Relay

Furthermore, the functioning is simple: a device must feed its VCC (3.3 – 5 V) and GND pins and each of the channels to control also should have a wire connection (ie. In5). In this way, when the Raspberry Pi or host device turns a certain GPIO its relay channel will also change.

It is important to remark two more specific characteristics. On the one hand, it can be seen that each relay channels has three holes. The reason for this is that a normally-open or a normally-closed configuration may be chosen. This defines the initial electrical state of that channel, when that relay is not triggered. If the wires are set in normally-open, the path will be open by default and, when the relay is triggered, the circuit will be closed. This configuration is the most intuitive for most of the cases.

Secondly, this relay is low-level triggered, which means that its inputs are triggered on receiving a 0. This is an extremely important decision also for the code, as there are also high-level relays.

Finally, *optocoupler* means that this design tries to isolate as much as possible the control and source supplier –Raspberry Pi or ESP32- from the high voltage from the outputs by means of a light system. In strict sense, an optocoupler interconnects two separate circuits by the emission of a light, usually IR, and its reception on the secondary circuit, by means of a light (IR) sensor. The following image of *Figure 34* [26] shows a simplified circuit to illustrate this idea, which may vary a little of that of the actual relay proposed.

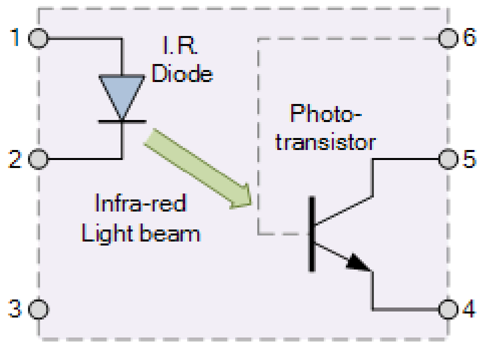


Figure 34. Optocoupler Relay Principle

In general terms, the relay is used in order to switch the outputs' state. This model is able to work both on 3.3 and 5 V, as long as the input source current is enough to allow the most quickly commutations in the major number of channels at the same time required. The fact of using an optocoupler relay ensures that the Raspberry Pi (wired modality) or ESP32 (wireless) that feeds the device is isolated from the high output voltage and currents that will be turn on/off continuously.

4.2 Electrical Diagram

According to the previous information, it is clear that the element that triggers the outputs in this system is the optocoupler relay, but it also needs power feed and GPIO signal from Raspberry Pi or ESP32. The next example shows a basic 1-channel commutation. As it can be understood, this example prefers 5V as VCC and uses only one GPIO, set as output.

As for the electrical part, the relay will close the circuit, so that current reaches the outlet and, finally, the output device could be controlled according to *Figure 35* [27].

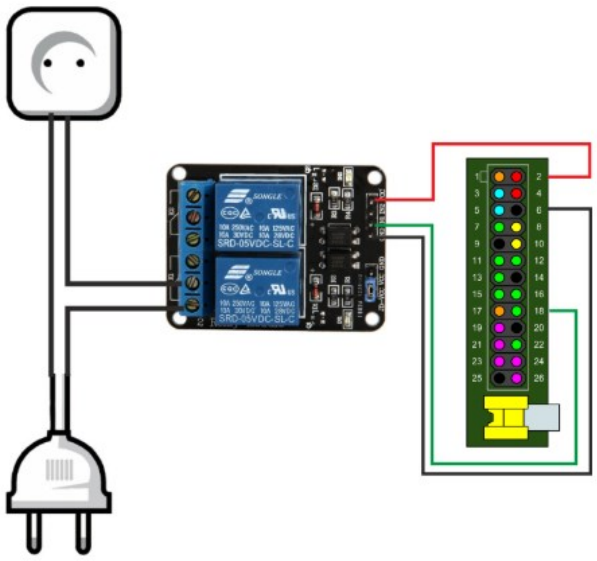


Figure 35. Optocoupler Relay Basic Circuit

It could be seen that the Raspberry Pi would require one GPIO, set as output, in order to activate each of the relays' channels. Also, the switching system requires to be fed from its power source, so commutations of many channels simultaneously would require a considerable power current from the Raspberry Pi.

In this same way, both the water pump and the extractor/intractor (same outlet) could be switched in this project. They use, therefore, two output GPIO and will respond to soil humidity and environmental conditions in order to change their state.

In case a particular implementation of the system also uses a lighting control, different options may be considered based on this basic circuit. Most of the plants follow 2 photoperiods: growing and flowering. On growing, lights requirements are usually high, which demand many artificial light hours, while the flowering tends to require less hours. Paradoxically, one of the main devices to turn on/off lights in this situation could be a one-outlet classic analog timer that closes the circuit for only specific hours. It is also possible to use a digital device that allows a complex schedule.

Finally, for security reasons all the electrical system could be fed from a master outlet that enables the whole electricity via IR remote control, but this option would remain in this project only as a proposal.

4.3 Power Source

DEVICES POWER		
RASPBERRY PI	Temperature, Humidity Sensor (DHT22)	3.3 V
	Atmospheric Pressure	3.3 V
	Raspberry Pi Internal Fan	3.3 V (Saves more energy, for long periods, usually enough) or 5 V
	LCD	3.3 V
	Presence Sensor (PIR)	5 V
	USB Camera (Security Camera)	5 V
ARDUINO UNO	Soil Moisture Sensor	5 V

Table 2. Prototype Device's Power Source

In a general view, digital sensors are feed by Raspberry Pi, with rather 3.3 or 5 V and analog devices are connected to Arduino UNO with a 5V supply. On analysing each of the devices' power implemented in this prototype:

It should also be noticed that Arduino UNO is powered via the USB which connects to Raspberry Pi. In case of using ESP32, each device needs an independent power source, which is the main reason of not proposing a wireless system in the prototype as a first option.

4.4 Wireless Alternatives

The main alternative found to wired configuration is by means of **ESP32** [31]. This is a small device that has the ESP8266 Wifi module integrated, so that it acts basically as an Arduino with Internet connectivity, for a low price (~5€). It is able to control many digital GPIO, supply 3.3V and, even, has an analogue input. The general pinout is as follows in *Figure 36* [28]

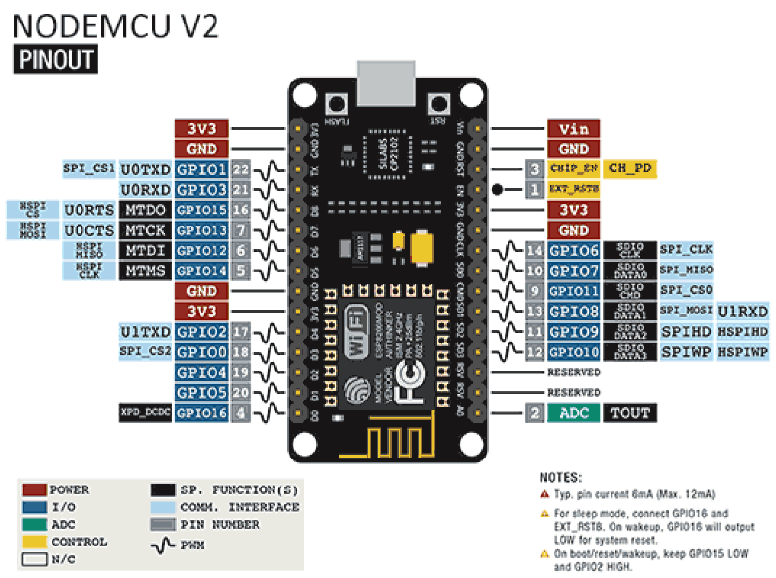

NODEMCU V2
PINOUT
3V3
GND
SPI_CS1 U0TXD GPIO1 22
U0RXD GPIO3 21
HSPI CS U0RTS MTDO GPIO15 16
HSPI MOSI U0CTS MTCK GPIO13 7
HSPI MISO MTDI GPIO12 6
HSPI CLK MTMS GPIO14 5
GND
3V3
U1TXD GPIO2 17
SPI_CS2 GPIO0 18
GPIO4 19
GPIO5 20
XPD_DCDC GPIO16 4
Vin
GND
3 CHIP_EN CH_PD
1 EXT_RSTB
3V3
GND
14 GPIO6 SDIO CLK SPI_CLK
10 GPIO7 SDIO DATA0 SPI_MISO
9 GPIO11 SDIO CMD SPI_CS0
13 GPIO8 SDIO DATA1 SPI_MOSI U1RXD
11 GPIO9 SDIO DATA2 SPIHD HSPIHD
12 GPIO10 SDIO DATA3 SPIWP HSPIWP
RESERVED
RESERVED
2 ADC TOUT
POWER
I/O
ADC
CONTROL
N/C
SP. FUNCTION(S)
COMM. INTERFACE
PIN NUMBER
PWM
NOTES:
Typ. pin current 6mA (Max. 12mA)
For sleep mode, connect GPIO16 and EXT_RSTB. On wakeup, GPIO16 will output LOW for system reset.
On boot/reset/wakeup, keep GPIO15 LOW and GPIO2 HIGH.

Figure 36. ESP 32 Pinout

There are two main benefits on using ESP32 on the actuators. It is clear that, firstly, it allows a wireless communication to the Raspberry Pi, so that an output device such as the irrigation pump does not need a physical connection, as long as the Wifi signal is enough.

Under other view, although the optocoupler isolation may be never-failing for most of the switching, if many commuting channels are used at the very same time, maybe extremely high transients may be induced to the control. This second situation is obviously avoided if the output systems are absolutely isolated from the Raspberry Pi.

On counterpart, a 5V supply is needed for feeding each ESP32, and a relay is also required for each output to be activated. Also a new problem appears, because the relay needs source power. Unfortunately, 3.3V source from ESP32 happens to not supply enough current as to feed not even a 2-channel relay.

For this problem a tricky solution was found. First, notice that ESP32 is able to receive its 5V power source from either its mini-USB or the *Vin* pin. This pin is a parallel connection to the mini-USB voltage. The solution to feed the relay uses *both* connections. On the one hand, the ESP32 is fed via mini-USB, while the *Vin* pin is also used, but not to feed itself -which in this case may even burn the microprocessor-, but to supply the relay VCC. Absolutely no problem on using this solution was found and *Figure 37* shows the mentioned elements connected.

In addition, using the ESP32 in a wireless system requires a server and, perhaps, it should be compatible with Alexa. Precisely this reasoning leads to configuring an MQTT server. All this issues are solved in different chapters of the present project.

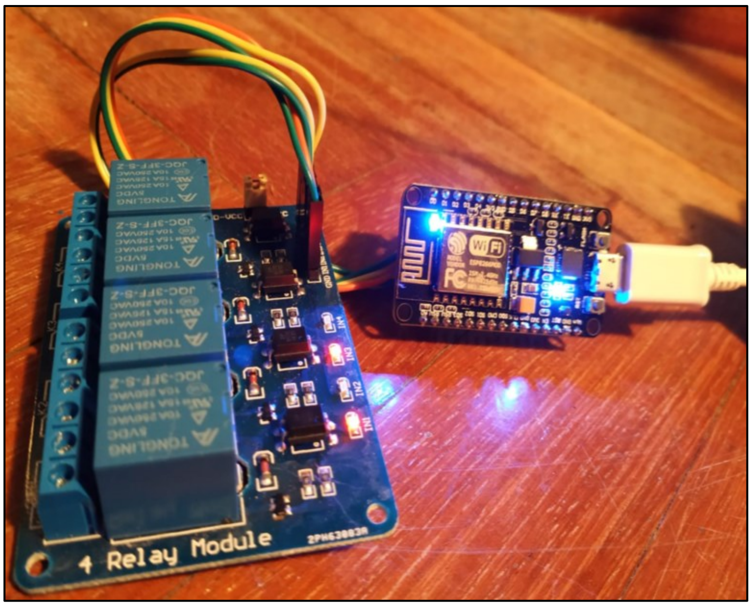


Figure 37. ESP32 and Optocoupler Relay

CHAPTER 5:

SYSTEM DESIGN

The present chapter explains the main characteristics of the developed prototype, its flowchart and the main functionalities and compatibilities of the whole system. It also includes design decisions and their justification in order to properly locate the scope of the project.

5.1 Flowchart

First of all, it should be noted that the Raspberry Pi is the main element of the prototype, as it processes all the sensors' information and provides the system with its network connectivity. The next flowchart corresponds to the Python script that runs on the Raspberry Pi and several considerations may be exposed.

On the one hand, the flowchart exposes the behaviour of the Python code by mentioning its functions, whose names were chosen to be as clear as possible. There is a main flow that starts automatically when the script is executed, which is firstly shown.

Afterwards the ISR's (Interrupt Service Routine) is explained, which is performed on triggering the presence sensor and on pressing *ctrl+c* by the user, respectively. The first case would result in starting the security camera service, while the second would exit the program.

In summary, one main flow and two ISR's are explained in the following flowcharts. In order to have a complete view, the main flowchart starts on the following page.

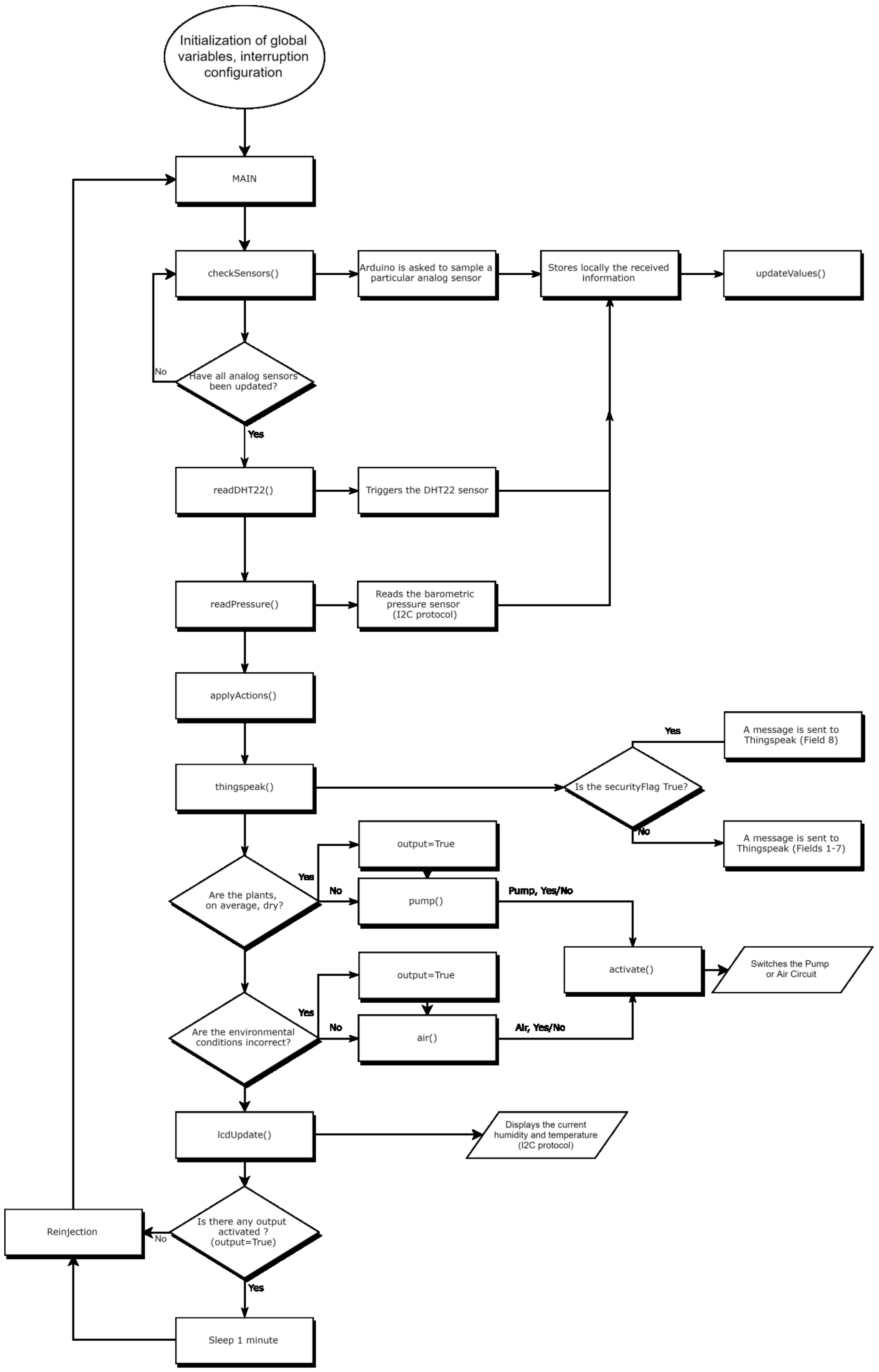

Initialization of global variables, interruption configuration
MAIN
checkSensors()
Arduino is asked to sample a particular analog sensor
Stores locally the received information
updateValues()
Have all analog sensors been updated?
No
Yes
readDHT22()
Triggers the DHT22 sensor
readPressure()
Reads the barometric pressure sensor (I2C protocol)
applyActions()
thingspeak()
Is the securityFlag True?
Yes
A message is sent to Thingspeak (Field 8)
No
A message is sent to Thingspeak (Fields 1-7)
Are the plants, on average, dry?
Yes
output=True
No
pump()
Pump, Yes/No
activate()
Switches the Pump or Air Circuit
Are the environmental conditions incorrect?
Yes
output=True
No
air()
Air, Yes/No
lcdUpdate()
Displays the current humidity and temperature (I2C protocol)
Is there any output activated ? (output=True)
No
Reinjection
Yes
Sleep 1 minute

Figure 38. Prototype Main Flowchart

Figure 38 shows that there are several functions executed when the *MAIN* is invoked. According to the main flowchart, *checkSensors()* is the first function to be called, which asks the Arduino Uno for the value of one of its analog sensors. Afterwards this information is stored and uploaded to an array of sensors in the program, which, in fact, stores for each an *id*, *information* (name) and its *current value*.

Once all the analog sensors configured have been updated, *readDHT22()* is called. This function returns the digital data from the humidity and temperature sensor directly connected to the Raspberry Pi. Next, its values would also be updated.

The last parameter to be sensed is the barometric pressure sensor, which is also connected to the Raspberry Pi GPIO. Its values are obtained via I2C protocol when a call to *readPressure()* is produced.

Afterwards, *applyActions()* is invoked in order to evaluate whether the new sensors' conditions need to change any output state. It is also remarkable to note in this point that *thingspeak()* is updated, so that this platform, explained in detail at the end of the chapter, stores the new values online. In addition, the LCD display updates its values.

If any output device is activated, an instant reinjection to the *MAIN* is produced, in order to check when the environmental characteristics return to the acceptable values and, therefore, turn off the output device. On the other hand, if the climate condition requires no output activation, a slow reinjection to the *MAIN* will occur, which refers to the fact of sleeping for 1 minute before restarting the measurement loop.

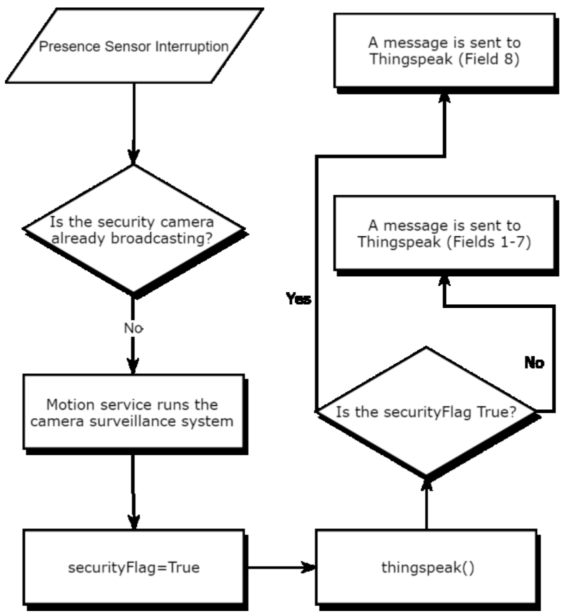

Presence Sensor Interruption
Is the security camera already broadcasting?
No
Motion service runs the camera surveillance system
securityFlag=True
thingspeak()
Is the securityFlag True?
Yes
No
A message is sent to Thingspeak (Fields 1-7)
A message is sent to Thingspeak (Field 8)

Figure 39. Prototype Presence ISR

The Interrupt Service Routine for the Presence Sensor would start broadcasting the security camera in case the Raspberry Pi is not already running that service -it checks a securityFlag-. It also triggers a specific action on sending a Thingspeak message that would instantly notify the user, via e-mail, that the security camera should be revised. This process is described in detail in *5.6 Camera A*, as it does not belong to the Python flow itself.

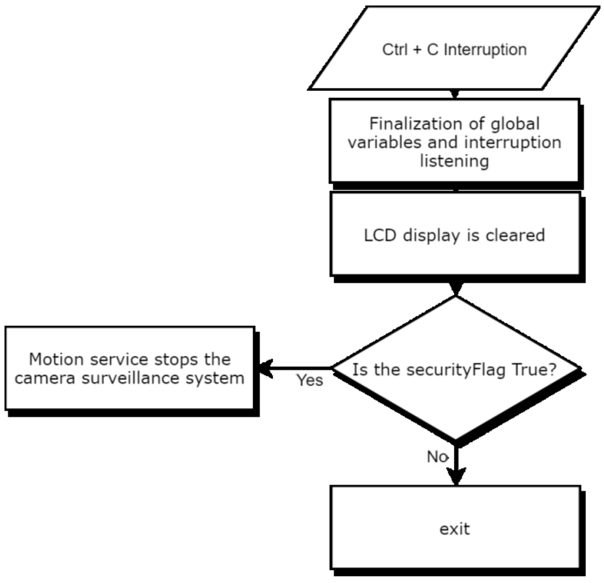

Ctrl + C Interruption
Finalization of global variables and interruption listening
LCD display is cleared
Is the securityFlag True?
Yes
Motion service stops the camera surveillance system
No
exit

Figure 40. Prototype Ctrl + c ISR

Finally, a typical keyboard interruption was integrated for the user to clean the resources that the Python script requires (see *Figure 40*). Moreover, in this case the LCD display would be cleared and the security camera broadcast be stopped. It is important to remark that the Raspberry is able to work without a keyboard, reason for which the present chapter also explains remote control options.

5.2 Arduino Sensing

The context of this section is that the prototype required both analog and digital sensors whose data is processed by the Raspberry Pi. According to the information provided in previous chapters, the digital sensors could directly be linked to the mini-computer. However, Raspberry Pi's GPIO do not support analog signals.

Therefore, several solutions to this problem may be offered. First of all, it could be used an external ADC of multiple channels in order to convert analog signals to digital, so that the Raspberry Pi was compatible to that information. On the other hand, any external device that fulfils this function and delivers the data in a compatible way to Raspberry Pi could also be used.

After the previous considerations in order to incorporate readings from analog sensors, the use of an Arduino was selected as the best choice. This decision required a searching for a method to interconnect it to the main Raspberry Pi system, and two alternatives were thought:

i. The main system could respond in an interruption callback when a periodical measurement was sent, for example, via USB from Arduino. This option could be the fastest and better way to contact a peripheral in general terms, requiring a periodical sample from Arduino.

ii. The main system will ask the Arduino, on considering also using USB connection, for each of its connected sensors, whenever was necessary, and it should check then its inputs' states and return them.

The second option was chosen in order to allow a future development that consists on a *manual supervision* that would precisely require the checking of the state of each sensor at the exact moment the user demands it from the Raspberry Pi, which seems a different approach to that of the interruption routine. Under the selected situation, Arduino acts as a slave on Raspberry Pi demand. However, regarding not using an interruption communication to Arduino, it could be stated that this decision is a personal preference of the author under the reasons explained and may be on considering also using USB connection, open to debate.

