

End of Career Project

Electronics, Robotics and Mechatronics
Engineering

Electronic system intended for the management of
sensors in agriculture.

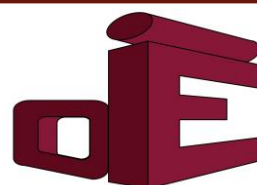
Sistema Electrónico destinado a la gestión de
sensores para la agricultura.

Author: Lucía Gálvez del Postigo Gallego

Tutor: Juan García Ortega

Department of Electronic Engineering
Electronic Technology Area
Higher Technical School of Engineering
University of Seville

Seville, 2020



Trabajo Fin de Grado
Ingeniería Robótica, Electrónica y Mecatrónica

Sistema Electrónico destinado a la gestión de sensores para la agricultura

Autor:

Lucía Gálvez del Postigo Gallego

Tutor:

Juan García Ortega

Profesor titular

Departamento de Ingeniería Electrónica

Área de Tecnología Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado: Sistema Electrónico destinado a la gestión de sensores para la agricultura

Autor: Lucía Gálvez del Postigo Gallego

Tutor: Juan García Ortega

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

*To all those who believed in me
even when I did not.*

*A todos aquellos que creyeron en
mí aun cuando yo no lo hacía.*

Acknowledgements

To all teachers who truly have a vocation and are passionate about what they teach, looking for the real interest of the students. To Juan, especially, for having welcomed me into this project, for his involvement and commitment both when he was my teacher and now as a tutor.

To my father, for teaching me that constancy is the key that opens all doors and reminding me anytime that "I can". To my mother, for teaching me that every "no" is just a step that brings you closer to the next "yes". To my sister, for always being there for me with the greatest smile to encourage me.

To my colleagues, because without them, the path would not have been the same. To Carmen, in particular, for filling these years with unforgettable memories. To all those who came into my life, sooner or later, to stay forever. Special mention of my friends Doru, for all the help, motivation and support; and Javi for providing me with a part of the material

A todos los profesores que verdaderamente tienen vocación y son apasionados de aquello que enseñan, buscando el interés real de los alumnos. A Juan, especialmente, por haberme acogido en este proyecto, por su implicación y empeño tanto cuando fue mi profesor, como actualmente como tutor.

A mi padre, por enseñarme que la constancia es la llave que abre todas las puertas y recordarme que "yo puedo". A mi madre, por inculcarme desde siempre que cada "no" es sólo un paso que te acerca más al próximo "sí". A mi hermana, por estar siempre ahí para mí con la mayor de las sonrisas y buscando comprenderme en cada momento.

A mis compañeros, porque sin ellos, el camino no hubiera sido el mismo. A Carmen, en particular, por llenar de recuerdos inolvidables estos años. A todos aquellos que llegaron a mi vida, más pronto o más tarde, para quedarse para siempre. Mencionar especialmente a mis amigos Doru, por toda la ayuda, motivación y apoyo; y Javi por proporcionarme parte del material.

The purpose of this study is to create a data logger from ground up. The aim is to provide the device with some flexibility compared to the existing ones. Therefore, firstly an analysis of a real data logger is performed, then the implementation is carried out. In this case, an Arduino is used as a microcontroller, to which some sensors and a SIM808 module are connected. The results showed how the device met the requirements preset. Nevertheless, a conclusion reached was that in future implementations, there is still some work to be done. For instance, the prototype could be improved in terms of the communication channel. In this case, only the temperature sensor was available, for this reason the range was defined by default. Nevertheless, an improvement would be to allow the user to decide the admissible range for each sensor.

Keywords

Microprocessor system, Data logger, Monitoring, Sensors, 3G communication, Arduino, Programming

Index

| | |
|---|-------------|
| Acknowledgements | IX |
| Abstract | XI |
| <i>Keywords</i> | <i>XI</i> |
| Index | XII |
| Index of tables | XIV |
| Index of figures | XV |
| Index of Illustrations | XVI |
| Notation | XVII |
| 1 Introduction | 2 |
| 1.1 <i>Requirements</i> | 3 |
| 1.2 <i>Objectives</i> | 4 |
| 2 System | 5 |
| 2.1 <i>Sensors</i> | 5 |
| 2.2 <i>Processing Unit</i> | 6 |
| 2.3 <i>Communication Channel</i> | 6 |
| 2.4 <i>User interface</i> | 7 |
| 2.5 <i>User</i> | 7 |
| 3 Implemented Solution | 8 |
| 3.1 <i>Requirements</i> | 8 |
| 3.2 <i>Justification and choices for each piece</i> | 9 |
| 3.2.1 <i>Sensors</i> | 9 |
| 3.2.2 <i>Processing Unit</i> | 13 |
| 3.2.3 <i>Communication channel</i> | 19 |
| 3.2.4 <i>User Interface</i> | 23 |
| 3.2.5 <i>User</i> | 24 |
| 3.3 <i>Implementation of the system</i> | 24 |
| 3.3.1 <i>Arduino board</i> | 24 |
| 3.4 <i>Final Outline</i> | 29 |
| 3.4.1 <i>Performance description</i> | 29 |
| 3.4.2 <i>Programming path</i> | 31 |
| 4 Tests | 34 |
| 4.1 <i>Sensors</i> | 34 |
| 4.1.1 <i>DS18B20 Sensor's test</i> | 35 |
| 4.1.2 <i>Sensor's simulation</i> | 35 |
| 4.2 <i>Arduino</i> | 36 |
| 4.2.1 <i>Arduino Nano vs Mega</i> | 36 |
| 4.2.2 <i>Arduino Tests</i> | 37 |
| 4.2.3 <i>Writing/Reading from EEPROM</i> | 37 |
| 4.2.4 <i>Clock</i> | 38 |
| 4.3 <i>SD tests</i> | 40 |

| | | |
|------------------|--|-----------|
| 4.3.1 | Writing a file | 40 |
| 4.3.2 | Reading a file | 41 |
| 4.3.3 | Conclusion | 42 |
| 4.4 | <i>SIM808 tests</i> | 42 |
| 4.4.1 | Sending SMS | 42 |
| 4.4.2 | Receiving SMS | 43 |
| 4.4.3 | Conclusion | 43 |
| 4.5 | <i>Test conclusion and recapitulation</i> | 44 |
| 5 | Demo | 45 |
| 5.1 | <i>Demo Description</i> | 45 |
| 5.1.1 | Getting Started | 45 |
| 5.1.2 | Program flow | 48 |
| 5.1.3 | Extra-Functionalities | 50 |
| 5.2 | <i>Achievements</i> | 56 |
| 5.3 | <i>Conclusions</i> | 57 |
| 6 | Further investigation and future Improvements | 58 |
| 6.1 | <i>System improvements</i> | 58 |
| 6.1.1 | Adding more sensors | 58 |
| 6.1.2 | Adding another device | 58 |
| 6.1.3 | Security Checks | 58 |
| 6.1.4 | Change phone number while working | 58 |
| 6.1.5 | User Interface | 58 |
| 6.1.6 | Change range | 58 |
| 6.2 | <i>New approaches</i> | 59 |
| 6.2.1 | New choices for each Module | 59 |
| 6.2.2 | Hardware wiring and implementation | 64 |
| 7 | Annex | 67 |
| 7.1 | <i>Introduction to Serial Communication</i> | 67 |
| 7.1.1 | Synchronous versus Asynchronous Serial communication | 67 |
| 7.1.2 | RS-485 | 68 |
| 7.1.3 | MODBUS | 69 |
| Reference | | 71 |

INDEX OF TABLES

| | |
|--|----|
| Table 1. DTH11 Sensor characteristics | 10 |
| Table 2 Sensor DS18B20 characteristics | 11 |
| Table 3 HD-38 Sensor's characteristics | 12 |
| Table 4 Arduino Uno, Mega, Nano comparison | 15 |
| Table 5. Micro SD adapter Characteristics | 18 |
| Table 6 SIM808 Characteristics | 20 |
| Table 7. AT commands for setting up the parameters to receive/send SMS texts | 22 |
| Table 8. AT command for sending an SMS | 22 |
| Table 9. SMS Response description. | 23 |
| Table 10. SPI Arduino Pins according to the board used | 27 |
| Table 11. Address keys | 29 |
| Table 12. Keywords for the different functions of the system. | 30 |
| Table 13. Arduino Mega VS Nano, memory comparison. | 36 |
| Table 14. Summary of Tests Conclusions | 44 |
| Table 15. Soil Moisture Sensor Characteristics | 60 |
| Table 16. Soil Temperature Sensor Characteristics | 61 |
| Table 17. Air temperature, Pressure and Humidity sensor characteristics | 62 |
| Table 18. Wind Speed Sensor Characteristics | 63 |
| Table 19. Object benches Types on Modbus | 69 |
| Table 20. Master Data frame in RS485-ModBus Communication | 69 |
| Table 21. Slave Data frame in RS485-ModBus Communication | 70 |

INDEX OF FIGURES

| | |
|--|----|
| Figure 1. System description. | 5 |
| Figure 2. Sensors description. | 6 |
| Figure 3. Processing Unit description | 6 |
| Figure 4. System characteristics Information requests, data frequency | 7 |
| Figure 5. System Characteristics | 7 |
| Figure 6. Sensor's parameters to observe | 9 |
| Figure 7. Sensors- System | 13 |
| Figure 8. Sensors final Choice | 13 |
| Figure 9. System details with Arduino functionalities | 17 |
| Figure 10. System details adding micro SD adapter | 19 |
| Figure 11. System Scheme Adding SIM808 | 23 |
| Figure 12. Complete system description after completing the choices | 24 |
| Figure 13. SIM808-Arduino connection | 28 |
| Figure 14. Final system | 34 |
| Figure 15. Arduino Serial Port Communication: Demo- Part1 | 46 |
| Figure 16. Arduino Serial Port Communication: Demo - Part 2 | 46 |
| Figure 17. Arduino Serial Port Communication: Demo - Part 3 | 47 |
| Figure 18. Arduino Serial Port Communication: Demo – Part 4 | 47 |
| Figure 19. SMS exchanged in Getting Started Demo Part | 48 |
| Figure 20. Arduino Serial Port Communication: Demo - Part 5 | 49 |
| Figure 21. SMS texts with the measures of the sensors | 50 |
| Figure 22. Arduino Serial Port Communication: Address functionality | 51 |
| Figure 23. SMS aspect of Address functionality | 51 |
| Figure 24. SMS exchanged in Consult functionality and Data record on micro SD | 52 |
| Figure 25. . Arduino Serial Port Communication: Demo - Consult Functionality | 53 |
| Figure 26. Arduino Serial Port Communication: Demo - Frequency Functionality | 54 |
| Figure 27. SMS example of frequency functionality User's response | 54 |
| Figure 28. SMS example of frequency functionality | 55 |
| Figure 29. Arduino Serial Port Communication: Demo - Asking for a particular sensor's information. | 55 |
| Figure 30. Arduino Serial Port Communication: Demo - Alerts | 56 |
| Figure 31. Switch implementation of the system | 65 |
| Figure 32. RS485 Interference rejection because of differential signal. | 68 |

INDEX OF ILLUSTRATIONS

| | |
|---|----|
| Illustration 1. Weather Station, using a data logger | 3 |
| Illustration 2. DTH11 Module Sensor – Sensor [17] | 9 |
| Illustration 3. . DS18B20 aspect. [18] | 10 |
| Illustration 4. Parasite and normal mode of DS18B20 sensor. [1] | 11 |
| Illustration 5. HD-38 sensor [20] | 12 |
| Illustration 6. Arduino IDE message area. | 16 |
| Illustration 7. Arduino IDE Buttons | 16 |
| Illustration 8. Baud rate agreement | 16 |
| Illustration 9. Micro SD card Adapter [19] and micro SD Card | 18 |
| Illustration 10. SIM808 aspect and connections [4] and 12 Volts Charger [21] | 20 |
| Illustration 11. Try AT commands serial output | 21 |
| Illustration 12. Arduino-Sensors connection | 25 |
| Illustration 13. Arduino Mega - MicroSD Adapter wiring | 27 |
| Illustration 14. Welding J19,J12,J13 and result. | 28 |
| Illustration 15. SIM808 wiring with Arduino Nano | 28 |
| Illustration 16. Arduino Serial Port Communication: DS18B20 record | 35 |
| Illustration 17. Arduino Serial Port Communication: Sensors record | 35 |
| Illustration 18. Arduino program with the libraries that will be used and its result in Arduino Nano's memory | 37 |
| Illustration 19. Arduino Mega's memory utilization when all the libraries are loaded | 37 |
| Illustration 20. Before and after writing on the EEPROM memory | 38 |
| Illustration 21. Arduino Serial Port Communication: Clock setting | 39 |
| Illustration 22. Arduino Serial Port Communication: Frequency Check | 40 |
| Illustration 23. The process of writing a file in the SD card. | 41 |
| Illustration 24. The process of writing a file in the SD card. | 41 |
| Illustration 25. Arduino Serial Port Communication: Sending an SMS and SMS received in the phone | 42 |
| Illustration 26. Arduino Serial Port Communication for receiving a message and the SMS sent in the phone | 43 |
| Illustration 27. System prepared for demo | 45 |
| Illustration 28. Veinasa Soil Moisture Sensor [13] | 59 |
| Illustration 29. Soil Temperature Sensor [16] | 60 |
| Illustration 30. Air pressure, temperature and humidity Sensor [14] | 61 |
| Illustration 31. Wind Speed Sensor [15] | 63 |
| Illustration 32. A real datalogger with keypad [22] | 65 |
| Illustration 33. Example of a PCB implementation in Eagle. | 66 |
| Illustration 34. Sampling data line in an asynchronous serial communication [23] | 67 |

| | |
|--------------------|---|
| OTP Memory | One Time Programmable Memory |
| PCB | Printed Circuit Board |
| Arduino IDE | Arduino Integrated Development Environment |
| UART | Universal Asynchronous Receiver-Transmitter |
| PWM | Pulse Width Modulation |
| SRAM | Static Random Access Memory |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| SPI | Serial Peripheral Interface |
| MISO | Master In Slave Out |
| MOSI | Master Out Slave In |
| SCLK | Serial Clock |
| CS | Chip Select |
| GPS | Global Positioning System |
| GPRS | General Packet Radio Service |
| SMT | Surface Mount Technology |
| LSB | Least Significant Bit |
| CRC | Cyclic Redundancy Check-Control |
| OSI | Open System Interaction |
| ADU | Application Data Unit |
| PDU | Protocol Data Unit |

1 INTRODUCTION

“Life can only be understood backwards; but it must be lived forwards.”

- Søren Kierkegaard-

Nowadays, everything is changing so fast, people tend to use technology to help themselves and improve their way of working. A challenging area on which to apply this progress is agriculture, data loggers have been invented for this scope. A data logger is an electronic device which registers data on time. By using its own or externally connected sensors, information about the state of some parameters is gathered. This gadget is generally based on microcontrollers and frequently equipped with a microprocessor along with internal memory to store the information. Some of them, communicate with a personal computer using specific software, while others have a local interface (keyboard, LCD screen) so they can be used as an independent dispositive.

The goal of using data loggers is to be able to supervise some nature parameters such as wind speed, humidity and temperature; thus, making decisions regarding the care of the crops. One of the benefits of using them is having data 24 hours per day on the state of the field. Besides, once activated, dataloggers are left without surveillance providing an autonomous way of checking the state. This device varies among those with general purposes for a wide range of applications, and the specific ones, to measure a designed environment or application. It is common for the first to be programmable; nonetheless, lots of them continue being static machines with a limited number of parameters. This study aims to design a data logger from the ground up with improvements pursuing a user-friendly solution where some limitations are fixed.

The present project is organized following a path for creating a new device; starting in the first chapter by designing a black boxes diagram explained in detail and completed throughout the following chapters. Once the diagram is understood, choices made for implementing each part are justified based on the requirements and possibilities. When the design is completed, the tests carried out are explained, after, a final experiment is performed, and the compliance of requirements is checked. The final chapter describes future improvements either in the system created or in a new system with more professional attributes.



Illustration 1. Weather Station, using a data logger

1.1 Requirements

The initial requirements were stated as:

- For the device to have better connectivity, offering the possibility of connecting several sensors without having as many limitations as currently available devices.
- Result in a user-friendly device.

Nevertheless, in order to create a data logger, different aspects must be taken into account. When the problem was analysed in-depth, some questions arose, and by giving them answers, several requirements were fixed:

- Where is supposed to be placed this device? In the champ, this aspect must be considered to create the structure case but also it is essential for the sensors that will be selected for the prototype because they must stand these conditions.
- How to communicate with the datalogger? Since it is going to be placed in the champ, in theory, no WiFi connection is available. Therefore, another method for the connection must be found.
- The aim is to leave the device without surveillance; hence the datalogger must work autonomously.
- The information collected by the sensor must be reliable, avoiding interferences.
- Sensors will be placed near the device, so there is no need to have a large cable, this affects the communication and interferences that may arise.
- The amount of information collected must be defined in order to manage data storage, memory, etc.

1.2 Objectives

This project aims to develop a datalogger to retrieve data from the environment of the corps and take actions accordingly for their best care. The development of the aim is structured in this path:

- Synthesize a real datalogger.
- Decide how to implement each of the parts identified in the first analysis.
- Investigate each component and get familiar with the programming and connections.
- Integrate the whole system.
- Present the final solution throughout a real experiment.
- Analyse the results and seek for improvements

2 SYSTEM

To have a better idea of what must be recreated; it is crucial to start with the analysis of a real datalogger. Thus, in the first step, a data logger was carefully studied. The analysis, based in a breakdown, resulted in some gathered modules which could be replicated afterwards. Those modules can be depicted in a simple schema like the one shown in Figure 1. From now on, references to this diagram will be made, for it the explanation to be easier to follow. It is observed that the datalogger can be considered as a system which will manage some information coming from the sensors. The information will be processed in a certain way, and then it will be sent through a communication channel. The user will be able to obtain or to ask the information by interacting with the User Interface. It is important to note that the Communication Interface supports a two-way flow of information. This description helps to understand, in general terms, how the system works. In the present section, specific details of the above-referenced modules will be given.

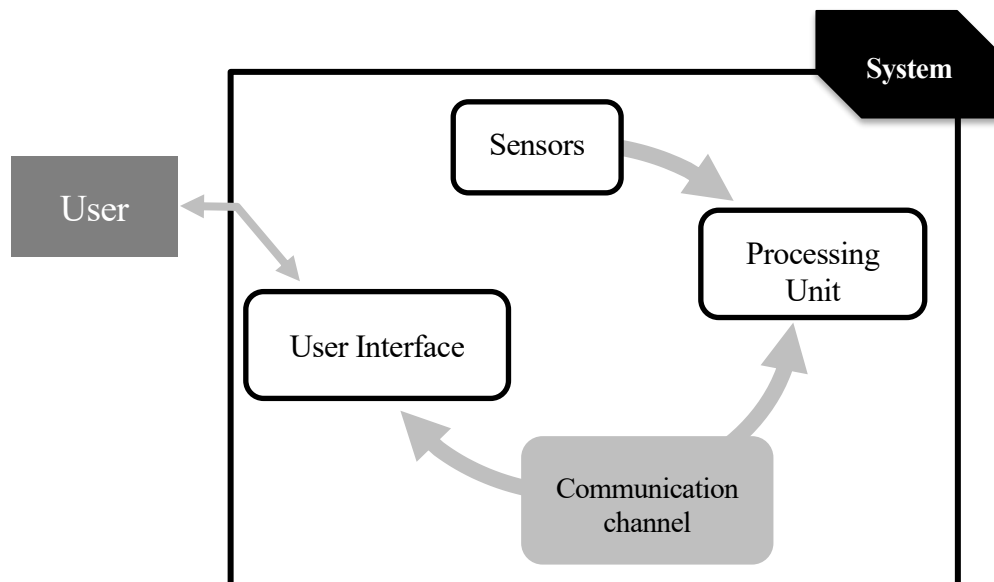


Figure 1. System description.

2.1 Sensors

In the first place, the way of collecting the information, and which information to monitor, must be decided. Figure 1 shows that the first Module is the Sensor one; for which some aspects have to be considered. Even if the parameters to monitor may change over time, for this project, some properties to observe were selected. Therefore, to understand what is happening with the crops under surveillance, soil temperature and moisture, humidity, ambient temperature, were supervised. It is important to note that Sensors consist of Hardware and Software, as appreciated in Figure 2; therefore, both aspects need to be studied.



Figure 2. Sensors description.

2.2 Processing Unit

The core of our system is found in the Processing Unit. This unit's functions vary from processing the information, collecting it, storing it and/or transmitting it. Additionally, other applications could be joined. Nevertheless, the functions listed on the preceding were selected to be those that our system will include. Another significant aspect of the Processing Unit is that the activities should be synchronized. For this reason, this unit must be provided with a clock signal or timing control. Bearing in mind that saving information implies providing this Module with an amount of memory; it is vital to define the frequency of data collection or the volume of data to be stored, in order to know how much memory would be necessary. Furthermore, errors must be prevented from happening. In the event of an error, the data should be stored in such a way that recovering it is possible.

Besides, some method must be thought in order to understand which sensors are connected. The decisions made about the Sensor's Module will also have its influence here. The reason is that this piece should be able to receive the data from the sensors, which will use a certain protocol, and then the Processing Unit will transmit it again to another Module. Beyond, even if the information flows only in one way, the Processing Unit must be able to receive information in both senses taking into account user's requests.

Figure 3 shows a summary of the characteristics described, in addition to the relationship of this Module to the preceding and subsequent elements.

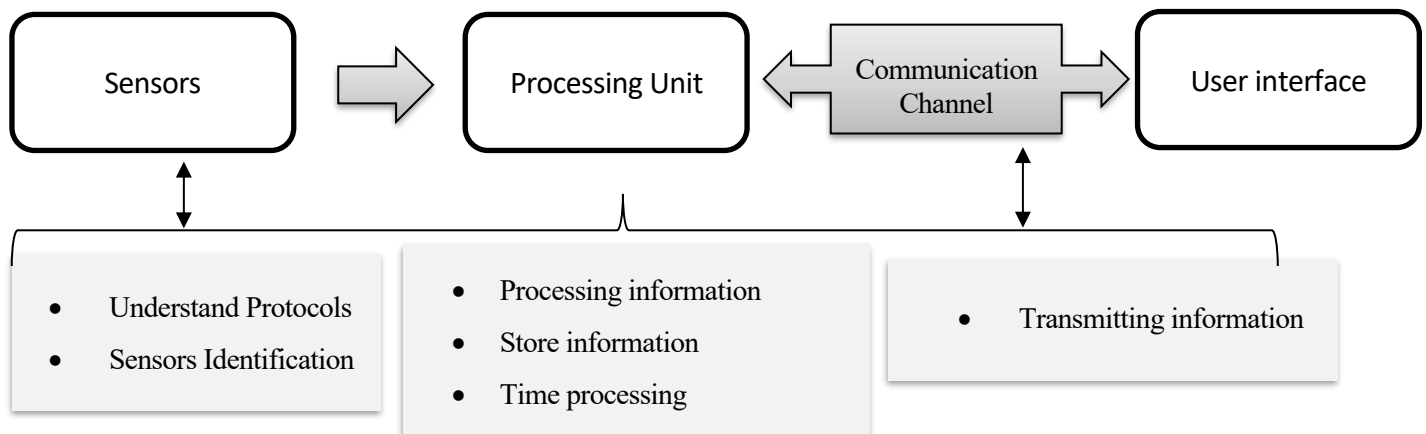


Figure 3. Processing Unit description

2.3 Communication Channel

As seen in Figure 3, the nature of such a device is bidirectional as far as communication is concerned; enabling both the Processing Unit and the final User to communicate. This characteristic implies that the Communication Channel will be the bridge; the requests for the information will first pass through this Module, but also the information will arrive at the client by the same mean. It is compulsory to create a "code" so that both ends can understand each other. Moreover, the conditions on which the communication will take place needs to be evaluated so that interferences, the fact that distance is not known in advance or the countryside conditions do not affect the performance.

2.4 User interface

The User Interface will present the information to the user whenever he or she asks for it. Requests for information from the user will also be made through this Module. In conclusion, both parties will dump the information on this platform.

2.5 User

The user stands for the person who will interact with the device. Consequently, he or she will be able to choose some of the parameters that will be decided as susceptible to change. For instance, in what refers to data frequency different approaches could be taken: To have the device working the whole day and then, certain times a day, transmitting the information; or communicating with the client only when a request is made, among others. Provided that versatility is one of the requirements, the best choice would be to include both options. In general, some requirements need to be defined in order to identify which will be the degree of freedom of the user in defining configuration parameters.

Once each Module has been described, little pieces of information can be added to the previous diagram, resulting in. Figure 5 The following section will review the modules trying to find a suitable implementation for each one.