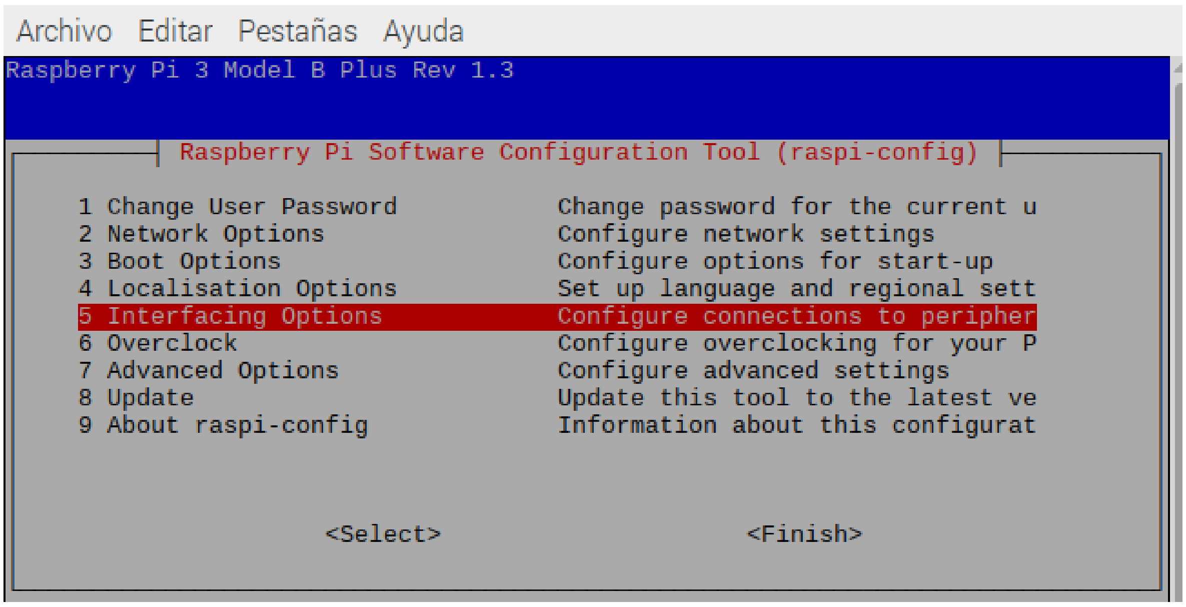
As it was explained in *Chapter 3*, the Arduino Uno is programmed using the Arduino IDE, in **C language**, and sends its information via serial when required. This approach makes the connection much easier. The C program code is simple: when a certain device's state is required, Arduino Uno reads its values, it maps it from 0-100 -so that it is easier to be processed on the Raspberry Pi- and sent it back using the info, id and values stablished by the author in order to identify clearly the communication.

5.3 Remote Control

Nowadays it is almost essential to use functionalities from telematics in any new project. In this sense, many options were configured, each of which has some strong and weak points. First, it is needed to enable some preferences of the Raspberry Pi by typing:

```
sudo raspi-config
```

Then, *Interfacing Options* should be selected (see *Figure 41*). In this page SSH and VNC should be enabled for remote purposes (see *Figure 42*). Also I2C needs activation, as both the pressure sensor and the LCD display requires it.



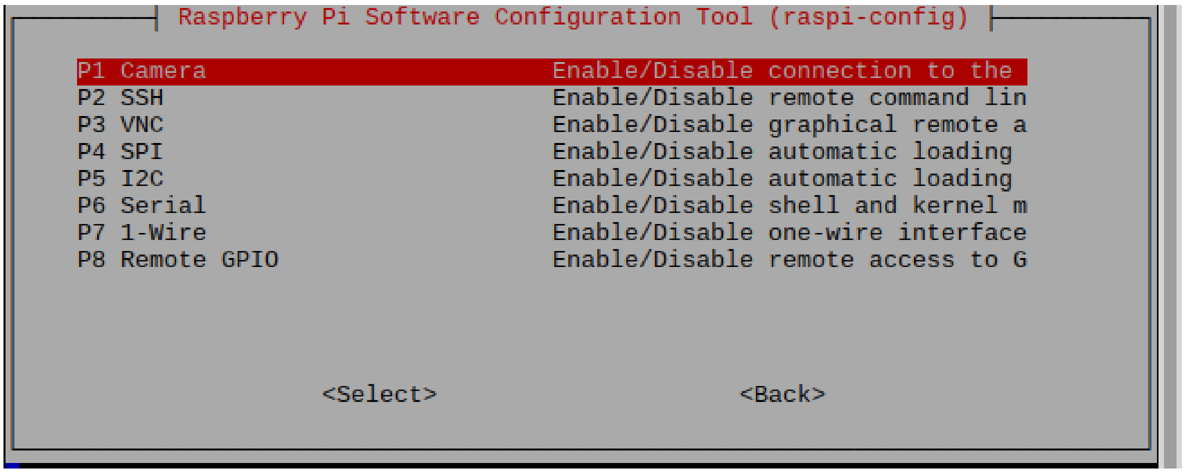
```
Archivo  Editar  Pestañas  Ayuda
Raspberry Pi 3 Model B Plus Rev 1.3

        ┤ Raspberry Pi Software Configuration Tool (raspi-config) ├

    1 Change User Password          Change password for the current u
    2 Network Options               Configure network settings
    3 Boot Options                  Configure options for start-up
    4 Localisation Options          Set up language and regional sett
    5 Interfacing Options           Configure connections to peripher
    6 Overclock                     Configure overclocking for your P
    7 Advanced Options              Configure advanced settings
    8 Update                        Update this tool to the latest ve
    9 About raspi-config            Information about this configurat

                   <Select>                    <Finish>
```

Figure 41. Raspi-config First Menu



```
        ┤ Raspberry Pi Software Configuration Tool (raspi-config) ├

    P1 Camera                       Enable/Disable connection to the
    P2 SSH                          Enable/Disable remote command lin
    P3 VNC                          Enable/Disable graphical remote a
    P4 SPI                          Enable/Disable automatic loading
    P5 I2C                          Enable/Disable automatic loading
    P6 Serial                       Enable/Disable shell and kernel m
    P7 1-Wire                       Enable/Disable one-wire interface
    P8 Remote GPIO                  Enable/Disable remote access to G

                   <Select>                    <Back>
```

Figure 42. Raspi-config Second Menu

Static IP

Once configured the Raspberry could be accessed remotely from other computer and, therefore, controlled its information remotely. At this point it will only be visible to the same LAN network, but afterwards it will be also exposed to the Internet. For SSH to work in the local network, the Raspberry Pi's IP should be known. Also, it is highly recommended that an static IP is set forever. In case the device also serves to the Internet, this is almost an unavoidable requirement.

The commands for this purpose are:

sudo nano /etc/dhcpcd.conf

This file should contain the following:

interface wlan0

static ip_address=192.168.0.157 // Raspberry Pi's new Static IP

static router=192.168.0.1

static domain_name_servers=192.168.0.1

The configuration may require internet services to restart or a reboot to take actions. Static IP may be checked with:

ifconfig wlan0

or, alternatively,

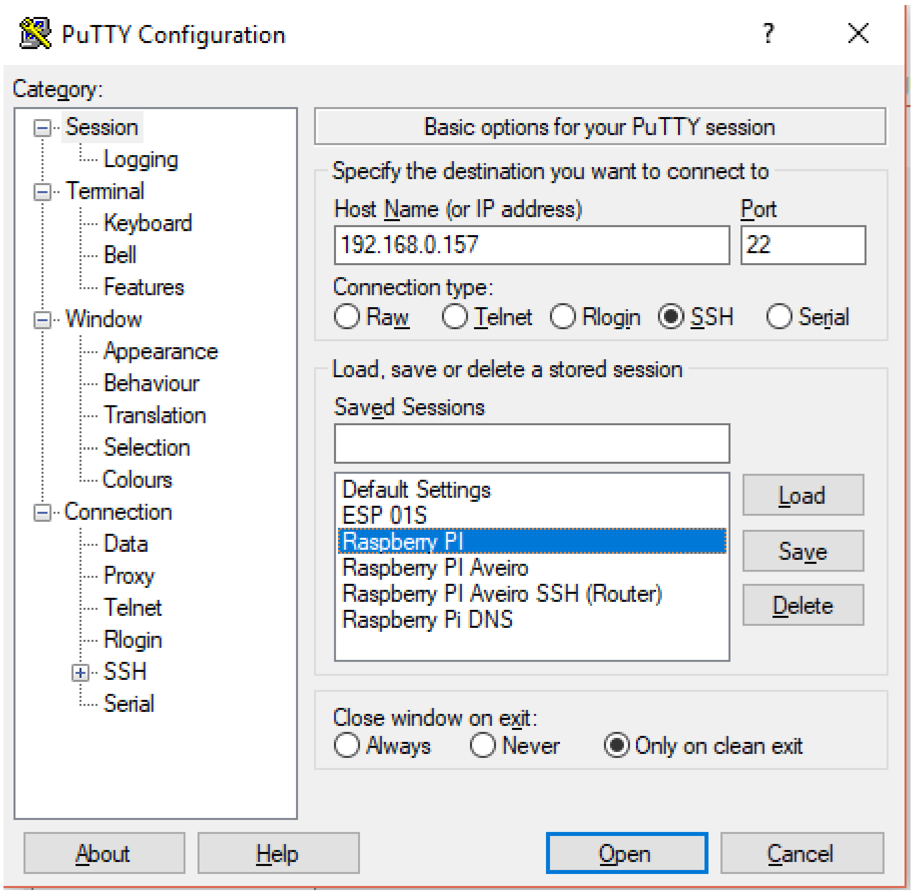
hostname -I

Note that if any other device in the network asks for a connection before the Raspberry Pi does, the router may assign it the Raspberry Pi's Static IP, as there is no IP reservation. For this reason it is appropriate to set the static IP to any of the extreme values in the available IP range, as they are less likely to be used.

SSH

If the previous configuration is followed, Raspbery Pi could be now accessed:

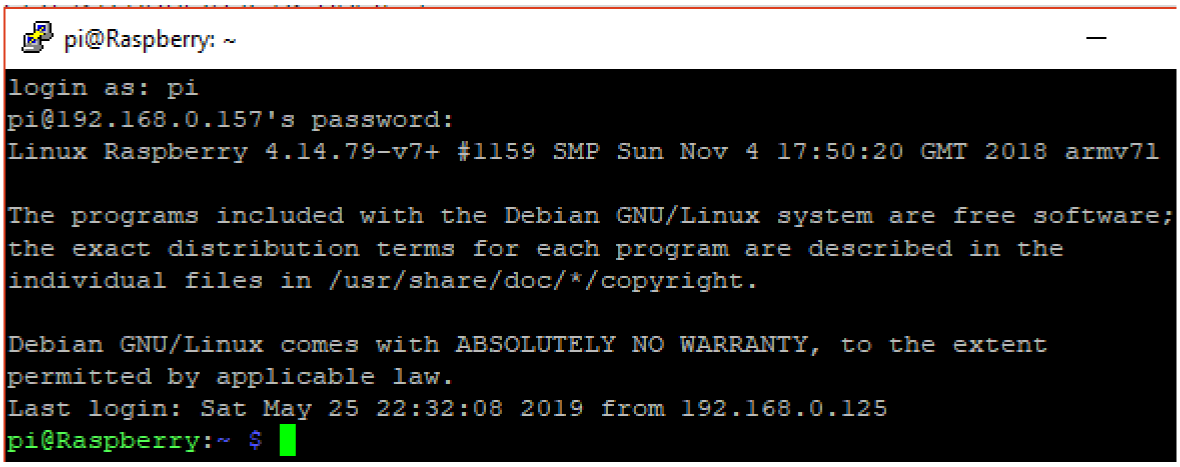
i. From Linux: A terminal should be opened and typed:
   ssh pi@192.168.0.157
ii. From Windows: Putty program should be installed. The configuration is shown in *Figure 43*:



Specify the destination you want to connect to
Host Name (or IP address): 192.168.0.157
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
ESP 01S
Raspberry PI
Raspberry PI Aveiro
Raspberry PI Aveiro SSH (Router)
Raspberry Pi DNS
Close window on exit: Always Never [✓ Only on clean exit]

Figure 43. Remote Monitoring using Static IP

SSH implements security methods and is, in fact, the only remote method recommended to be used over Internet. It is limited to using commands from a remote terminal, which is enough in order to execute tfg.py or do any other checking. However, operations that required viewing the graphical interface are really difficult in this way. SSH remote connection is shown in *Figure 44*.



```
login as: pi
pi@192.168.0.157's password:
Linux Raspberry 4.14.79-v7+ #1159 SMP Sun Nov 4 17:50:20 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat May 25 22:32:08 2019 from 192.168.0.125
pi@Raspberry:~ $
```

Figure 44. Remote Monitoring using Static IP

VNC

The Virtual Network Computing allows to access remotely to the very same instance that is running on the Raspberry Pi. Therefore, the image information is also transmitted and the mouse could be used. It requires a server (computer to be remotely commanded) and a client configuration (external computer).

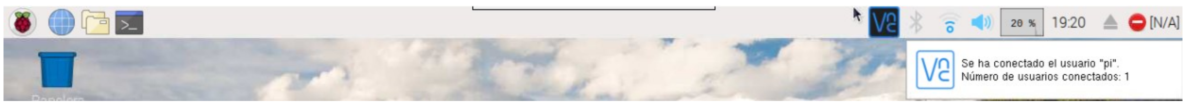
VNC Server in Raspberry Pi:

sudo apt-get remove xrdp vnc4server tightvncserver //In case it was installed
sudo apt-get install tightvncserver
sudo apt-get install xrdp

sudo apt-get update //Upgrades system's package
sudo apt-get dist-upgrade //Upgrades all installed packages

sudo apt-get remove realvnc-vnc-server
sudo apt-get remove real-vnc-viewer //In case it was already installed
sudo apt-get install realvnc-vnc-server //Raspberry Pi serves
sudo apt-get install real-vnc-viewer //Raspberry Pi could even view others VNC connections as a client

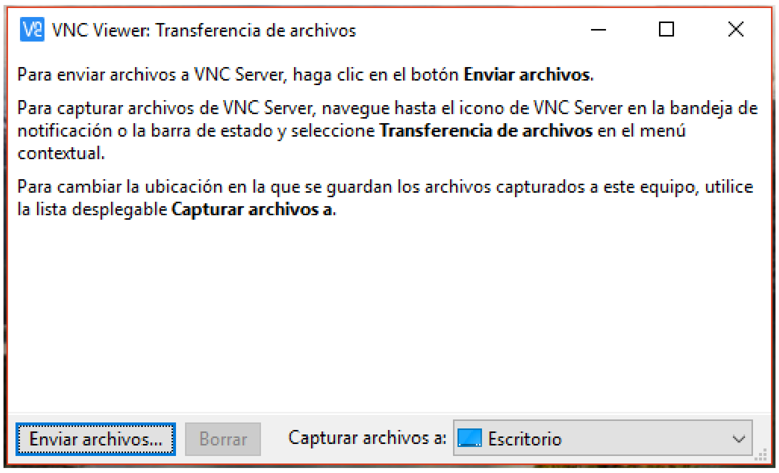
In order to connect to the VNC service, VNC Viewer (client) should be opened in the remote computer. On supposing the previous static IP example, 192.168.0.157 would be used. The default port for this connection is **5900**.



Se ha conectado el usuario "pi".
Número de usuarios conectados: 1

Figure 45. Detail on VNC Connection

VNC offers a low-latency view control that works well for configurations on the same local network. Moreover, it is possible to easily transfer files to the remote computer:


VNC Viewer: Transferencia de archivos

Para enviar archivos a VNC Server, haga clic en el botón **Enviar archivos**.

Para capturar archivos de VNC Server, navegue hasta el icono de VNC Server en la bandeja de notificación o la barra de estado y seleccione **Transferencia de archivos** en el menú contextual.

Para cambiar la ubicación en la que se guardan los archivos capturados a este equipo, utilice la lista desplegable **Capturar archivos a**.

Figure 46. VNC File Trasfer

In conclusion, the Raspberry Pi is configured so that it is accessible to any device on its local network, under the typical user and password restriction. On the one hand, **SSH protocol** is useful for the *main review tasks* and it is stablished means terminal (Linux) or **PuTTY** (Windows). Although it is limited to *sending terminal commands*, it provides the *safest connection* even if linked through Internet. Some security methods are explained later.

On the other hand, **VNC** option allows to remotely view and command the very same instance that the Raspberry Pi would *show in screen*. Therefore, the VNC client is able to control the whole computer environment –with still certain restrictions-. This configuration exposes the system to dangers if connected through an external network, but it is easier to work, as any operations could be done exactly the same way as *in situ*, with even a file transmission option.

For monitoring and telematics access from Internet a **DNS provider** is configured, **no-ip.com**, and a domain name was reserved: **gardenhost666.ddns.net**.
Once again, it is vital to have a static IP for the Raspberry Pi configured.

After the following section about DNS configuration, the Raspberry will be visible to the Internet and accessible via its domain name, if the SSH port is requested with the correct password.

NO-IP Configuration (DNS)

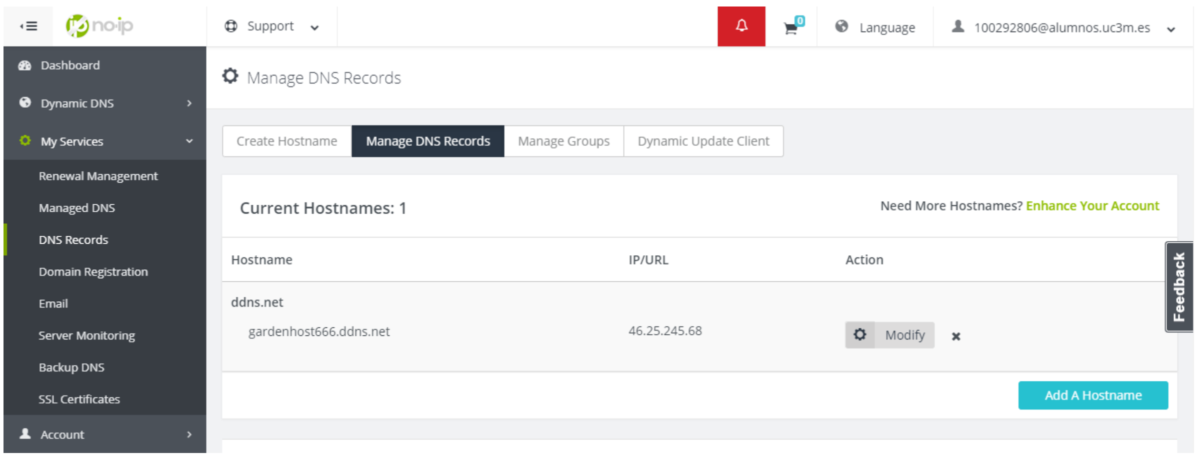
No-ip service allows the *Raspberry Pi* to act as a server and provide its multimedia and information. Instead on referring to our device as 192.168.0.157 as in the internal network, it could onwards be also accessed from anywhere else using an URL:

https://gardenhost666.ddns.net

Routers along the Internet network would broadcast a message asking for the IP which resolves that address. Eventually someone may have a route to it. Then its public IP is forwarded back. However, this public IP suffers periodical changes, for both traffic fluctuations as well as for security reasons. This fact increases complexity, as the *Raspberry Pi* needs to be accessed from anywhere and anytime but may have different public IP each time.

The exposition of this problem clearly shows that there is a difficult requirement: **Public Raspberry Pi's IP Address should be accessed without knowing its IP**. (ie. 46.25.245.68, at a time). No-ip throws its IP on a URL petition.

For this reason it is also needed to relate the https://gardenhost666.ddns.net traffic from our router to the Raspberry Pi's local IP. This step may forwards the communication from a given Internet port to any port on the Pi. Many people use different ports relationships (ie. **1883**, typical MQTT port (Router) **->** **1983** (Raspberry Pi)), so that security on our device increases and the chance of being intercepted is less likely.

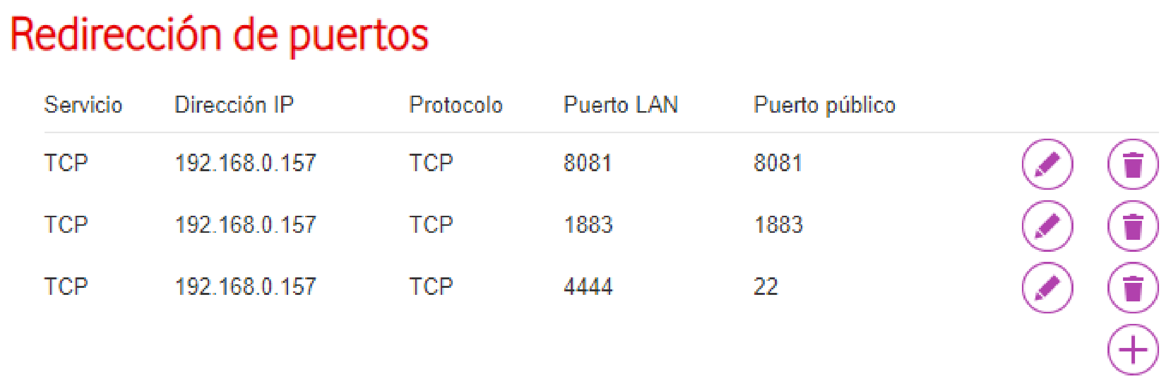


Current Hostnames: 1

Hostname	IP/URL	Action
ddns.net		
gardenhost666.ddns.net	46.25.245.68	Modify

Figure 47. DNS Web Domain for the System

Therefore, DNS via No-ip allows any device to communicate to the Raspberry. However, it is still necessary to allow some forwarding rules on our router. As said, these are simply permissions so that traffic coming to a certain public port could be forward to a certain Raspberry Pi port. This association requires **a fixed IP**, which, in this case, was 192.168.0.157.



Redirección de puertos

Servicio	Dirección IP	Protocolo	Puerto LAN	Puerto público
TCP	192.168.0.157	TCP	8081	8081
TCP	192.168.0.157	TCP	1883	1883
TCP	192.168.0.157	TCP	4444	22

Figure 48. Port Forwarding Configuration

The previous screenshot of *Figure 46* shows the needed forward configuration for the Raspberry Pi acting as a server of multiple services. Each of the forwarded port has a use in this project:

Raspberry Pi Services

PORT NUMBER	FUNCTION	DESCRIPTION
8080 (Default)	**Security Camera**	**Configuration Options**
8081	**Security Camera**	**Fullscreen**
1883	**MQTT Server**	**Wireless Compatibility**
22*	**SSH**	**Raspberry Control Over Internet**

*Forwarded to 4444 on Raspberry Pi for security reasons. SSH default port is 22.

Table 3. Prototype Port Services

However, for this reason, it is one of the first ports to be knocked by Internet intruders. This redirection would forward, then, incoming traffic with destiny gardenhost666.ddns.net:22 to 192.168.0.157:4444. Of course, Raspberry Pi should change its SSH port in order to listen to the new 4444.

As stated above, each of the ports supplies a different service. Then, listening at https://gardenhost666.ddns.net on public port 8081 would return the on-live system's security camera at fullscreen, while listening on 8080 would show a little screen and some basic configuration about it. Moreover, messages on 1883 would respond to MQTT server, which is exposed in *Chapter 6*, and SSH will listen to port 4444 connections.

In order to set the Raspberry Pi linked to the No-IP DNS service, it is needed to first create an account on this platform. Afterwards, the following instructions should be typed in a Raspberry Pi terminal:

```
sudo bash
cd /usr/local/src/
wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
tar xf noip-duc-linux.tar.gz
cd noip-2.1.9-1/
make install
```

At this point the email and password used in the No-IP account are asked. Then:

```
n // no script run at update
sudo nano /etc/rc.local //The server needs to run at Raspberry Pi boot
```

Inside this editor, in previous line to the exit 0, type:

```
sudo /usr/local/bin/noip2
```

It is fundamental to note that the use of a DNS and, in general terms, exposing the Raspberry Pi to the Internet access presents some serious drawbacks that should be carefully considered. Therefore, some Internet security suggestions are included in the following chapter, whose purpose is to protect the system to any stranger's control as much as possible.