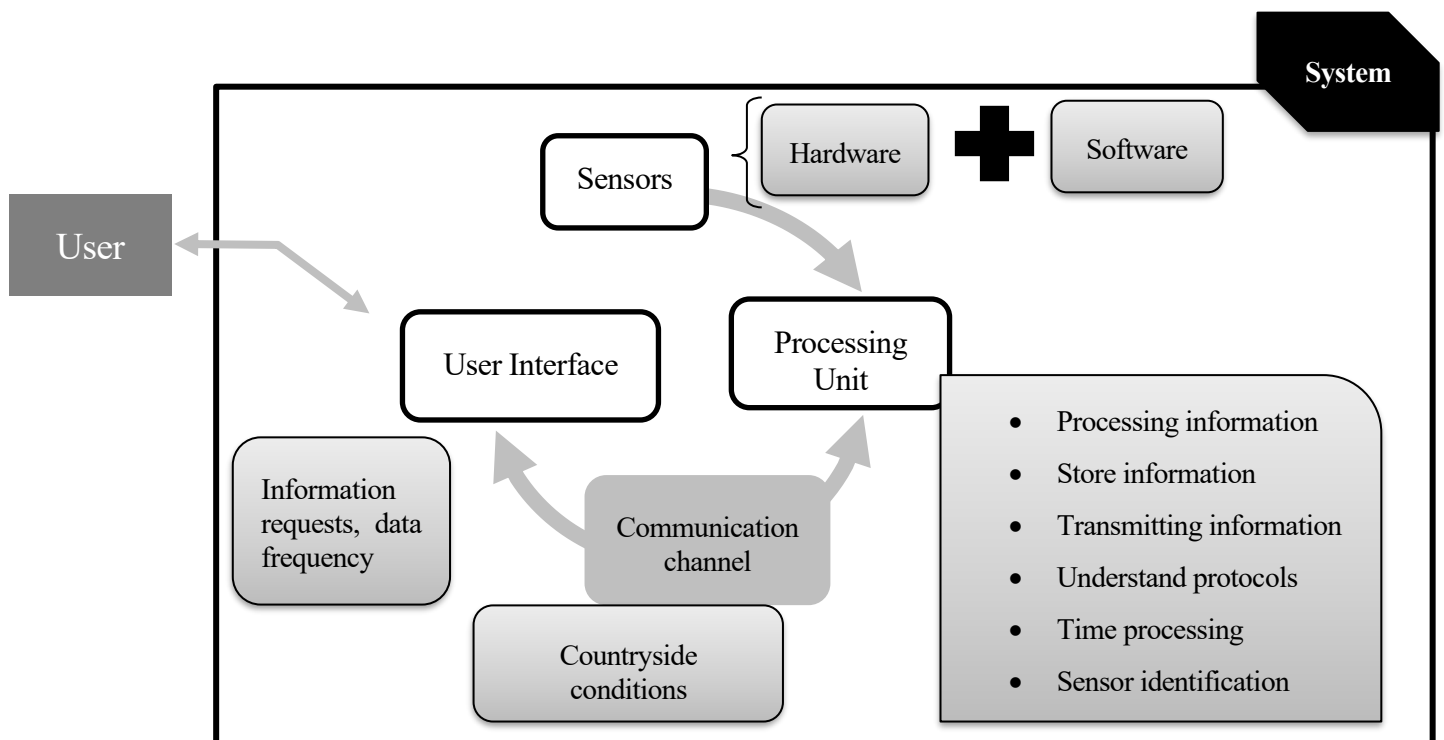


Figure 5. System Characteristics

3 IMPLEMENTED SOLUTION

The main characteristics of the system have been defined in broad terms. Therefore, right now, it is proceeded by giving details about how each Module is implemented. First of all, the main features of the machine are presented, then the selection of each device that will perform the tasks, lastly the connection between them is described.

3.1 Requirements

Talking about performance, and bearing in mind how a real data logger works, the **functionalities/features** of the system are thought to be the following:

- Ask the user which sensors are connected.
- Ask the user to define a frequency for receiving information.
- Get back the data at a particular time already passed.
- Retrieve data at the current time.
- Obtain all the data.
- Alert the user when a parameter goes out of a defined range.

Besides, to get in shape the system, it is compulsory to define specific **requirements** in such a way that the device can be implemented according to predefined guidelines:

1. Having as maximum five sensors connected: For a prototype, it seems like a great option to focus on not having so much information but to obtain a functional performance. For this reason, the number of sensors is limited to five.
2. Define an internal frequency of measurement: Even if the user receives information the whole time or just in certain moments, as mentioned above, it is important to define a frequency for regular measurement. This parameter is thought to be “factory default”, in such a way that the user can not change it.
3. Define how each Module will communicate with each other: For instance, it is essential to understand how the sensors will transmit the information, by defining the protocol to be used, so that the Processing Unit will understand it.
4. Define a bidirectional communication with the user: So, not only information can be sent to the user, but also the user can make requests for information.
5. Provide versatility in the sensor’s implementation/connection
6. Provide memory to the system: Provided that accessing to past data is desired, it is relevant to have an amount of memory available.

In this project, these requirements will be tried to achieve. In the end, an assessment will be made to understand the degree of achievement of the objectives.

3.2 Justification and choices for each piece

An overview of different possibilities was made on each Module to build-up our solution. The process of choosing each device and the justification for it is presented below. At the end of each section, the conclusion for the selection is given. The information here will be presented in the same order as in the previous section.

3.2.1 Sensors

Firstly, to understand what the state of the crops is, the proper parameters to observe must be chosen. The choice of them will lead us to the conclusion of what needs to be done to maintain the good conditions of the crops. It seems a great decision to focus on the soil moisture, humidity, and both ambient and soil temperature.

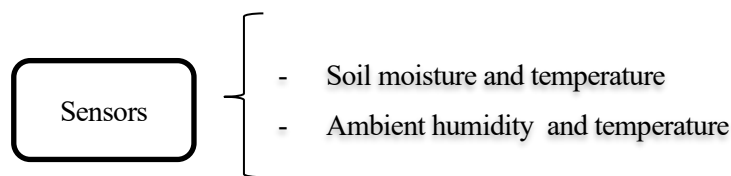


Figure 6. Sensor's parameters to observe

After selecting the parameters, specific sensors are chosen for each; the details are found in the following.

3.2.1.1 DTH11

DTH11 sensor allows measuring Temperature and Relative Humidity. This sensor is equipped with two resistive sensors, a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. In addition, the sensor is factory calibrated and hence easy to interface with other microcontrollers. It is important to highlight that the calibration coefficients are stored as programmes in the One Time Programmable (OTP) memory. These parameters are used by the sensor's internal signal detecting process. Not only the precision is high, because the integrated microcontroller on the sensor guarantees accuracy over time, but also the measure response is fast. Besides, it has a small size, low energy consumption as well as excellent transmission up to 20 meters of distance.

As far as communication is concerned, single-bus data is used (1-wire protocol), the serial interface protocol makes system integration quick and easy. DHT11 sensors use their two-way communication system via a single driver, using timed signals. In each measurement transmission, the sensor sends 40 bits, in 4ms. These 40 bits correspond to 2 bytes for moisture measurement, 2 bytes for temperature measurement, plus a final byte for error checking (8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit checksum).

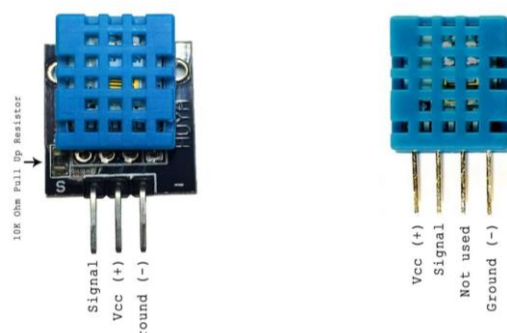


Illustration 2. DTH11 Module Sensor – Sensor [17]

| | | |
|---------------------------|-----------------------------|------------|
| Power supply | 3Vdc \leq Vcc \leq 5Vdc | |
| Output Signal | Digital | |
| Temperature Measure Range | From 0 to 50 °C | |
| | Accuracy | ± 2 °C |
| | Resolution | 0.1°C |
| Humidity Measure Range | from 20% to 90% RH | |
| | Accuracy | 4% RH |
| | Resolution | 1%RH |
| Time response | 1s | |
| Size | 12 x 15.5 x 5.5mm | |

Table 1. DHT11 Sensor characteristics

3.2.1.1.1 Sensor Module.

DHT11 sensor can be purchased either separately or integrated within a module; still, the performance of the sensor is the same. The Module will come with three pins, whereas the sensor will come as a 4-pin package out of which only three pins will be used as it is observed in Illustration 2. The only difference between the sensor and Module is that the Module will have a filtering capacitor and pull-up resistor inbuilt, and for the sensor, they have to be used externally if required.

3.2.1.1.2 Software

It is possible to program this sensor by using previously created libraries. For example, the Adafruit “*DHT.h*” library, which is available for download on the internet. Working with a library makes it easier to obtain the measurement data from the sensor, avoiding the previously described work with data bytes.

The detailed information above leads us to the conclusion that this sensor is perfectly suitable for the desired application. Therefore, it will be the one used to measure the temperature and humidity in our prototype.

3.2.1.2 DS18B20

The DS18B20 [1] is a digital thermometer that provides 9-bit to 12-bit Celsius temperature measurements. This sensor has an alarm function with non-volatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that, by definition, requires only one data line (and ground) for communication with a central microprocessor. Besides, multiple sensors can be connected to the same data bus; each sensor identifies itself by a unique serial number. Thus, one microprocessor can easily control many DS18B20s distributed over a large area. In Table 2, electrical characteristics are found.

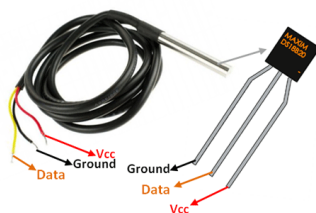


Illustration 3. DS18B20 aspect. [18]

This sensor can operate in **normal** or **parasite mode**. In normal mode, a 3-wire connection is needed while in parasite mode the sensor derives its power from the data line in such manner that only two wires, data and ground, are required. The advantage of using parasite mode is to eliminate the need for an external power supply.



Illustration 4. Parasite and normal mode of DS18B20 sensor. [1]

3.2.1.2.1 The normal mode of operation

In normal mode, each sensor is connected to a power line and ground (pins 3 and 1 seen in Illustration 4), and the DQ pin connects to a data line. The data output (pin 2) is a 3-state or open-drain port and requires a *4k7 pull-up resistor*, nevertheless, up to 20m cable length has been successfully reported with a lower pull-up resistor value of 2K. Normal mode is recommended when many devices and/or long cable runs are involved.

| | |
|-------------------------|---|
| Temperature range | from -55°C to $+125^{\circ}\text{C}$ (-67°F to $+257^{\circ}\text{F}$) |
| Accuracy | $\pm 0.5^{\circ}\text{C}$ in the range of -10°C to $+85^{\circ}\text{C}$, $\pm 2^{\circ}\text{C}$ in the rest. [2] |
| Programmable Resolution | From 9 Bits to 12 Bits |

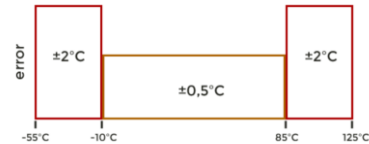


Table 2 Sensor DS18B20 characteristics

3.2.1.2.2 Software.

In this case, two libraries should be used for programming DS18B20 sensors: *“DallasTemperature.h”* and *“OneWire.h”* protocol communication library. The use of libraries allows us to work with the sensor straightforward.

This sensor suits for obtaining the soil temperature, withstanding the conditions to which it will be exposed. Besides, it provides the accuracy wanted for this application.

3.2.1.3 HD – 38

A soil moisture sensor can read the amount of moisture present in the surrounding soil. Particularly, HD-38 sensor uses the two probes to pass current through the soil; then it reads the resistance to get the moisture level. For moisture calculation, the following is considered: More water makes it easier for the soil to conduct electricity (less resistance), while dry soil conducts electricity poorly (more resistance). HD-38 sensor has low power consumption and high sensitivity. In addition, a built-in potentiometer for sensitivity adjustment of the digital output (D0), a power LED and a digital output LED is provided.



Illustration 5. HD-38 sensor [20]

In the case of this sensor, shielding is critical, therefore it uses Immersion Gold which protects the nickel from oxidation. Electroless Nickel Immersion Gold (ENIG) has several advantages over more conventional (and cheaper) surface platings such as hot air solder levelling, including excellent surface planarity (particularly helpful for PCB's with large BGA packages), good oxidation resistance, and usability for untreated contact surfaces such as membrane switches and contact points. The following table gives some information about electrical characteristics of the sensor.

| | |
|---------------------|------------------|
| Voltage Range | 3.3-12VDC |
| Working Current | <20mA |
| Output Current | <30mA |
| Working Temperature | -25 ~ 85 Celsius |
| Output Interface | DO: digital |
| | AO: Analog |

Table 3 HD-38 Sensor's characteristics

3.2.1.3.1 Software

The output of this sensor is an analogical value. There is no need for a library to control the sensor and retrieving its value.

After gathering all this information, the sensor is selected for having soil moisture data. It has been considered both the price and the characteristics, but also, as seen in the description, it is well prepared for performing in the system's conditions.

3.2.1.4 Sensors conclusion

The choices described above allow us to complete the diagram of the sensors outlined in the previous section, Figure 2. Thus, the resulting diagram is the one shown in Figure 8. The parameters to monitor have been chosen and, taking them into account, the sensors as well. The protocol is defined by the characteristics of the sensors.

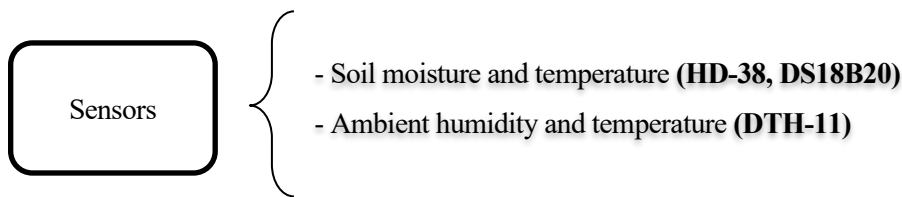


Figure 8. Sensors final Choice

On the other hand, the diagram of the system as a whole can be updated with the choices made, leaving the system as follows in Figure 7.

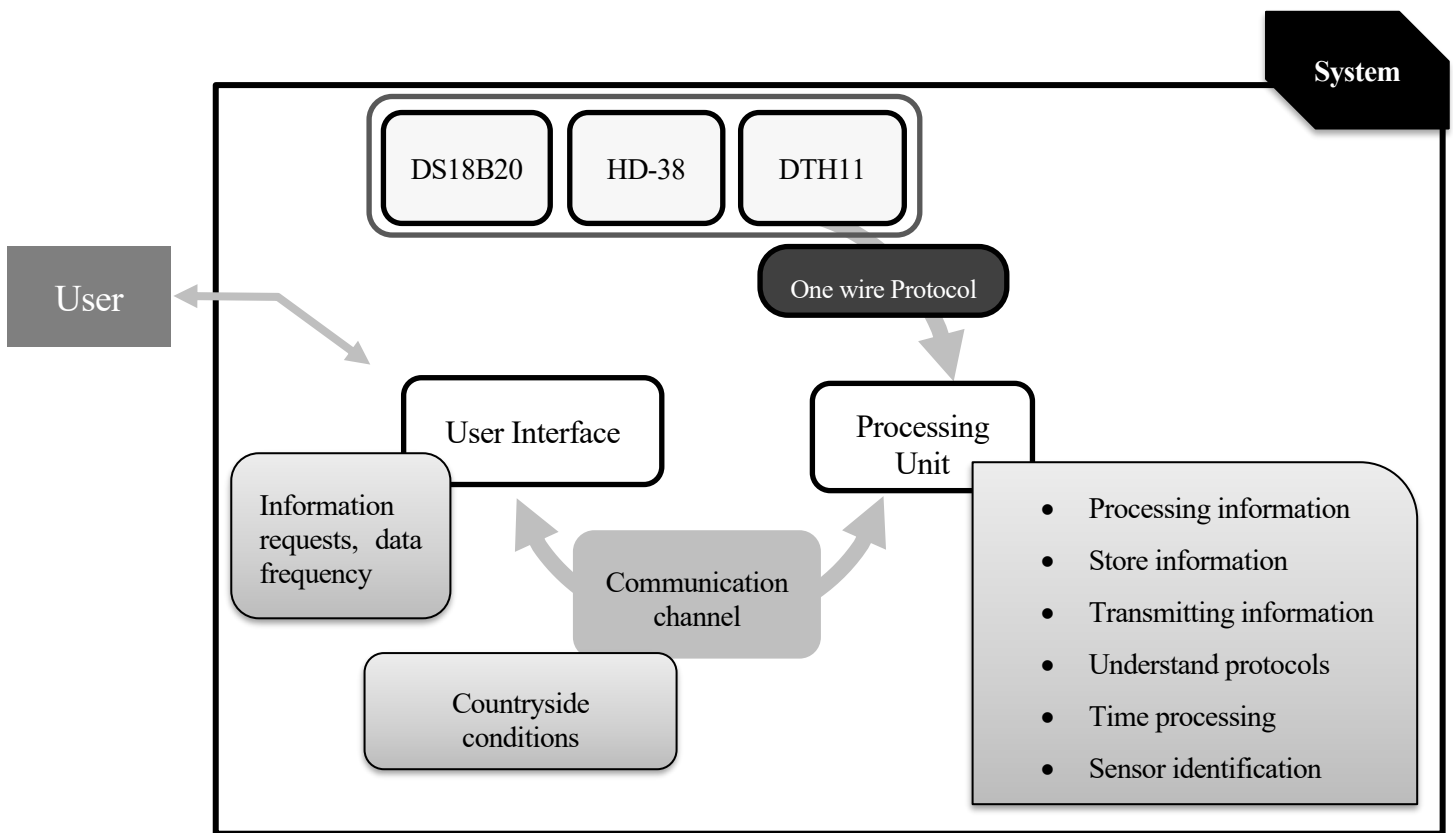


Figure 7. Sensors-System

3.2.2 Processing Unit

Considering our scope with the project, but also the must-have characteristics of the processing unit, using a microcontroller was thought to be the best option to implement this Module. A microcontroller contains one or more processor cores (CPUs), along with memory and programmable input/output peripherals. The flexibility we want to achieve in the system as a whole has its major influence on this Module. This kind of project involves different tests to be undertaken during the development of the system. For this reason, a versatile solution that would allow for changes in development needed to be chosen.

3.2.2.1 Arduino

Arduino is an open-source hardware and software company, project, and user community. This enterprise designs and manufactures single-board microcontrollers as well as microcontroller kits for building digital devices. The platform that this company holds is based on easy-to-use hardware and software.

Arduino boards can read inputs and turn them into outputs. The board operates according to a set of instructions sent to the microcontroller on the board. To set these instructions, Arduino programming language (based on Wiring), and the Arduino Software (IDE) (based on Processing), are used. Among all the features that can be found on the manufacturer's website [3], the most interesting for this project are:

- Open source and extensible software, due to the fact that The Arduino software is published as an open-source tool, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can switch from Arduino to the AVR C programming language on which it is based.
- Open source and extensible hardware, because the plans of the Arduino boards are published under a Creative Commons license; therefore, expert circuit designers can create their version of the Module, extending it and improving it.

Considering, on the one hand, this information about this type of microcontroller; and, on the other hand, the fact of having worked with them previously throughout the university degree; Arduino was chosen for implementing the processing unit.

3.2.2.1.1 Arduino boards

Different Arduino boards are available; an assessment should be made to have an idea of which one will be suitable for the present work. There are different kinds of Arduino boards. In order to find the one that best suits our system, some of the alternatives are analysed below:

- Arduino Uno: It consists of 14-digital I/O pins, where 6-pins can be used as Pulse Width Modulation outputs (PWM), 6-analog inputs, a reset button, a power jack, a USB connection and more. When purchasing Arduino One, everything for the microcontroller to work is integrated into the board, so it is only needed to plug it into the computer and upload the code. In more advanced applications, which may require extra supply, AC-to-DC adapter or battery are used.
- Arduino Mega: The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analogue inputs. Besides, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an In Chip Serial Programmer (ICSP) header, and a reset button. It contains everything needed to support the microcontroller. It also can connect to the computer by a USB cable and using it directly or add an AC-to-DC adapter or battery.
- Arduino Nano: The Arduino Nano is a small and complete board based on the ATmega328 (Arduino Nano 3.x). It has 22 digital input/output pins out of which six can be used as PWM outputs, eight as analogue inputs. It works with a Mini-B USB cable instead of a standard one and lacks a DC power jack.

| | UNO | MEGA | NANO |
|------------------------------------|---|---|--|
| Architecture | AVR | AVIR | AVR |
| Operating Voltage | 5V | 5V | 5 V |
| Input Voltage (recommended) | 7-12V | 7-12V | 7-12 V |
| Input Voltage (limit) | 6-20V | 6-20V | 6-20V |
| DC Current per I/O Pin | 20 mA | 20 mA | 40 mA (I/O Pins) |
| DC Current for 3.3V Pin | 50 mA | 50 mA | - |
| Power consumption | 20mA | 50mA-200mA | 19 mA |
| Flash Memory | 32 KB (ATmega328P) of which 0.5 KB used by bootloader | 256 KB of which 8 KB used by bootloader | 32 KB of which 2 KB used by bootloader |
| SRAM | 2KB (ATmega328P) | 8 KB | 2 KB |
| EEPROM | 1 KB (ATmega328P) | 4 KB | 1 KB |
| Clock Speed | 16 MHz | 16 MHz | 16 MHz |
| Length | 68.6 mm | 101.52 mm | 18 x 45 mm |
| Width | 53.4 mm | 53.3 mm | |
| Weight | 25 g | 37 g | 7 g |

Table 4 Arduino Uno, Mega, Nano comparison

3.2.2.1.2 How to programme

In order to use the Arduino boards, it is necessary to program code and then upload it. To be able to do this, there is a tool that the Arduino platform itself provides to the user, the Arduino Integrated Development Environment (IDE). This program is a cross-platform application, written in functions from C and C++. It supplies a software library from the Wiring project, which includes many standard input and output procedures. Programs written using Arduino Software IDE are called **sketches**, which are saved with the file extension *.ino*.

Looking now at the aspect of this tool, the message area, shown in Illustration 6, is found at the bottom of the window and gives feedback while saving and exporting codes, but it displays errors as well. The console, with a black background, displays text output by the Arduino Software IDE, including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port.

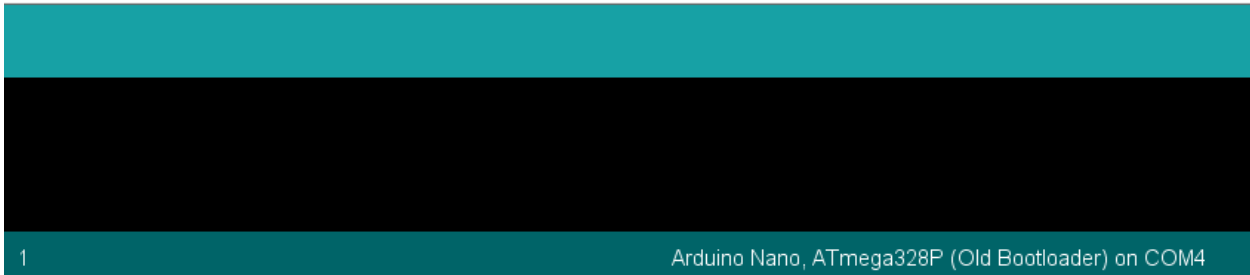


Illustration 6. Arduino IDE message area.

The serial port is used for communication between the Arduino board and a computer. All Arduino boards have at least one serial port (also known as a UART or USART). On Uno, Nano, Mini, and Mega, pins 0 and 1 are used for communication with the computer. Connecting anything to these pins can interfere with serial communication, including causing failed uploads to the board. The Arduino environment's built-in serial monitor can be used to communicate with an Arduino board. It can be opened by clicking on the button in the right-hand corner at the top of the window (See Illustration 7). In order to use the serial communication, the same baud rate as the one used in the call to `begin()` in the code must be selected.



Illustration 7. Arduino IDE Buttons

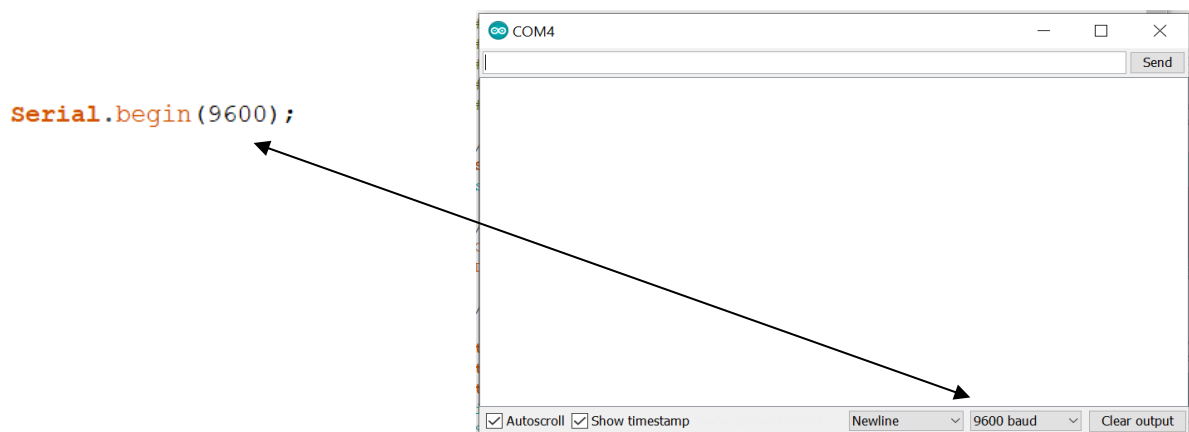


Illustration 8. Baud rate agreement

Moreover, it is worth mentioning the use of libraries for programming in Arduino. Libraries provide sketches with extra functionality, for example, working with external hardware or manipulating data. For using a library in a sketch, it must be selected from the menu on the upper part: Sketch > Import Library menu. The libraries are inserted one behind the other by `#include` statements at the top of the sketch. The library is compiled with the sketch. Because libraries are uploaded to the board with the sketch, they increase the amount of space it takes up; therefore, special attention must be paid not to overload the board with unnecessary information.

3.2.2.2 Board Usage

As seen in Table 4, characteristics overall are similar; it seems an excellent choice to choose Arduino Mega because of the abundance of memory and pins. However, on the other hand, the Arduino Nano offers similar features and tiny size. The risk of choosing Arduino Nano is the possible lack of memory for performing all the features wanted in the project, seeing that the quantity of memory available in this model is significantly reduced. For the moment, we are considering both options and details of the final choice will be given later. The decisions regarding the implementation of the following modules are taken based on the choice of Arduino. A considerable benefit of Arduino, from which this project takes advantage, is the wide variety of easy-to-connect extra hardware that is found on the market. The following Figure gives an idea of how the system looks like as decisions are taken.

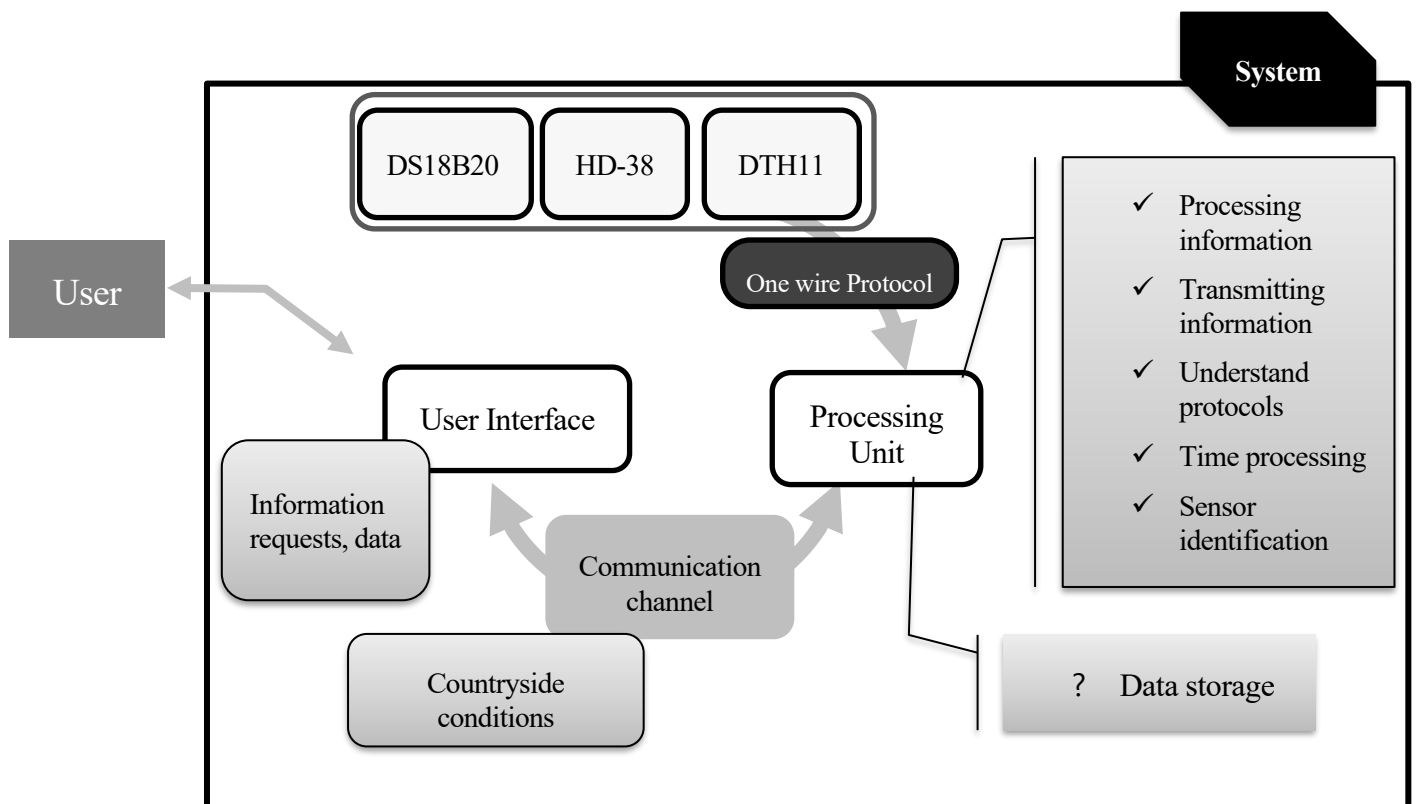


Figure 9. System details with Arduino functionalities

Although the Arduino platform satisfies the general aspects considered for the Processing Unit, one significant point is missing. This Module must consider the possibility of saving data acquired by the sensors, allowing the user to retrieve them in the future. Therefore, taking advantage of the hardware extensiveness offered by Arduino, an extra module to perform this task should be considered.

3.2.2.3 Data storage

Arduino boards have three different kinds of memory with different functions each: Static Random Access Memory (SRAM); a volatile memory, where the sketch creates and manipulates the variables when executed, it is a limited resource and must be monitored to avoid depletion, EEPROM; a non-volatile memory to keep data after a reset, Flash, which is the program memory, where the sketch is saved once it has been compiled, it also stores the bootloader. Although all of them are necessary and allow the Arduino to perform its functions, external memory is required to perform the sensor data storage function. By proceeding this way, the benefit accomplished is that in the event of an Arduino failure data is separately stored so it could be recovered.

A popular and easily obtainable option is the usage of an external SD memory. In order to insert an SD memory on the system, an SD adapter compatible with Arduino must be included. There is a wide range of these on the market; a micro SD was already available; therefore, a micro SD adapter was the model chosen. The model selected is shown in Illustration 9, and also the electrical characteristics of the module are found in Table 5.

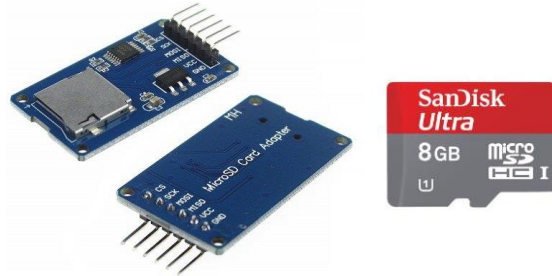


Illustration 9. Micro SD card Adapter [19] and micro SD Card

| | Min | Typical | Max |
|------------------------------------|-------------------------------|---------|-----|
| Power Voltage VCC (V) | 4.5 | 5 | 5.5 |
| Current (mA) | 0.2 | 80 | 200 |
| Interface Electrical Potential (V) | 3.3 or 5 | | |
| Support Card Type | Micro SD Card/Micro SDHC Card | | |
| Size (mm) | 41 x 24 x 12 | | |
| Wight (g) | 5 | | |

Table 5. Micro SD adapter Characteristics

3.2.2.3.1 Software

As was the case of the sensors, for using this device, software libraries are needed, the name of them are “*SD.h*”, “*SPI.h*”. The first one allows for reading and writing to SD cards, the formats supported of the SD are FAT16 and FAT32, this last one will be used. The second library is needed because the communication between the microcontroller and the SD card uses **Serial Peripheral Interface (SPI)**, which takes specific pins of the Arduino in order to perform this communication.

The **SPI protocol** is a synchronous serial data protocol used by microcontrollers for communicating with own or more peripheral devices in a quick way over short distances. In this protocol, a master is always found, usually, a microcontroller which controls the peripheral devices. This communication uses three lines, common for the devices that will use SPI. Those three lines are Master In Slave Out (MISO), which allows the slave for sending data to the master; Master Out Slave In (MOSI), Master’s line for sending data to the peripherals, Serial Clock (SCLK) which allows data transmission synchronization, it is generated by the master; and one specific line for every device, Save Select (SS) by the pins connected to this line, the master is able to decide which device will enable or disable. When the Slave Select is at a low value, the device connected is communicating with the slave, whereas when its value is high, the peripheral device is ignoring the master.

Figure 10 is updated with the last choices made, also adding the protocols needed to communicate between the parts.

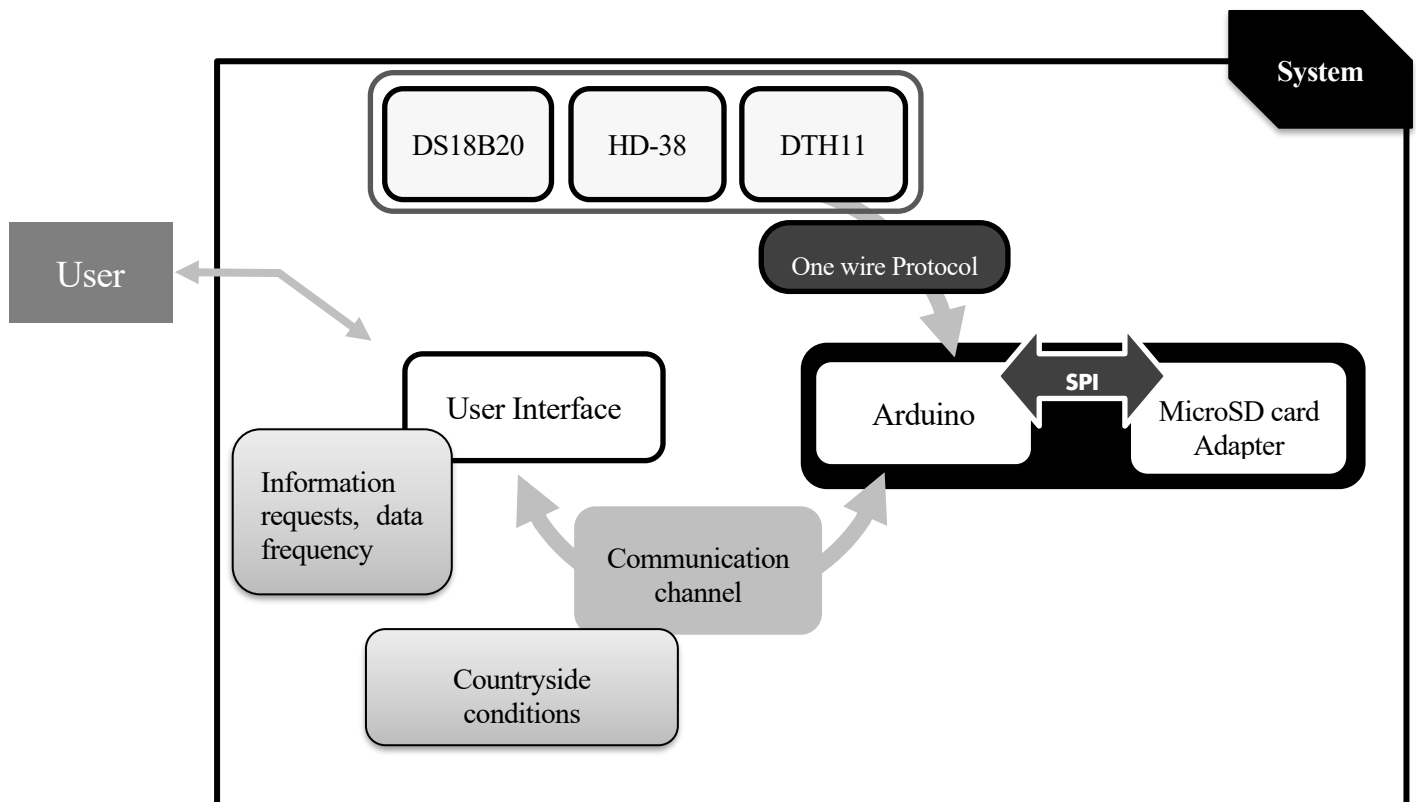


Figure 10. System details adding micro SD adapter

3.2.3 Communication channel

Different possibilities are available for providing an Arduino Board with communication skills. Some of the requirements limit the options, for instance, the fact that there is no WiFi connection where the datalogger will be placed, discards the choice of a WiFi module. Therefore, a more reasonable alternative was to consider a 3G module for communication. Having a 3G transmission provides several benefits; for example, the distance between the user and the device is not a problem. In the market, several compatible 3G Arduino shields are found, the **SIM808 evb-v3.2** model was used.

3.2.3.1 SIM808 evb-v3.2

SIM808 module is a complete Quad-Band GSM/GPRS module which provides Global Positioning System (GPS) technology for satellite navigation. This compact design integrates General Packet Radio Service (GPRS) and GPS in a Surface-Mount Technology (SMT) package. It will let users develop GPS enabled applications. Featuring an industry-standard interface and GPS function, it allows variable assets to be tracked seamlessly at any location and anytime with signal coverage [4]. Furthermore, sending and receiving SMS messages or making and receiving phone calls is also possible with this Module; those are indeed the most relevant characteristics for our project. Some features are:

| | |
|------------------------------------|--|
| Quad-band | 850/900/1800/1900MHz |
| GPRS multi-slot class | 12/10 |
| GPRS mobile station class | B |
| Compliant to GSM phase 2/2+ | Class 4 (2 W @ 850/900MHz / Class 1 (1 W @ 1800/1900MHz) |
| Bluetooth | compliant with 3.0+EDR |
| FM | 76~109MHz worldwide bands with 50KHz tuning step |
| Dimensions | 24.0*24.0*2.6mm |
| Control | via AT commands (3GPP TS 27.007, 27.005 and SIMCOM enhanced AT Commands) |
| Supply voltage range | 3.4 ~ 4.4V |
| Operation temperature: | -40°C ~85°C |

Table 6 SIM808 Characteristics

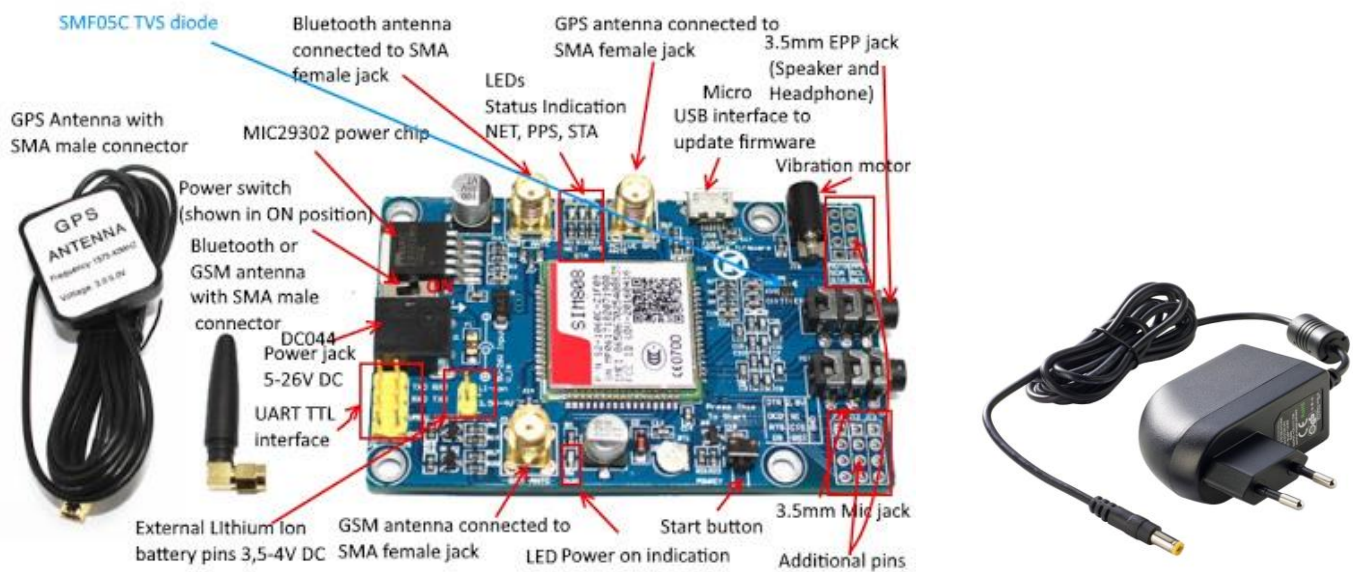


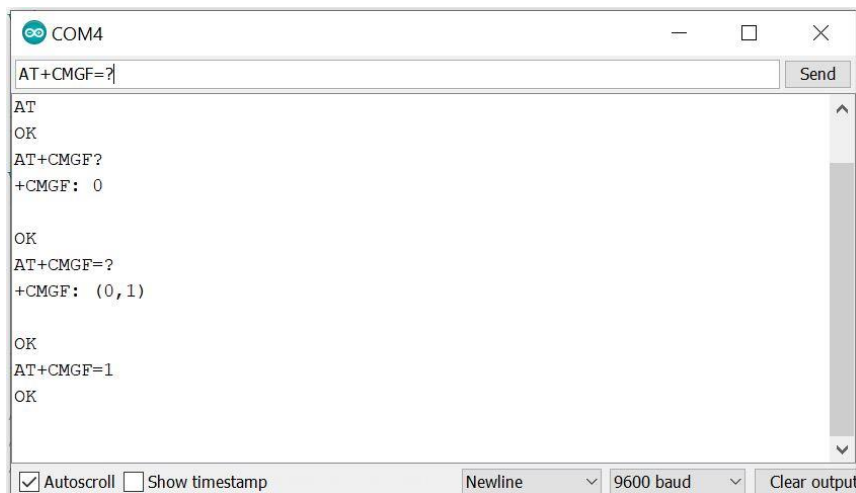
Illustration 10. SIM808 aspect and connections [4] and 12 Volts Charger [21]

Using this shield involves having an external power supply adapter. In this case, a 12Volts charger was acquired like the one seen in Illustration 10.

3.2.3.1.1 AT commands

For the communication with the SIM808 board, **AT commands** are used. The Hayes command set (also AT command set) is a specific command language originally developed for the Hayes Smartmodem 300 baud modem in 1981. The command set consists of a series of short text strings which can be combined to produce commands for operations such as dialling, hanging up, and changing the parameters of the connection. The vast majority of dial-up modems use the Hayes command set in numerous variations [5]. The AT commands are required to use the SMS communication of this shield. In the provided datasheet of SIM808 module, the syntax for programming is laid out [6].

The procedure starts with specific commands used for configuring parameters to deploy communication; then, SMS communication takes place by using other instructions. A simple code was made to analyse the responses of the shield. The performance of that program helped in order to have a better understanding of how the Module works. In this first code, it is possible to send AT commands by serial port, writing them directly by the computer keyboard. This test not only allowed us to observe the responses but also to know which are the parameters to send for a given setting command. Writing the command followed by a question mark '?' gives the current value of the command, whereas writing the command followed by '=' results in the values that the parameter accepts—this process is observed in Illustration 11



```
COM4
AT+CMGF=?
Send
AT
OK
AT+CMGF?
+CMGF: 0

OK
AT+CMGF=?
+CMGF: (0,1)

OK
AT+CMGF=1
OK
```

At the bottom of the window, there are control options: Autoscroll, Show timestamp, a dropdown menu set to 'Newline', a dropdown menu set to '9600 baud', and a 'Clear output' button.

Illustration 11. Try AT commands serial output

3.2.3.1.2 Using SIM808 for sending and receiving SMS

The specific commands for setting up the Module to receive and send SMS are found on the following table.

| AT+CMGF [6] | Select SMS Message format |
|--|---|
| AT+CNMI=<mode>[,<mt>[,<bm>[,<ds>[,<bfr>]]]] | <p>Terminal Adapter (TA) selects the procedure for how the receiving of new messages from the network is indicated to the Terminal Equipment (TE) when TE is active.</p> <p>The parameters used in our case: <i>AT+CNMI=2,2,0,0,0</i> <i><mode> = 2</i></p> <p>Buffer unsolicited result codes in the TA when TA-TE link is reserved (e.g. in on-line data mode) and flush them to the TE after reservation. Otherwise, forward them directly to the TE</p> <p><i><mt> = 2</i></p> <p>the rules for storing received SMS depend on its data coding scheme preferred memory storage (+CPMS) setting and this value</p> <p>SMS-DELIVERs (except class 2) are routed directly to the TE using unsolicited result code: +CMT: [<i><alpha></i>],<i><length></i><CR><LF><pdu> (PDU mode enabled) Class 2 messages result in indication as defined in <i><mt>=1</i></p> |

Table 7. AT commands for setting up the parameters to receive/send SMS texts

After setting all these parameters, the Module is prepared for receiving the SMS messages coming from the user. This is the path to follow for sending an SMS:

| AT+CMGS [6] | Send SMS Message |
|---|---|
| <p>Write Command</p> <p>1) If text mode (+CMGF=1): +CMGS=<da>[,<toda>]<CR> text is entered<ctrl-Z/ESC> ESC quits without sending</p> <p>2) If PDU mode (+CMGF=0): +CMGS=<length><CR>PDU is given<ctrl-Z/ESC></p> | <p>Parameters</p> <p><da> GSM 03.40 TP-Destination-Address Address-Value field in string format(string should be included in quotation marks); BCD numbers (or GSM default alphabet characters) are converted to characters of the currently selected TE character set (specified by +CSCS in TS 07.07); type of address given by <i><toda></i></p> <p><toda> GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of <i><da></i> is + (IRA 43) default is 145, otherwise default is 129)</p> <p><length> integer type value (not exceed 160 bytes) indicating in the text mode (+CMGF=1) the length of the message body</p> <p><data> (or <i><cdata></i>) in characters; or in PDU mode (+CMGF=0), the length of the actual TP data unit in octets (i.e. the RP layer SMSC address octets are not counted in the length)</p> |

Table 8. AT command for sending an SMS

A response is given from the module whenever a message is sent, the description is found on the following table.

| |
|--|
| <p>Response</p> <p>TA sends a message from a TE to the network (SMS-SUBMIT). Message reference value <mr> is returned to the TE on successful message delivery. Optionally (when +CSMS <service> value is 1 and network supports) <scts> is returned. Values can be used to identify message upon unsolicited delivery status report result code.</p> <p>1) If text mode(+CMGF=1) and sending successful: +CMGS: <mr> OK</p> <p>2) If PDU mode(+CMGF=0) and sending successful: +CMGS: <mr> OK</p> <p>3) If an error is related to ME functionality: +CMS ERROR: <err></p> |
|--|

Table 9. SMS Response description.

After selecting the device which stands for the Communication Channel, the whole system diagram can be updated, resulting as the one in Figure 11.

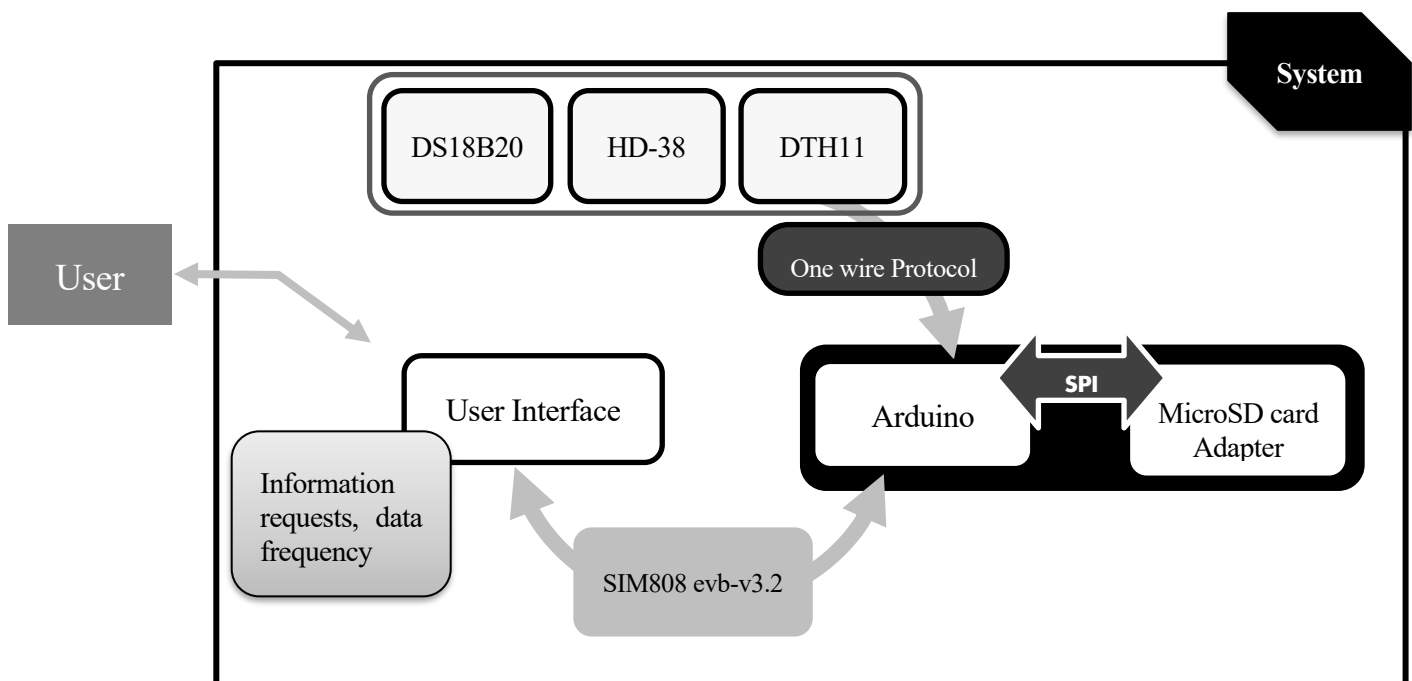


Figure 11. System Scheme Adding SIM808

3.2.4 User Interface

Provided that the communication will employ SMS, the User Interface will be the application installed in the mobile phone to read and send SMS. The phone number to which refer the communication is the one found in the SIM card installed on the SIM808-ev2.3 Module.

3.2.5 User

The user is responsible for setting up the frequency and demanding information when he or she requires it. The customer can define a frequency whereby receive an SMS with the information given by the sensors, but also is able to ask for the measurement at a particular past time as well as to obtain the current status. These functionalities are defined considering the milestones defined in the epigraph 3.1, Processing Unit oversees the coordination between all of them. The aim is to avoid disturbances between one another. For instance, if the client wants to know some past data acquired, access to the data in the SD is coordinated by the microcontroller.

The final aspect of the system is the one seen in Figure 12.

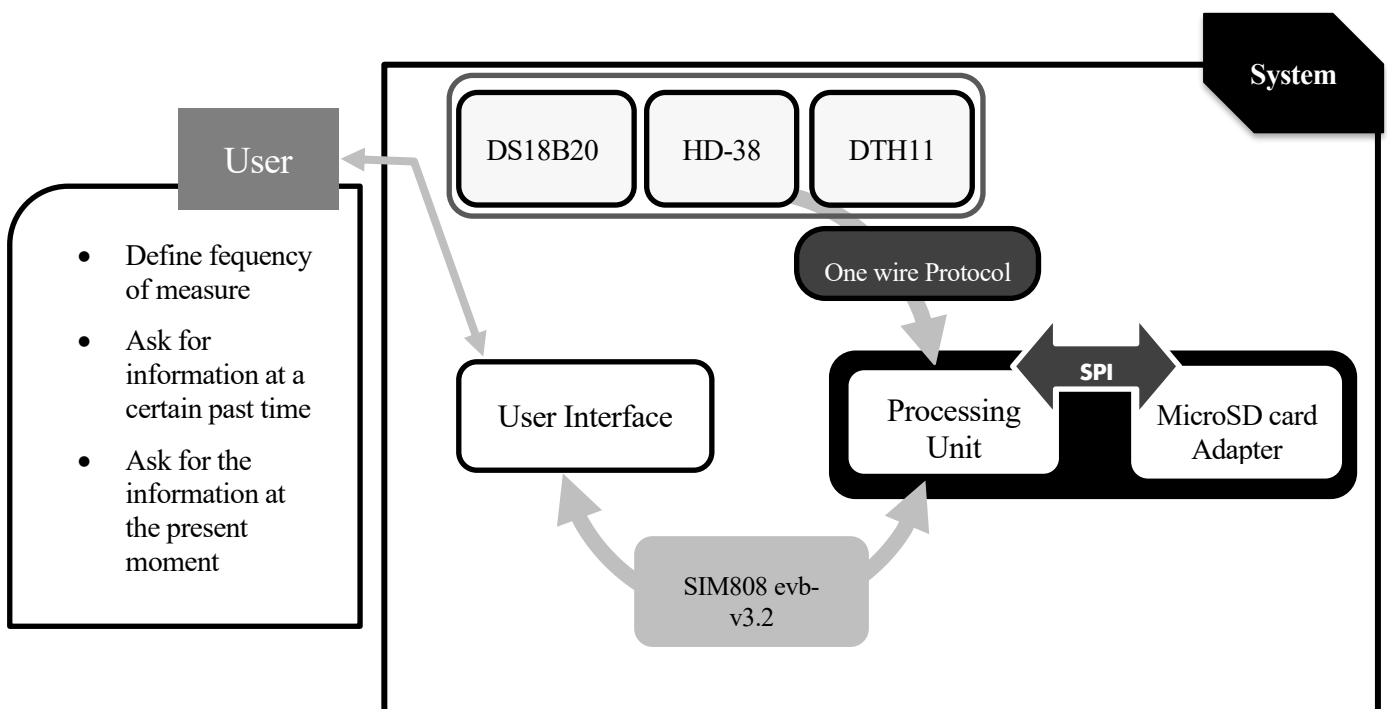


Figure 12. Complete system description after completing the choices

3.3 Implementation of the system

In the present section, the implementation, not only about hardware but also regarding software, is explained. A section is found for each component. Details about wiring are explained concretely but also the software, which focuses on the programming part. Everything is related to the core of the system, the Arduino board, that is why a first description is made on how the programming is made in the Arduino IDE.