5.4 Internet Security

Due to the vulnerability introduced by exposing the system to potential intruders' control, some security methods are explained in this section. However, notice that it is always better to have local control as much as possible.

Suggestions:

- **Not allowing Root login over SSH**.
  This will make that *sudo* commands are not available via SSH. In case we are really interested, we may edit the following file using:
  sudo nano /etc/ssh/sshd_config

  The sshd_config is the configuration file for ssh server process, which is executed on boot. There we can set "PermitRootLogin no" (strong security, protects main functions) or "PermitRootLogin yes" (root functions are now also available for remote control). This file also sets the SSH port (Default: 22). Finally, it is needed to reboot the Raspberry Pi and, in case there is not, set a root password:
  sudo passwd my_password

- **Using different ports in the public and local connection.**
  As said before, SSH port was changed to 4444, which is a very easy practice to defend a system exposed to internet, as usually no one would search for every single possible port on searching for a response.

- **Installing software to monitor undesired logins and block IP's**.
  Fail2ban checks for log files and notifies if there is too many password failures, among other options.

- **Restricting valid incoming IPs to only those of a given country**

- **SSL (Secure Socket Layer, or Transport Layer Security) certificates**, whose purpose is to authenticate sites and use symmetric encryption

- **Port Knocking**: This is a more sophisticated security method. According to its name, this action means that connection will only be accepted if, in a very limited amount of time, certain ports are knocked in a particular order. In this way, the port to be protected (ie. Port 22 (SSH)) will be closed unless the correct sequence of ports is called. In addition, each port call may use a different protocol, which increases a lot its security.

5.5 Voice Control

The latest advances on IOT of course have lot to do with Alexa and Google Home global strategies. Their point is simple: voice is the easiest instruction to use. In this way, different devices may be turn on/off by means of a calling to an identifier (device’s name).

This voice recognition is not new itself, as many systems existed before. The originality is that these instructions are processed *on their servers* and, finally, responses may trigger the requested actions on our houses. Domotics and industrial implementations, usually a part of a full system, make voice control systems to be a rising market.

Alexa

As it was explained in *2.2 Criticism to the State of Art*, this project uses the Alexa voice-assistant. Currently there are many functionalities and compatible devices to Alexa, which also justifies its inclusion in the project. The model selected is the ***Amazon Echo Dot (2nd generation)*** [32], because it is able to process English instructions. On the contrary, 3rd generation, at least in Spain, only responds to commands in Spanish.

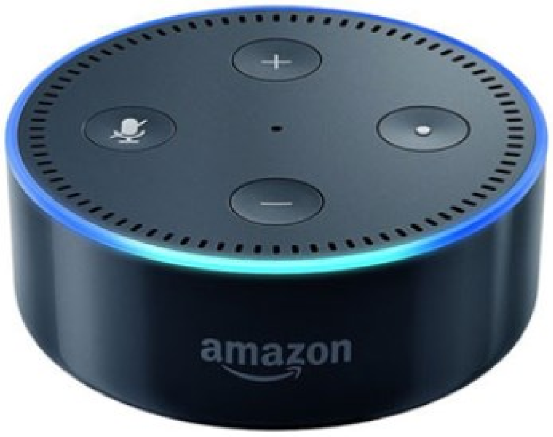


Figure 49. Amazon Echo Dot (2nd generation)

The main technical difference between the second and third generation is that the last version uses *four* far-field mics, which improves voice recognition at different angles and at a greater distance. Instead, second generation uses just *one* mic, settled at the centre, as illustrated in *Figure 49* [29]. Even so, no relevant difficulty was found on communicating to second-generation Alexa.

A research found out several Alexa implementations:

1) **Alexa + Relay (Raspberry Pi)**

A system may be able to turn on/off the different outputs of a relay connected to the Raspberry Pi by means of a Python script that listens to Alexa requests. It registers virtual devices, so that each one of the relay's outputs has a different name, and reacts to different Alexa messages. Although this option is not explored, a code referred to Github [33] that implements it was found. On a quick consideration of this reference, the name the Raspberry Pi reacts could be easily changed with the following lines:

TRIGGERS = {"garden": 52000}
if name=="garden": #Action to be applied (function)

Now, if "Alexa, discover devices" is said, it should respond with "Discovery is completed. I've found one smart home device. Try saying "turn on garden"". Finally, "Alexa, turn on garden" would be able to activate a certain Raspberry Pi GPIO, where a channel relay is supposed to be connected.

On a general view, this example was not explored further, as the prototype links to the relay in a wired connection, but introduces the idea of an ID/trigger/name that is able to respond to Alexa requests.

2) **Alexa + ESP32 + MQTT server + Relay**

The most important and complex option in this project is this one. On one side, it uses ESP32 devices which, as explained before, are wireless solutions that may be operated on sending MQTT messages and are programmed in C. On the other, Alexa should also be able to communicate to them, so a double requirement is presented.

The base code [34] shows how to use ESP devices to subscribe to an MQTT topic and, therefore, could be compatible with *Setapp*, the developed Android App that also triggers different devices.

The final ESP32 code developed was called "*Alexa_MQTT_ESP32.ino*" and reacts both to MQTT messages and Alexa requests. The main point is:

const char* MQTT_LIGHT_COMMAND_TOPIC = "garden"; // This is the topic that ESP32 listens to change its state.
const char* MQTT_LIGHT_STATE_TOPIC = "gardenAnswer"; // This is the topic where the ESP32 publishes its activity, each time a change is produced

This proposal will be fully understood in *Chapter 6*: *Android APP and MQTT Protocol Server*. In order to activate the ESP32, this MQTT chapter will provide further answer and exact instructions.

5.6 Camera and Surveillance

Both for personal or industrial uses, a video surveillance system could not be forget in this project. A-B comparison was offered with two very different proposals. On the one hand, a free camera server was stablished via the software *Motion* and a typical USB camera. Opposite, a camera controlled via App was also installed. A comparative review could be found at the end of this part.

Camera A

This system uses any USB camera but requires an important initial configuration as well as the previous DNS provider.

Initially, the USB camera may be plugged in. The image can be checked by installing **Luvcview**, a free camera viewer for Linux. The application that configures camera options and well as the broadcast port (8080) is called **Motion**, a free and open-code module for GNU and Linux. This camera could be found and operated in the network by tying:

http://192.168.0.157:8080, (Local network)
http://gardenhost666.ddns.net:8080 (Internet)

This URL will resolve showing the camera view as well as some configuration. When "All" is pressed some views are expanded, but the full system is designed with the configuration file ***/etc/motion/motion.conf.***

LUVCVIEW INSTALLATION:

lsusb //It allows to see if the USB camera is detected
sudo apt-get install luvcview
sudo **luvcview**
sudo luvcview -s 1080x720 //Sets a different resolution

MOTION SETUP:

sudo apt-get install motion
sudo nano /etc/motion/motion.conf //Configuration file
Next, the configuration should be change to include the following:
# Start in daemon (background) mode and release terminal (default: off)
daemon on //Runs service on boot
# Restrict stream connections to localhost only (default: on)
"stream_localhost" off ->M
stream_port 8081 //Defines the fullscreen port

Lastly, there is a small configuration to unable motion broadcasting on start:

sudo nano /etc/default/motion
start daemon=no //Broadcast does not start on boot

For simplicity, it is recommended to reboot at this moment the Raspberry Pi.

The service could now be started:

> **sudo service motion start** //Unnecessary now, as service starts on boot
> **sudo motion** //Broadcast starts

The reason for *not* broadcasting on startup is that, for security reasons, it is not recommended to have a security camera permanently working. Of course, nothing may occur and, hopefully, the security walls are enough. Moreover, there is no need to waste Raspberry Pi's energy into a service that is rarely used.

The surveillance system, therefore, would work as follows:

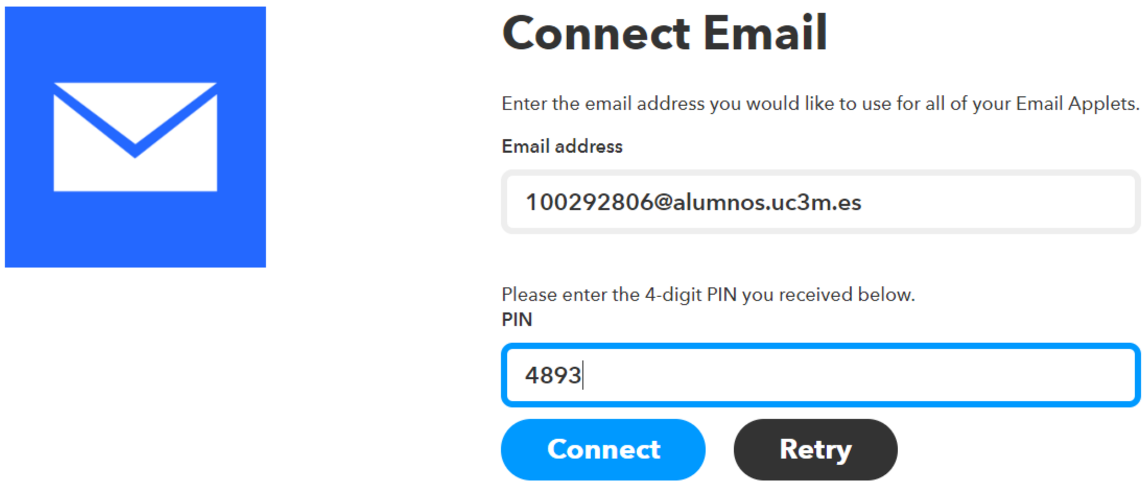
i. Motion service is initially running but *it does not broadcast*
ii. The PIR presence sensor is set so that, if activated, *it triggers a global interruption* in the Python script
iii. This code calls the system and requires to *start motion broadcasting*
iv. The event is notified to *Thingspeak* register, which *reacts* and allows IFTTT service, explained now, to send a pre-configured email with the camera URL

Smart functionality starts here. IFFF stands for "*If this then that*", and is a service that allows the interconnection of devices in many ways. When any trigger is configured ("this"), an action applies, as to pop-up a notification from the IFFF app mobile phone or activate a device via Alexa.

Figure 50. IFTTT Service

IFTTT (*If This Then That)* is an external service that allows the connection of systems. The target of this service in this project is to send an alert email, but it is also able to do different automatic tasks that the author considers not to develop further, due to being distant to the present focus. For the purpose of sending that email, *Thingspeak*, platform used to register device's activity explained at the end of this chapter, is used as bridge.

The best way to understand this flow and configuration is by using screenshots of the different steps. First, it is needed to connect IFTTT to our email:


Connect Email

Enter the email address you would like to use for all of your Email Applets.

Email address

100292806@alumnos.uc3m.es

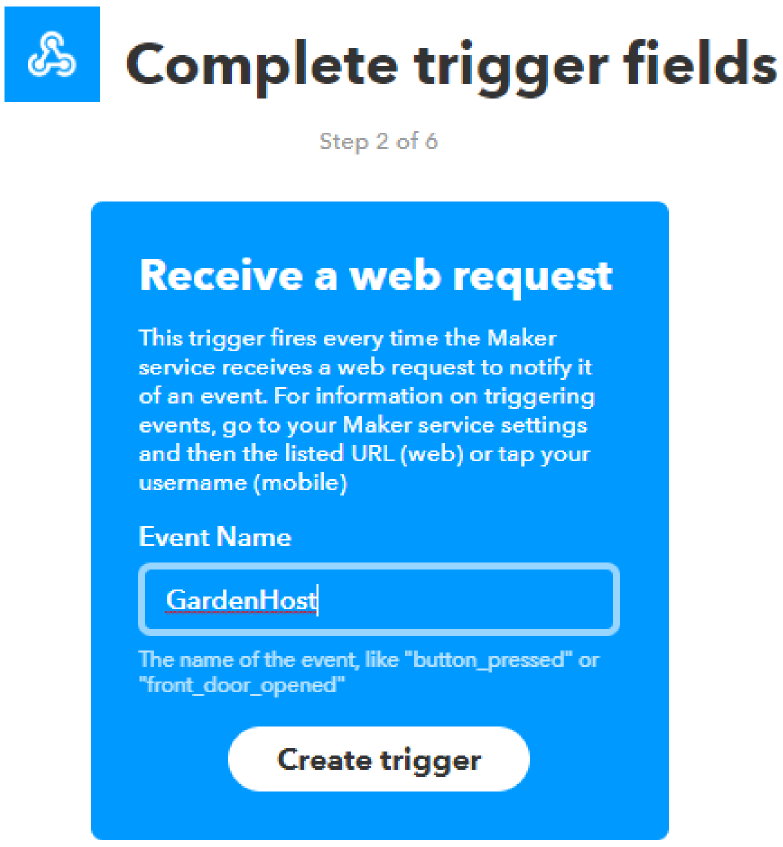
Please enter the 4-digit PIN you received below.

PIN

4893

Figure 51. IFTTT Email Configuration (1)

Next, a *this* is chosen, which is the service that will *trigger* an action. In this case, it is a web request from Thingspeak. A name needs to be provided.


Complete trigger fields

Step 2 of 6

Receive a web request

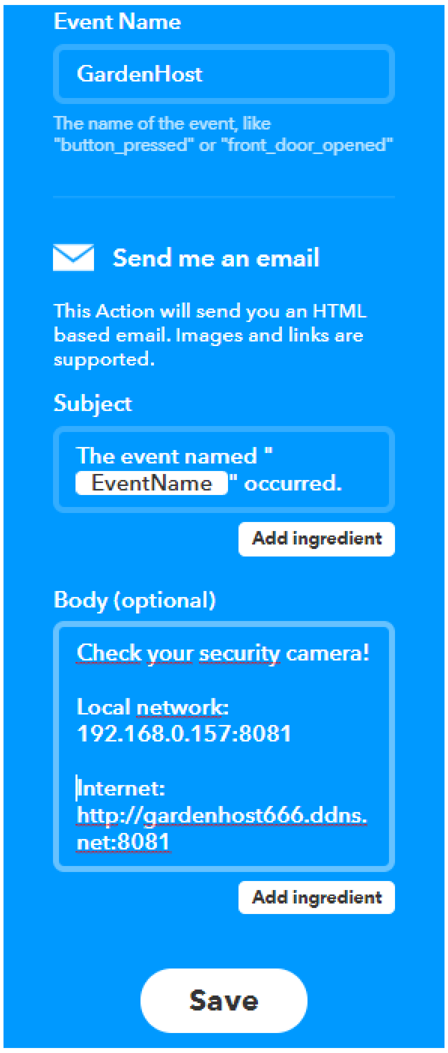
This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Event Name

GardenHost

The name of the event, like "button_pressed" or "front_door_opened"

Figure 52. IFTTT Web Request


Event Name

GardenHost

The name of the event, like "button_pressed" or "front_door_opened"

Send me an email

This Action will send you an HTML based email. Images and links are supported.

Subject

The event named " EventName " occurred.

Body (optional)

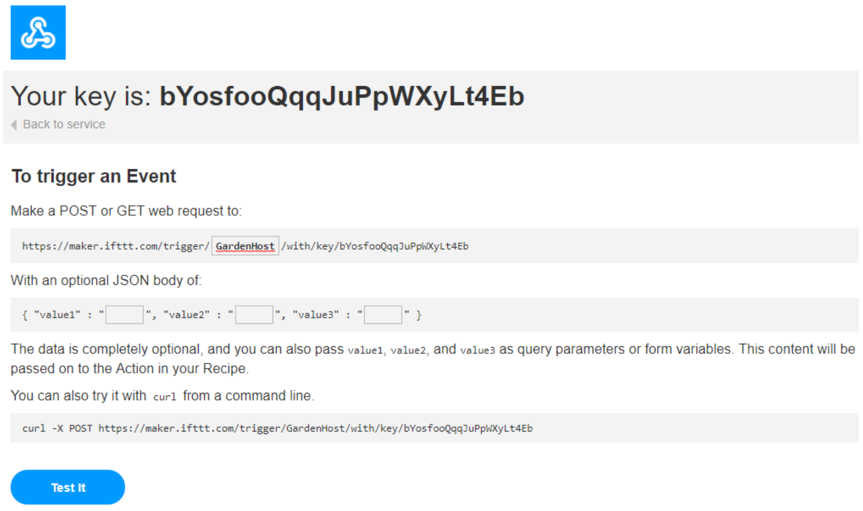
Check your security camera!

Local network:
192.168.0.157:8081

Internet:
http://gardenhost666.ddns.net:8081

Figure 53. IFTTT Email Configuration (2)

Afterwards, a *that* is set: when the previous condition occurs an email is sent. The information in *Figure 54* shows a URL to be used in a POST/GET request so that this flow is triggered. Therefore, this is the direction that Thingspeak would use to trigger IFTTT response.


Your key is: **bYosfooQqqJuPpWXyLt4Eb**

To trigger an Event

Make a POST or GET web request to:

```
https://maker.ifttt.com/trigger/ GardenHost /with/key/bYosfooQqqJuPpWXyLt4Eb
```

With an optional JSON body of:

```
{ "value1" : "      ", "value2" : "      ", "value3" : "      " }
```

The data is completely optional, and you can also pass `value1`, `value2`, and `value3` as query parameters or form variables. This content will be passed on to the Action in your Recipe.

You can also try it with `curl` from a command line.

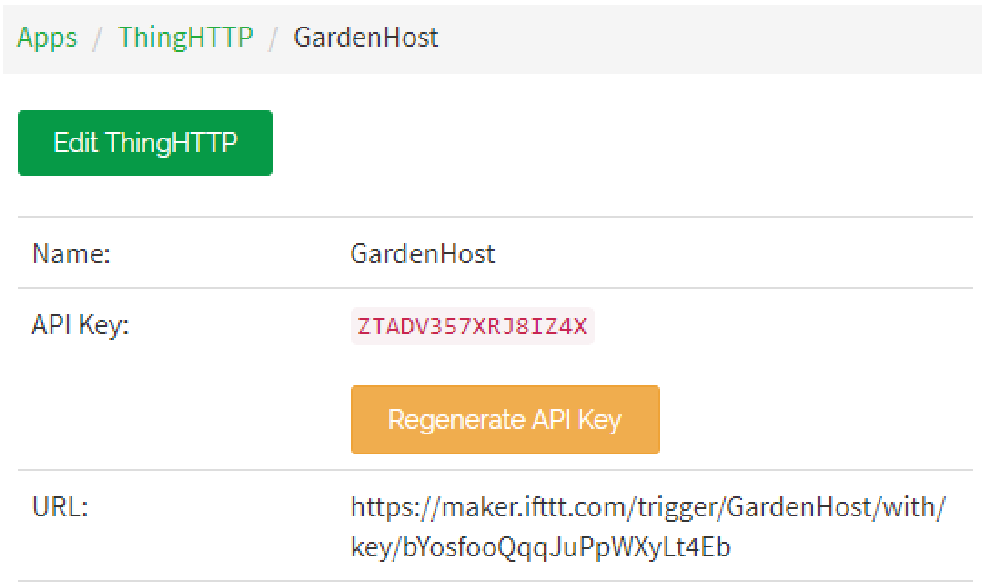
```
curl -X POST https://maker.ifttt.com/trigger/GardenHost/with/key/bYosfooQqqJuPpWXyLt4Eb
```

Figure 54. IFTTT Example Key

In summary, IFTTT configuration would respond its trigger event on sending the email above written. In the following page Thingspeak would be configured so that, if a certain condition from the presence sensor field is met, a message is sent to the following:

[https://maker.ifttt.com/trigger/GardenHost/with/key/bYosfooXXXqqPpXXXyLt4Eb]

Thingspeak needs to know first which action to do, which, in this case is sending an HTTP message to IFTTT. The second requirement is to also set the condition (*react*) which needs to be fulfilled so that it happens.

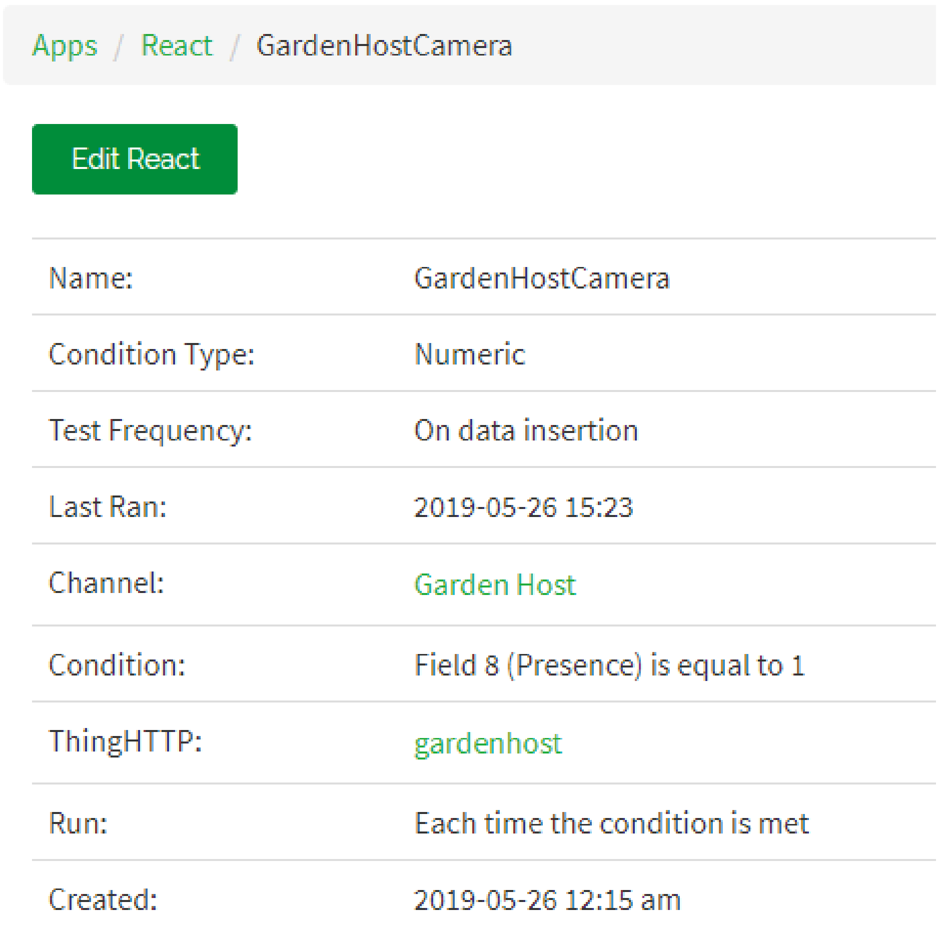

Apps / ThingHTTP / GardenHost

Edit ThingHTTP

Name:	GardenHost
API Key:	ZTADV357XRJ8IZ4X Regenerate API Key
URL:	https://maker.ifttt.com/trigger/GardenHost/with/key/bYosfooQqqJuPpWXyLt4Eb

Figure 55. Thingspeak Action: Sending HTTP Request

Then the reaction is configured:

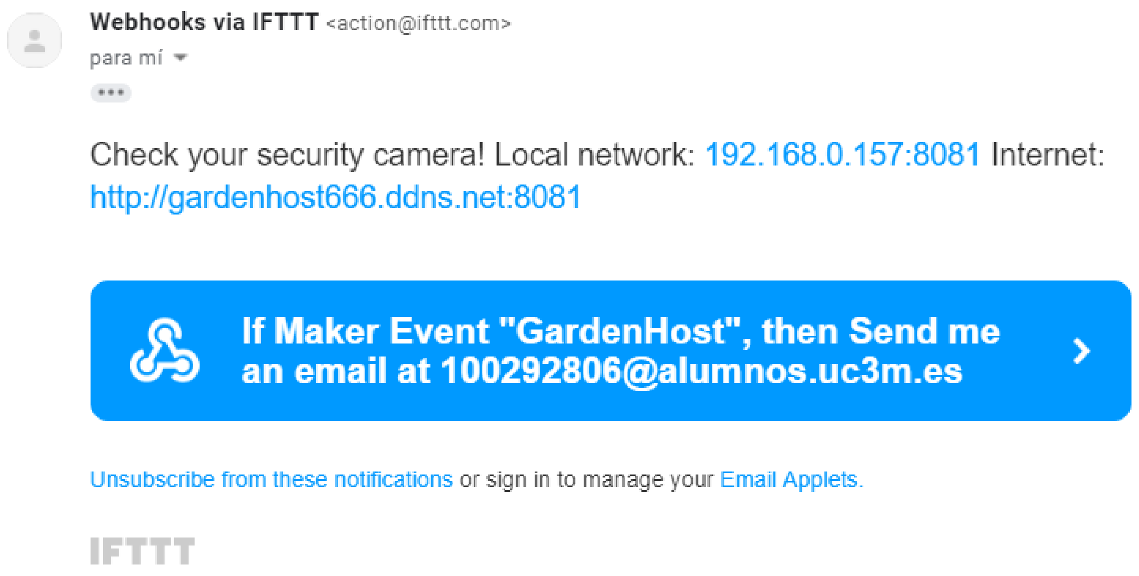


Apps / React / GardenHostCamera

Name:	GardenHostCamera
Condition Type:	Numeric
Test Frequency:	On data insertion
Last Ran:	2019-05-26 15:23
Channel:	Garden Host
Condition:	Field 8 (Presence) is equal to 1
ThingHTTP:	gardenhost
Run:	Each time the condition is met
Created:	2019-05-26 12:15 am

Figure 56. Thingspeak React: Condition to be met

After all this whole process, *any time the presence sensor detects an obstacle the security camera will start broadcasting and an email as the following will be received*:



Webhooks via IFTTT <action@ifttt.com>
para mí

Check your security camera! Local network: 192.168.0.157:8081 Internet: http://gardenhost666.ddns.net:8081

If Maker Event "GardenHost", then Send me an email at 100292806@alumnos.uc3m.es

Unsubscribe from these notifications or sign in to manage your Email Applets.

IFTTT

Figure 57. IFTTT Received Email

To sum up, Camera A proposal needed an USB camera, the installation of the Motion service, which does not broadcast on boot, and the previous No-ip DNS. The main system has an interruption code that is read whenever the presence sensor detects a body and consists of a call to the system in order to start serving the security camera. Also, it requests to write a 1 to the Thingspeak channel, in field 8. Its *react* will listen this 1 and act sending a specific HTTP request to IFTTT. At the same time, it would receive this intention and, finally, the email will be sent.

Notice that, of course, as the IFTTT can be triggered with a HTTP message, it is not necessary to *bridge* over the Thingspeak platform. However, not using the exposed final option was discarded as, in this case, there would be no Thingspeak record. Also, IOT has no problem on combining technologies, so this excuse seems perfect to design a global interconnected system.

Camera B

The *Sricam Indoor IP Camera* [35] was studied. In order to work it needs an Android APP to be downloaded, so that the view is only available by means of a mobile phone/tablet. This proposal has some strong points, as a much quicker configuration by means of using an *id*, on the app. Moreover, it has a network port to be directly connected to the router, 10 IR LEDS so that night view is also acceptable, remote movement commands of the camera and bidirectional mic communication, so that the device can both hear and forward our voice, captured on the app. Finally, it has even the option of triggering a sound alarm in case any detection is produced. The interface is as included:

In general terms, it is a cheap camera that *does* provide many of the imaginable functions. However, no integration to any system seems possible and the view is transmitted by means of using external servers that may even store the full communication.

Comparison

On a general view, it could be said that:

Camera A: Uses the free software Motion to broadcast over the Internet. It requires some knowledge and specific configuration on Raspberry Pi as well as on the local router (*port forwarding*). This option offers full control of the stream and integrates easily to a global system. Moreover, the design for this camera allows to enable the server at a specific action, e-mail notified, so bandwidth is saved and privacy more protected.

Camera B: It incorporates lots of functionalities, such as voice communication. Its configuration is easy to follow. However, it is only viewed on its particular Android App and there is no control over the stream. Unfortunately, there is no easy implementation of this device into a whole system.

On reading both options it is clear that the **Camera A system** is the one implemented to the main system, as needing to be part of a communicated system and being an independent module are the requirements for this project. From a Law point of view, it is also important to remark that it would be more difficult to use Camera B under sales purposes.

5.7 AUTOBOOT

Autoboot (*Daemon*) refers to an automatic system start when the Raspberry Pi is turn on. This means that there are several services, modules or actions that may be desired to start on its startup, as the MQTT service. The following lines explain how this file opens and what to write in order to execute a program. Note that there is also a similar file for ssh sessions.

Runs a program on startup:

cd /etc

nano rc.local

Write (as example):

(sleep 10; python scriptname.py) & //Here *scriptname* is under etc folder

However, any other directory may be pointed, as in:

/home/pi/scriptname, por ejemplo

Runs a program on start of ssh session:

cd/home/pi

sudo nano .bashrc

At the end of the document, type:

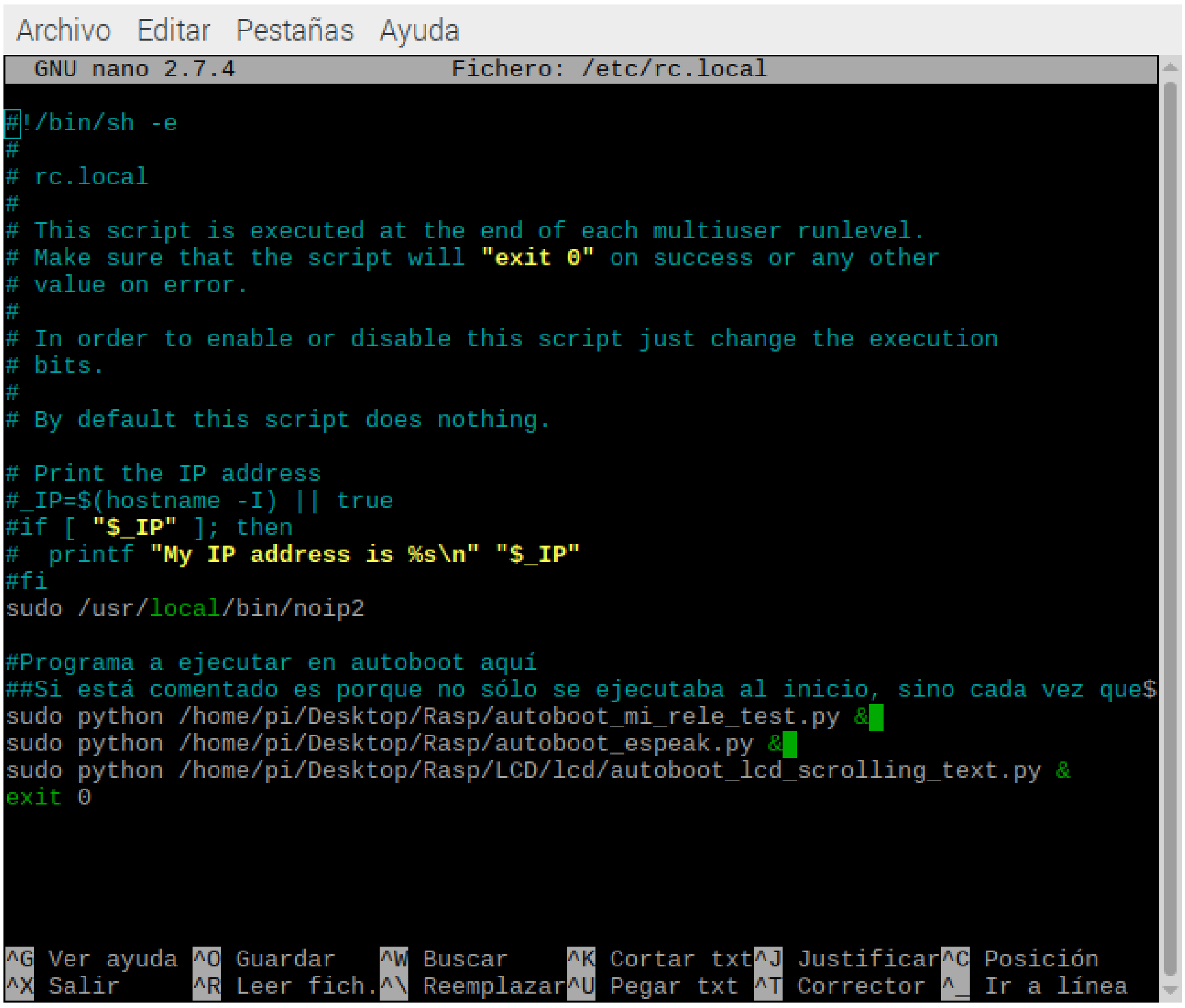
python example.py (Se encuentra en home/pi)

In this prototype there are several actions started:

1) **NO-IP SERVICE**: Reads the configuration files for No-Ip; to supply.
2) **RELAY TEST**: Executes a Python script that enables and disables relay channels
3) **SPEECH MESSAGE VIA ESPEAK**: A robotic male voice says: “Hey, welcome to Garden Host”. Afterwards, a woman version says: “Hola, bienvenido a Garden Host”.

4) **LCD WELCOME MESSAGE**: A message is displayed on the LCD display for several seconds.

After the previous configuration, the **rc.local** file that runs on the boot looks as in the *Figure 58*:



```
  GNU nano 2.7.4                 Fichero: /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
#_IP=$(hostname -I) || true
#if [ "$_IP" ]; then
#  printf "My IP address is %s\n" "$_IP"
#fi
sudo /usr/local/bin/noip2

#Programa a ejecutar en autoboot aquí
##Si está comentado es porque no sólo se ejecutaba al inicio, sino cada vez que$
sudo python /home/pi/Desktop/Rasp/autoboot_mi_rele_test.py &
sudo python /home/pi/Desktop/Rasp/autoboot_espeak.py &
sudo python /home/pi/Desktop/Rasp/LCD/lcd/autoboot_lcd_scrolling_text.py &
exit 0



^G Ver ayuda ^O Guardar    ^W Buscar     ^K Cortar txt^J Justificar^C Posición
^X Salir     ^R Leer fich.^\ Reemplazar^U Pegar txt  ^T Corrector  ^_ Ir a línea
```

Figure 58. Autoboot Configuration

Espeak and Consideration

Espeak [36] converts text to emulated voices. This project uses it under Python and its configuration is explained as follows:

sudo apt-get install espeak //Installs Espeak

sudo nano autoboot_espeak.py //Generates an empty Python script to write:

#! /usr/bin/env python

from subprocess import call

call(['espeak "Welcome to Garden Host" 2>/dev/null'], shell=True)

It is important to write simple and double quotation commas exactly as there.

See that a call to the system is required to speak. This project uses it as a simple welcome message while the desktop is still loading.

As an extra, note that every rc.local instruction uses **&** symbol. This means that the instructions written there must end –or be ended- after a limited time. Without the ampersand the system boot might be paralyzed on a Python script and even not complete.

However, any boot process can be traced writing results to file. Rc.local may contain:

```
sudo python /home/pi/autoboot_espeak.py & > /home/pi/Desktop/trace.txt 2>&1
```

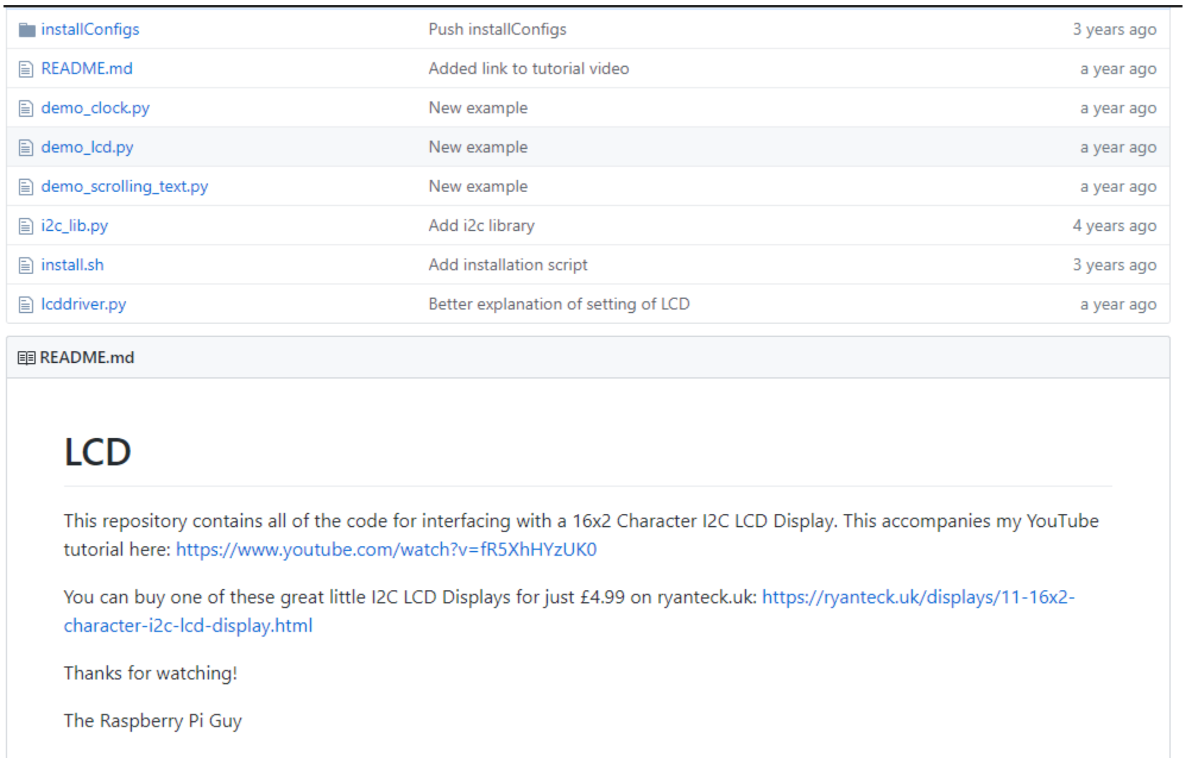
On summary, autoboot allows to execute several functions on Raspberry Pi boot, which is necessary in case the prototype code should execute automatically when turning on. This procedure allows the Raspberry Pi to start its services and scripts without a monitor or remote connection, just when plugged in to an electrical source, and may include the main Python script that executes the whole design system.

5.8 Monitoring and Data Visualization

This section studies the methods used in order to watch the information provided by the sensors. On the one hand, an LCD display is used under I2C (Inter-Integrated Circuit) protocol, which needs only a few wires and its configuration is based on addresses. Secondly, all of the information data is sent to Thingspeak, a monitoring platform.

I2C Protocol: LCD and Pressure Sensor

The connections used are only 4: VCC (3.3V), GND, SDA (Pin3) and SCL (Pin5). The module works on a scrolling text LCD manner. The following reference (see *Figure 59* [31]) was partially introduced in **tfg.py**, in a function that refreshes the screen when humidity and temperature sensors are picked up.



installConfigs	Push installConfigs	3 years ago
README.md	Added link to tutorial video	a year ago
demo_clock.py	New example	a year ago
demo_lcd.py	New example	a year ago
demo_scrolling_text.py	New example	a year ago
i2c_lib.py	Add i2c library	4 years ago
install.sh	Add installation script	3 years ago
lcddriver.py	Better explanation of setting of LCD	a year ago

README.md

LCD

This repository contains all of the code for interfacing with a 16x2 Character I2C LCD Display. This accompanies my YouTube tutorial here: https://www.youtube.com/watch?v=fR5XhHYzUK0

You can buy one of these great little I2C LCD Displays for just £4.99 on ryanteck.uk: https://ryanteck.uk/displays/11-16x2-character-i2c-lcd-display.html

Thanks for watching!

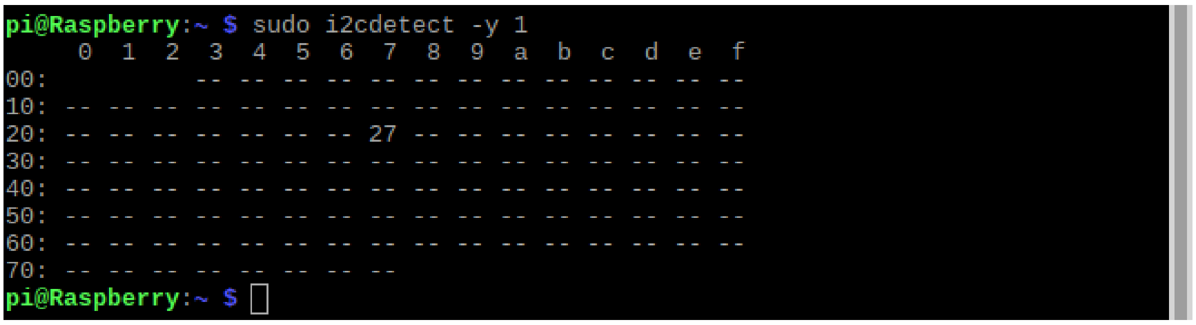
The Raspberry Pi Guy

Figure 59. Screenshot from LCD References

The reference, thanks to Raspberry Pi Guy [37], could be downloaded:

```
git clone https://github.com/the-raspberry-pi-guy/lcd
cd lcd
sudo sh install.sh //It will reboot
```

In I2C protocol it is extremely important addressing the information. In simple, each I2C device should have an own address. When connecting any I2c device, it could be checked: **i2cdetect -y 1**

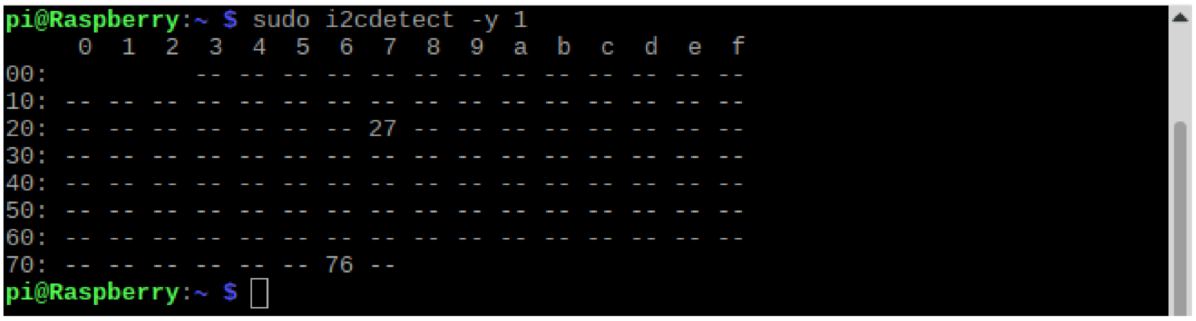


```
pi@Raspberry:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@Raspberry:~ $
```

Figure 60. I2C Address for LCD

Also, see that the output returns ADDRESS=0x27. Therefore, the LCD display is accessible via I2C Protocol, address 27.

Similarly, the Pressure Sensor is used via I2C. After some datasheet investigation, it could work at 0x76. Therefore, on doing the same command, it could be seen:



```
pi@Raspberry:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
pi@Raspberry:~ $
```

Figure 61. I2C Address for LCD and Pressure Sensor

This means that the Raspberry finds the two I2c devices, each one on a different address.

However, it is also possible to change to another I2C address of the pressure sensor, in case it is needed. In order to do it VCC (pin1) and SDO (pin6) will need to receive 3.3V. *Figure 62* [32] shows an I2C connection as example. Nevertheless, notice that the design in this project connects I2C devices to Raspberry Pi, not to the Arduino microcontroller.

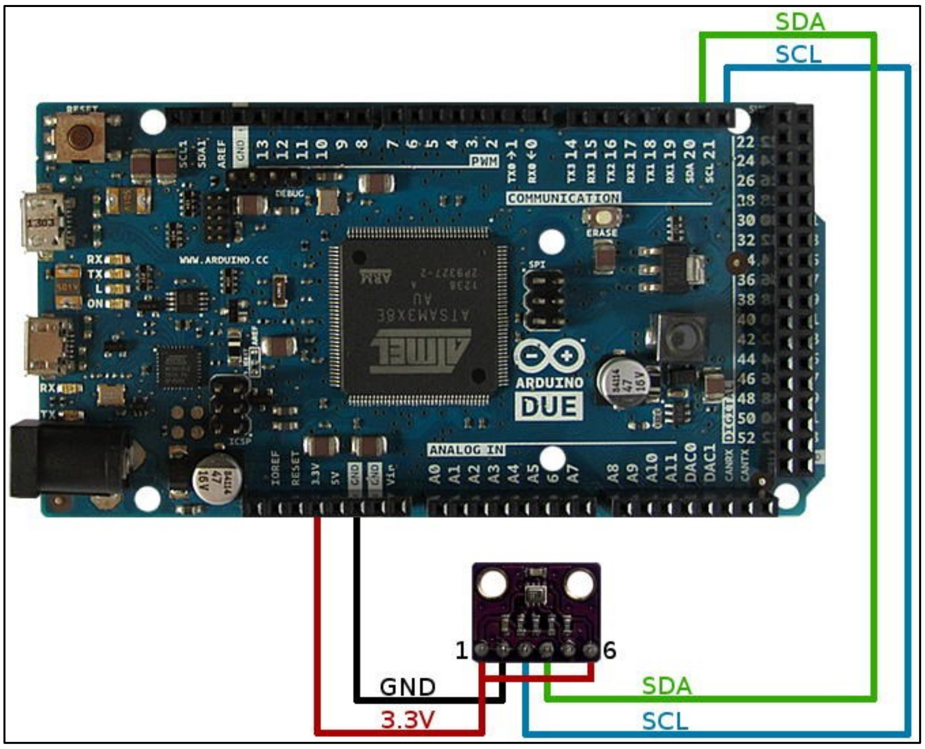


Figure 62. I2C Wiring to 3.3V Arduino Due

For this module to work a library was needed, which is simply installed as follows:

sudo pip install bmp280

Python instructions should now read the correct address:

bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0X77)

The summary of this section information is:

LCD DISPLAY	PRESSURE SENSOR
Address: 0x27	Address: 0x76 (Default) or 0x77
Sense: Go	Sense: Return

Table 4. Prototype I2C Connections

Thingspeak

Figure 63 [33] introduces this service, which is intimately related to IOT technologies:

Figure 63. Thingspeak General View

This platform stores the project information. All the sensor-data from Raspberry Pi as well as the MQTT operations done on *Setapp* Android app are notified here.

The good point of this service is that it represents graphically the information received on-line (synchronized) and can interconnect other services by *reactions*. As stated before, Thingspeak reacts to its field 8 (PIR Presence Sensor) in this project, so that the IFTTT service receives an HTTP message from Thingspeak and sends a pre-configured email.
It is also interesting to know that Thingspeak is linked with ***Mathworks***, reason for which the University account has been used for this registration.

This service has a strict limitation that should be considered: I*t only allows to process an update message every 15 seconds* –on nominal. Accordingly, this project only updates it at the end of *all* sensor's reading loop, after the *updateValues()* in the main flow; or when the PIR sensor activates, on interruption.

As the loop last longer than 15 seconds, there is no problem with this limitation, and neither on inducing an interruption temporary close to a previous sending of the Thingspeak message, because it accepts some peak rates. Therefore, by design all the information should be included in a *key-value* relation in only *one* message.