3.3.1 Arduino board

The Arduino IDE uses special rules of code structuring. User-written code requires two essential functions, **setup()** and **loop()**, both are compiled and linked with a program stub **main()** into an executable cyclic executive program. The **setup()** function, only executes once; usually, all the configurations are made here, like setting pin modes or initializing libraries. On the other hand, the **loop()** function executes repeatedly; this is the part of the code which will concentrate the performance. Both functions must be included in the sketch, even if one of

them is not needed for a particular application. The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board. By default, avrdude is used as the uploading tool to flash the user code onto official Arduino boards.

3.3.1.1 Sensors

In order to get the measurements from the sensors, we have to program their usage into the Arduino IDE. For this purpose, we use the libraries previously mentioned in the description of each sensor. The use of the libraries is not necessary, but it simplifies the work enormously. Illustration 12 shows the connection of an Arduino nano board with the sensors, from right to left: DS18B20, DTH11, HD-38. A description of the wiring and the methods to be used for programming each one is presented below.

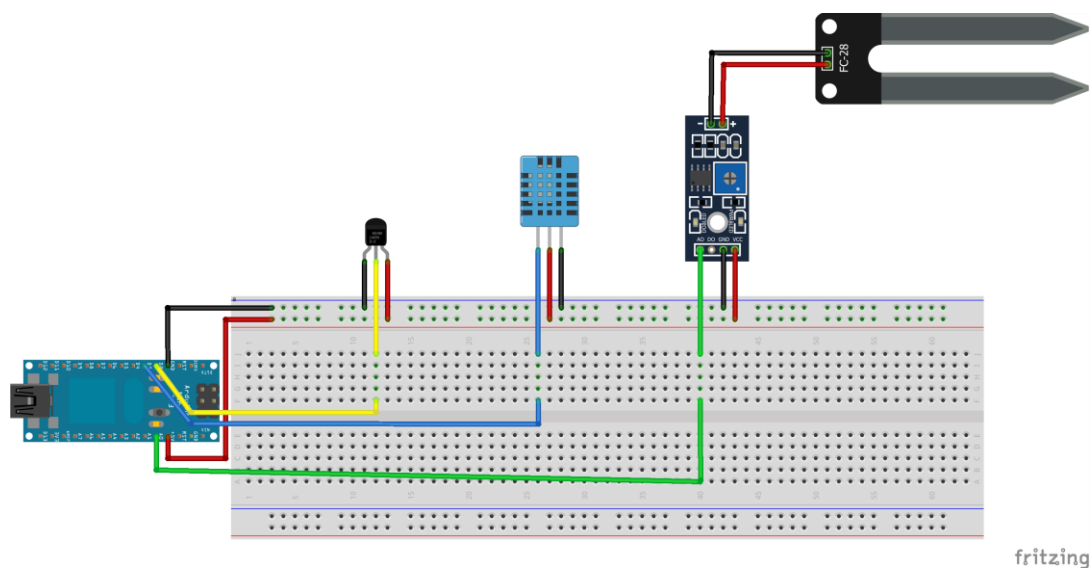


Illustration 12. Arduino-Sensors connection

3.3.1.1.1 DTH11

3.3.1.1.1.1 Wiring

As previously seen in Illustration 2, the module has three output pins, DOUT, VCC, GND. These are connected to a digital pin of the Arduino (blue cable), 5V (red cable), Ground (black cable) respectively. The advantage of using the module version of this sensor is that there is no need to connect extra resistors or capacitors.

3.3.1.1.1.2 Programming

The library “*DTH.h*” allow us to create an instance of the sensor as follows:

```
DHT dht(DHTPin, DHTTYPE);
```

In this instruction, the pin of the Arduino chosen for the sensor’s connection is identified, DHTPin. The DHTTYPE stands for the kind of sensor used because there are some versions of DTH supported by this library, DHTTYPE are:

```

// #define DHTTYPE DHT11 // DHT 11
// #define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
// #define DHTTYPE DHT21 // DHT 21 (AM2301)

```

The one used will be uncommented in order to give the entry to the declaration first seen. The instance is declared as global for being able to use the sensor on the code globally. Afterwards, the sensor must be initialized with the instruction: `dht.begin();`. This method could be used either at the set up or create a function where each time is needed, it will be initialized.

The following methods allow to easily retrieve the measures of the sensor, both of them:

```
// Reading temperature or humidity takes about 250 milliseconds.
float h = dht.readHumidity();
float t = dht.readTemperature();
```

3.3.1.1.2 DS18B20

3.3.1.1.2.1 Wiring

This sensor has two possibilities, the normal or parasite mode; in our case, **normal mode** is used, which means that the three cables need to be used. According to what is seen in Illustration 12, the red one stands for VCC, 5 volts, in this case, the black one is Ground, and the yellow gives the data, therefore is connected to a digital pin of the Arduino.

3.3.1.1.2.2 Programming

In the case of this sensor, two libraries are needed: “*OneWire.h*” and “*DallasTemperature.h*”. “*OneWire.h*” allows defining the pin from which the connection will be made, whereas the second one provides the functionalities needed for obtaining the measures of the sensor. On the code below, these steps are followed. The address of OneWire is firstly created, for giving it afterwards to an instance of DallasTemperature’s sensor. From the moment that the object “sensors” is created, the rest of the configuration can be performed.

```
//Sensor DS18B20
OneWire ourWire(NumberOfPins); //Establishes the pin 'NumberOfPins' as bus OneWire
DallasTemperature sensors(&ourWire); //An object is declared for our sensor
sensors.begin(); //The sensor is initialized
```

The functions offered by “DallasTemperature” allow retrieving the values easily. In the first instruction, the measure is performed and then, the temperature of the sensor selected is obtained. Note that an index is indicated to the function that retrieves the value, the reason is that more of one sensor could be connected to the same OneWire address. Each sensor has a unique identification address; therefore, there are no interferences if more than one is added. For this project, only one sensor is considered, but this possibility is highlighted for being interesting for future applications.

```
sensors.requestTemperatures(); //Requests the measures
temp = sensors.getTempCByIndex(0); //Obtains temperature from the sensor with index 0
```

3.3.1.1.3 HD-38

3.3.1.1.3.1 Wiring

This sensor includes a potentiometer to regulate its exit value. The two cables of the sensor must be connected to the potentiometer. The positive goes to the ‘+’ and the negative to the ‘-’ indicated in the potentiometer board. Next, the exits of the potentiometer are connected to the Arduino’s ground and power. This time, the pin for the data returns an analogic value; therefore, it must be connected to an analogic Arduino pin such as the A0. This connection can be observed in Illustration 12

3.3.1.1.3.2 Programming

Bellow, a code where this sensor is set up, is presented. First, the Arduino pin is declared in order to access its value later on. The value of an analogic pin of the Arduino is given by the function “analogRead”. Once the analogue value is read, a mapping is done to obtain a percentage of moisture. The mapping is done because the range of measures that could be obtained is known in advance,

```
const int hygrometer = A0; //Hygrometer sensor analogue pin output at pin A0 of Arduino
// When the plant is watered well the sensor will read a value 380~400
value = analogRead(hygrometer); //Read analog value
value = constrain(value,400,1023); //For keeping the ranges
Value = map(value,400,1023,100,0); //Map value : 400 will be 100 and 1023 will
be 0
```

3.3.1.2 SD card Adapter connection

3.3.1.2.1 Wiring

As mentioned before, this module uses SPI communication. In order to communicate by SPI means with Arduino, special pins must be used. These vary according to the board that is used; Table 10 shows a summary of SPI pins on the Arduino Boards:

| Arduino Board | MOSI | MISO | SCK | SS (slave) | Level |
|----------------------|--------------|--------------|--------------|------------|-------|
| Uno or Duemilanove | 11 or ICSP-4 | 12 or ICSP-1 | 13 or ICSP-3 | 10 | 5V |
| Mega1280 or Mega2560 | 51 or ICSP-4 | 50 or ICSP-1 | 52 or ICSP-3 | 53 | 5V |
| Nano | 11 or ICSP-4 | 12 or ICSP-1 | 13 or ICSP-3 | 4 | 5V |

Table 10. SPI Arduino Pins according to the board used

Error! Reference source not found. depicts the wiring performed in an Arduino MEGA.

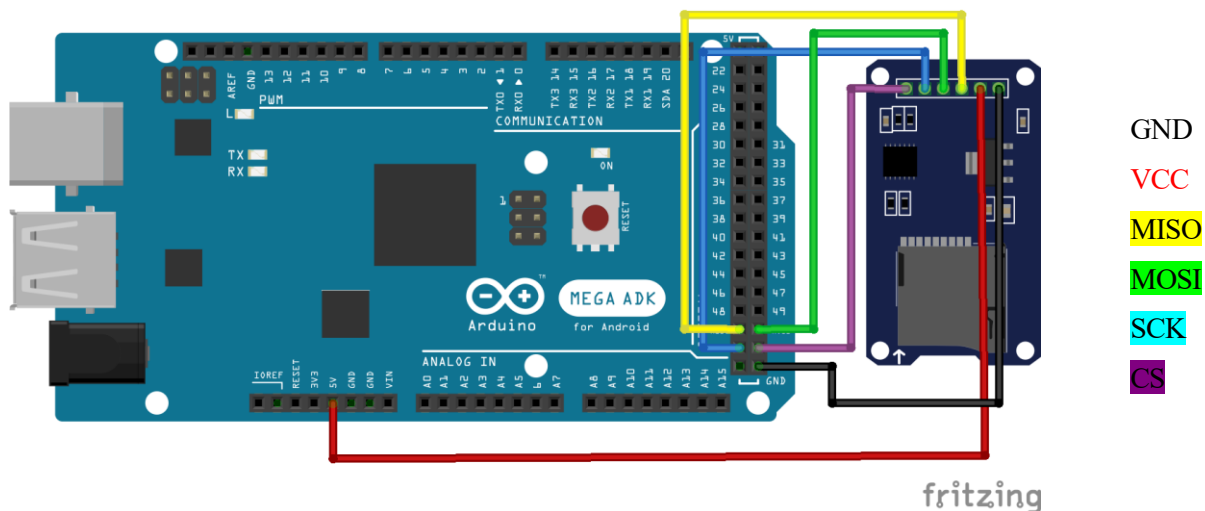


Illustration 13. Arduino Mega - MicroSD Adapter wiring

3.3.1.2.2 Software

“*SD.h*” and “*SPI.h*” allow to manage file creation, the memory storing and reading. First of all, the CS pin must be identified, according to what is listed in Table 10. The selected pin is configured as Output in order to work with the microSD. The SD module is initialized in the next instruction, and then everything is prepared. Arduino has a File class which allows for reading from and writing to individual files on the SD card. Therefore, a variable of type File will be created. In order to make easier the file managing such as opening, closing or creating; and the memory saving or reading; some functions are created for those specific tasks.

```
pinMode(CS_PIN, OUTPUT); // This instruction configure the CS pin as output
SD.begin(); // Allows to initialize Serial communication
```

3.3.1.3 SIM808

3.3.1.3.1 Wiring/welding

When this module is connected to the external power supply, the 3G connection is not automatically activated. There is a button to press for this communication to begin. Pressing the button could be avoided throughout additional welding, which allows instructing to turn on or off the module programmatically. Therefore, the welding was made; in Illustration 14, the process is shown. The pin that gives this functionality is the J19, it must be connected to a digital pin of the Arduino so that by giving a pulse the turning off/on function is performed.

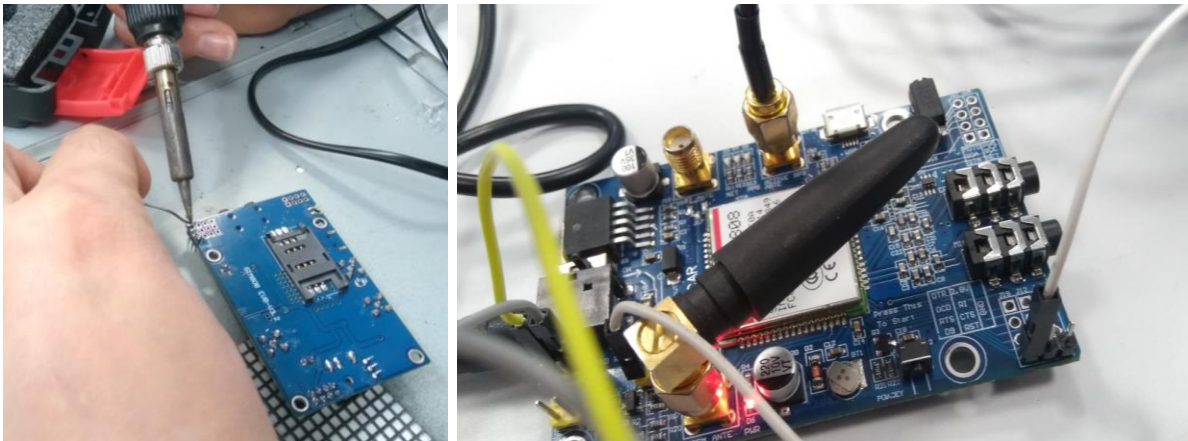


Illustration 14. Welding J19,J12,J13 and result.

The rest of the connections with the Arduino are shown in Illustration 15. Two pins are needed to perform Serial Communication, both TXD and RXD are going to digital pins of the Arduino. The ground must be unified, seeing that external supply is provided (Illustration 10); therefore, another cable must be connected between the SIM808's ground and Arduino's.

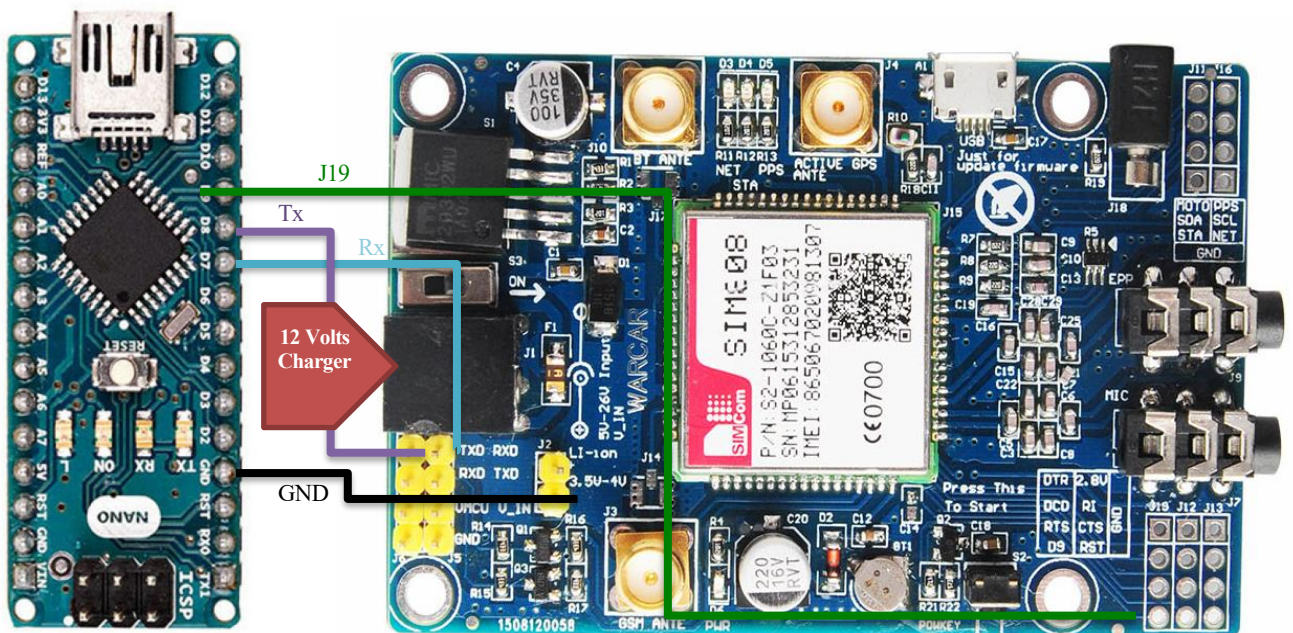


Illustration 15. SIM808 wiring with Arduino Nano

3.3.1.3.2 Software

In this case, the library *“SoftwareSerial.h”* is imported. The aim of using this library is to allow Serial Communication using digital pins of the Arduino, besides the preset 0 and 1 pins so that it will not interfere with

the Computer's Serial Communication. Therefore, an instance standing for SIM808 is created, which will allow the interaction between Arduino and SIM808 Module. The code is found below.

```
//CONEXION SIM
SoftwareSerial SIM808(10, 11); //Configuring serial pins using software
static const int PowerPin =9; //This pin is used to turn on-off the Module
```

After setting up the communication, it is noted that it works the same as Serial Computer Communication. Therefore, the baud rate must be selected to coincide with the one of Arduino. These instructions must be accomplished before using communication.

```
SIM808.begin(9600); //Arduino communicates with SIM808 at 9600bps
```

Afterwards, the interaction between both is made reading and writing in one another. Whereas for sending AT commands into SIM808 those are printed into the port, SIM808.print instruction.

```
Serial.write(SIM808.read()); //This commans
SIM808.print("AT+CMGF=1\r"); // AT command
```

In order to accomplish the process of sending or receiving SMS, it is compulsory to use the commands for configuration described in the previous section. Then, the corresponding command is written to perform the activity. (See Table 8)

3.3.1.4 User Interface

No set up is required to carry out the communication between the phone and the Arduino. If the device is intended to start the communication, the user's phone number must be known in advance. Otherwise, the user could send an SMS with identification, giving the system the number.

3.4 Final Outline

In this section, the final aspect of the system is presented. Here, the aim is to meet the objectives outlined throughout the research and work, as well as solving possible final implementation problems that may arise. First of all, the description of the system's performance is presented.

3.4.1 Performance description

So far, the versatility of the system has been discussed, but no details have been given on how to achieve it. It is foreseen to provide the user with a **manual** containing information that he or she will need in order to configure the device. For instance, since the wiring of the sensors was not supposed to be preset, this manual will provide some details about how to identify each sensor. For this reason, prior identification of how many and which sensors are connected is necessary. Table 11 reveals how each sensor will have an address to be identified. The same codification will be internally stored in the device in order to understand which sensor is connected. Consequently, the kind of sensor connected to a particular pin is not preset, allowing this way the flexibility wanted.

| Sensor | Address |
|----------------------------|---------|
| DS18B20 (Soil Temperature) | 10 |
| DTH11 (Humidity) | 11 |
| DTH11 (Temperature) | 12 |
| HD-38 (Soil Moisture) | 13 |
| ... | ... |

Table 11. Address keys

The assumption here is that even if the sensors used are right now defined, in the future, some others may be added. The intention is to avoid a considerable change in the implementation to extend the device's functionalities. Proceeding in this way, allows the user to perform the connections of the sensor in a more versatile way. Besides, the first of the functionalities: "Ask the user which sensors are connected." would be covered. The maximum number of sensors has been limited to five (Requirement 1) for this system. In a real performance, this is the first parameter that the user will be asked to define.

Secondly, the user must define a frequency corresponding to the frequency of SMS reception. These SMS will include the measure of each sensor. The **limitation will be 30 seconds** because between each SMS there is an unavoidable delay. This information will also be displayed on the User's manual. If the set frequency does not meet this requirement, another one is asked. However, in a real-life test of the system, a much lower frequency should be considered. It would not make sense to observe the parameters with such insistence since changes will generally occur gradually over time. The second feature: "Asking the user to define a frequency for the regular notification" is accomplished.

On the other hand, the device will count with an **internal frequency** to monitor data (Requirement 2). This frequency is defined as 30 seconds in order to have information for each moment throughout a day. This requirement is not subject to be changed by the user. The data will be stored in the external memory of the device, and it is possible to consult this information anytime.

To sum up, it is found that the user will have to define two features when first setting up of the system: the number of sensors connected as well as its identification according to Table 11, and the SMS frequency. Once determined, the system is prepared to start measuring.

At this point, the device works autonomously. There are also some extra functionalities: by using specific keywords, like the ones seen in Table 12, the user is able to perform different functions. These extra functionalities allow completing the remaining features of the system.

| Keyword | Function |
|--|---|
| Address | Change the identifications of the sensors connected |
| Consult | Ask for the values of sensors connected in a specific passed moment of time |
| Frequency | Change actual period of user's sms update |
| Keyword of the sensor In our case: <ul style="list-style-type: none"> - Temperature is established for DS18B20 - Moisture would be established for HD38 - Humidity would be established for the humidity value of sensor DTH11. - Ambient Temperature would be established for DTH11. | Each sensor will have a particular keyword in order to retrieve its measures at the moment. Other keywords could be configured according to future sensors connected to this device. |

Table 12. Keywords for the different functions of the system.

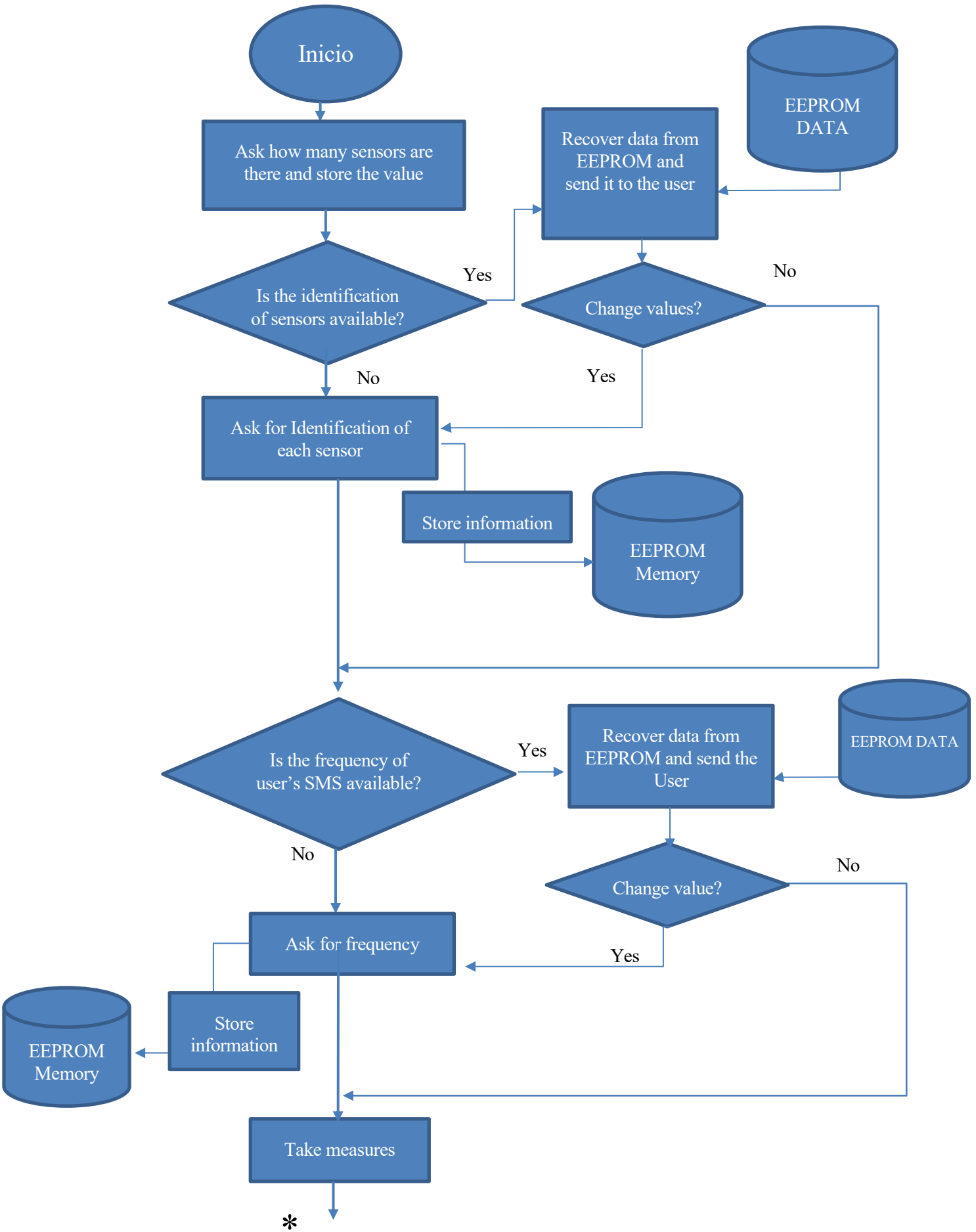
Moreover, when a parameter goes out of a specific range, the user will receive an alert. This functionality corresponds to the last of the features presented in 3.1.