In order to use Thingspeak in Python a library for HTTP connections is needed and is installed in the following way:

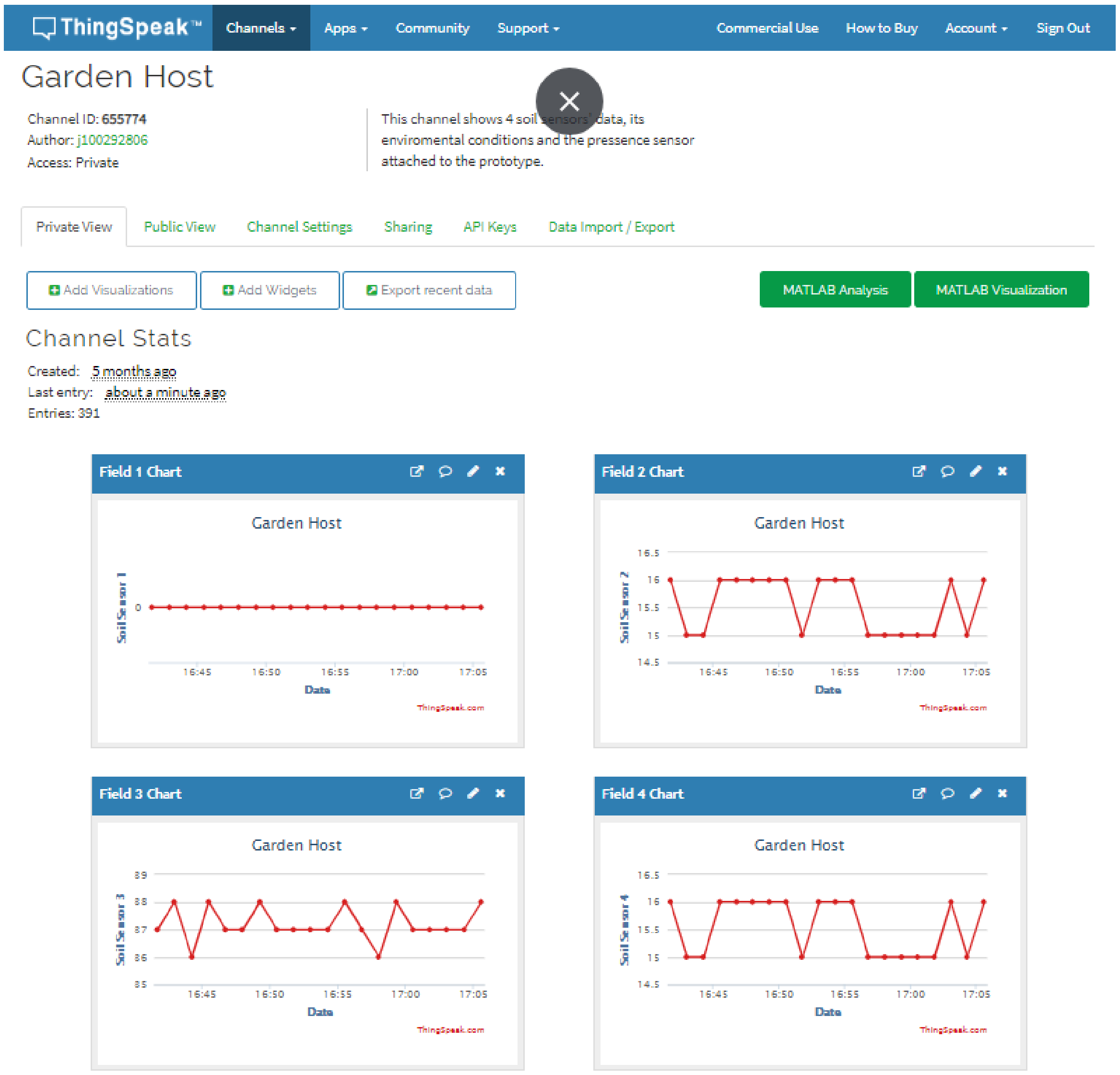
> **sudo pip install requests** //Installs the library
> **API (WRITE) (Example): N11SQBG8R087791I** //The code requests to use it
> **sudo python thingspeak.py** //Execute a Thingspeak script

Thingspeak is used in this project in 2 different channels:

- **Garden Host** - Sensors' trace from tfg.py
- **Setapp** - MQTT devices' trace

On the one hand, Garden Host fields will be explained. Afterwards, Setapp will show their register fields, to be complemented with the next section about this app development.

The first 4 fields from Garden Host correspond to 4 soil sensors' data.



Garden Host

Channel ID: 655774
Author: j100292806
Access: Private

This channel shows 4 soil sensors' data, its enviromental conditions and the pressence sensor attached to the prototype.

Channel Stats

Created: 5 months ago
Last entry: about a minute ago
Entries: 391

Figure 64. Thingspeak: Prototype Channels (1)

This channel also plots environmental information (fields 5 and 6), which refers to humidity and temperature readings. On their right, a plugin is used so that the last reading is easily identified in a green-to-red scale.

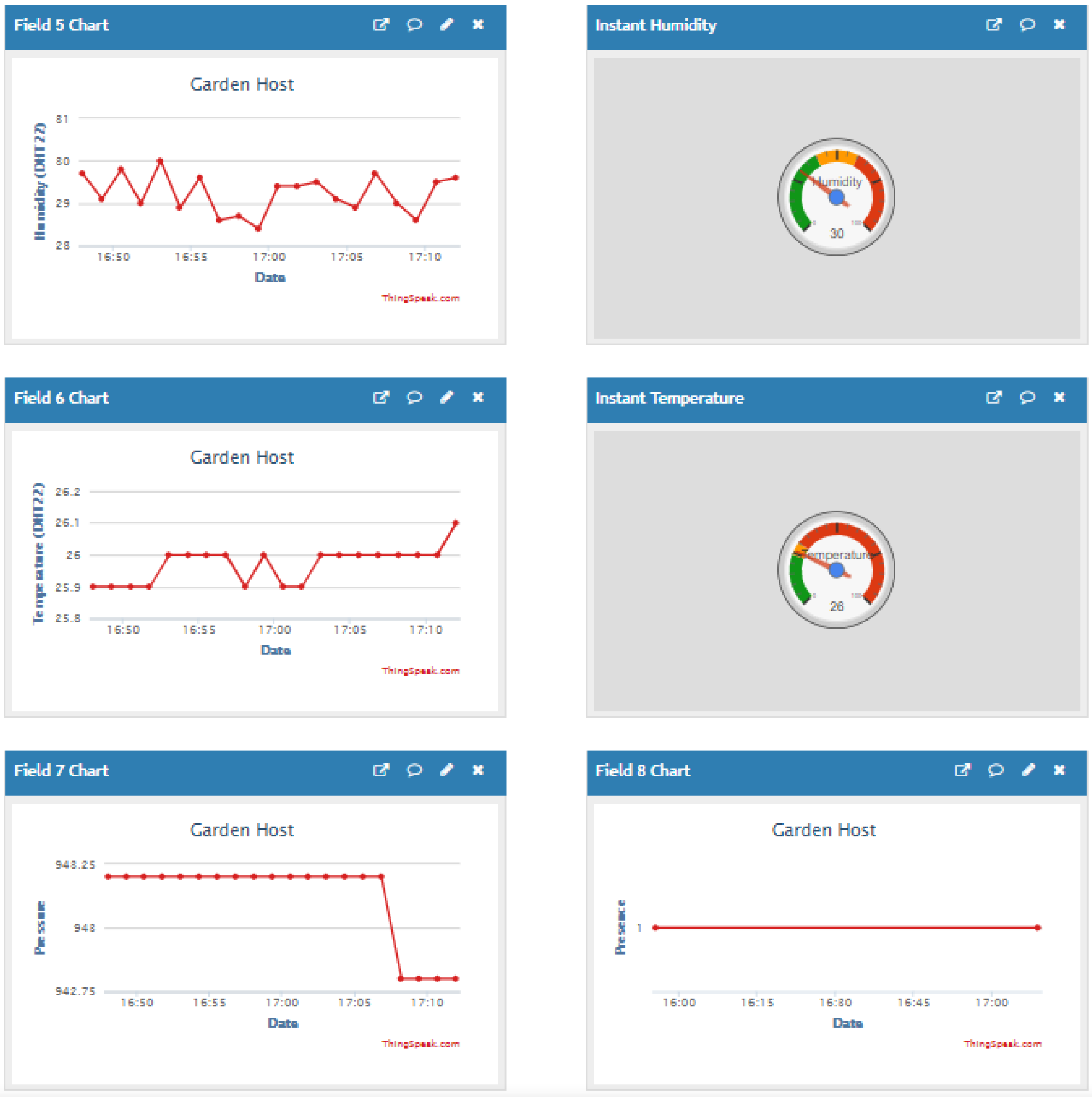
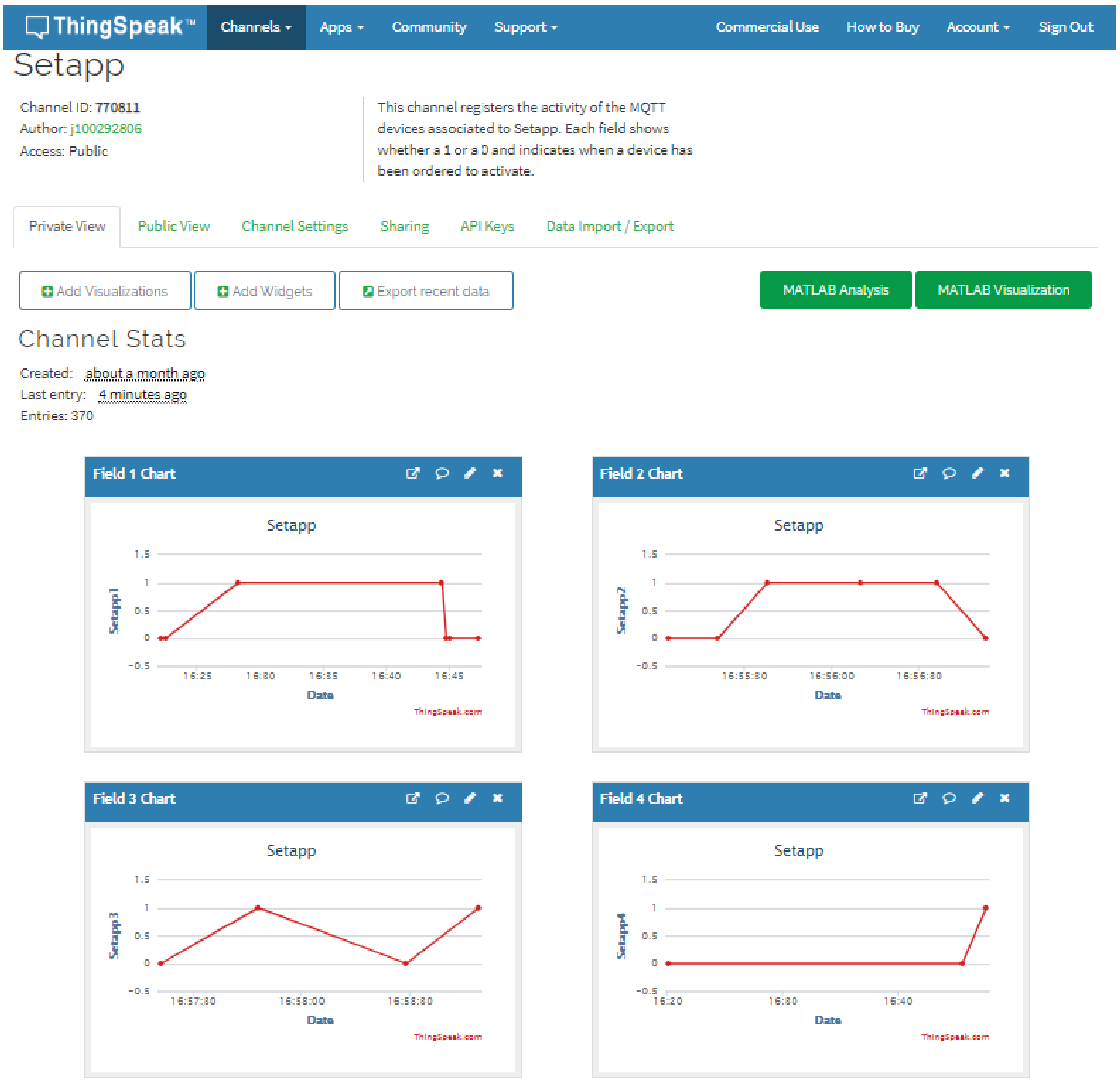


Figure 65. Thingspeak: Prototype Channels (2)

Field 7 corresponds to the data provided by the barometric pressure sensor, while field 8 receives a 1 if the presence sensor is triggered. As explained before, this event would react on sending a message to IFTTT, which also reacts creating an email with the URL of the camera via the No-Ip server set on the Raspberry Pi. The Python code would have commanded to start the broadcast, so the security camera would be available on clicking to the URL received in the e-mail.



Setapp

Channel ID: **770811**
Author: j100292806
Access: Public

This channel registers the activity of the MQTT devices associated to Setapp. Each field shows whether a 1 or a 0 and indicates when a device has been ordered to activate.

Channel Stats

Created: about a month ago
Last entry: 4 minutes ago
Entries: 370

Figure 66. Thingspeak: Developed Android App Channels

The figure above shows the register of four MQTT devices controlled via Android App. For simplicity, a 1 was coded to represent an ON instruction, while, consequently, a 0 would switch off a certain MQTT device.

In summary, Thingspeak is used in this project as a cloud and smart database. It stores in one channel the information from the sensors of the Smart Agriculture Prototype, while in a second it stores the information from Setapp, the Android app. Also a *reaction* that triggers other service -IFTTT- is proposed for field 8, whose consequence is receiving a warning email.

As a global chapter consideration, IOT pays a lot of importance to presenting an attractive user-friendly interface. On simply using a LCD and showing real-time graphical information there is a significant gain. Finally, Thingspeak has proven to be a service that is compatible to reactions, but it is also able to execute big data analysis.

CHAPTER 6

ANDROID APP AND MQTT PROTOCOL SERVER

6.1 MQTT General Idea

Message Queuing Telemetry Transport is a protocol that allows to send mostly short messages inside a network. Four main concepts play an important role, summarised here:

MQTT broker: It is the MQTT server that distributes the messages.
Topic: Each of the categories whose messages a certain device listens to.
Publish: Act of sending a message to the broker to be forward to a specific topic.
Subscribe: Act of start listening to a certain topic.

Therefore, MQTT broker processes all the *publish* messages to be forwarded to the different topics and manages the subscriptions. This protocol is widely spread in IOT systems because it is able to run even under devices with very *limited computational requirements* as well as in networks with extremely *low bandwidth*.

This project configures an MQTT server in order to serve for the ESP32 wireless devices. *Setapp*, the Android app, sends messages to the broker so that the different devices are switched. In a general view, MQTT is a perfect protocol for non-heavy traffic. It needs a server configuration and is able to turn on many actuators at once with wireless communication. Finally, note that once a connection is stablished to the broker, the TCP link remains open for further announcement, which is also a very pragmatic approach.

6.2 Mosquitto: Mqtt server configuration

Eclipse Mosquitto implements the MQTT protocol in a lightweight way. For this reason and due to its simple installation it is used in this proposal.

In order to have an idea of how light a MQTT message could be, it could be considered the following composition: 2-bytes obligatory header, 4-bits optative header and, under usual conditions, 2-4kB of payload. It is clear, therefore, that MQTT could be able to communicate under low bandwidth scenarios and be compatible to devices with limited computational resources.

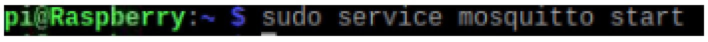
As for the configuration, it is absolutely recommended to use a *password* to protect the topics. If it is used over the Internet, for security reasons it is also necessary to use *SSL certificates*.

It is important to highlight the next instructions:
sudo service mosquitto stop //Stops services (recommended before changes)

sudo mosquitto_passwd -c /etc/mosquitto/pwfile jesus //Password is introduced
sudo nano /etc/mosquitto/mosquitto.conf //Configuration file opens. There:
allow_anonymous false //Sets to false in case of using password

Once set, the MQTT broker is able to serve messages. In order to check it, several terminal instances of the Raspberry Pi may communicate with each other by means of the Mosquitto server.

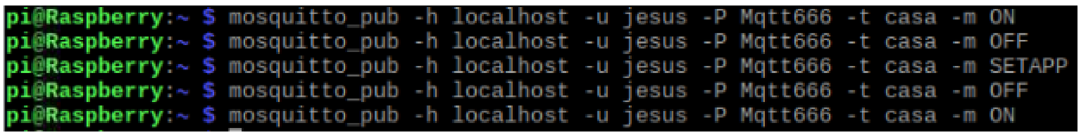
An example with topic *casa* is shown in terminal:
Console: After configuration, it runs the Mosquitto service on the Raspberry Pi:



```
pi@Raspberry:~ $ sudo service mosquitto start
```

Figure 67. Mosquitto Start

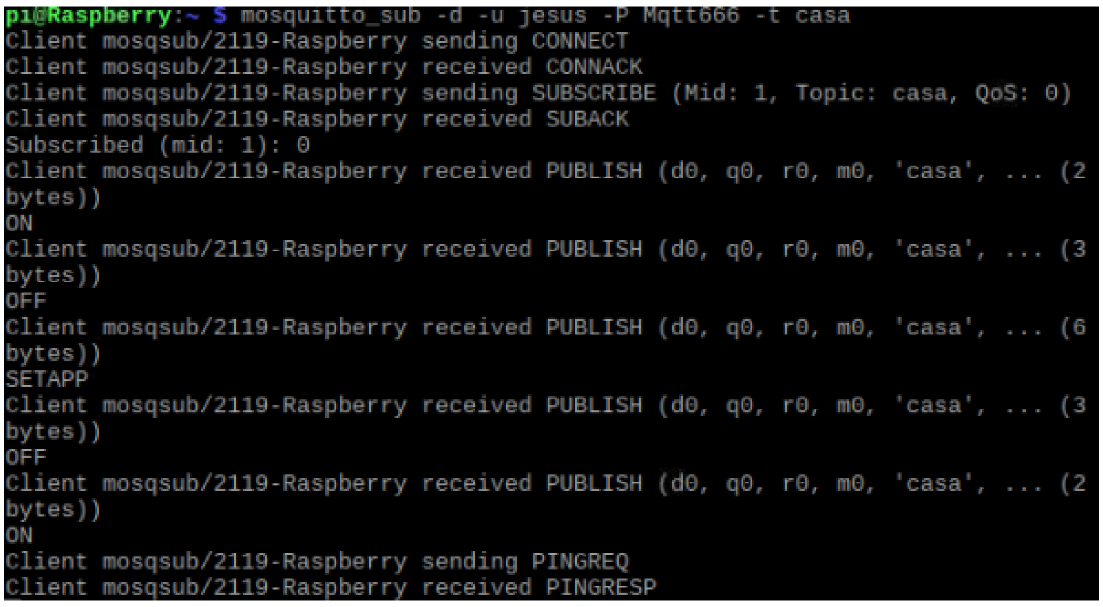
Console 1: It acts as the publisher device:



```
pi@Raspberry:~ $ mosquitto_pub -h localhost -u jesus -P Mqtt666 -t casa -m ON
pi@Raspberry:~ $ mosquitto_pub -h localhost -u jesus -P Mqtt666 -t casa -m OFF
pi@Raspberry:~ $ mosquitto_pub -h localhost -u jesus -P Mqtt666 -t casa -m SETAPP
pi@Raspberry:~ $ mosquitto_pub -h localhost -u jesus -P Mqtt666 -t casa -m OFF
pi@Raspberry:~ $ mosquitto_pub -h localhost -u jesus -P Mqtt666 -t casa -m ON
```

Figure 68. Mosquitto Publish Example

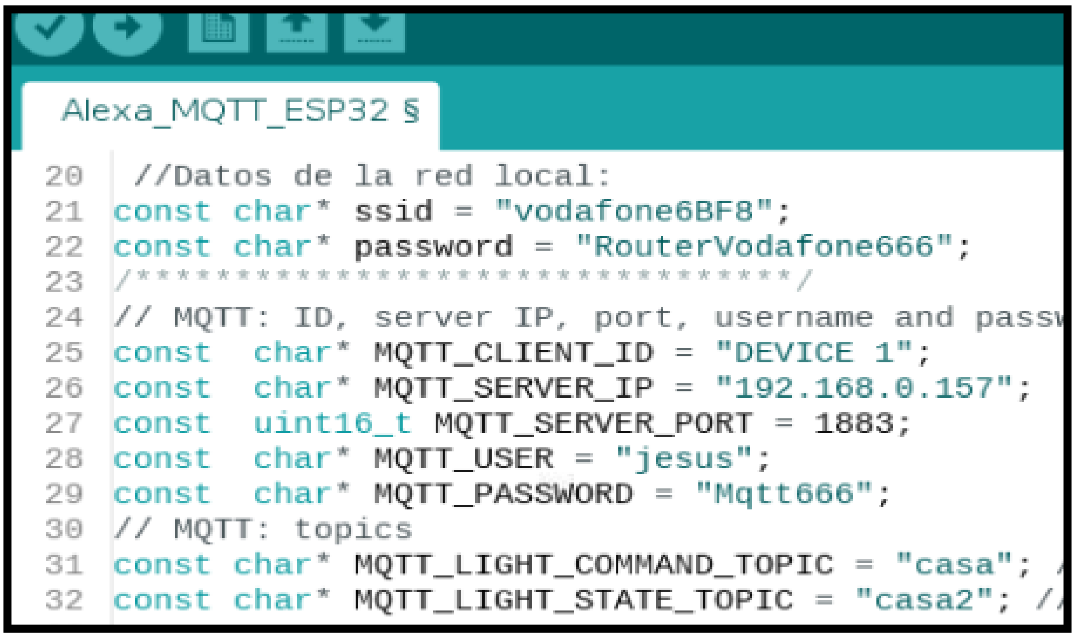
Console 2: If subscribed to *casa*, the history of the received of *Figure 69* messages is shown.



```
pi@Raspberry:~ $ mosquitto_sub -d -u jesus -P Mqtt666 -t casa
Client mosqsub/2119-Raspberry sending CONNECT
Client mosqsub/2119-Raspberry received CONNACK
Client mosqsub/2119-Raspberry sending SUBSCRIBE (Mid: 1, Topic: casa, QoS: 0)
Client mosqsub/2119-Raspberry received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/2119-Raspberry received PUBLISH (d0, q0, r0, m0, 'casa', ... (2
bytes))
ON
Client mosqsub/2119-Raspberry received PUBLISH (d0, q0, r0, m0, 'casa', ... (3
bytes))
OFF
Client mosqsub/2119-Raspberry received PUBLISH (d0, q0, r0, m0, 'casa', ... (6
bytes))
SETAPP
Client mosqsub/2119-Raspberry received PUBLISH (d0, q0, r0, m0, 'casa', ... (3
bytes))
OFF
Client mosqsub/2119-Raspberry received PUBLISH (d0, q0, r0, m0, 'casa', ... (2
bytes))
ON
Client mosqsub/2119-Raspberry sending PINGREQ
Client mosqsub/2119-Raspberry received PINGRESP
```

Figure 69. Mosquitto Subscribe Example

With respect to the ESP32, it uses a modified code whose main structure is referenced. Note that it is able to respond both to MQTT and Alexa messages, so a combination was required. After all this process, ESP32 connects to the local Wifi, starts listening on a topic (*casa*) so that an attached device is turn on/off and, finally, publishes back a message, to a different topic (*casa2*), stating its new state.


Alexa_MQTT_ESP32 §

```
 //Datos de la red local:
const char* ssid = "vodafone6BF8";
const char* password = "RouterVodafone666";
/*********************************/
// MQTT: ID, server IP, port, username and passw
const  char* MQTT_CLIENT_ID = "DEVICE 1";
const  char* MQTT_SERVER_IP = "192.168.0.157";
const  uint16_t MQTT_SERVER_PORT = 1883;
const  char* MQTT_USER = "jesus";
const  char* MQTT_PASSWORD = "Mqtt666";
// MQTT: topics
const char* MQTT_LIGHT_COMMAND_TOPIC = "casa"; /
const char* MQTT_LIGHT_STATE_TOPIC = "casa2"; //
```

Figure 70. ESP32 Code for MQTT and Alexa Compatibility

At this point it is demonstrated that MQTT server works well on using terminal commands. This method is useful in combination with a Python script that calls the system and requires it to execute them. However, there is an alternative solution that only requires user cooperation and supposes the final proposal of this Final Project: *Setapp*, an Android app.

6.3 SETAPP (Designed Android App for MQTT Control)

The MQTT server may work in combination with the Android App to activate devices. This last functionality is able to send MQTT messages to different topics and even to different brokers, under password, so that multiple devices are remotely controlled by once in a simple touch. In summary, this application provides a *Configuration* activity where the parameters, including a photo of each device, are registered, and a *Set* activity, which controls the registered devices with an intuitive ON/OFF button.

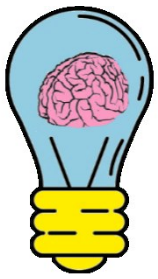
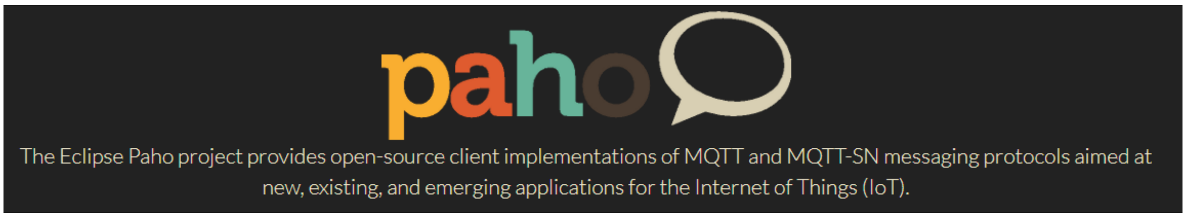


Figure 71. Developed Android App Logo

SQLite database [38]is used in order to store the devices' configuration and each time a device is set a Thingspeak message is sent. *Register* activity can open the Thingspeak channel –if it is public- where all the register graphs of present and previous states are available.

In a technical perspective, MQTT functions are implemented using the **Eclipse Paho** [39] library, which tries to simplify some of the process to connect via MQTT and defines itself as (see *Figure 72* [34]):



paho

The Eclipse Paho project provides open-source client implementations of MQTT and MQTT-SN messaging protocols aimed at new, existing, and emerging applications for the Internet of Things (IoT).

Figure 72. Paho Definition

The MQTT operations are called whenever an ON/OFF button is pressed on a registered device, and needs to first connect as a client of the MQTT broker and, seconds later, send the message with that client ID and the switching content itself.
The control of a system by means of an Android app is an essential possibility that allows sending complex MQTT messages in a very intuitive way, which, according to IOT dictates, should be one of the first priorities, together with providing new or smarter functionality.

The next image shows the main screen of *Setapp*, where the different options by means of ImageButtons. *Register* will show several choices, which the most important opens the Thingspeak channel set on the configuration. *HowTo* shows information about the MQTT protocol and the designing of the project. It also includes the logo of the app and a small tribute to ESP32 and the optocoupler relay that allows the devices' switching.

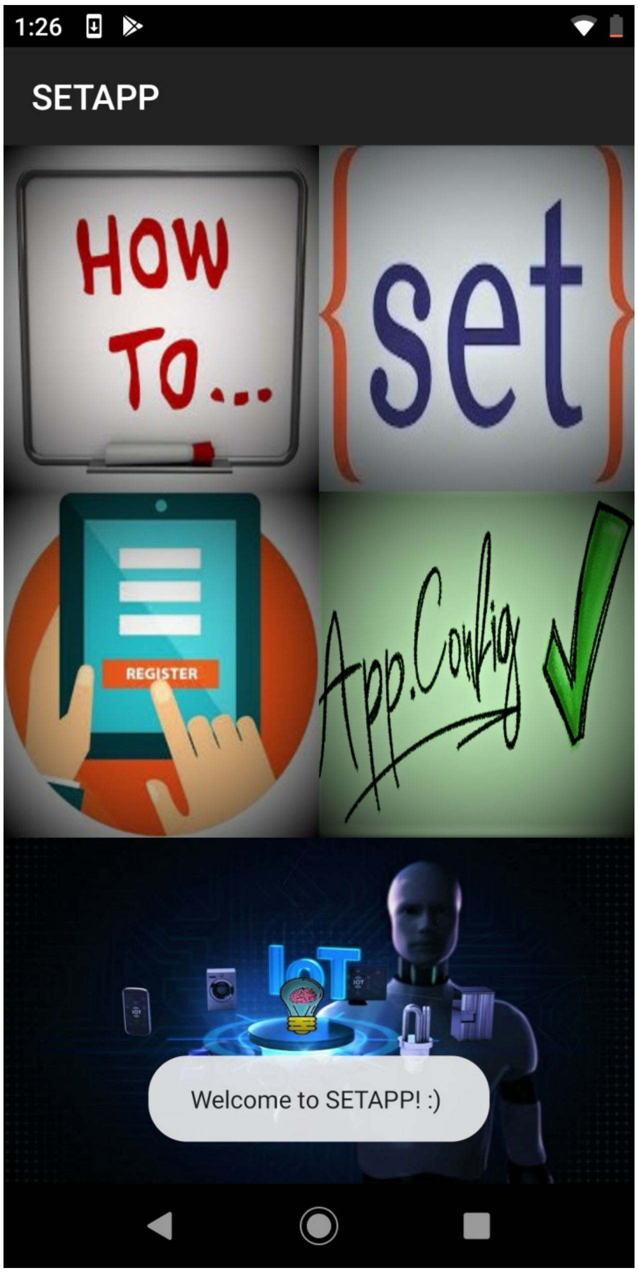
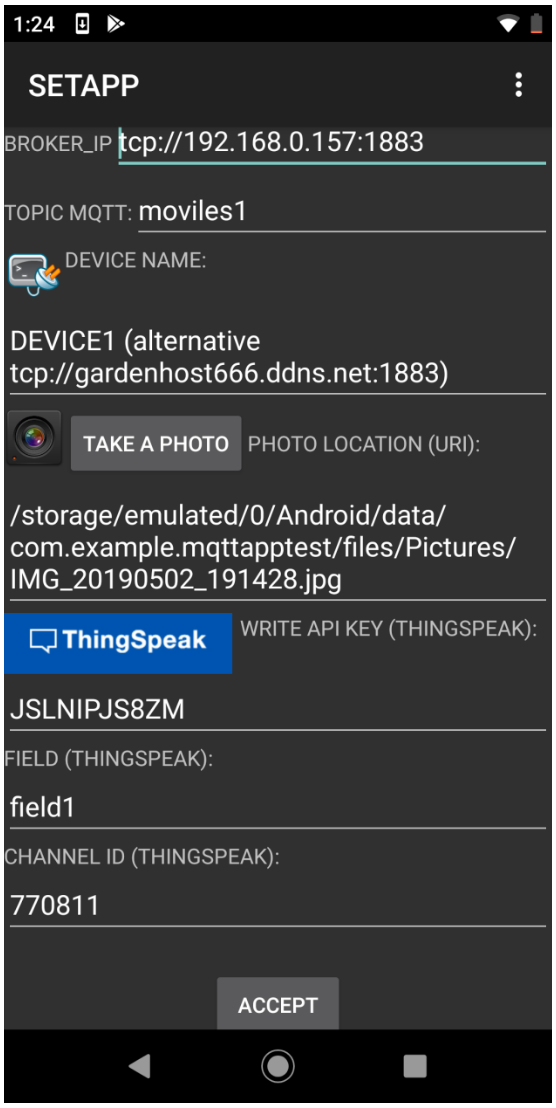


Figure 73. Developed Android App Main Activity

As stated, *Configuration* allows the storage of a device's MQTT information to the SQLite database. At the left there is a screenshot of its fields, while on the right *Set* shows the different registered devices so that. When selected one, its photo will be called from the database. If ON/OFF is pressed, an MQTT message would be sent.



1:24

SETAPP

BROKER_IP tcp://192.168.0.157:1883

TOPIC MQTT: moviles1

DEVICE NAME:

DEVICE1 (alternative tcp://gardenhost666.ddns.net:1883)

TAKE A PHOTO PHOTO LOCATION (URI):

/storage/emulated/0/Android/data/com.example.mqttapptest/files/Pictures/IMG_20190502_191428.jpg

ThingSpeak WRITE API KEY (THINGSPEAK):

JSLNIPJS8ZM

FIELD (THINGSPEAK):

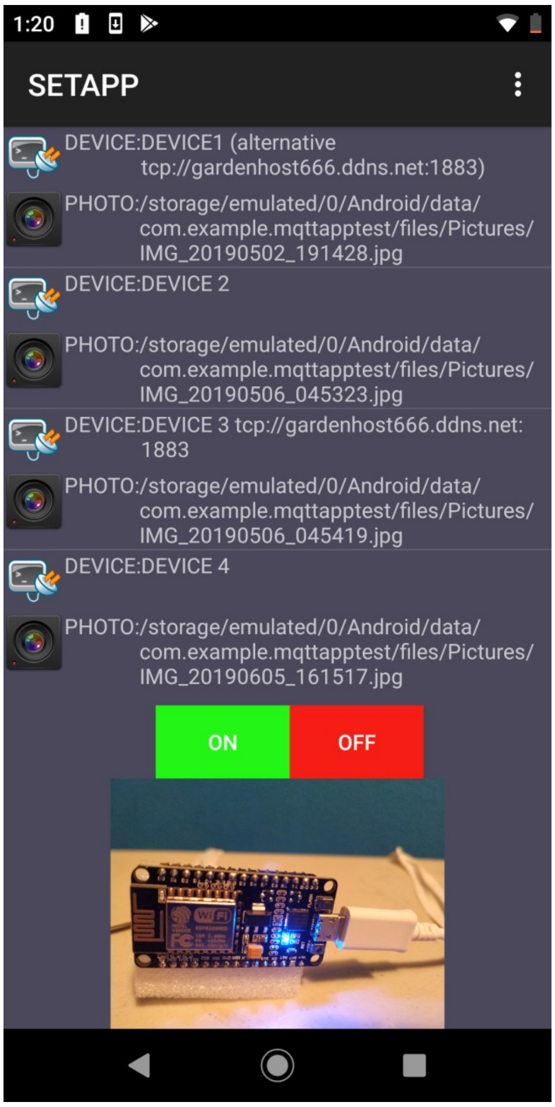
field1

CHANNEL ID (THINGSPEAK):

770811

ACCEPT

Figure 74. Developed Android App: CONFIGURE Activity



1:20

SETAPP

DEVICE:DEVICE1 (alternative tcp://gardenhost666.ddns.net:1883)

PHOTO:/storage/emulated/0/Android/data/com.example.mqttapptest/files/Pictures/IMG_20190502_191428.jpg

DEVICE:DEVICE 2

PHOTO:/storage/emulated/0/Android/data/com.example.mqttapptest/files/Pictures/IMG_20190506_045323.jpg

DEVICE:DEVICE 3 tcp://gardenhost666.ddns.net:1883

PHOTO:/storage/emulated/0/Android/data/com.example.mqttapptest/files/Pictures/IMG_20190506_045419.jpg

DEVICE:DEVICE 4

PHOTO:/storage/emulated/0/Android/data/com.example.mqttapptest/files/Pictures/IMG_20190605_161517.jpg

ON OFF

Figure 75. Developed Android App: SET Activity

On a general view, *Setapp* interact as in the following flow (see *Figure 76*). The arrows indicate a change in the visible activity and, therefore, an interface change.

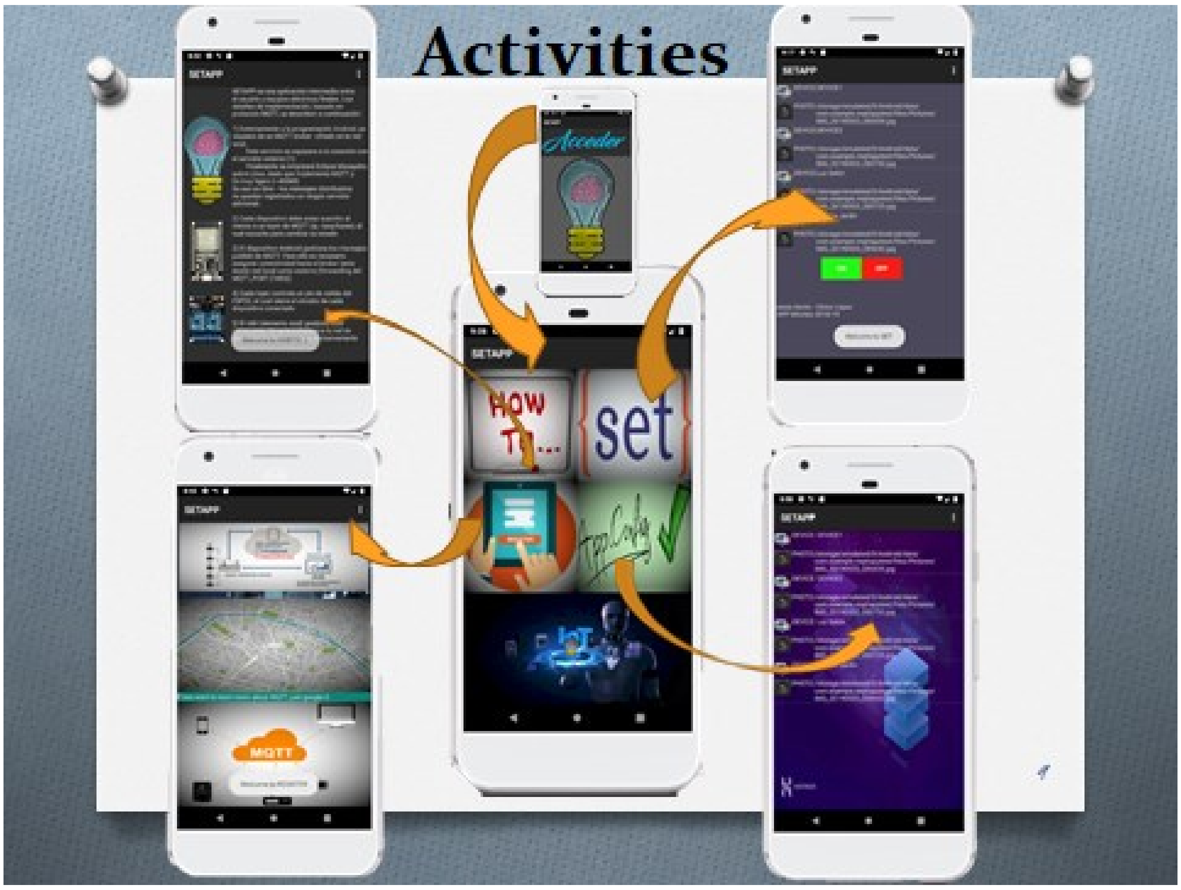


Figure 76. Developed Android App Activities

In conclusion, *Setapp* is a designed Android app that makes sending MQTT messages easier. According to the increasing demand for IOT systems, this proposal attempts to offer an easy connection to this protocol, the probably most extended for computational-limited devices and light communications.

In this project this is used in cooperation to **Mosquitto** server (MQTT broker), **Thingspeak** register and **SQLite database**, in the software part; while it also requires as many ESP32 devices as wanted, connected each one to a relay.

This system is able not only to turn ON/OFF a bulb, but also a fan, a water pump and basically any electrical device that turns on when receiving electrical supply. MQTT messages are sent to each of the topics configured, to one or many different brokes, and finally allow the activation of the devices.

As usual, the development of a full Android app last long, but it is one fundamental piece of the IOT scene and modern-time society. For this reason this section, which is really summarized in order to explain the main idea the app tries to cover, needs to have a special consideration inside this project.

Although *Setapp* is one of the strongest proposals of the present paper, note that the Smart Agriculture Prototype does not work under MQTT protocol. This decision is taken for several reasons that are now expressed.

On supposing the Smart Agriculture Prototype worked under ESP32 wireless devices, an evident danger appears if the LAN communication fails, as each device would remain at a certain state, which is especially risky for some elements such as the water pump. In this situation occurs in the current prototype, the wired devices would still be working, with only being Thingspeak registration and, obviously, the security camera broadcasting the affected features. From another point of view, as this project is complex and offers many different options, creating two different proposals –wired VS wireless- allows a simple exposition of both the prototype and the *Setapp* + ESP32 devices.

In order to be short, using MQTT protocol is perfectly compatible in detail, on also considering the possible system's eventual failures. Wireless devices may not be as reliable as wired options, especially on a fixed location. However, wireless technology and Android app is currently compatible for other devices in this prototype.

CHAPTER 7:

PLANNING AND TESTS

7.1 Design Stages and Results

The design of this whole Final Project has not been linear, reason for which the present exposition tries to be as understandable as possible. It is clear that many sections interconnect and that alternative options are explained as well as those that belong to the Smart Agriculture Prototype. The following lines summarize the whole process.

The original idea was to create a ***simple irrigation system***, which is by itself not a new concept. Initially it was thought that a periodical activation of a water pump would be enough. It is true: many simpler solutions use periodical approaches. However, this simple solution is quite limited, as supposes a linear absorption of water, to be periodically supplied. And, of course, this solution takes no environmental conditions into account. It is also evident that ***air-flow*** devices were needed too, which, at the beginning, would be thought to be turn on periodically, with the limitation of conceiving stationary conditions of the environment.

The next conception of the project understood that ***sensing was needed***. This refers to extracting soil and environmental characteristics by means of sensors so that the different output devices switch automatically, as an ***autonomous* system**. The simplest sensing considers both the *Soil Sensor* and the *Environmental Humidity and Temperature* device (*DHT22*). This purpose could be achieved by means of a *microcontroller with GPIO pins*. However, soil sensor delivers *analog signals*, while DHT22 uses *digital signals*.

Raspberry Pi was the first important element to be introduced. Its GPIO structure supports the reading of digital signals, so that the *Humidity and Temperature Sensor* was able to be directly connected. As for the *Soil Sensor*, many of them are sold together with a tiny circuit that allows the output of both digital and analog readings. This digital output returns a high voltage whenever a potentiometer-controlled threshold is surpassed, so that a Dry/Wet binary information was available.

Nevertheless, obviously the digital signal created by the threshold does not describe the data as well as with analog readings. For this reason it was necessary to consider the best way to incorporate ***both*** digital and analog sensors, for the previous and also next sensors.

After considering other options as independent ADC modules in order to convert *each* analog signal into a digital one, to be processed by the Raspberry Pi, finally the use of a microcontroller with ADC was required. **Arduino UNO** provides enough GPIO with this characteristic in order to be used in the Prototype. However, Arduino MEGA might be required for using *all* the analog sensors presented. As explained, an ***optocoupler relay*** is used in order to short/open the output devices.

At this point, an important decision was taken in this project:
Raspberry Pi will receive digital sensors (ie. DHT22) and, via serial communication, it will ask Arduino for its analog readings (ie. soil sensors).
The present implementation requires a **main Python code** to be run on the Raspberry Pi and a **C code** that commands the Arduino actions when a serial instruction is received.

Up to here, the prototype is not very different from any other sensing system and its communication is *wired* to all of the sensors. An initial research about *wireless technologies* suggests a tendency to connect devices that share the same LAN by means of light protocols such as MQTT and independent devices.

Therefore, **IOT technologies** apply here. They play an important role in the project on, first, *connecting wireless devices* and providing an alternative method to turn on actuators by means of ESP32; and, second, on allowing a user-control by means of an **Android app** that simplifies the ON/OFF messages in the MQTT network.

For wireless alternative to run, *Mosquitto* service was installed and configured on the Raspberry Pi, which starts on boot. **ESP32 devices**, programmed in C, would here be able to answer to MQTT messages so that a pin was set to HIGH/LOW, which is connected to an optocoupler relay that turns ON/OFF the output element.

It is also important to mention the **IOT Security Camera** flow. It uses a PIR Presence Sensor in order to detect movement and, on a Python code interruption, it calls the system in order to execute the Motion software, and, consequently, the broadcast of images from an USB camera starts.

Alexa was introduced later and ESP32 code was modified to respond also to its messages. Voice control devices have emerged as one of the most important IOT technologies, so integrating Alexa as an alternative communication seemed a perfect extra.

Setapp is the designed Android app that allows the register and management of MQTT devices (ESP32). It configures each of the MQTT parameters and sends an ON/OFF message on command.

The **result presentation** was also taken into account, so that both Smart Agriculture Prototype and *Setapp* registers show graphical representations by means of Thingspeak. Moreover, this service is able to trigger certain actions and, together with IFTTT, will notify the user when the security camera is turn on due to a stranger presence next to the system. An **LCD display** shows the last DHT22 reading.

In a general view, the wired technology was researched and implemented first. Each of the sensors studied or finally used required a specific understanding of the physical magnitude to be measured and the type of technology that transduces.

The wireless alternatives were studied afterwards, on special consideration to Alexa technology and the explained Android app developed for this purpose.

Finally, this project also studies network connectivity by means of **No-Ip**, which allows the Raspberry Pi to communicate to the Internet and supply its services, whose security methods have been exposed.

Figure 77 shows a very general view of the elements that compose the Prototype. However, the defense session of this Final Project would provide multimedia demonstration of its working and of the wireless options that use ESP32 devices.

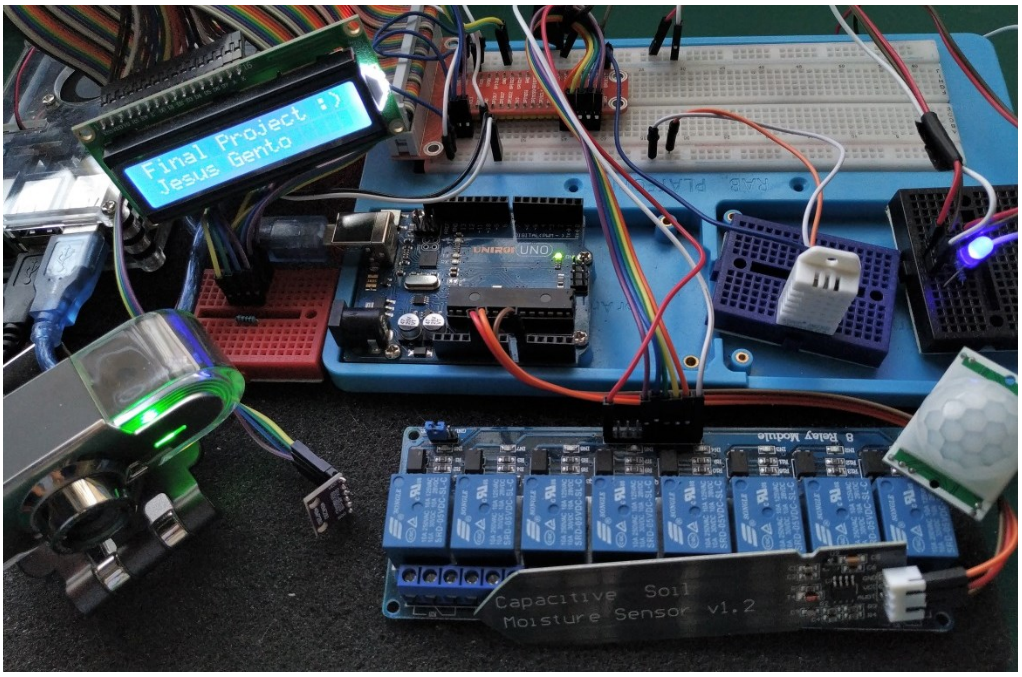


Figure 77. Smart Agriculture Prototype

7.2 Modular Integration Idea

This Final Project offers a new model of product based on independent modules that required a progressive integration and continuous testing. For this reason, each of the sensors that compose the Smart Agriculture Prototype, as well as those that have been explained but finally are not integrated, work perfectly on individual scripts.

Therefore, the project was oriented to first research on a particular type of sensor, then designing a small script that make it work and, finally, the main project considerations for a particular target project may use it or not. For instance, in the case of the Prototype, there was no need of integrating the *Gas Sensor*, as its functioning is explained in detail and it may not offer information about a magnitude that is not important for a specific design.

Tfg.py code is the main Python script and represents the whole Smart Agriculture Prototype with IOT functionalities.

7.3 Planning

According to this Degree program, the Final Project should represent at least 360 hours of work. This project extends further than that due to a constant research that was hardly affordable to be done all at once at the very beginning. Moreover, the modular integration implies that a certain sensor’s functions may be implemented to the main code while some other sensor may be studied on its theoretical terms.