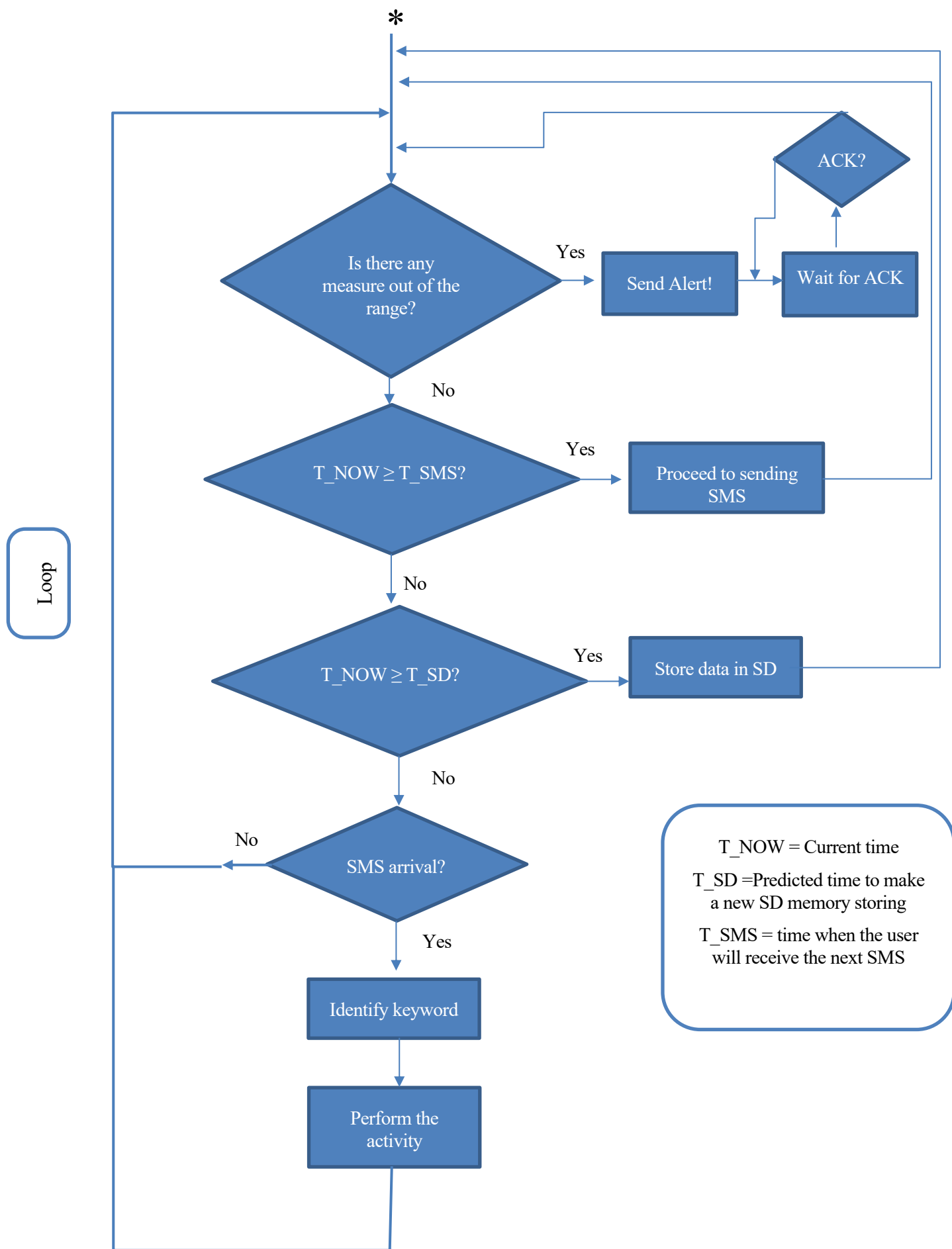
If the device is accidentally or intendedly turned off, when it is turned on again, it asks for the number of sensors connected. It would then send a confirmation of the data it had stored from when it was in operation. An SMS

will be sent to inform the user of the addresses it contains and the frequency of messages. The user is asked to confirm whether or not he/she wants to change them and then continues the execution regularly, as explained above.

3.4.2 Programming path

So far, the behaviour of the system is summarised in broad terms. Right now, all the information is summarized in a flowchart. The scheme describes the path that the future program will have to follow more transparently.





4 TESTS

All the sensors described in the preceding section were not available. For this reason, some modifications on the first outline of the system had to be made. To avoid losing versatility on the whole system, the remaining sensors were simulated; more details about the process are given in the present section. The equipment available was:

- DS18B20
- Arduino Nano/Arduino MEGA.
- Micro SD card Adapter
- SIM808 evb-v3.2

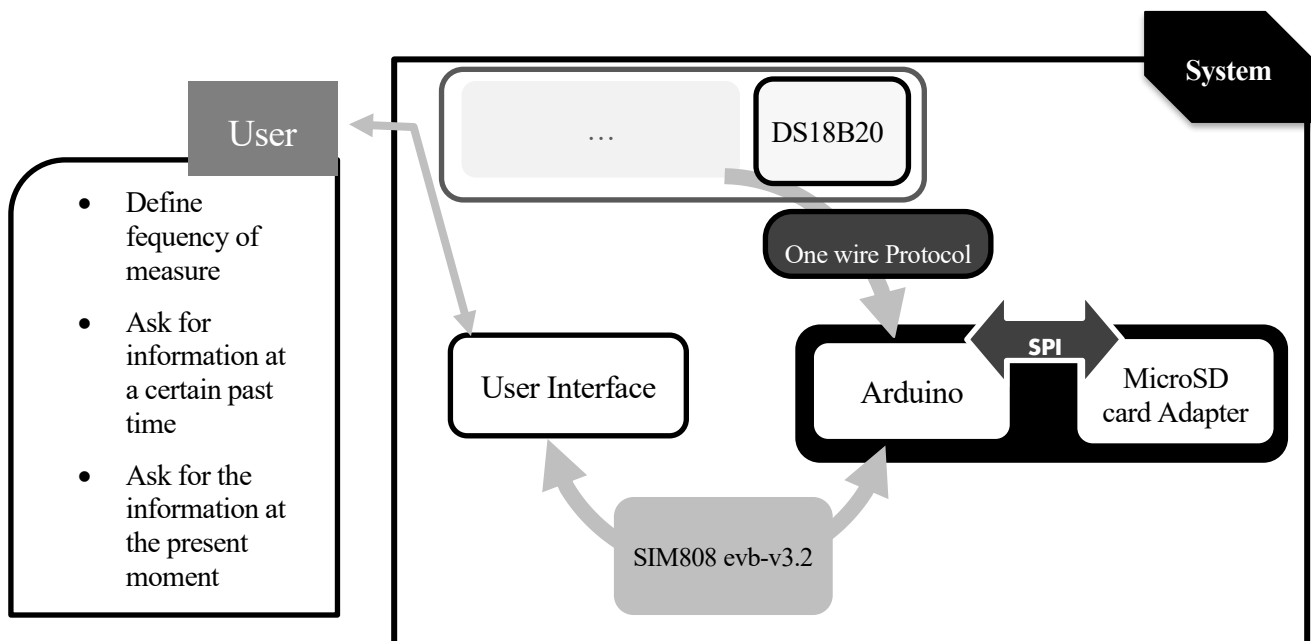


Figure 14. Final system

Subsequently, a series of tests were carried out with this material. The importance of these tests resides in the fact that they help to get acquainted with each Module separately. Once it is understood how each one works, it is possible to proceed to the shaping of the final system. Each section below describes the tests carried out for each Module. In addition, tests are referred to the requirements that they enable to satisfy, those raised in Section 3.1.

4.1 Sensors

Provided that only one of the sensors were available, the way of creating a more realistic system was to simulate the remaining sensors by reading random values for each one. Therefore, it is proceeded with the programming of the real sensor, based on the information given in the previous section. Besides, some trials were made in order to implement the function that was going to provide the values.

4.1.1 DS18B20 Sensor's test

Using the previously described libraries, but also studying its way of working and making some tests, the sensor was quickly launched. A method was created in order to obtain the measure each time it was called. The result of the tests is found in Illustration 16.

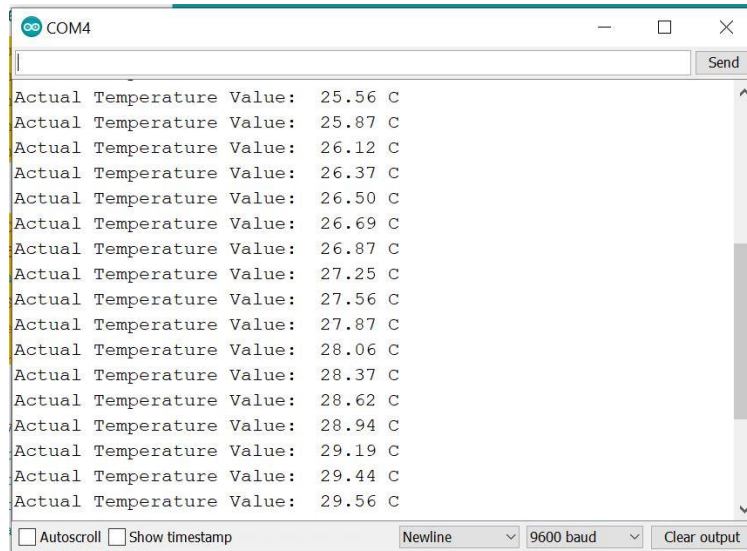


Illustration 16. Arduino Serial Port Communication: DS18B20 redord

✓ **Test check:** Completing this test allows for obtaining the soil temperature measurement. Therefore, at this point, the requirement of observing this parameter is fulfilled. The result of the test is not only significant for this reason but also because it is possible to work with the value obtained: transmitting it or saving it.

4.1.2 Sensor's simulation

The fact of monitoring several parameters renders the system much more effective and realistic. Since this idea was to be retained, the rest of the sensors were simulated with random values. In order to obtain those random values, a function was created using Arduino's own "random()" method. This function returns pseudo-random values up to the number given between brackets. Illustration 17 demonstrates how this function is combined with the previous temperature value and how all sensors are gathered together.

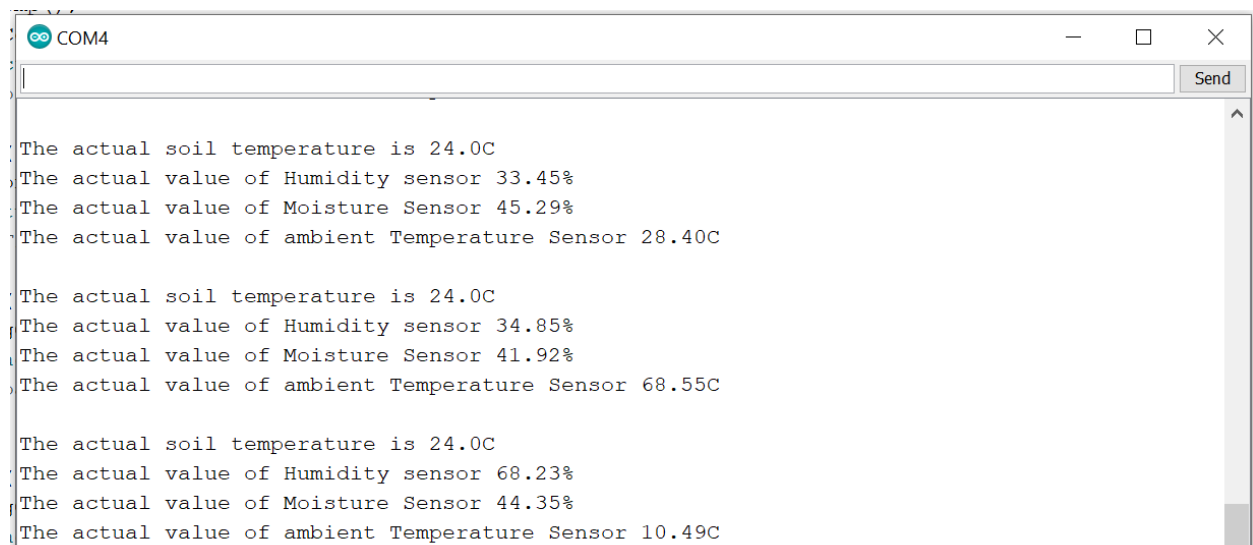


Illustration 17. Arduino Serial Port Communication: Sensors record

✓ **Test check:** At this point, a more generic implementation of the system is allowed because in the future, when having all the sensors available, it will be easier to integrate them, at the programming level. On the other hand, obtaining these values and being able to operate with them, allows the selected parameters in 3.2.1 section, “soil and ambient parameters”, to be successfully monitored.

Concretely, it is possible to refer here to the requirement number 5: Provide versatility in the sensor’s implementation/connection.

4.2 Arduino

Up to now, due to the fact that the rest of the parties were not yet considered definitive, the alternatives of Arduino Mega or Nano, to implement the Processing Unit, have both been considered. However, at this moment, it is convenient to choose one of them in order to continue with the project. The characteristics of both boards have already been discussed in section 3.2.2.1.1, and the differences between them can be found in Table 4.

4.2.1 Arduino Nano vs Mega

The price of both can be considered a determinant factor to decide since Arduino Nano costs 20 euros [7] whereas Arduino Mega costs 35 euros [8]. We have mentioned above aspects of size, resulting Arduino nano advantageous over mega, if integrating everything into a small system is aimed. However, above all this, it is found the factor related to memory. Once the other elements have been defined, it is verified that a large number of external libraries is required. The size of the sketch increases whenever a library is added. The comparison between the memories follows in Table 13:

| | MEGA | NANO |
|---------------------|---|--|
| Flash Memory | 256 KB of which 8 KB used by bootloader | 32 KB of which 2 KB used by bootloader |
| SRAM | 8 KB | 2 KB |
| EEPROM | 4 KB | 1 KB |

Table 13. Arduino Mega VS Nano, memory comparison.

As a recall: The Flash memory (program space), is where the Arduino sketch is stored. The SRAM is where the sketch creates and manipulates variables when it runs. EEPROM is a memory space that programmers can use to store long-term information. The three of them will be used in the implementation of the system, but the first two are essential to decide which board to use.

Just by loading the ten libraries needed to use all the elements and leaving the program empty; it can be noticed how Arduino Nano is quite far from being able to be used without the risk of running out of memory. In Illustration 18, it is appreciated how this program requires almost 40% of the total SRAM memory and leaves only around 80% for the programming of functions and general code. When compared to the 9% and 2% occupation of Flash memory and SRAM respectively of Arduino Mega, which can be seen in Illustration 19, there is no doubt that Arduino Mega must be chosen in order to continue with the implementation.

```

sketch_may24a | Arduino 1.8.12
File Edit Sketch Tools Help
sketch_may24a
#include <Time.h>
#include <TimeLib.h>
#include <DallasTemperature.h>
#include <OneWire.h>
#include <String.h>
#include <SoftwareSerial.h>
#include <EEPROM.h>
#include <SD.h>
#include <SPI.h>

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

Done compiling.

Sketch uses 5250 bytes (17%) of program storage space. Maximum is 30720 bytes.
Global variables use 795 bytes (38%) of dynamic memory, leaving 1253 bytes for local variables. Maximum is 2048 bytes.

9 - 1 Arduino Nano on COM4

```

Illustration 18. Arduino program with the libraries that will be used and its result in Arduino Nano's memory

```

Done compiling.

Sketch uses 5736 bytes (2%) of program storage space. Maximum is 253952 bytes.
Global variables use 795 bytes (9%) of dynamic memory, leaving 7397 bytes for local variables. Maximum is 8192 bytes.

9 - 1 Arduino Mega or Mega 2560 on COM4

```

Illustration 19. Arduino Mega's memory utilization when all the libraries are loaded

4.2.2 Arduino Tests

Now the tests needed to implement the parts that Arduino controls of the Processing Unit are explained. There are different functionalities of the Arduino, not related to the units connected to it, which need to be programmed for proper operation. These functionalities, such as time control or error forecasting, need to be addressed separately. How to implement those functionalities and the trials made is presented below.

4.2.3 Writing/Reading from EEPROM

Provided that if the system is turned off, it has to work again; some information must be stored to permit re-launch without asking for it again. The EEPROM is a memory whose values are kept when the board is turned off, as previously mentioned. Therefore, when it was decided to use this solution, a necessary test to be carried out was to read and store data in this memory. Thanks to a library so-called “*EEPROM.h*”, performing the activities of reading and writing was straightforward employing its functions: `read()`, `write()`, `update()`.

For this trial, since each address is a byte available, some values were written at specified addresses, then the Arduino board was disconnected and connected again. A program for reading the whole EEPROM was then uploaded to check whether the value was still the one previously assigned.

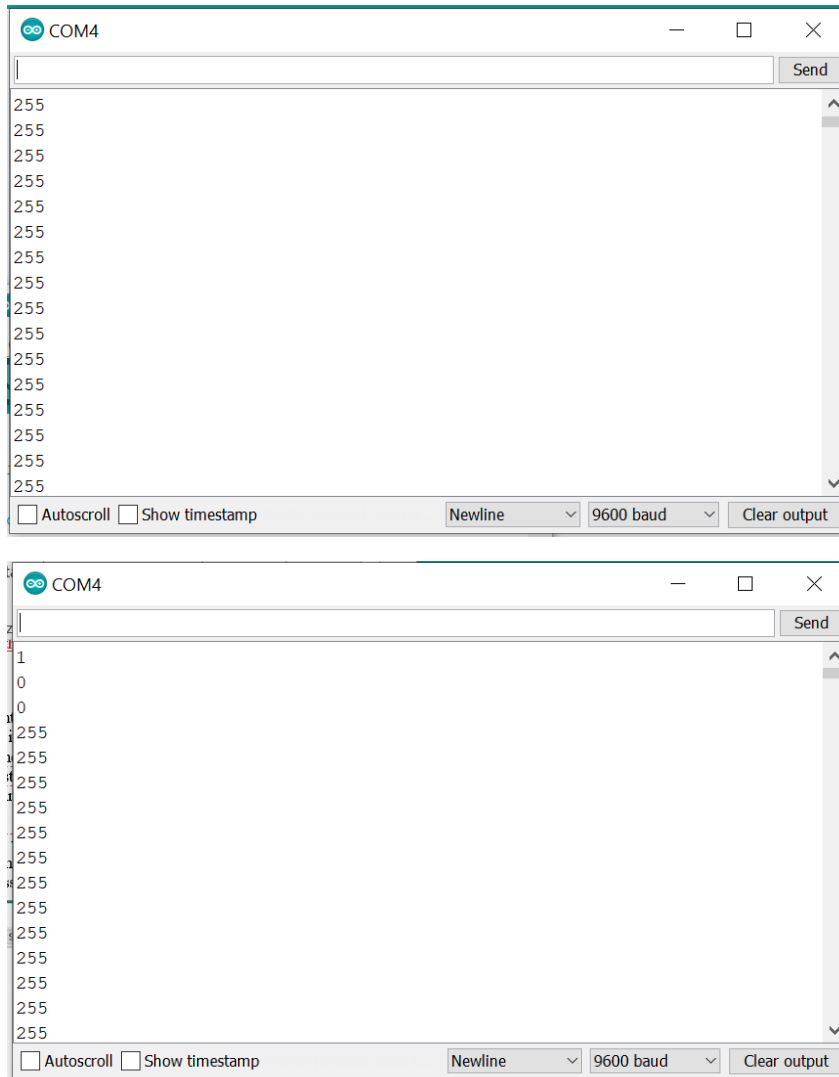


Illustration 20. Before and after writing on the EEPROM memory

It is observed in Illustration 20 that the default value of the memory is 255. At each address, only a byte is supported. In the final implementation, this memory will store information about the frequency and the identification of the sensors.

✓ **Test check:** Anticipate a possible turning off error.

4.2.4 Clock

It is crucial to managing time in our system; Arduino boards can create clock signals and interact with timers or dates. Provided that a physical clock module is not available, again we resort to the time libraries available in Arduino. The Time library adds timekeeping functionality to Arduino with or without external timekeeping hardware. When included in a sketch, some macros, functions, and new variables are available to create timers.

It is relevant to mention the *time_t* variable, having 32 bits, allows to store time format data in a precise way: The number of seconds transurred from a date until the first of January of 1970 [9]. The variables containing time can be added or subtracted, a great advantage, for instance, to compare different dates. Once this is understood, it is essential to set the time in order to match it with the current time. The `setTime()` function allows doing the coordination accepting as parameters the current time and date. In conclusion, the first step regarding the clock point will be to set the time.

Additionally, knowing how to create moments of time will also have relevance. The structure `tmElements_t` implements this functionality, some details must be taken into account about its values,


```

tmElements_t  tm ;

tm.Second= // Seconds    0 to 59
tm.Minute= // Minnutes   0 to 59
tm.Hour=   // Hours     0 to 23
tm.Wday=   // Day of the week 0 to 6 (It is not used in mktime)
tm.Day=    // Day       1 to 31
tm.Month=  // Month     1 to 12
tm.Year=   // Year     0 to 99 (Diference since 1970)

```

once a structure `tmElements_t` is created, to generate the time it is necessary to do:

```
time_t Time = makeTime(tm);
```

Illustration 21. Displays the result of setting up a clock by using the `setTime` function, giving as parameters each number.

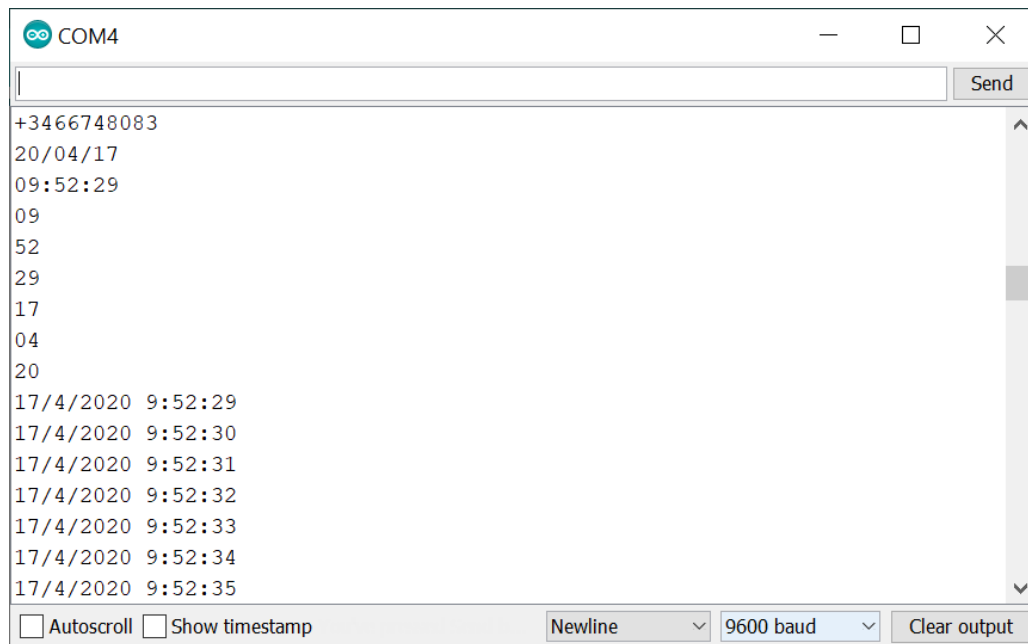


Illustration 21. Arduino Serial Port Communication: Clock setting

✓ **Test check:** In this case, it has been proved that it is possible to manage time with Arduino libraries and its functionalities. This test allows to include time in the implementations and helps to overcome time checks that are needed to fulfil the requirements.

4.2.4.1 Frequency checks

Whenever a clock is defined, there is a function called “`now()`” which returns the actual time; it can be stored into a `time_t` variable. By utilizing this function, the actual time can be compared to an objective one in such a way that it is possible to check if the quantity of time defined has already elapsed. Illustration 22 shows the first try comparing a moment to another ten seconds after. Note that during a whole second, time objective is equal to the actual time, that is why “Time reached!” is printed several times.

```

COM4
08:22:02.070 -> Setting up clock...
08:22:02.070 -> 10/04/20
08:22:02.070 -> 18:00:29
08:22:02.103 -> The time we want to reach: 18:0:39
08:22:12.159 -> Time reached!
08:22:12.159 -> Time reached!
08:22:12.159 -> Time reached!
08:22:12.193 -> Time reached!
08:22:12.193 -> Time reached!
08:22:12.227 -> Time reached!
08:22:12.227 -> Time reached!
08:22:12.260 -> Time reached!
08:22:12.260 -> Time reached!
08:22:12.294 -> Time reached!
08:22:12.294 -> Time reached!
08:22:12.327 -> Time reached!

```

Illustration 22. Arduino Serial Port Communication: Frequency Check

✓ **Test check:** This test is simple but helps enormously to understand how to manage a frequency check.

At the end of this section, we are able to provide a “timing system” to the device and also to manage frequencies to perform operations. From now on, it is possible to meet the requirement of having a periodicity on the measurements and communication. The requirement: Define an internal frequency of measurement, presented in 3.1 is now achievable.

4.3 SD tests

How to initialize the SD card was previously explained in section 3.2.2.3. The aim of the tests regarding the micro SD card was first to get familiar with the File class of Arduino and how to work with it. There are some basic predefined functions to work with files, `SD.open()`, `SD.remove()`, `SD.exists()`. `SD.open()` receives, as inputs, the name of the file and a macro that enables reading and writing access, `FILE_WRITE`; it returns an object of File class. On the other hand, the function `SD.exists()` checks if the file given as argument exists, whereas the `SD.remove()`, deletes the file.

4.3.1 Writing a file

An object of file class uses `file.println()` in order to write into the file. This test performs the writing and file creation employing the functions detailed above. A function which encapsulates the work of opening the file or creating it if it does not exist, writing and closing the file, is created for this test. In order to have a clue of what is happening, some Serial Port prints are included. The result of the file is what is seen on the right side of the Illustration 23.