Nevertheless, with purpose on a global understanding, a good linear approximation of the whole process could be summarized in the following Gantt diagram:

TASK	JANUARY				FEBRUARY				MARCH				APRIL				MAY				JUNE	
	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2
IDEA GENERATION																						
Documentation on State of Art	🟩																					
Documentation on Soil Sensing		🟩																				
Documentation on Climate Sensing		🟩																				
Raspbery Pi Research on Similar Targets			🟩																			
Arduino Research on Similar Targets			🟩																			
Documentation on Electrical Connections and				🟩																		
SOFTWARE UNDERSTANDING																						
Raspbian Configuration					🟩																	
Python Learning					🟩	🟩	🟩	🟩	🟩													
Raspberry Pi First Scripts								🟩	🟩	🟩	🟩											
BASIC PROTOTYPE DEVELOPMENT																						
Soil Sensing Incorporation									🟩													
Climate Sensing Incorporation										🟩												
Arduino Incorporation										🟩												
First Test										🟩	🟩											
BASIC PROTOTYPE FUNCIONALITIES																						
No-IP Network Service (Remote Control)												🟩										
Security Camera System												🟩										
Thingspeak													🟩									
I2C Protocol (LCD, Presssure Sensor incorporati													🟩									
Second Test													🟩									
IDEA EXPANSION: Wireless Options																						
Documentation on IOT Technologies														🟩	🟩							
Documentation on MQTT Protocol														🟩								
Android Fundamentals					🟩		🟩		🟩		🟩			🟩	🟩							
ESP 01S (Option Finally Refused)															🟩							
ESP 32															🟩	🟩						
Alexa																🟩	🟩	🟩				
Mosquitto Server																🟩		🟩				
Setapp (Android App)								🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩					
FINAL TESTS																						
Smart Agriculture Prototype Tests																	🟩	🟩	🟩	🟩		
IOT Technologies Proposal and Android Debug																		🟩	🟩	🟩	🟩	🟩
MEMORY WRITING																						
Preliminar Information	🟩	🟩	🟩						🟩	🟩	🟩		🟩	🟩	🟩		🟩					
Final Version																	🟩	🟩	🟩	🟩	🟩	🟩

Gantt Diagram

CHAPTER 8:

BUDGET FOR IOT SOLUTIONS AND LAW

8.1 Services according to Requirements

According to the information provided in the present paper, it is clear that many options are offered and each one have required a certain amount of time, technological resources and, if used under commercial uses, needs a different money retribution. For this reason, the budget for this project is not easy to propose. This section explains the main options studied here and, for simplicity, it estimates the most relevant cases:

Smart Agriculture Prototype:
The following proposal would be based on *tfg.py* script and, therefore, targets on measuring environmental conditions. This can be applied to a specific growing system or, with some modifications, to any environmental control for industrial purposes.

Base products:

- **i.** **Purely Wired**: It is the most studied solution.
- **ii.** **Hybrid**: Sensors *or* actuators may communicate wireless.
  It requires MQTT server configuration.
- **iii.** **Wireless**: Sensors *and* actuators communicate wireless.
  It requires MQTT server configuration.

Moreover, there are ***extra services*** that increase the total cost of particular option:

- ❖ **Large number of sensors** required: If code needs modification there should be a cost increase. It is currently able to work for 10 different sensors.
- ❖ **Historical record**, via Thingspeak
- ❖ **Private security camera**, via Motion
- ❖ **Remote connectivity option**, via No-Ip.

Total product: Base product + Extra Service(s)
As an example, a Purely Wired option for sensors' control increases its cost if also it needs to be remotely controlled.

IOT Technologies:

- **i.** **MQTT server configuration**: Needed for every wireless solutions and included in Hybrid and Wireless options
- **ii.** **1-point IOT domotic system**: ESP32. May or not be compatible with Alexa
- iii. **N-point IOT domotic system**: ESP32. May or not be compatible with Alexa
- **iv.** ***Setapp* Android App:** Outputs are controlled via an Android device

It is also needed to consider the engineering cost of designing the different system as well as its supervision. Therefore, the following page shows the main budget for this project:

BUDGET

CATEGORY	DESCRIPTION	PRICE/UNITS (€)	MAIN FEATURES	UNITS	TOTAL PRICE
COMPUTING	Raspberry Pi 3B+	33,78	1 GB Ram (DDR2 SDRAM), 4 Processors (1,4 GHz, 64 bits), 2.4GHz and 5GHz WiFi, HDMI, 40 GPIO'S, 4 x USB 2.0	1	33,78
	Arduino kit	29,99	Arduino UNO (6 Analog Inputs), Raspberry Pi and Arduino UNO base, LCD 1602, Protoboard, Wires and Extra Components	1	29,99
	Arduino UNO	9,99	Arduino UNO (6 Analog Inputs)	1	9,99
ACCESORIES	Smraza Kit for Raspberry Pi 3B+	23,99	Raspberry Pi 3 b+ Case, Power Supply, Fan, 3x Heatsink, SD (16Gb), SD-USB reader, HDMI cable	1	23,99
	Kuman Kit for Raspberry Pi 3B+	11,53	GPIO T-Expansion, 40 pin Rainbow Ribbon Cable, 65 Dupont cables	1	11,53
	SD 32 GB card	6,39	Fluent working on using the full system, Motion (Camera streaming) and Mosquitto serving (MQTT) would require more SD space	1	6,39
SENSORS	Soil Moisture Sensor	9,59	4 Capacitive sensors, which are much stronger to corrosion processes than resistance-based	1	9,59
	Temperature, Humidity Sensor (DHT22)	7,09	Accuracy: Temperature +/- 0.5°C, Humidity +/-2RH; Minimum Detection Time: 2 seconds	1	7,09
	Water Level Sensor	0,87	May suffer corrosion on long term	1	0,87
	Atmospheric Pressure	1,15	Acurracy: +/- 1 hPa	1	1,15
	Light Sensor (Top and Roots)	1,88	On having two light sensors it could be deduced the light dispersion in our room	2	3,76

	Air Quality Sensor (MQ135)	2,78	Detection of: NH3, NOx, Alcohol, Benzene, Smoke, CO2	1	2,78
	Presence Sensor (PIR)	2,3	Detection Range (adjustable): 3-7 m; Detection angle: 120°; Minimum Detection Time: 3 seconds	1	2,3
	PH Sensor	34,95	Manual sensor Adwa AD11. Accuracy: +/-0,1 pH, +/-0.3°C	1	34,95
	EC/TDS Sensor	37,95	Manual sensor Adwa AD31. Accuracy: +/-0,2, +/-0.3°C	1	37,95
OUTPUTS	8-Channel Relay	4,47	Works under 5V (15-20mA per channel), Built-in Leds to indicate each state	1	4,47
	1-Channel Relay	1,99	Needed for each ESP32	1	1,99
	Mini Pump	9,99	Works under 12 (<400 mA) and under water, 240 l/hour, Silent (<35 dbA)	1	9,99
EXTRA	ESP 32	10,09	520 KiB SRAM, 2 Processors (160MHz, 32 bits), 2.4GHz WiFi, 32 GPIO's, 1 Analogue Input	4	40,36
Option A	USB Camera (Security Camera)	9,95	In this prject runs on gardenhost666.ddns.net:8081. Requires Motion installation and Server Configuration on home router.	1	9,95
Option B	IP Camera (Security Camera)	32,99	Alternative solution. Infrared light for night vision, bidirectional MIC, used under APP on external servers.	1	32,99
MINIMUM DEVICES	Smart Agriculture Prototype Minimum	114,37	Raspberry Pi 3B+ + Arduino Uno+4 x Soil Moisture Sensor + Temperature, Humidity Sensor (DHT22) + 8-Channel Relay + Mini Pump	1	114,37

It could be seen that the minimum materials cost 120€, which fulfils the low-cost target for the Smart Agriculture Prototype. However, other options are also explored in order to show a complete economic analysis

SERVICES	DESCRIPTION	PRICE/UNITS (€)	DESCRIPTION	UNITS	TOTAL
	Purely Wired Smart Agriculture Prototype	0	Basic Product. Included in the ENGINEERING DESIGN price	1	0
	Hybrid Smart Agriculture Prototype	200	Enhanced Product. Includes MQTT Server Configuration	1	200
	Wireless Smart Agriculture Prototype	300	Enhanced Product. Includes MQTT Server Configuration	1	300
EXTRA SERVICES	Large Number of Sensors	100	In case more than 10 sensors implemented	1	100
	Historical Record	100	Shows graphical records of devices	1	100
	Private Security Camera	150	Includes Remote Connectivity Option	1	150
	Remote Connectivity Option	50	Remote Connectivity Option	1	50
IOT PRODUCTS	MQTT Server Configuration	100	MQTT Server Configuration	1	100
	1-point IOT Domotic System + 1-channel relay	15	1 wireless point to be turn ON/OFF. Requires MQTT Server Configuration	1	16,99
	10-points IOT Domotic System + 10 x 1-channel relay	100	10 wireless points to be turn ON/OFF. Requires MQTT Server Configuration	1	119,9
	Alexa Compatibility (1 unit)	2	1 wireless compatibilty with Alexa	1	2
	Alexa Compatibility (10 units)	2	10 wireless compatibilty with Alexa	10	20
	Setapp: Android Remote Control	30	Includes the app installation and configuration in 2 Android devices. Requires MQTT Server Configuration	1	30

ENGINEERING DESIGN	DESCRIPTION	PRICE/UNITS (€)	DESCRIPTION	HOURS OF WORK	TOTAL
	Engineer work	20	Research and designing work for any of the services	300	6000
	Supervisor advice	50	Tutor's proposals and supervision	20	1000
					7000

TOTAL BUDGET**= COMPUTATION + ACCESORIES + REQUIRED SENSORS+ OUTPUTS + EXTRA + ENGINEERING DESIGN + SERVICES**

BUDGET EXAMPLES	TOTAL	DESCRIPTION
1	7114,37 €	Purely Wired Smart Agriculture Prototype. Minimum devices
2	7314,37 €	Hybrid Smart Agriculture Prototype. Minimum devices
3	7414,37 €	Wireless Smart Agriculture Prototype. Minimum devices
4	7464,37 €	Purely Wired Smart Agriculture Prototype. Minimum devices + All Extra Services
5	7269,9 €	MQTT Server Configuration + 10-points IOT Domotic System + 10 x 1-Channel Relay
6	7269,9 €	MQTT Server Configuration + 10-points IOT Domotic System + 10 x 1-Channel Relay + Setapp: Android Remote Control
7	7544,37 €	Wireless Smart Agriculture Prototype. Minimum devices + MQTT Server Configuration + 2-points IOT Domotic System + 2x1-Channel Relay + Setapp: Android Remote Control
8	7564,37 €	Wireless Smart Agriculture Prototype. Minimum devices + MQTT Server Configuration + 10-points IOT Domotic System + 10x1-channel relay + Setapp: Android Remote Control + Alexa Compatibility

8.2 Socio-economic Summary

The next information is stated in order to abstract the whole socio-economic environment studied along the different sections of this project. However, it should be noted that the global context presented in *Chapter 1* completes this section.

In a general perspective, the previous budget shows that the main cost of the project is that from its engineering design and supervision, which even seems assumable as long as commercial solutions may charge a proportional amount to the commercialization of the prototype or its derivatives. Accordingly, it could be stated that the minimum components required for this sensing system to work suppose not more than **120€**. Nevertheless, this price could be visibly increased in case more of the minimum sensors or functionalities were required. It is also important to consider that the proposed offers should only be used as indicators in order to locate each of the different designs' associated costs comparatively.

In detail, the product that includes the Wireless Smart Agriculture Prototype with all its IOT functionalities, including remote control via the developed Android app and Alexa compatibility, is of approximately **7.500€**. This possibility could offer solutions not only for the Smart Agriculture sector but also to domotics and any automation processes for home and industrial control.

In conclusion, the strategic situation for the budget proposed under its target market seems very positive. On the one hand, it offers a Smart Agriculture option according to the latest Raspberry Pi model for June 2019, which has been demonstrated that, in fact, it is a completely efficient mini-computer with network connectivity that allows an easy IOT integration.

8.3 Regulatory Framework and Law

The Smart Agriculture Prototype and any IOT-based solutions need to follow several commercialization rules. The following paragraphs refer to the regulation required for each of their elements to be used for commercial purposes and are referenced.

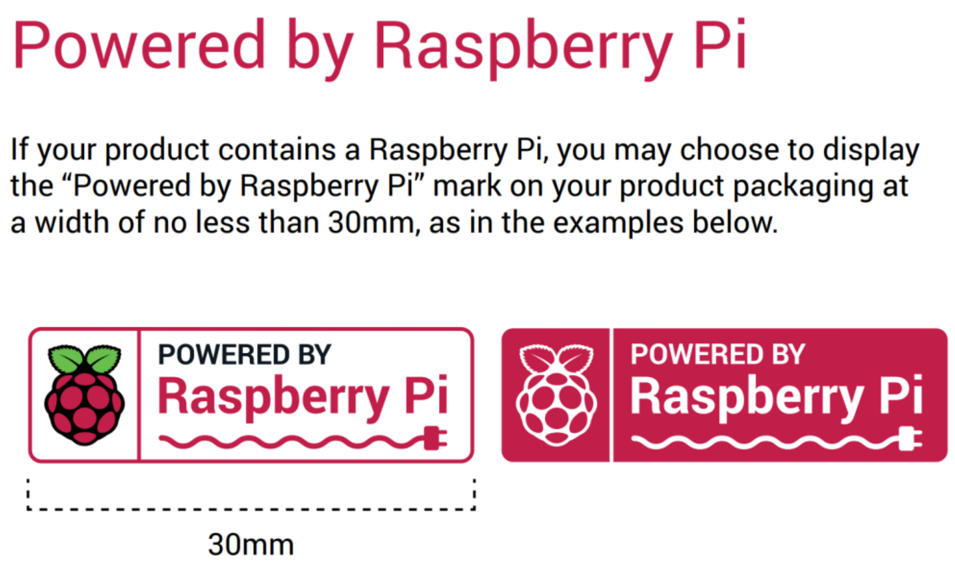
- **Raspberry Pi**

  Fortunately, the core of the project does not require any specific license to be marketed. On the other hand, it is important to note that a project based on Raspberry Pi architecture needs to show that it is "***Powered by Raspberry Pi***" [40].

  Under a closer analysis, the words "Raspberry Pi" are defined as the "Raspberry Pi Marks" and should be the only ones that respond to this product. Therefore, references to Raspberry or Pi would not be allowed for strict uses, though they may be understood colloquially. In the same way, "*Raspberry Pi is a trademark of the Raspberry Pi Foundation*".

  Moreover, if the Raspberry Pi is used as based for a project or product, there are also some restrictions on the using of Raspberry Pi Mark. It should use expressions such as "*runs on*" or "*compatible with*" to indicate the product *needs* the Raspberry Pi to work.

  Regarding the view appearance, a product based on Raspberry Pi should follow strict *Visual Identity Guidelines* [41]. The most important are described:



Powered by Raspberry Pi

If your product contains a Raspberry Pi, you may choose to display the "Powered by Raspberry Pi" mark on your product packaging at a width of no less than 30mm, as in the examples below.

POWERED BY Raspberry Pi
POWERED BY Raspberry Pi
30mm

Figure 78. Raspberry Pi Visual Identity Reference

Finally, Raspberry Pi Foundation states that any charitable donation in concept of using their product may be appreciated. It also insists on a good use of the product and does not reject applying newer conditions eventually.

In a general view, Raspbian distribution is protected under GPL (General Public License). It is one of the most extended license in open-source fields and allows the use, modification and sharing of the OS.

- **Arduino:**
  Its hardware design is regulated under the Creative Commons Attribution Share-Alike [42], an open-source definition. On the other hand, the Arduino software is also open-source, which, as exposed in the previous page, introduces no restriction to selling.

  Consequently, it is a free product to be totally or partially modified, as long as it credits Arduino and, for gentleness, any source that might be implied in the modification. It is also possible to be used for commercial purposes.

- **ESP32:**
  This product belongs to ESPRESSIF [43]. The following extract was highlighted in order to show it is also available to be used as a part of a product or system:

```
ESPRESSIF
MIT
License
             Copyright (c) 2018 <ESPRESSIF SYSTEMS (SHANGHAI) PTE LTD>
             Permission is hereby granted for use on all ESPRESSIF SYSTEMS products, in
            which case,
             it is free of charge, to any person obtaining a copy of this software and
            associated
             documentation files (the "Software"), to deal in the
            Software without restriction, including
             without limitation the rights to use, copy, modify,
            merge, publish, distribute, sublicense,
             and/or sell copies of the Software, and to permit persons to
            whom the Software is furnished
             to do so, subject to the following conditions:
             The above copyright notice and this permission notice shall be included in
            all copies or
             substantial portions of the Software.
             THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
             IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
            FITNESS
```

- **Alexa:**

This voice assistant is used only in compatibility with the wireless proposal of the project. It is important to notice that, therefore, Alexa is not a part of the project, but its control is compatible to the system's message reception. In this sense, its use is legitimate. Obviously, in case an industry or particular was interested on implementing a system based on this project it would not be possible to gain benefit of the selling of the assistant itself. On the contrary, configuration costs of the Alexa-compatible system may be charged, but not the selling itself.

In case the project would like to fully integrate Alexa, there is also a more complex solution. On this perspective, the system should be considered as a Alexa *built-in* product and several restrictions apply, on accepting the Amazon Alexa Developer legal terms. This option was absolutely discard due to the using of Alexa only as an optional compatibility of the system.

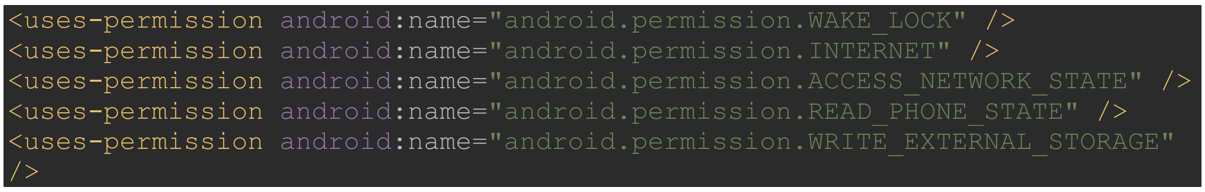
- **Security Camera and Remote Control**
  The using of a surveillance system and its broadcast exposes the user to a certain degree of vulnerability, even on applying some or all of the Internet security techniques that this research offers. Accordingly, this exposition should be notified to the user and, if possible, a conformity agreement should be signed.

  In case of using this system for alternative uses, it should be known that there is a high level of controversy and there is no specific regulation that applies categorically to each of the cases. As a general rule, any camera system must be used with caution, with notification to the user or, in case of recording a working space, also informing the workers of its presence, with respect to their dignity and privacy as much as possible. The final section of the present chapter details further cybersecurity regulation.

- **Thingspeak:**
  Although this service imposes some restrictions, it is able to be used for pure commercial purposes and also for personal uses.

  a) *Commercial purposes* require a standard commercial license
  b) *Personal uses* is the option proposed. On the one hand, once an account is created, it is free. However, it has some limitations, such as being impossible to process more than one message in 15 seconds, a 3 million messages limitation per year and the option of only managing up to 4 channels

- **Setapp, Android App.**
  The regulatory framework may also consider the developed Android app, specifically its required permissions. The most important are the accessing to Internet, in order to be able to send the MQTT messages, and the writing to external storage permission, for the SQLite database to work.



```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

Figure 78. Android Developed App Required Permissions

If not agreed, Setapp will not run and, as a result, the user will not be able to control MQTT devices remotely.

- **Industrial supervision:**
  Nowadays it is not allowed to use an Arduino or Raspberry Pi system for *quality test* or *certification*. The main reason is that there is no homogeneous composition of some of the elements. Also the system design itself may infer disperse data samples. Moreover, please consider the great importance in the results of using a defective sensor or wire. As an example, a based-on-Arduino sound measurement device is currently not able to be compared to the accuracy of a sound level meter device and, therefore, has no Law consideration.

  Fortunately, those systems *are* perfectly employable for *industrial supervision*, for either a certain magnitude indicator or as a complement to a control system.

- **Data protection and cibersecurity:**
  The Law that may apply to the data transmission from the different services of the present prototype could only be referred under general legislatives, at the current time (June 2019), due to the diffuse framework that still regulates the new IOT sectors. As long as there is no further particular legislation, therefore, this section could accurately only relate to the European Union global framework.

  First of all, the "***REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016***" [45], concerns the protection of *natural persons* in sense of both using and transferring their personal data. According to the previous reference, the *Reglamento* is valid since 25th May of 2018 and could be consulted if more detailed data protection is required.

In other matters, cybersecurity is considered under the "***DIRECTIVA (UE) 2016/1148 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 6 de julio de 2016*** *[46]",* which aims to guarantee a common level of security of networks and information systems in the European Union. It also targets on improving the whole Union coordination in case a severe cyberattack occurs.

Finally, the "***DIRECTIVA (UE) 2016/680 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016***" [47] focuses mainly on the protection of *natural persons* with regard to the processing of personal data by the competent authorities, with the purpose of preventing, investigating and detecting any criminal offense.

In summary, the devices and services that compose this Final Project have proven to be easily valid for commercial purposes, which was precisely the initial approach.

On the one hand, Raspberry Pi could be used as part of a product as long as the words "*Powered by Raspberry Pi"* appear, while Arduino and ESP32 do not impose limitations due to its regulation under open-source. To continue with the recapitulation, Alexa voice assistant is only a device compatible to the product, so no Law infringement would proceed in this sense.

Moreover, the surveillance system and the remote control functions require the emission of data through the Internet, which supposes no legal obstacle if, even with security methods, the user agrees some remote risk of data interception. Thingspeak has been proposed to be used with a personal license, whose limitations are enough to many project's implementations even in industrial control. In addition, the Android app Setapp would require the agreement of its permissions to function.

The Law that explains the data protection valid for today (June 2019) was referred under the European Union perspective. However, new specific regulation for IOT sectors may be imposed further.

CHAPTER 9:

CONCLUSIONS

9.1 Adversities during the Project

The project was initially conceived for a smaller system, for which a 16 GB SD was thought to be enough. However, Raspbian installation and the described modules finally required a migration to a 32 GB card in order to continue working with no compromise on the fluency of the Raspberry Pi.

It is also remarkable the discarded option of the ESP01-S. This device belongs to the **ESP8266 family** [], as well as the ESP32 does, and was the first wireless option to be studied (see *Figures 79* [36] and *80* [37]). It only offers 2 GPIO (extensible to 4) but it appeared to have many drawbacks. First, it was uncomfortable to flash, process which needs many *dupont* wires or either an external USB adapter plus forcing a certain shortcut between pins so that the device enters flashing mode.

Some source feeding problems were also perceived in several units, which resulted in involuntary resets. A general opinion of this model is that, despite of being one of the first and smallest WiFi modules and, therefore, supposed an important revolution in electronic devices related to IOT, it is precisely difficult to be operated and owns insufficient GPIO. For all the previous reasons, this option was abandoned and substituted by the ESP32 module already described.

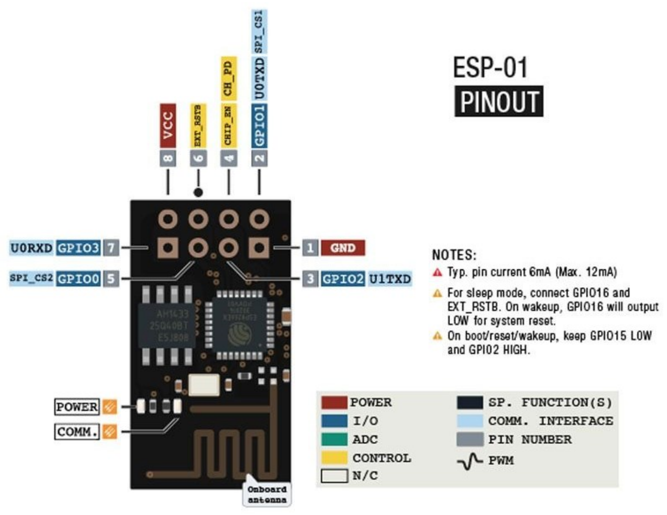
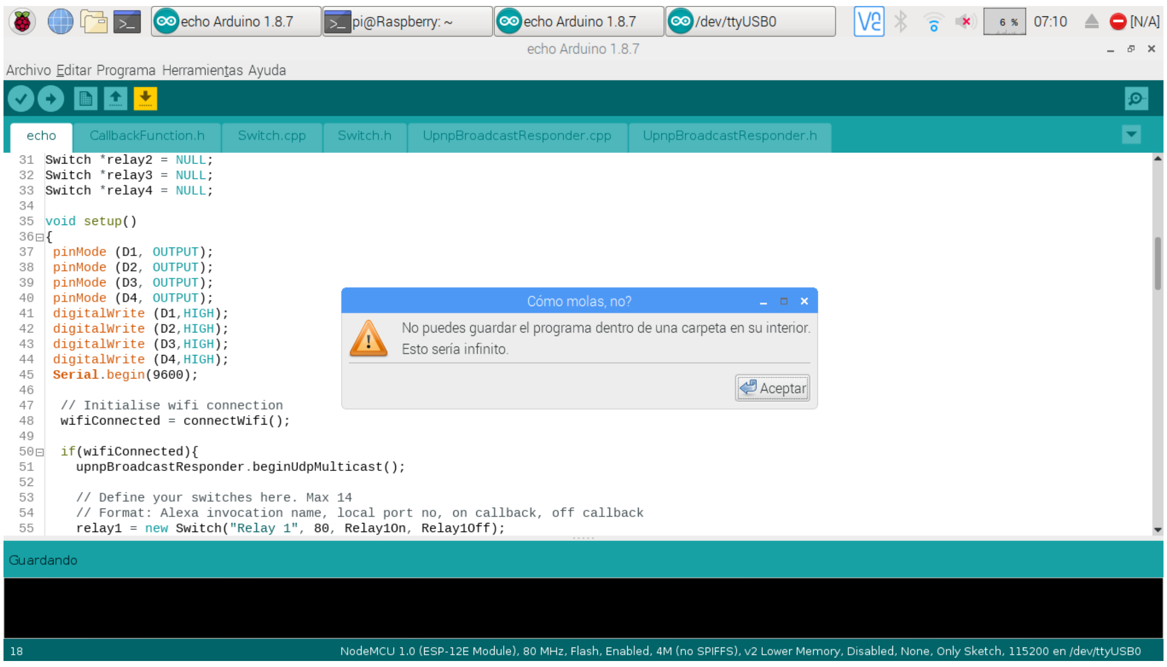


Figure 79. ESP 01 Pinout (Unused option, left)
Figure 80. ESP 01S View (Unused option, right)

As a special consideration with no academic purpose, *Figure 81* shows an unusual screenshot that the author finds it deserved a place in this Final Project.



```
Switch *relay2 = NULL;
Switch *relay3 = NULL;
Switch *relay4 = NULL;

void setup()
{
 pinMode (D1, OUTPUT);
 pinMode (D2, OUTPUT);
 pinMode (D3, OUTPUT);
 pinMode (D4, OUTPUT);
 digitalWrite (D1,HIGH);
 digitalWrite (D2,HIGH);
 digitalWrite (D3,HIGH);
 digitalWrite (D4,HIGH);
 Serial.begin(9600);

  // Initialise wifi connection
  wifiConnected = connectWifi();

  if(wifiConnected){
    upnpBroadcastResponder.beginUdpMulticast();

    // Define your switches here. Max 14
    // Format: Alexa invocation name, local port no, on callback, off callback
    relay1 = new Switch("Relay 1", 80, Relay1On, Relay1Off);
```

Cómo molas, no?

No puedes guardar el programa dentro de una carpeta en su interior. Esto sería infinito.

Aceptar

```
Guardando
```

```
18        NodeMCU 1.0 (ESP-12E Module), 80 MHz, Flash, Enabled, 4M (no SPIFFS), v2 Lower Memory, Disabled, None, Only Sketch, 115200 en /dev/ttyUSB0
```

Figure 81. True Arduino IDE Screenshot from the Author

Lastly, it is important to explain the final idea of running the MQTT server on the Raspberry Pi. At the beginning there was a consideration of using an external computer to this purpose, which, obviously, had some advantages, because the MQTT devices could be switched even if the Raspberry Pi was busy on another subject or off, and, especially, less computation would be required from the Raspberry Pi. However, a second computer for running a lightweight protocol such as MQTT saves no energy. Moreover, Raspberry Pi was finally able to run simultaneously all the services proposed in this project perfectly.

9.2 Future

As a prototype, the Smart Agriculture project may propose only a coarse approximation to the actual industry methods and considers little interconnection between the different sensors' data, which could provide very useful information.

For this reason, the first line of future improvements is using *big data analysis* in order to understand better how sensors interact. It is also possible to use this approach to find more efficient configurations or schedules such as to minimize the electrical consumption. The importance of proposing the storage of data in Thingspeak was not only to create graphical records but specifically to apply *Mathworks* computations in this sense. Once channels receive real-time information, Matlab processing is easy to implement.

The next proposal tries to avoid even using a Raspberry Pi. It is a very similar approach, but translates its computation to Thingspeak. On the one hand, ESP32 wireless devices needs to sense a certain characteristic and *send* the information to the platform, where the information is combined. Note that the present prototype does not consider the option of devices itself updating data, as the Raspberry Pi manages the process.

Another consideration is to test the MQTT service with several clients in order to determine its functional limit. It is also possible to change the Quality of Service (QoS), currently at 0, in order to strengthen the performance, although no issue was found yet. QoS 0 means that eventually a message may not be delivered through the Internet. Setting a QoS to 1 or 2 could ensure that any message could reach to the broker.

Finally, scatterplot and regression curves may be obtained from the sensors, so that the global accuracy of the system is improved, especially for industrial purposes. On a first perspective, same type of sensors should present similar characteristics, so *outlayer* sensors might be discarded. This information is also important in order to compensate any calibration deviation that a certain unit may suffer.

9.3 Final Message

Once the present design is exposed and its appropriate justification presented, it is accurate to state that the initial targets were successfully covered. On the one hand, a Smart Agriculture Prototype, whose sensing responds automatically to environmental stimuli, has been exposed. Wireless alternatives were studied too, so that the project focuses on a modern approach.

IOT technologies play a very significant role in the project, especially due to the use of a local server and a developed Android app in order to control devices remotely, which respond to an increasing tendency of connecting devices via MQTT lightweight protocol.

In the economical perspective, the system explained uses low-cost sensors without compromising the overall performance, which could be understood as a global target achieved. In this way, there is also a strong proposal on integrating mini-computers into the market, for which the present design introduces an only **120€** material costs proposal, in the minimum prototype exposed, which also includes the main functionalities described along this exposition.

In line with the open-source fundamentals, the author would like to consider this Final Project as a short contribution to a whole sector which is in considerable expansion, with the hope of having exposed some of the ideas that could globally help and be developed in the future.

The legal aspects that concern the commercialization of any external device as part of a system, such as the Raspberry Pi and the Arduino Uno, were studied in detail and are exposed under a global context where Internet security arises as a serious issue to be strengthened, while domotics-related devices are still defining their particular legislation on industry.

The shown technology, design and research aims also to be considered as a proof of merit that completes this Final Project of the Bachelor Degree in Audiovisual System Engineering (UC3M).

Moreover, the Prototype is obviously open to further improvements and new modular integrations so that it could be adapted to a different sector than that of the Smart Agriculture. The economy market considered in this project may indicate that the socio-economic position of products related to IOT technologies is completely profitable, which is applied to any of its fields.

Finally, it is remarkable to state that future lines suggest that sensor-based systems could get benefit from the big data processing, in order to contribute to general flow automatization and optimize any design to a greater extent.

REFERENCES:

[1] "Internet of Things – number of connected devices worldwide 2015-2025". Statista.com.
https://www.statista.com/statistics/512673/worldwide-internet-of-things-market/.
(Online: June 16th, 2019).

[2] "IoT Project List: Database Of Enterprise IoT Projects 2018". Statista.com https://www.statista.com/topics/4134/smart-agriculture/.
(Online: June 16th, 2019).

[3] "MQTT".
http://mqtt.org/.
(Online: June 16th, 2019).

[4] "Raspberry Pi"
https://www.raspberrypi.org/
(Online: June 16th, 2019).

[5] "Arduino"
https://www.arduino.cc/
(Online: June 16th, 2019).

[6] "Alexa"
https://developer.amazon.com/es/alexa
(Online: June 16th, 2019).

[7] "Google Home"
https://store.google.com/es/product/google_home
(Online: June 16th, 2019).

[8] "Alexa Echo Dot (Third generation)". Amazon.es https://www.amazon.es/Echo-Dot-3-%C2%AA-generaci%C3%B3n-inteligente/dp/B0792HCFTG/ref=cm_wl_huc_item
(Online: June 16th, 2019).

[9] "Google Home Mini". Google.es
https://store.google.com/es/config/google_home_mini?sku=_google_home_mini_chalk
(Online: June 16th, 2019).

[10] Kinsella, Bret. "Google Assistant Actions Total 4,253 in January 2019, Up 2.5x in Past Year but 7.5% the Total Number Alexa Skills in U.S.”. Voicebot.ai. https://voicebot.ai/2019/02/15/google-assistant-actions-total-4253-in-january-2019-up-2-5x-in-past-year-but-7-5-the-total-number-alexa-skills-in-u-s/
(Online: June 16th, 2019).

[11] “Mobile Operating System Market Share Spain”. Gs.statcounter.com.
http://gs.statcounter.com/os-market-share/mobile/spain
(Online: June 16th, 2019).

[12] “Thingspeak service”.
https://thingspeak.com/
(Online: June 16th, 2019).

[13] “Raspbian”. Raspberrypi.org.
https://www.raspberrypi.org/downloads/raspbian/
(Online: June 16th, 2019).

[14] “Debian”
https://www.debian.org/index.es.html
(Online: June 16th, 2019).

[15] "Raspberry Pi Compute Module 3+ (Datasheet)". Raspberrypi.org
https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf
(Online: June 16th, 2019).

[16] “Raspberry Pi 3 Model B+”. Raspberrypi.org.
https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf
(Online: June 16th, 2019).

[17] “Python 3.0 Release”
https://www.python.org/download/releases/3.0/
(Online: June 16th, 2019).

[18] “Arduino IDE 1.8.9”
https://www.arduino.cc/en/Main/Software
(Online: June 16th, 2019).

[19] “Temperature and humidity module. AM2302 Product Manual” Akizukidenshi.com. https://akizukidenshi.com/download/ds/aosong/AM2302.pdf
(Online: June 16th, 2019).

[20] “Temperature and humidity module. DHT11 Product Manual”. Akizukidenshi.com. https://akizukidenshi.com/download/ds/aosong/DHT11.pdf
(Online: June 16th, 2019).

[21] “Arduino Libraries”. Learn.adafruit.com.
https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use/arduino-libraries
(Online: June 16th, 2019).

[22] “DHT-sensor-library”. Github.com.
https://github.com/adafruit/DHT-sensor-library
(Online: June 16th, 2019).

[23] “Gas Sensor Comparison”, “Datasheets: SNS-MQ135.pdf”. Mysensors.org.
https://www.mysensors.org/build/gas
(Online: June 16th, 2019).

[24] “BMP280 (Datasheet)”. Bosch.com.
https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001.pdf
(Online: June 16th, 2019).

[25] “BME280 (Datasheet)”. Bosch.com.
https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280-DS002.pdf
(Online: June 16th, 2019).

[26] “Electrical specifications of GY-BMP280 Module”. Mantech.co.za.
http://www.mantech.co.za/datasheets/products/GY-BMP280-3.3_BG.pdf
(Online: June 16th, 2019).

[27] “Ultrasonic Module HC - SR04”. Mouser.com
https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf
(Online: June 16th, 2019).

[28] “AD11 & AD12 Waterproof pH Testers (User Manual)”. Adwa instruments.
http://www.adwainstruments.com/manuals/manual_ad11_12.pdf
(Online: June 16th, 2019).

[29] “AD31 & AD32 Waterproof EC/TDS Testers (User Manual)”. Adwa instruments.
http://www.adwainstruments.com/manuals/manual_ad31_32.pdf
(Online: June 16th, 2019).

[30] “How PIRs Work”. Learn.adafruit.com.

https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work
(Online: June 16th, 2019).

[31] "ESP32 Series (Datasheet)". Espressif.com.
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
(Online: June 16th, 2019).

[32] "Amazon Echo Dot (2nd Generation) specs". Cnet.com.
https://www.cnet.com/products/amazon-echo-dot-2nd-generation/specs/
(Online: June 16th, 2019).

[33] Nassir-Malik. "IOT-Pi3-Alexa-Automation". Github.com
https://github.com/nassir-malik/IOT-Pi3-Alexa-Automation
(Online: June 16th, 2019).

[34] Nicholas3388. "ESP32 voice control via Amazon Echo Dot (Alexa)". Github.com
https://github.com/Nicholas3388/alexa-esp32
(Online: June 16th, 2019).

[35] "Sricam Indoor IP Camera". Sricam.com.
www.sricam.com/download/id/263a06190e7f4394ad099043ddbd5df9.html
(Online: June 16th, 2019).

[36] "eSpeak text to speech".
http://espeak.sourceforge.net/
(Online: June 16th, 2019).

[37] Raspberry Pi Guy. "LCD". Github.com
https://github.com/the-raspberry-pi-guy/lcd
(Online: June 16th, 2019).

[38] "SQLite".
https://www.sqlite.org/index.html
(Online: June 16th, 2019).

[39] "Eclipse Paho".
https://www.eclipse.org/paho/
(Online: June 16th, 2019).

[40] “What licensing conditions do I have to take care about when distributing a commercial product based on the Raspberry Pi?”. Raspberrypi.stackexchange.com.
https://raspberrypi.stackexchange.com/questions/61729/what-licensing-conditions-do-i-have-to-take-care-about-when-distributing-a-comme
(Online: June 16th, 2019).

[41] “Trademark rules and brand guidelines”. Raspberrypi.org.
https://www.raspberrypi.org/trademark-rules/
(Online: June 16th, 2019).

[42] “Frequently Asked Questions”. Arduino.cc.
https://www.arduino.cc/en/Main/FAQ#toc3
(Online: June 16th, 2019).

[43] “Copyrights and Licenses”. Docs.espressif.com.
https://docs.espressif.com/projects/esp-idf/en/latest/COPYRIGHT.html
(Online: June 16th, 2019).

[44] “ThingSpeak™ Licensing FAQ”. Thingspeak.com.
https://thingspeak.com/pages/license_faq
(Online: June 16th, 2019).

[45] “REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO, de 27 de abril de 2016”. Boe.es.
https://www.boe.es/doue/2016/119/L00001-00088.pdf
(Online: June 16th, 2019).

[46] “DIRECTIVA (UE) 2016/1148 DEL PARLAMENTO EUROPEO Y DEL CONSEJO, de 6 de julio de 2016”. Eur-lex.europa.eu.
https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32016L1148
(Online: June 16th, 2019).

[47] “DIRECTIVA (UE) 2016/680 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016”
https://www.boe.es/doue/2016/119/L00001-00088.pdf
(Online: June 16th, 2019).

[48] “ESP-01 WiFi Module (Datasheet)”. Microchip.ua.
http://www.microchip.ua/wireless/esp01.pdf
(Online: June 16th, 2019).

[49] “ESP8266 AT Instruction Set (Datasheet)”. Espressif.com. https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf
(Online: June 16th, 2019).

REFERENCES (FIGURES):

Figure 1. [1] "Internet of Things – number of connected devices worldwide 2015-2025".
Statista.com. https://www.statista.com/statistics/512673/worldwide-internet-of-things-market/. (Online: June 16th, 2019).

Figure 2. [2] "IoT Project List: Database Of Enterprise IoT Projects 2018". Iot-Analytics
https://iot-analytics.com/product/list-of-1600-enterprise-iot-projects-2018/. (Online: June 16th, 2019).

Figure 3. [3] "IoT Project List: Database Of Enterprise IoT Projects 2018". Iot-Analytics
https://www.csl.cornell.edu/curie2014/projects.html. (Online: June 16th, 2019).

Figure 6. [4] "Raspberry Pi Pinout Diagram | Circuit Notes" Jameco.com
https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html (Online: June 16th, 2019).

Figure 7. [5] "GPIO Electrical Specifications" Mosaic-industries.com.
http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications (Online: June 16th, 2019).

Figure 11. [6] "Festnight-humedad-detección-Detección-higrómetros". Amazon.es.
https://www.amazon.es/Festnight-humedad-detecci%C3%B3n-Detecci%C3%B3n-higr%C3%B3metros/dp/B07RYC2K83/ref=cm_wl_huc_item
(Online: June 16th, 2019).

Figure 12. [7] "Daorier-humedad-higrómetro-sensibilidad-Arduino". Amazon.es.
https://www.amazon.es/daorier-humedad-higr%C3%B3metro-sensibilidad-Arduino/dp/B06VY3LB92/ref=cm_wl_huc_item
(Online: June 16th, 2019).

Figure 13. [8] "Soil Moisture Sensor – How to Use it?". Invent.module143.com.
http://invent.module143.com/soil-moisture-sensor-how-to-use-it/
(Online: June 16th, 2019).

Figure 14. [9] "ARCELI-Capacitivo-detección-analógica-Electrónico". Amazon.es.
https://www.amazon.es/ARCELI-Capacitivo-detecci%C3%B3n-anal%C3%B3gica-Electr%C3%B3nico/dp/B07MY2RNTT/ref=cm_wl_huc_item
(Online: June 16th, 2019).

Figure 15. [10] “Sensor DHT22 Temperatura y Humedad para Arduino”. Tiendatec.es. https://www.tiendatec.es/arduino/sensores/790-sensor-dht22-temperatura-y-humedad-8472496009812.html?gclid=EAIaIQobChMIuO62xsXm4gIVBrXtCh3Rkg53EAQYBCABEgLSQ_D_BwE (Online: June 16th, 2019).

Figure 16. [11] “Sensor DHT11 Temperatura y Humedad para Arduino”. Tiendatec.es. https://www.tiendatec.es/arduino/sensores/588-sensor-dht11-temperatura-y-humedad-para-arduino-8405881480002.html?utm_source=doofinder&utm_medium=results
(Online: June 16th, 2019).

Figures 17, 18, 19. [12] “Temperature and humidity module. AM2302 Product Manual”. Akizukidenshi.com.
https://akizukidenshi.com/download/ds/aosong/AM2302.pdf
(Online: June 16th, 2019).

Figure 20. [13] “InisIE MQ135 Módulo Sensor de Calidad de Aire nocivo detección de Gases”. Amazon.es. https://www.tiendatec.es/arduino/sensores/588-sensor-dht11-temperatura-y-humedad-para-arduino-8405881480002.html?utm_source=doofinder&utm_medium=results
(Online: June 16th, 2019).

Figures 21, 22. [14] “Datasheets: SNS-MQ135.pdf”. Mysensors.org.
https://www.mysensors.org/build/gas
(Online: June 16th, 2019).

Figure 23. [15] “HKWX fotorresistencia photosensitive de detección módulo de sensor de luz para Arduino”. Amazon.es.
https://www.amazon.es/gp/product/B01G53DYVC/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
(Online: June 16th, 2019).

Figure 24. [16] “Module: GY-BMP280-3.3”. Mantech.co.za.
http://www.mantech.co.za/datasheets/products/GY-BMP280-3.3_BG.pdf
(Online: June 16th, 2019).

Figure 25. [17] “Uzinb 1PC Lluvia líquido del Agua del módulo del Sensor de Nivel de Profundidad de detección para Arduino”. Amazon.es
https://www.amazon.es/gp/product/B07G99442C/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
(Online: June 16th, 2019).

Figure 26. [18] “Módulo HC-SR04 sensor distancia por ultrasonidos para Arduino y Raspberry Pi”. Tiendatec.es
https://www.tiendatec.es/arduino/modulos/392-modulo-hc-sr04-sensor-distancia-por-ultrasonidos-para-arduino-y-raspberry-pi-8403921180011.html
(Online: June 16th, 2019).

Figure 27. [19] “HC-SR04 Working Principle”. Electronicwings.com
https://www.electronicwings.com/sensors-modules/ultrasonic-module-hc-sr04
(Online: June 16th, 2019).

Figure 28. [20] “Influence of soil pH on plant nutrient availability”. PDA.org
https://www.pda.org.uk/pda_leaflets/24-soil-analysis-key-to-nutrient-management-planning/influence-of-soil-ph-on-plant-nutrient-availability/
(Online: June 16th, 2019).

Figure 29. [21] “Medidor de pH Adwa AD11 waterproof”. Medidordeph.com.
https://medidordeph.com/medidor-de-ph-adwa-ad11.html?___store=default&gclid=EAIaIQobChMIvJW9pYvr4gIVAflRCh3-VgidEAQYASABEgIRPPD_BwE
(Online: June 16th, 2019).

Figure 30. [22] “Medidor de Conductividad EC/TDS y Temperatura Adwa Waterproof AD31”. Medidordeph.com .
https://medidordeph.com/medidor-de-ec-temperatura-adwa-waterproof.html
(Online: June 16th, 2019).

Figure 31. [23] “Single Element Pyroelectric Detector”. Cdn-learn.adafruit.com.
https://cdn-learn.adafruit.com/assets/assets/000/010/138/original/an2105.pdf
(Online: June 16th, 2019).

Figure 32. [24] “About PIR Motion sensor”. Osoyoo.com
http://osoyoo.com/2017/07/27/arduino-lesson-pir-motion-sensor/
(Online: June 16th, 2019).

Figure 33. [25] “ELEGOO 8 Channel DC 5V Relay Module with Optocoupler”. Elegoo.com
https://www.elegoo.com/product/elegoo-8-channel-dc-5v-relay-module-with-optocoupler/
(Online: June 16th, 2019).

Figure 34. [26] “Phototransistor Optocoupler”. Electronics-tutorials.ws.
https://www.electronics-tutorials.ws/blog/optocoupler.html
(Online: June 16th, 2019).

Figure 35. [27] “Control de relé mecánico”. Rsppi.blogspot.com.
http://rsppi.blogspot.com/2013/07/control-de-rele-mecanico.html
(Online: June 16th, 2019).

Figure 36. [28] “PINOUT DEL NODEMCU”. Luisllamas.es.
http://rsppi.blogspot.com/2013/07/control-de-rele-mecanico.html
(Online: June 16th, 2019).

Figure 49. [29] “Amazon Echo Dot (2nd Generation, Black)”. Thegadgetshop.co.za.

https://www.thegadgetshop.co.za/audio-visual/amazon-echo-dot-2nd-generation-black
(Online: June 16th, 2019).

Figure 50. [30] “If this then that “.
https://ifttt.com/
(Online: June 16th, 2019).

Figure 59. [31] Raspberry Pi Guy “LCD “. Github.com
https://github.com/the-raspberry-pi-guy/lcd
(Online: June 16th, 2019).

Figure 62. [32] “I2C Wiring to 3.3V Arduino Due“. Startingelectronics.com
https://startingelectronics.org/tutorials/arduino/modules/pressure-sensor/
(Online: June 16th, 2019).

Figure 63. [33] “Thingspeak“.
https://thingspeak.com/
(Online: June 16th, 2019).

Figure 72. [34] “Eclipse Paho “.
https://www.eclipse.org/paho/
(Online: June 16th, 2019).

Figure 78. [35] “Powered by Raspberry Pi “. Static.Raspberrypi.org.
https://static.raspberrypi.org/files/Raspberry_Pi_Visual_Guidelines_2018.pdf
(Online: June 16th, 2019).

Figure 79. [36] “ESP-01 modules programming board “. Mischianti.org.

(Online: June 16th, 2019).

Figure 80. [37] “ESP-01S Wifi Module “. Electrodragon.com.
https://www.electrodragon.com/product/esp-01-esp8266-wifi-module/
(Online: June 16th, 2019).