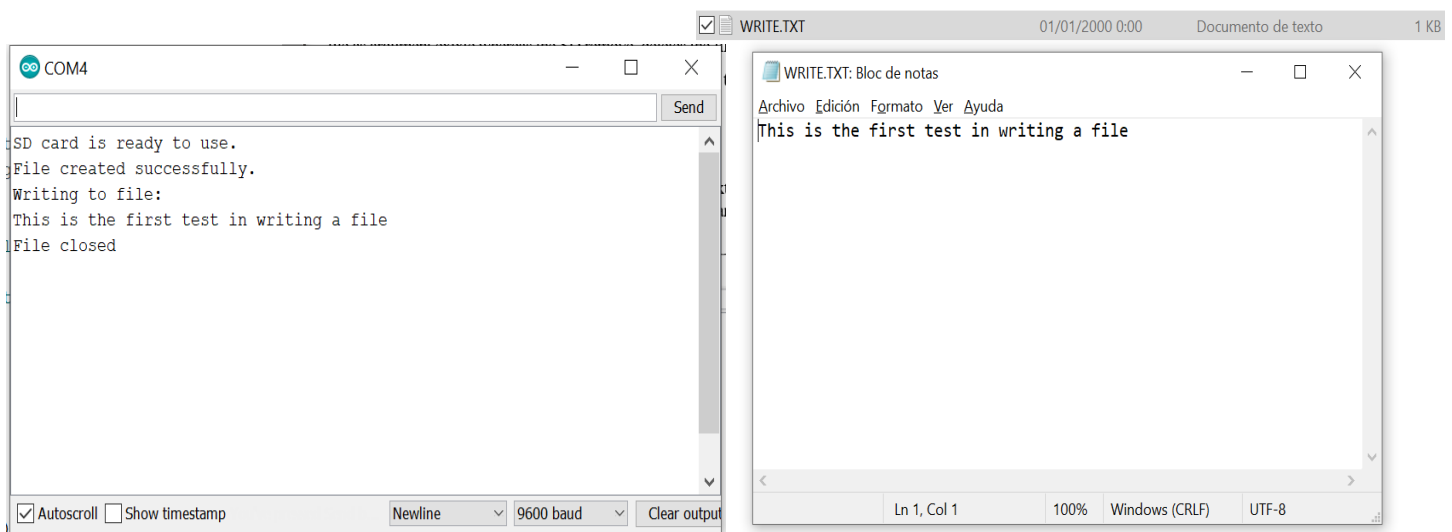


Illustration 23. The process of writing a file in the SD cad.

✓ **Test check:** Being able to write information into a file, allows to accomplish the requirement of storing information in the external memory.

4.3.2 Reading a file

File.read() is the function for reading something from the file which has been already opened. The reading is performed character by character. The final of a line can be identified when the character equals a “\n” which is the terminator character; this is how reading is performed “line by line”. In Illustration 24, this process is appreciated. A file is prepared with some information (on the right side), then the result of reading the lines is displayed on the left side.

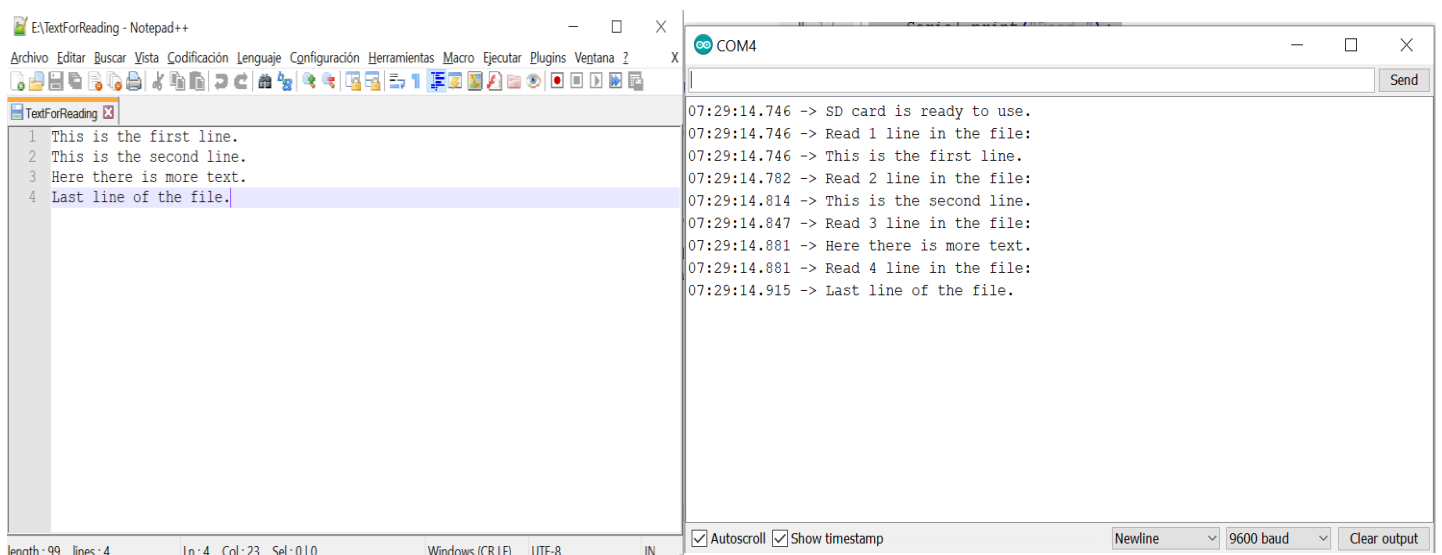


Illustration 24. The process of writing a file in the SD cad.

✓ **Test check:** One of the functionalities wanted for the device is to retrieve the information of past times that are in a file. This test demonstrates how to perform the activity.

4.3.3 Conclusion

These tests are closely related to Requirement number 6: Provide memory to the system, having completed both functions of reading and writing, it is possible to implement this specification in the final system without problems.

4.4 SIM808 tests

Knowing well how this module works is of vital importance for the subsequent implementation of the final program. The aim of performing these tests was to automate the entire process of communication with the user, after understanding how it works. It has been previously described, on 3.2.3.1.1 section, how to set up the module. The parameters to configure the SMS transmission only have to be used once. Once the set up is made, it has been proceeded with tests that provide a general overview of how the module works sending and receiving SMS.

4.4.1 Sending SMS

The process for sending an SMS and the commands to use are previously detailed as well on 3.2.3.1.1 section. The first try has been programmed to work by sending the commands on the Serial Port. Proceeding this way allows to understand which are the module's responses because each operation is printed on the Serial Port. Having this information enables the creation of a function which implements the piece of code needed for sending an SMS. The only particularity is that each SMS needs "CTRL + Z" as terminator character since it is not possible to write this, it is considered on the code, using the following instruction:

```
SIM808.write(0x1a);
```

Provided that 0x1a is the corresponding code in hexadecimal. The results of this test can be observed in Illustration 25

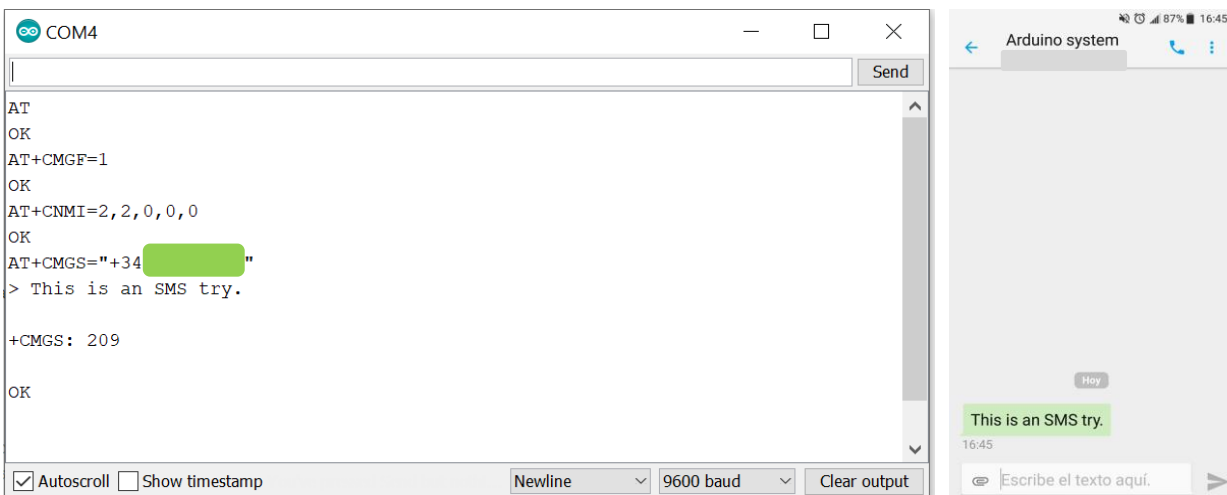


Illustration 25. Arduino Serial Port Communication: Sending an SMS and SMS received in the phone

✓ **Test check:** This test proves that it is possible to start communication from the Arduino with the phone.

4.4.2 Receiving SMS

As a first try for the Arduino to receive SMSs, the arrival of the message is waited. The information that comes in is printed on the Serial Port; the result is shown in Illustration 26. It can also be observed the fact that each message arrival is linked with a String containing the number from which the SMS proceeds, the date (yy-mm-dd format) and the hour.

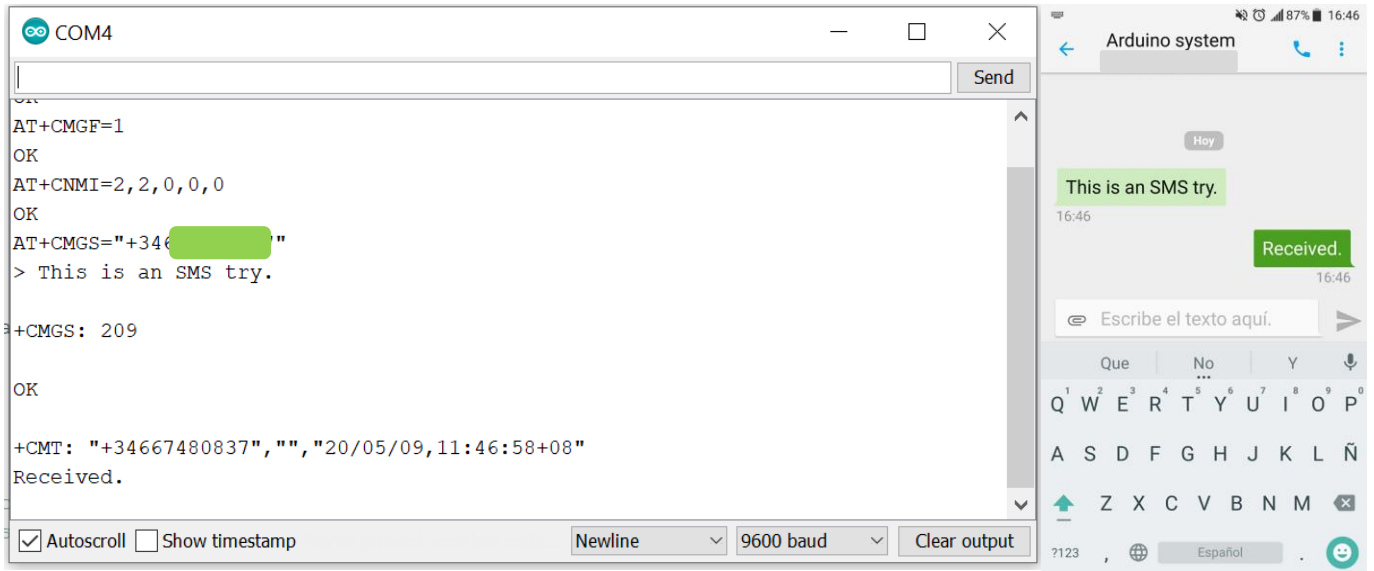


Illustration 26. Arduino Serial Port Communication for receiving a message and the SMS sent in the phone

✓ **Test check:** This trial shows how the User will be able to send information to the system., but also the ability to read and parse the incoming data.

4.4.3 Conclusion

As a whole, it is possible to conclude by saying that the bidirectional communication is accomplished. Hence, the requirement of receiving and sending information to the user is covered. The immediate goal after these tests is to automate the process through functions that perform both the sending and receiving of information. The result of the performance of the functions created is the same; therefore, no Figures are attached. Right now, the requirement number 4, Define a bidirectional communication with the user, is fulfilled.

4.5 Test conclusion and recapitulation

The conclusions of the tests are gathered together in Table 14.

| Sensors | |
|--|--|
| DS18B20 Sensor's test | <ul style="list-style-type: none"> • Obtaining Soil Temperature's value • Operating with that information • Transmit the value • Define a range for alerts |
| Sensor's simulation | <ul style="list-style-type: none"> • "Monitor" all the parameters |
| ✓ Requirement 5: <u>Provide versatility in the sensor's implementation/connection</u> | |
| Arduino Tests | |
| Writing/Reading from EEPROM | <ul style="list-style-type: none"> • Storing information after reset → anticipate a possible turn off |
| Clock | <ul style="list-style-type: none"> • Provide the device with a "timing system". |
| Frequency checks | <ul style="list-style-type: none"> • Regulate frequency activities |
| ✓ Requirement 2: <u>Define an internal frequency of measurement:</u> ✓ Features: Define a frequency for receiving information | |
| SD tests | |
| Writing a file | <ul style="list-style-type: none"> • Storing information |
| Reading a file | <ul style="list-style-type: none"> • Retrieving information |
| ✓ Requirement 6: <u>Provide memory to the system</u> ✓ Feature: Provide access to past data. | |
| SIM808 tests | |
| Sending SMS | <ul style="list-style-type: none"> • Start communication with the User |
| Receiving SMS | <ul style="list-style-type: none"> • Receive information from the User |
| ✓ Requirement 4: <u>Define a bidirectional communication with the user</u> ✓ Features: Ask the user which sensors are connected, Ask the user to define a frequency for receiving information, interact with the User to perform the rest of activities | |

Table 14. Summary of Tests Conclusions

5 DEMO

Once the tests and experiments have been finished, it has been possible to develop the diverse functionalities that were foreseen for the system in 3.1 section. The present section includes the demonstration of the whole system working. A Demo is performed searching to illustrate all the previously described functionalities. Thus, the system is launched for the first time, and the program follows the steps demonstrated in the above flowchart also included in section 3.

5.1 Demo Description

The illustration below shows all the elements available for the demo and the whole system connected. All the available features are going to be performed by order. First of all, the configuration is made and then each keyword on Table 12 is tried. For the demonstration, a picture of SMS messages is attached and also the record of Arduino Serial Port communication.

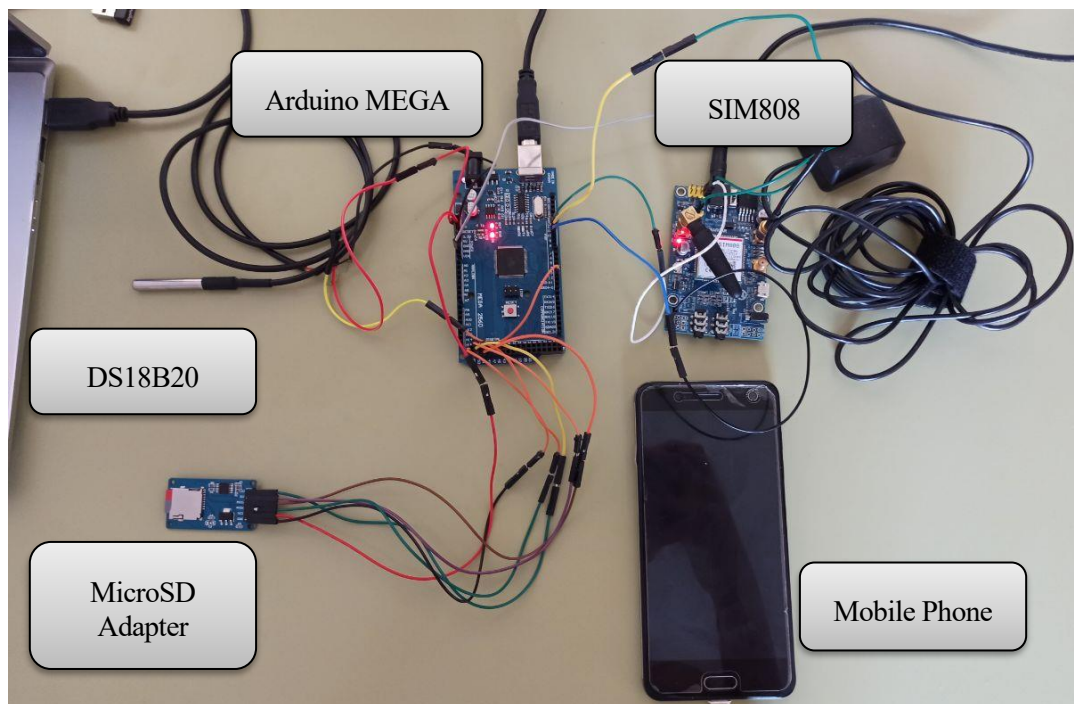


Illustration 27. System prepared for demo

IMPORTANT NOTE: The buffer of the Serial communication between Arduino and SIM808, sometimes is not bigger enough to display the whole SMS text that is going to be sent to the user. However, even if it sometimes appears incomplete in the Serial Port communication, the text arrives complete to the mobile phone. Please check the Illustrations of the mobile interface in each section.

5.1.1 Getting Started

Figure 15 displays the previous configuration needed for the SIM808 and checks if the SD card is correctly connected. Then, when the program is launched for the first time, the number of connected sensors is requested. In addition, it is important to mention that when the user sends a message for the first time, **the time and date** received in the chain together with the message is used to set up the clock for the first time. This chain is not here displayed, but it can also be seen in Illustration 26. The reason why this choice is made is to synchronize the time with the one on the mobile phone.

```

SD card is ready to use.
Set sim
AT+CMGF=1
OK
AT+CNMI=2,2,0,0,0
OK
Inside 'sendMessage' function...AT+CMGS="6 [REDACTED]"
> How many sensor are there?
+CMGS: 253
OK
Waiting for response...
4
Setting clock...
20/05/20
12:19:57

```

System configuration.

First SMS text sent to the user

User's response to the user

Clock Setting to the user

Figure 15. Arduino Serial Port Communication: Demo- Part 1

The next step is to ask for the sensors' identification. An SMS is sent asking for the sensor in turn's number. When the answer is given, this information is stored in the EEPROM. Figure 16 displays this process for the

```

Inside 'sendMessage' function...AT+CMGS="6 [REDACTED] 7"
> Identify sensor number 1
+CMGS: 254
OK
Waiting for response...
11
Inside 'sendMessage' function...AT+CMGS="6 [REDACTED]"
> Identify sensor number 2
+CMGS: 255
OK
Waiting for response...
12
Inside 'sendMessage' function...AT+CMGS="6 [REDACTED]"
> Identify sensor number 3
+CMGS: 0
OK
Waiting for response...
10
Inside 'sendMessage' function...AT+CMGS="6 [REDACTED]"
> Identify sensor number 4
+CMGS: 1
OK
Waiting for response...

```

Ask for Sensor One's identification

User's response

Figure 16. Arduino Serial Port Communication: Demo - Part 2

four sensors in this example. The numbers correspond with the ones previously seen in Table 11.

Whenever all the sensors are identified, a copy of the address assigned is sent to the user. This SMS is used as a double-check that everything is correct. The process can be seen in Figure 17.

```

Inside 'sendMessage' function...AT+CMGS="6 [redacted] 7"
> Address assigned:
1: 11
2: 12
3: 10
4:
+CMGS: 2
OK
Inside 'sendMessage' function...

```

Sensor's identification

Autoscroll Show timestamp Newline 9600 baud Clear output

Figure 17. Arduino Serial Port Communication: Demo - Part 3

The last part of the setting up regards defining the frequency for the SMS texts that will be sent automatically to the user. The frequency is asked as a quantity of time in a hh:mm:ss format. The system takes into account the requirement that 30 seconds is the maximum frequency. In fact, in this example, a lower frequency is given to demonstrate this functionality. In Figure 18, the details about this section is shown.

```

Inside 'sendMessage' function...AT+CMGS="6 [redacted] "
> Define frequency for receiving SMS: hh:
+CMGS: 3
OK
Waiting for response...
00:00:20
Inside 'sendMessage' function...AT+CMGS="6 [redacted] "
> Frequency must be equal or bigger than
+CMGS: 4
OK
Waiting for response...
00:01:00
The next time when the User recieves a message: 12:25:55
The new time when a measure is stored: 12:25:30

```

SMS asking for the frequency

Response of the user

SMS asking for another frequency seeing that the value was not correct

Response of the user

Next time when a measure is sent/stored is calculated

Autoscroll Show timestamp Newline 9600 baud Clear output

Figure 18. Arduino Serial Port Communication: Demo – Part 4

The user interface stands for the SMS app on the mobile phone. The aspect of the process described until now will result as follows in Figure 19.

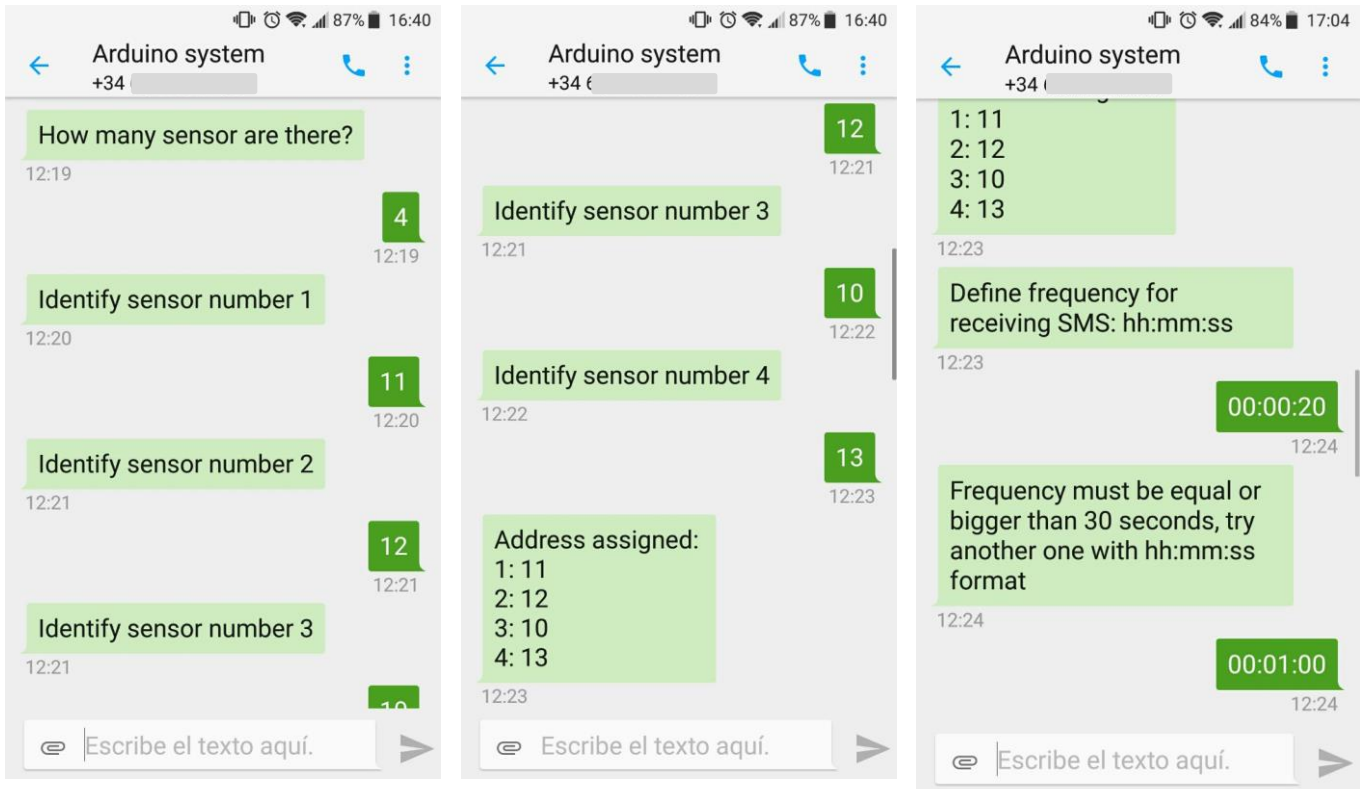


Figure 19. SMS exchanged in Getting Started Demo Part

5.1.2 Program flow

In the regular workflow of the program, messages with the measurements will be sent, according to the frequency. Also, when a measure is stored in SD, a message is displayed on the screen. In Figure 20 the process is displayed; in this case, the User's frequency is one minute. It has been waited for the process to occur twice. The new time when sending or storing data is again performed is calculated only after completing the activity, to avoid not taking into account possible delays on the network.

```
File created successfully.                                     Storing measures in SD
Writing to file:
12:25:30 34.25 77.90 24.81 49.31
The new time when a measure is stored: 12:26:6              Updating storing time
Inside 'sendMessage' function...AT+CMGS="66[REDACTED]7"
> The actual value of Humidity sensor 47.                  SMS texts with
+CMGS: 5                                                    sensors' measurement

Inside 'sendMessage' function...AT+CMGS="66[REDACTED]7"
> The actual value of Ambient Temperature
+CMGS: 6

OK

Inside 'sendMessage' function...AT+CMGS="66[REDACTED]7"
> The actual Soil Temperature is 24.81C

+CMGS: 7

OK

Inside 'sendMessage' function...AT+CMGS="66[REDACTED]7"
> The actual value of Moisture Sensor 9.6

+CMGS: 8

OK

The next time when the User recieves a message: 12:27:26   Updating time
File created successfully.
Writing to file:
12:26:26 59.69 5.95 24.75 78.17
The new time when a measure is stored: 12:27:2
File created successfully.
Writing to file:
12:27:2 7.55 19.10 24.81 51.8
The new time when a measure is stored: 12:27:38
Inside 'sendMessage' function...AT+CMGS="66[REDACTED]7"
> The actual value of Humidity sensor 39.
```

Autoscroll Show timestamp Newline 9600 baud Clear output

Figure 20. Arduino Serial Port Communication: Demo - Part 5

The following illustration shows the aspect of the SMS that update the status of the sensors periodically on the phone. Concretely the illustrations correspond to the previous Arduino transcription in Figure 20.

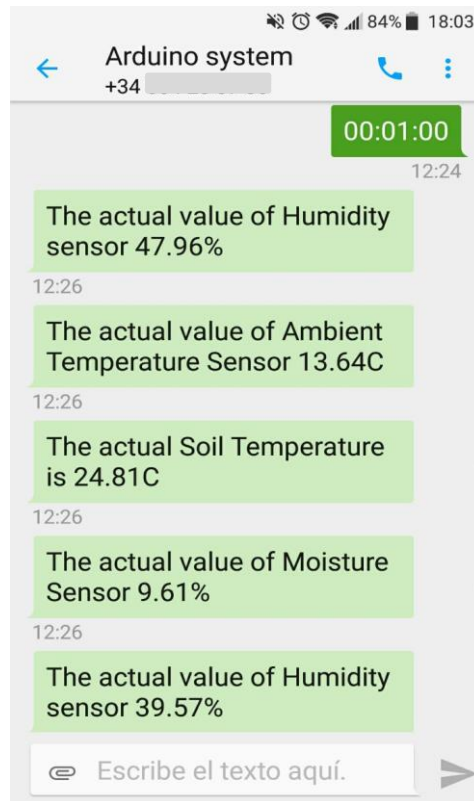


Figure 21. SMS texts with the measures of the sensors

5.1.3 Extra-Functionalities

In 3.4.1 section, some functionalities were presented in Table 12, right now each of them is tried.

- **Address:** Provided that the sensors may vary during the use of the device, it is possible to change the identification by using this keyword. Also, which sensors are currently connected is asked using "Address". An affirmative answer must be given in case of wanting to change it. This process is shown in Figure 23 and Figure 22.

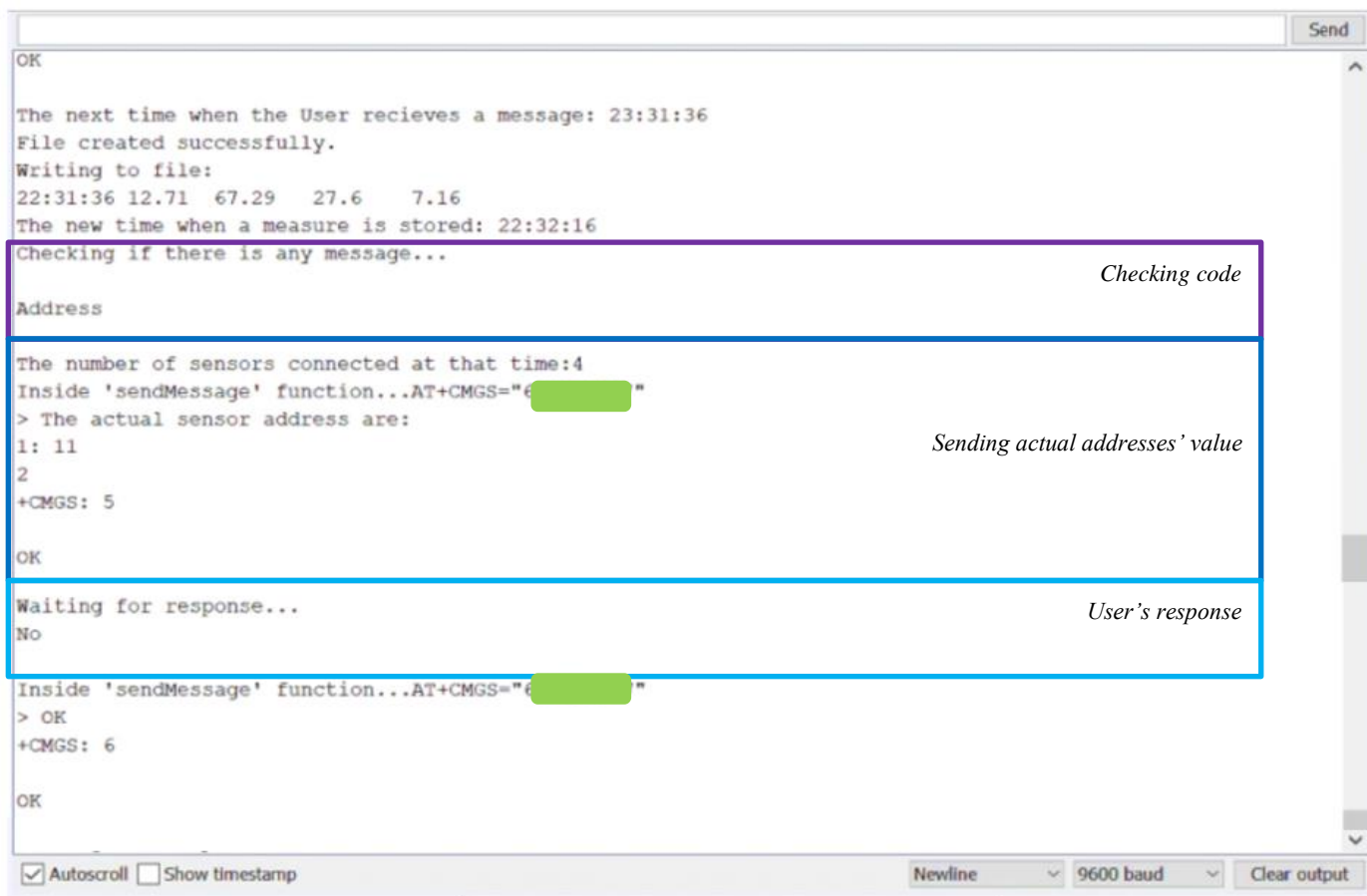


Figure 23. Arduino Serial Port Communication: Address functionality

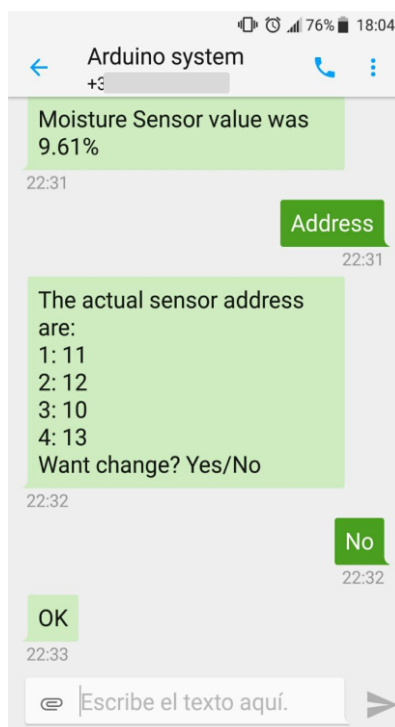


Figure 22. SMS aspect of Address functionality

- **Consult:** This functionality allows the user to know which were the values of the sensors at a particular

past moment. After sending the keyword, the time at which the consult wants to be made is asked. When the system receives this information, the data is searched all over the file, which is stored in SD. If the exact time is not present on the record for any reason, the solution adopted returns the closest value, either over or under it. Whenever the data is found, the quantity of values is checked, because the number of sensors connected could change over time. Both Figure 24 and Figure 25 show the workflow of this functionality.

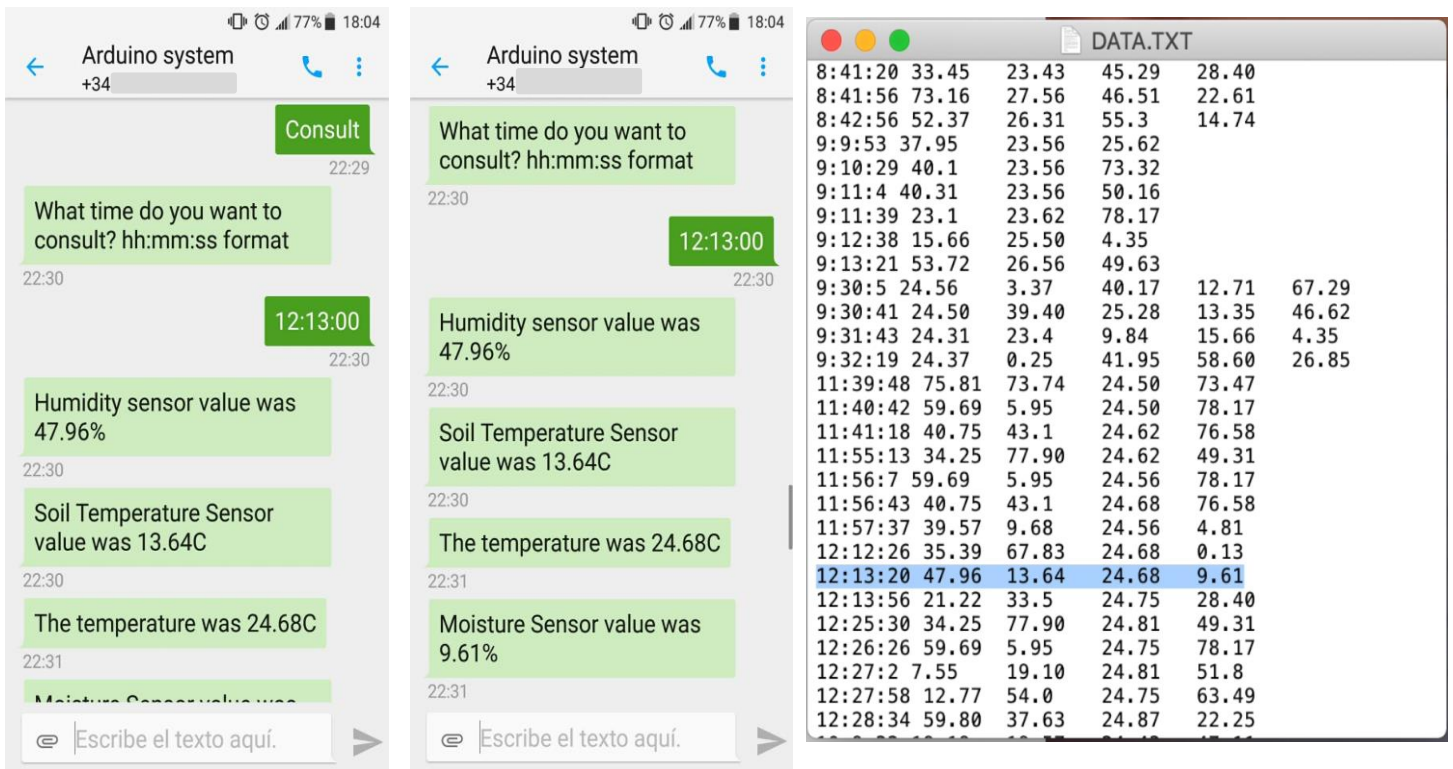


Figure 24. SMS exchanged in Consult functionality and Data record on micro SD

```
Checking if there is any message...
Consult
Code: Consult
Inside 'sendMessage' function...AT+CMGS="6674
> What time do you want to consult? hh:mm
+CMGS: 252
OK
Waiting for response...
12:13:00
File opened with success!
The same hour was stored
The number of sensors connected at that time:4
Inside 'sendMessage' function...AT+CMGS="6674
> Humidity sensor value was 47.96%
+CMGS: 253
OK
Inside 'sendMessage' function...AT+CMGS="6674
> Soil Temperature Sensor value was 13.64
+CMGS: 254
OK
Inside 'sendMessage' function...AT+CMGS="6674
> The temperature was 24.68C
+CMGS: 255
OK
Inside 'sendMessage' function...AT+CMGS="6674
> Moisture Sensor value was 9.61%
+CMGS: 0
OK
```

Autoscroll Show timestamp Newline 9600 baud Clear output

Figure 25. . Arduino Serial Port Communication: Demo - Consult Functionality

- **Frequency:** The possibility of changing the SMS reception frequency during the execution of the programme is under consideration as well. For that purpose, it is necessary first to ask what the current frequency is, sending “Frequency” as a keyword. If an affirmative answer is given, a new frequency is asked. In the example shown below, the frequency started at one minute, is changed to one hour so that the SMS will not interfere with the course of the demo. In the end, the frequency is again asked in order to check that the value has been updated.

| | |
|--|---|
| Checking if there is any message... | <i>Checking code</i> |
| Frequency | |
| Inside 'sendMessage' function...AT+CMGS=" [REDACTED] " | <i>Sending actual frequency value</i> |
| > The frequency period is: 00:01:00 | |
| Want | |
| +CMGS: 238 | <i>Sending actual frequency value</i> |
| OK | |
| Waiting for response... | <i>User's response</i> |
| Yes | |
| Inside 'sendMessage' function...AT+CMGS=" [REDACTED] " | <i>Asking for the new frequency value</i> |
| > Define frequency for receiving SMS: hh: | |
| +CMGS: 239 | <i>Asking for the new frequency value</i> |
| OK | |
| Waiting for response... | <i>User's response</i> |
| 01:00:00 | |
| The next time when the User receives a message: 23:23:38 | <i>Operation result</i> |
| Inside 'sendMessage' function...AT+CMGS=" [REDACTED] " | |
| > OK | |
| +CMGS: 240 | |
| OK | |
| File created successfully. | |
| Writing to file: | <i>Internal data storing</i> |
| 22:24:7 33.45 45.29 27.12 28.40 | |
| The new time when a measure is stored: 22:24:48 | |
| Checking if there is any message... | <i>Checking code</i> |
| Frequency | |
| Inside 'sendMessage' function...AT+CMGS=" [REDACTED] " | <i>Sending actual frequency value</i> |
| > The frequency period is: 01:00:00 | |
| Want | |
| +CMGS: 241 | <i>Sending actual frequency value</i> |
| OK | |
| Waiting for response... | <i>User's response</i> |
| No | |
| Inside 'sendMessage' function...AT+CMGS=" [REDACTED] " | |
| > OK | |
| +CMGS: 251 | |
| OK | |

Figure 26. Arduino Serial Port Communication: Demo - Frequency Functionality

The process described in las paragraph has this aspect on the phone interface:

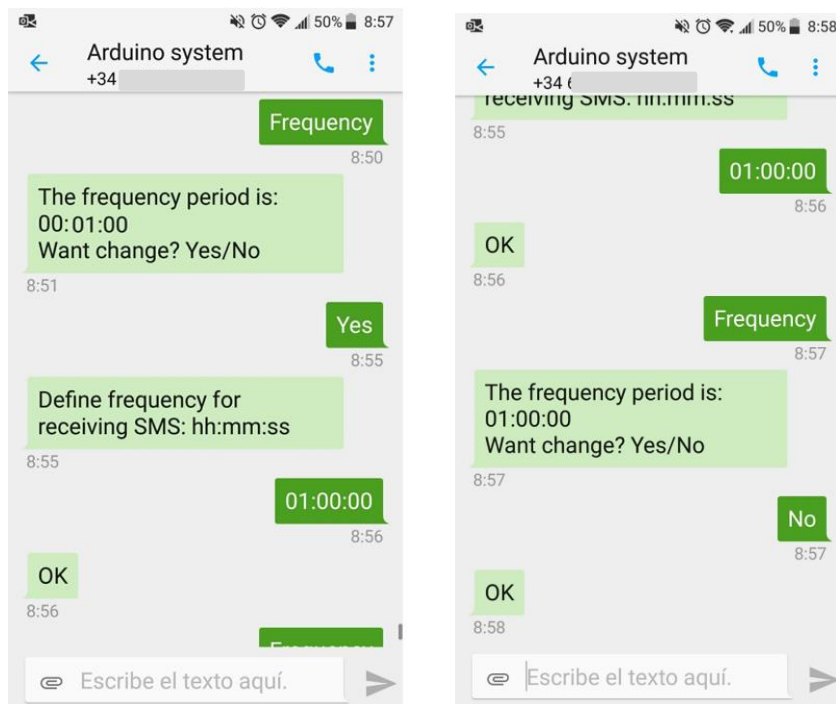


Figure 28. SMS example of frequency functionality

- **Ask for a sensor’s value:** Each sensor will have a keyword to ask for its current value. This functionality is useful in case of having a low SMS reception frequency. Therefore, the user can have an independent update if wanted. The system answers with a text containing the current measure of the sensor. The results are shown in the following Figure:

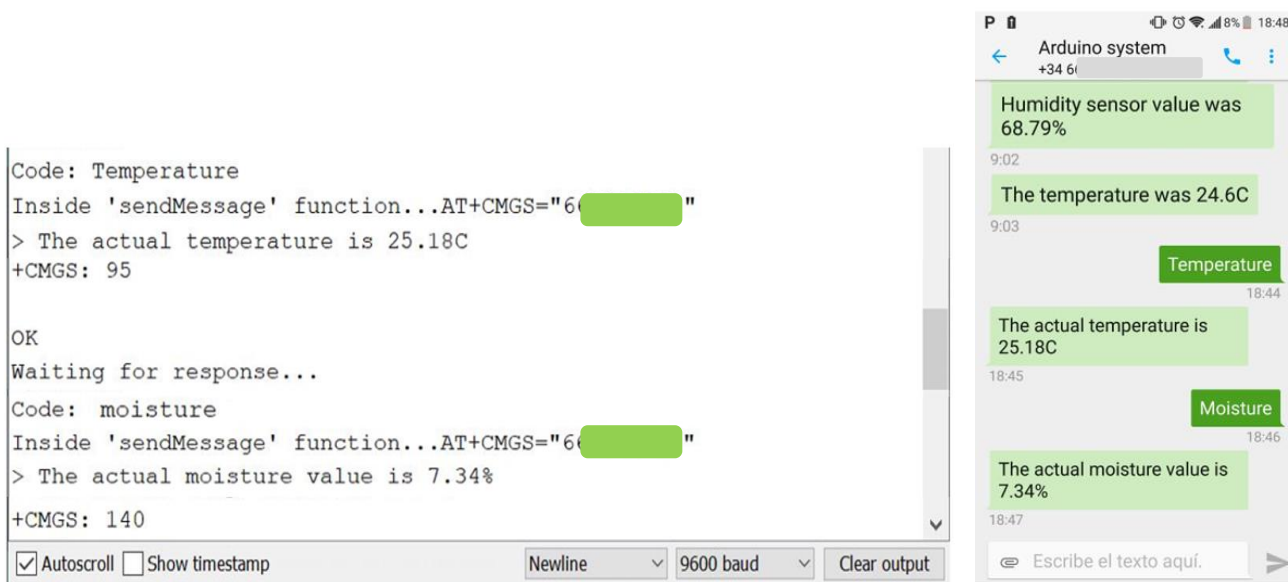


Figure 29. Arduino Serial Port Communication: Demo - Asking for a particular sensor's information.

- **Alerts:** The device is provided with an alert functionality, in case any value measured goes out of a defined range. Since only the Temperature sensor is available, it has been only implemented for temperature. The alerts can be easily added for the rest of sensors, but having an alert for a random value seems not to be the best option for regular performance, receiving alerts SMS all the time was risked. An experiment was made bringing the sensor closer to a flame to receive the real alert. Then, the user has to answer with an ACK message, to ensure that he or she is aware of the situation. Figure 30 displays the result of this functionality.



Figure 30. Arduino Serial Port Communication: Demo - Alerts

5.2 Achievements

- ✓ Ask the user which sensors are connected.
- ✓ Ask the user to define a frequency for the regular notification.
- ✓ Get back the data at a particular time already passed.
- ✓ Retrieve data at the current time.
- ✓ Alert the user when a parameter goes out of a certain range.
- ❖ Obtaining all the data

All but one feature have been implemented directly (❖). Obtaining all the data on the phone seemed to be unsuitable since the communication method is by SMS means. However, this feature is accomplished by extracting the micro SD. Opening the file from the computer is possible and allows to revise the log; therefore, the activity can still be performed.

A check of the requirements achieved must also be carried out before concluding: It is verified that all of them have been achieved in the implementation of this prototype.

5.3 Conclusions

At this point, an assessment can be made of the performance and usefulness of the system. From the demo, some conclusions can be drawn. It is verified how the prototype meets specifications and behaves as desired. However, it should be noted that in terms of being user-friendly, the system has some disadvantages. It is known that the communication method may not be the most appropriate, firstly because of the cost involved. Besides, it is also noted that the sending and receiving of SMS sometimes implies a delay that is too long (up to 12 seconds); therefore, the process is not automatic. On the other hand, a limitation of the system has been the fact that it is not possible to have multitasking in Arduino.

Multitasking refers to the possibility of timing tasks in a non-blocking way. That is, to execute one or more tasks at the same time, without this meaning that nothing else can be done. In Arduino, it is necessary to implement this functionality through programming since it does not support parallelism. In the system, the aim is to prevent a task from not being executed due to a delay. For instance, this problem may occur if the system has not finished a task when another activity must be performed. An activity (Sending messages, saving data, etc.) may be blocked because of this; it must be considered.

These limitations, which appeared when performing the tests, have been taken into account when designing the system. As an alternative, an attempt was made by creating a database. Dumping the information there and providing the user access to it would have solved the problem of communication delays. However, with the SIM808 module, it could not be achieved. One of the solutions adopted, to avoid these constraints, was to minimize the number of messages sent and received and to plan standard answers. Besides, it has been tried that the main functions continue working, they do not get blocked, even if a delay occurs; this was a vital milestone to achieve since the main functionalities depend on frequency.

6 FURTHER INVESTIGATION AND FUTURE IMPROVEMENTS

This system has the advantage to be able to move into a future version with many improvements. An investigation has been made in this direction. In the present chapter, first, a description of improvements for the resulting system is made. Afterwards, the results of the in-depth investigation made are presented.

6.1 System improvements

6.1.1 Adding more sensors

Having only five sensors can, at any given time, limit activity. In this prototype, five were chosen because what mattered was not the number of parameters to be observed but the design of the whole. However, with Arduino MEGA, it is possible to include many more sensors, taking into account the number of pins available.

6.1.2 Adding another device

It could be the case that a vast area is involved and that more than one device is wanted for control. To include more than one device and implement the intercommunication between them, would be a significant improvement to be made. It could be implemented by adding another module to the current system, which would allow remote communication.

6.1.3 Security Checks

Using a 3G module for communication makes the device susceptible to receiving messages from any phone number. A solution would be to start communication oppositely, i.e. the user would identify himself instead of having preprogrammed the phone number. From that moment on, it should be foreseen to only accept messages from the first phone number that sent a message.

6.1.4 Change phone number while working

One possibility for improvement regarding the phone number which could be easily implemented would be to offer the possibility of changing the phone number at runtime. Perhaps this improvement would not be much used, but it would allow the device to gain more flexibility.

6.1.5 User Interface

If the system is improved by the addition of a database, it is possible to ameliorate the User Interface by creating a dedicated platform: a web page or an app for the mobile phone; where data could be displayed. Also, interaction with the system could be made using the new interface; the result would be a more userfriendly system.

6.1.6 Change range

In this case, only the temperature sensor was available, for this reason the range of admissible values was defined by default. Nevertheless, an improvement would be to allow the user to decide the threshold from which an alert will be sent for each sensor.

6.2 New approaches

Since the objective was to create a prototype similar to a real data logger, a more in-depth study has been carried out. Thus, the results of this study are shown below as complementary information. This new information gives ideas for implementing a more professional prototype.

6.2.1 New choices for each Module

This section examines additional components that could be used to implement the new system, with its characteristics. The layout used so far (recall Figure 1. System description.) can continue to be used. Therefore, it is possible to talk about the same modules.

6.2.1.1 Sensors.

Some companies are specially orientated towards agriculture and crops caring. Therefore, an investigation was made in order to find more suitable and professional sensors. The study focused on two companies, Meter and Veinasa. Dataloggers are found in both companies. Veinasa company was chosen for the analysis of its sensors because there was more information available to collect.

6.2.1.1.1 Veinasa Company Sensors.

These sensors were thoroughly analysed to use them as part of a future solution. The parameters of the sensors described below are the same as those of the created system. However, it has been sought to study one more parameter: the wind speed, in order to complete the capacity of five sensors defined in the original system.

- TR-TS01 Soil moisture sensor:

It has been already explained that Soil moisture sensors measure the volumetric water content in the soil. In the case of this sensor, the process is made by an indirect calculation. It uses some property of the soil, such as electrical resistance, dielectric constant, or interaction with neutrons, as a proxy for the moisture content. With high sensitivity and stable performance, Veinasa's sensor provides information for the occurrence, evolution, improvement and dynamics of saline soil. The measurement provides the percentage of soil moisture, meeting current international standards. Additional characteristics are:

- The electrode adopts specially treated alloy material, which can withstand a strong external impact and is not easy to damage.
- Completely sealed, resistant to acid and alkali corrosion, can be buried in the soil or directly into the water for long-term dynamic detection.
- High precision, fast response, good interchangeability, probe plug-in design to ensure accurate measurement and reliable performance.
- Perfect protection circuit.

Some specific characteristics can be observed in Table 15, as well as the aspect of the sensor in Illustration 28. In conclusion, this sensor seems to meet the requirements for excellent performance in the environment where the system is placed.



Illustration 28. Veinasa Soil Moisture Sensor [13]

| | |
|-----------------------------------|---|
| Signal output | RS485, 4-20 mA |
| Power Supply | DC5-24V |
| Static power consumption | 6mA@24V |
| Soil moisture Measurement | Optional Range: 0-100% Resolution: 0.01% Accuracy: 5% |
| Principle and mode of measurement | FDR Method for soil Moisture. A direct test of soil in situ insertion or immersion in culture medium and water-fertilizer integrated nutrient solution |
| Protection Level | IP68 |
| Working environment | -40~85°C |

Table 15. Soil Moisture Sensor Characteristics

- Veinasa-TW Soil temperature sensor:

Earlier, it has been explained that soil temperature sensors come in a variety of designs using thermistors, thermocouples, thermocouple wires, and averaging thermocouples. The electrical signals transmitted from the sensors can be converted to different units of measurement, including °C, °F, and °K. In this case, Veinasa sensor adopts a high precision thermistor as an induction component, which has the characteristics of high measuring precision and optimal stability. The signal transmitter adopts an advanced integrated circuit module, which can convert the temperature into the corresponding voltage or current signal according to the different needs of the user. Among the characteristics of this sensor, it is found that it is compact, convenient to install and portable, reliable in performance, with a special circuit, it has a good linearity, strong load capacity, long transmission distance and strong anti-interference ability. Table 16 gathers characteristics and parameters of this sensor.



Illustration 29. Soil Temperature Sensor [16]

| | | |
|---------------------------|---|----------------|
| Measuring Range | -50~100°C | |
| Accuracy | ±0.5°C | |
| Power supply mode | DC 2.5V DC 5V DC 12V DC 24V Other | |
| Output form | Current | 4~20mA |
| | Voltage | 0~2.5V |
| | Voltage | 0~5V |
| | RS485 | |
| | Other | |
| Load Capacity | Current-mode output impedance $\leq 300 \Omega$ | |
| | Voltage-mode output impedance $\geq 1K\Omega$ | |
| Work environment | Temperature | -50°C~80°C |
| | Humidity | $\leq 100\%RH$ |
| Product power consumption | 0.5mW | |

Table 16. Soil Temperature Sensor Characteristics

- Veinasa-THP Air pressure, temperature and humidity sensor:

In this case, Veinasa's sensor uses a piezoresistive silicon micro-sensor for measuring atmospheric pressure. A capacitive polymer moisture sensor allows the humidity sensing part, whereas the temperature sensing parts of a gap material using a temperature measurement device. The information that regards the remaining characteristics are gathered in Table 17. It is found that these sensors have high accuracy, rapid response, anti-interference ability, good stability, among others.

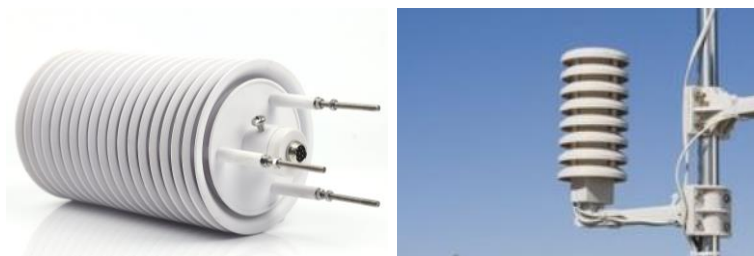


Illustration 30. Air pressure, temperature and humidity Sensor [14]

| | | |
|-----------------------|------------------------|--------------|
| Atmospheric pressure | Measuring range | 10 ~ 1100hPa |
| | Relative accuracy | ± 0.3hPa |
| | Resolution | 0.1hPa |
| Temperature | Measuring range | -50 ~ 100 °C |
| | Resolution | 0.1 °C |
| | Accuracy | ± 0.3 °C |
| Humidity | Measuring range | 0 ~ 100% RH |
| | Resolution | 0.1% RH |
| | Accuracy | ± 2% RH |
| Power Supply | DC 5V | |
| | DC 12V | |
| | DC 24V | |
| Signal Output | RS232 | |
| | RS485 | |
| Operating Environment | Temperature -50°C~80°C | |
| | Humidity≤100%RH | |
| Protection grade: | IP45 | |

Table 17. Air temperature, Pressure and Humidity sensor characteristics

- Wind Speed Sensor:

The last but not least is wind direction and speed sensor. This one is a kind of physical device that detects and senses the wind direction information. It works by the rotation of the arrow of the wind direction, and transmits it to the coaxial code plate; at the same time, it outputs the relevant values of the wind direction. Veinasa's wind speed sensor uses a three-cup wind speed sensor structure, whose structure is made with carbon fibre material. Several uses of this sensor among agricultural areas are found.

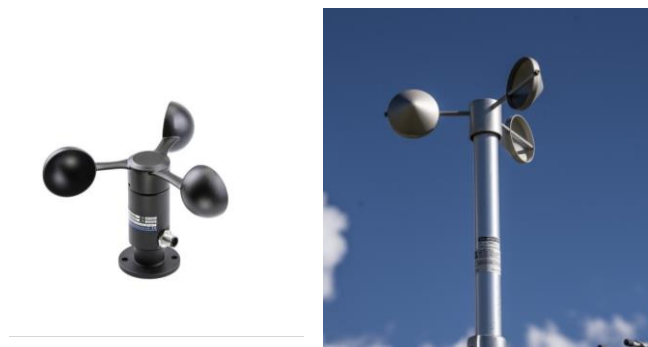


Illustration 31. Wind Speed Sensor [15]

| | |
|---------------------|------------------------------------|
| Measurement range | 0~45m/s; 0~70m/s |
| start-up wind speed | $\pm(0.3+0.03V)m/s$ (V-wind speed) |
| Resolution | 0.1m/s |
| Power supply | DC5V, DC 12V, DC24V, Other |
| Signal output | 4-20mA, 0~5V, RS485, RS232 |
| Load resistance | Voltage type: $RL \geq 1K\Omega$ |
| | Current mode: $RL \leq 600\Omega$ |
| Working temperature | -40°C~50°C |
| Relative humidity | 0~100%RH |
| Power consumption | 50 mW |

Table 18. Wind Speed Sensor Characteristics

6.2.1.1.1 Sensor's protocol

In order to integrate all these sensors into a system, the common characteristic among them should be used. As regards to communication and power supply, all of them count with RS485 protocol as well as 5V signal. In order to apply this specific communication protocol, a far-reaching study of the protocol is required, it can be found in section 7.1.2. This choice also affects the processing unit since it must understand the information provided by the sensors.

6.2.1.2 Processing Unit

Following on from what was described immediately above, it can be assumed that a new processing unit must understand and communicate with the sensors using the same protocol, RS485. This time, instead of using a general-purpose microcontroller, a great solution would be to implement a specific solution. Integration of all the components must be made, but also an allocation of memory and dimensioning battery. These will be the milestones to achieve when designing the new and specific PU. In this way, former extra-modules such as SD adapter could be avoided, prioritizing the integration of components.

6.2.1.3 Communication

6.2.1.3.1.1 Communication channel

Including a more efficient communication system without cost for the user will be an essential consideration to make. In this direction, information is gathered from various possibilities, for example, communicating with a database hosted on the internet, for which a wifi connection would be required; Bluetooth transmission, among others. In particular, a promising option was found when getting information about **LoRa** [10].

IOT's wireless technology has been influenced by the emergence of low-power, high-range networks like LoRa. This is a wireless technology like WiFi or Bluetooth, which uses a type of radiofrequency modulation called Chirp Spread Spectrum (CSS). This modulation technology has been used in military and space communications for decades. There are several advantages found in LoRa for communication such as security, bidirectional communication, high tolerance to interferences, high sensitivity to receive data (-168dB), low power consumption, long-range 10-20 km, point to point connection, among others. In turn, LoRaWAN is a network protocol that uses LoRa technology for low power and wide area networks. The extensive coverage range would allow the design of a system that, through the use of this technology, could communicate successfully.

6.2.1.3.1.2 User interface

In the case of the user interface, it should change accordingly to the choice made above. Thus, if the database option was taken, a web page could be created to show the data, or simply allow the user to access the platform where the data is located. Another option could be to create an application for the phone. The real goal is to make the user experience simpler and more accessible than the one implemented in the prototype.

6.2.2 Hardware wiring and implementation

By investigating these new options, new ways of implementing the system have also been considered. The fact of having new components involves a reconsideration of some aspects. The following are new possibilities raised during the study for future improvements.

6.2.2.1 Sensors identification.

As regards the connection of the sensors to the system, these are supposed to be connected and disconnected indistinctly. Identification is needed so that Processing Unit can have that versatility of connecting sensors indistinctly, which continues to be a requirement in this research. Right now, two options are explained, their advantages and disadvantages.

6.2.2.1.1 Keyboard and screen

Providing the system with a keyboard and screen allows the user to provide the required information for the sensor's identification. In this case, the procedure would be similar to the one proposed in the prototype of this work.

Comparing this system with the current one, it would be much more versatile and perhaps more straightforward, without having to take care of the number of messages to make it minimal. However, it implies that the user must be physically present where the device is when this configuration is made. Besides, a module that allows wireless communication must also be included to transmit the data; therefore, the product would be more expensive.

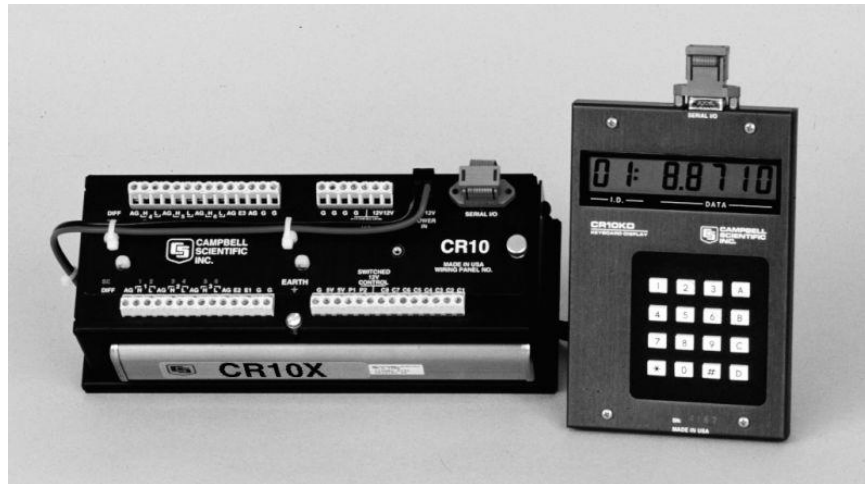


Illustration 32. A real datalogger with keypad [22]

6.2.2.1.2 Switch

An efficient and highly versatile way of performing sensor identification would be to carry out this identification using electronics. A potential implementation is detailed in Figure 31. It is shown how the sensor connected would first be identified through a switch, that will contain an identification, understandable by the Processing Unit. Then, employing shift registers, the information identifying each sensor connected would reach the Processing Unit. The shift register will have parallel and serial input and serial output; an example of this is the 74LS165, depicted in Figure 31. When operating, the sensors will send their information in the same order as the unit receives the identification numbers, in this way, the PU would always know to which of them corresponds the information received. This implementation will require a lower level of electronic design. For example, it is assumed that this part would be attached via a HUB to the Processing Unit.

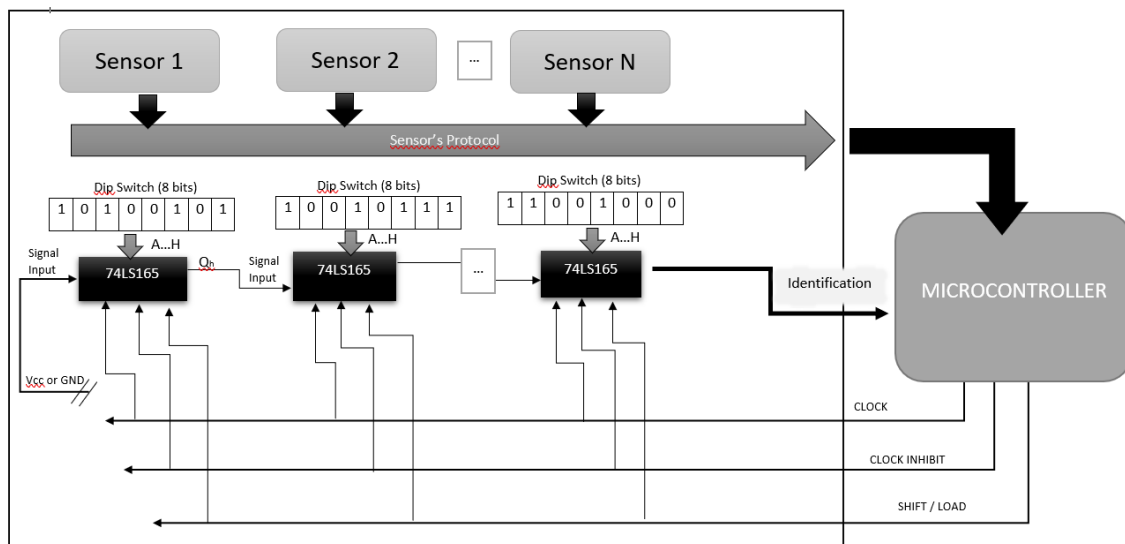


Figure 31. Switch implementation of the system

This option has excellent advantages, e.g. in case of failure or restart, the identification can be made automatically; furthermore, no user intervention in the process is required.

6.2.2.2 Electronic design

The fact of trying to implement a custom solution, entails that the coupling of all the parts would be done by means of a Printed Circuit Board (PCB), The design process of a PCB generally goes through 4 phases, search and design, schematic capture and simulation, board layout, verifying. At the end of this process, the board will be ready for printing. The possibilities in terms of programs that exist in the market for the design of PCBs have also been investigated; some of them are: KiCad, Multism, EasyEDA, Altium, EAGLE, DipTrace. They vary in the interface, price, etc. Concretely, during the development of the research, the use of PCB implementation programs was recalled through EAGLE in its free version.

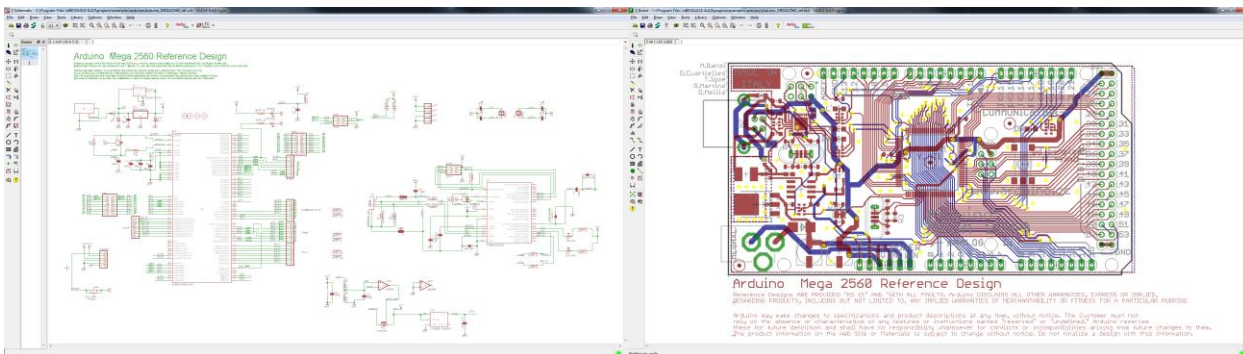


Illustration 33. Example of a PCB implementation in Eagle.

7 ANNEX

7.1 Introduction to Serial Communication

In this section, information about Serial communication is provided. This communication is used when exchanging information between Arduino and the computer, but also between the SIM808 module and Arduino. There are Serial communication protocols, in particular, details about the RS485 and Modbus are included since the last section's sensors use this protocol.

In this section, some words are underlined, a box follows with a concrete explanation. This way, a better comprehension of the descriptions can be reached.

7.1.1 Synchronous versus Asynchronous Serial communication

The main characteristic of Serial Communication is that the data is managed in series. When the transmitter and the receiver cannot have the same clock signal, since it is not possible to send the clock signal over the communication line, asynchronous communication is performed. In that case, both transmitter and receiver have the same signal frequency, (Some possible values have been standardized: 110, 300, 600, 1200, 2400, 4800, 9600, 19 200, 57 600, 115 200 Hz) but different clock signal. Provided that the speed of the communication is highly related with frequency, because with each clock pulse a bit of information is transmitted, speed is measured on Bits Per Seconds or Baudies; the transmission period is measured in *Bit Time* (T_b).

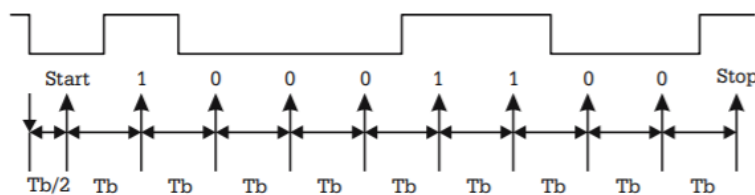


Illustration 34. Sampling data line in an asynchronous serial communication [23]

Once asynchronous communication has been established, it is essential to define the beginning of the transmission sequence. For that purpose, a *start bit* is introduced on the sequence. The start bit is none other than a synchronization bit, it has the polarity opposed to the idle line status, and its duration equals one T_b . On the other hand, the moment when transmission finishes is recognized because the length of the sequence is known in advance, usually 8 bits packets. An additional bit is transmitted to make sure that the communication line returns to the idle status after the packet reception, the *stop bit*. It has the idle value of the line. The first bit transmitted is the Least Significant Bit (LSB), the idle value of the Serial Line is HIGH, regarding what has been explained, the start bit has a LOW value as it is appreciated in Illustration 34.

7.1.1.1 Error Detection

When using asynchronous communication, error detection is crucial. Hardware and software solutions have been implemented in order to solve errors. On the first hand, a hardware solution is, for instance, parity control, where an extra bit is transmitted to grant the integrity of the data frame. It could be either the LSB or a ninth additional bit associated to the byte. The number of ones in the data frame is an odd or even number according to the parity chosen, completed with the parity bit. On the other right hand, software solutions could be packets with a preset structure as well as a reserved field for checksum control such as Cyclic Redundancy Check Control (CRC).

7.1.2 RS-485

A standard of bus communications of the Open Systems Interconnection (OSI) model is the RS485; it is placed on the Physical Layer.

Physical Layer of OSI model is the first and lowest one. It defines the topology of the network and the global connections of the computer to the network. Information about the physical mean used and the way of transmitting the data is gathered in this layer. The physical layer must contain details about:

- *The characteristics of materials (components) but also the electrical ones (Volts values, etc.) which must be used on the transmission of the data by these physical channels.*
- *Functional aspects of the interface, such as the establishment, maintenance, and hard link freeing.*

Hard link: is a directory entry that associates a name with a file on a file system. All directory-based file systems must have at least one hard link giving the original name for each file.

File system: Operative system's component in charge of administrating and facilitate peripheral memories' usage.

- *How the bits flow is transmitted in the physical mean.*
- *Electrical signal managing in the physical mean.*
- *In charge of granting the connection.*

RS485 uses a twisted pair cable whose maximum length is 1200 meters transmitting in this case 300 bits per second, while the maximum speed is 10 Mbits/s for a length of 12 meters. Characterized for the differential interface; the signal travels through two cables so-called A and B, one is the inverted version of the other, which allows an excellent interference rejection. A graphic explanation is given in Figure 32 about differential signal and interference rejection. It is seen how the difference between both signals remains the same, whether there is interference or not. Besides, RS485 uses multipoint connection with a total top number of 32 stations.

In a multipoint network, each communication channel can be used to contact different nodes therefore, only one line of communication exists which use is shared by all the terminals.

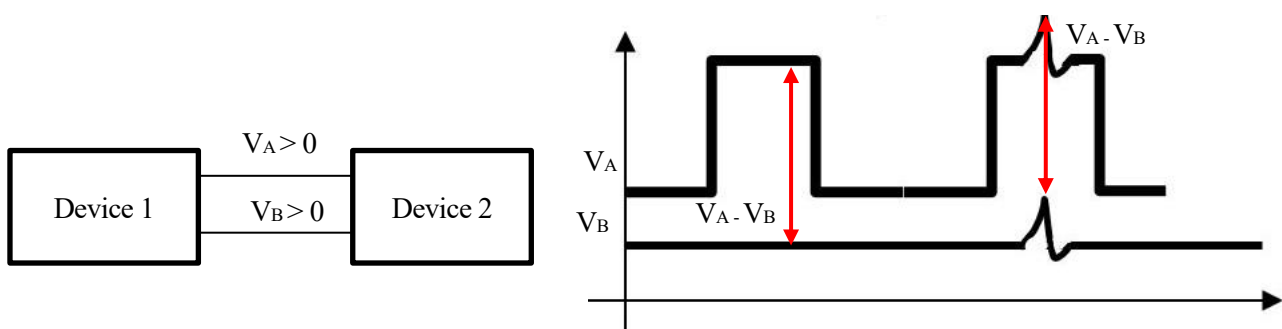


Figure 32. RS485 Interference rejection because of differential signal.

Among the electrical characteristics, it is found that the network uses 5 volts, a pull-up resistor may be added, but it is not compulsory if the distances are short, the bus value ranges from -7V till 12V. Also, for the receptors to identify the two possible logic states of line A and B correctly, the difference between both must be at least 0.2 V.

7.1.2.1 Data transfer

Once the method is defined, it is also worth to mention the way data is managed in this standard. The information in each packet is codified in ASCII. All data transfer is initiated by the master because this protocol is based in a master-slave configuration. Thus, it is the master who sends an interrogation or individual command packets to the slaves. Each slave has a unique address. More details about the data package are given in the following section.

7.1.3 MODBUS

Modbus is a protocol with a request-response workflow using a master-slave configuration. Two versions of this protocols are standardized: RTU, TCP, ASCII Modbus where the data model and calls to methods are identical, just the encapsulation is different; and Eron, Pemex, Plus, UDP, Modbus, which have non-interoperable variations. On the OSI model, this standard is placed on the application layer. While RS485 is the physical mean, the Modbus explains how the information is distributed.

The Application layer offers the applications the possibility to access the other layers' services, and it defines the protocols that are used by those applications to exchange data. The number of protocols is as many as applications can be found. Usually, the user does not interact directly with this layer.

7.1.3.1 Specifications

Modbus frame consists of an Application Data Unit (ADU), which encapsulates a Protocol Data Unit (PDU).

The Protocol Data Unit defines basic concepts of the access and data handling; it is conformed by the data and the function code. Moreover, it is considered the core of the specifications of the Modbus protocol, and its full size must not be bigger than 253 bytes.

- The **function code** has flexible behaviour that the slaves can implement based on the application desired. It defines the kind of request that is made to the slave, codified by a number going from 1 until 127, among which public functions, user functions and reserved functions are defined.
- The **Data**:

All the data is stored in one of the four data or direction benches, detailed in Table 19, they define the type and access rights of the data contained in these blocks. Additionally, the bench selected provides the ability to restrict or allow access to different types of data.

| Object type | Access |
|------------------|-----------------|
| Discrete input | Just reading |
| Coil | Reading/Writing |
| Input register | Just reading |
| Holding register | Reading/Writing |

Table 19. Object benches Types on Modbus

Each block contains a memory space, maximum 65536 items. The address range of these elements starts at 0 and ends at 65535; however, each data element is numbered from 1 to 65536. Each data frame has a preset configuration. Any frame coming from the master can be distinguished from one coming from the slave because of its configuration.

- Master Data Frame: Table 20 displays how a data frame coming from the Master is made up. Below, an explanation of each field is gathered.

| Byte 01 | Byte 02 | Byte 03-04 | Byte 05-06 | Byte 07-08 |
|----------|----------|------------|------------|------------|
| Slave ID | Function | Address | Length | CRC |

Table 20. Master Data frame in RS485-ModBus Communication

Slave ID: Indicates the device to which the frame is addressed.

Function: Action to be performed.

Address: Where to start looking for the information requested through the function.

Length: From the address onwards, how many bytes should be taken.

CRC: Modbus Checksum.

- Slave Data Frame: Table 21 shows the data frame of the Slave. The main difference between the Master's data frame and Slave's is that the slave frame has a more significant number of bytes, according to the quantity of information requested.

| | | | | |
|----------|----------|------------|-------------|--------------|
| Byte 01 | Byte 02 | Byte 03 | Byte 04...n | Byte n+1-n+2 |
| Slave ID | Function | # of Bytes | Response | CRC |

Table 21. Slave Data frame in RS485-ModBus Communication

The Application Data Unit (ADU) includes the PDU, and a different ADU exists depending on the network protocol, the three standard formats are TCP, RTU, ASCII.

REFERENCE

- [1] Learn OpenEnergyMonitor, “Learn OpenEnergyMonitor,” [Online]. Available: <https://learn.openenergymonitor.org/electricity-monitoring/temperature/DS18B20-temperature-sensing>. [Accessed 04 2020].
- [2] L. d. V. Hernández, “Porgramar Facil,” [Online]. Available: <https://programarfacil.com/blog/arduino-blog/ds18b20-sensor-temperatura-arduino/>. [Accessed 2020].
- [3] “Arduino,” Arduino, 2020. [Online]. Available: <https://www.arduino.cc/en/Guide/Introduction>. [Accessed April 2020].
- [4] “ACOPTEX,” ACOPTEX, 11 January 2018. [Online]. Available: <http://acoptex.com/project/264/basics-project-053d-sim808-gsm-gprs-gps-bluetooth-evolution-board-evb-v32-at-acoptexcom/#sthash.XoACMLoR.dpbs>. [Accessed April 2020].
- [5] “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Hayes_command_set.
- [6] S. S. W. S. Ltd., “Shangai SIMCom Wireless Solution Ltd.,” Shangai SIMCom Wireless Solution Ltd., 24 December 2010. [Online]. Available: http://www.espruino.com/datasheets/SIM900_AT.pdf. [Accessed 2020].
- [7] Arduino, “Arduino,” Arduino, [Online]. Available: <https://store.arduino.cc/arduino-nano>. [Accessed 05 2020].
- [8] Arduino, “Arduino,” Arduino, [Online]. Available: <https://store.arduino.cc/arduino-mega-2560-rev3>. [Accessed 05 2020].
- [9] Prometec, “Prometec,” [Online]. Available: <https://www.prometec.net/time-arduino/>.
- [10] CatSensors, “CatSensors,” CatSensors, [Online]. Available: <https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan>. [Accessed 2020].
- [11] R. S. S. a. A. G. Barto, Reinforcement Learning: an introduction, 2018.]
- [12] “Components101,” 7 May 2018. [Online]. Available: <https://components101.com/sensors/ds18b20-temperature-sensor>. [Accessed 2020].]
- [13] “AprendiendoArduino,” 28 June 2016. [Online]. Available: <https://aprendiendoarduino.wordpress.com/tag/dht/>. [Accessed 2020].]
- [14] “Makerfabs,” [Online]. Available: <https://www.makerfabs.com/soil-moisture-sensor.html>. [Accessed 2020].]
- [15] Sunny, your only power source, “Sunny,” [Online]. Available: <https://www.sunny-euro.com/es/productos/sys1308-2412-w2e-europe-2-1x5-5x11-s-rc-1-8m-6ft>. [Accessed 2020].]
- [16] B. Smith, “Campbell Scientific,” 9 January 2019. [Online]. Available: <https://www.campbellsci.com/blog/keypad-commands-edlog-data-loggers>. [Accessed 2020].]

- [17 I. Susnea and M. Mitescu, "Using the Asynchronous Serial Interface," in *Microcontrollers in Practice*, Springer, 2005, p. 262.
- [18 Alibaba, "Alibaba," [Online]. Available: https://www.alibaba.com/product-detail/Veinasa-TWSD-Integrated-Agricultural-Systems-Moisture_62320339342.html.
- [19 "Alibaba," [Online]. Available: https://weinasa.en.alibaba.com/product/62103713348-807301649/Veinasa_THP_rh_Sensor_Temperature_Pressure_Sensor_Meteo_Station_Outdoor_Sensor.html. [Accessed 2020].
- [20 "Alibaba," [Online]. Available: https://weinasa.en.alibaba.com/product/62032893887-807301649/Veinasa_FS_Digital_Zigbee_Wind_Speed_Sensor_Forest_Anemometer_Output_RS485.html. [Accessed 2020].
- [21 "indiaMart," indiaMart, [Online]. Available: <https://www.indiamart.com/proddetail/sd-card-module-20741748862.html>. [Accessed 2020].
- [22 SIMCom, 16 January 2014. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/SIM808_GPS_Application_Note_V1.00.pdf. [Accessed 2020].
- [23 Alibaba, "Alibaba," [Online]. Available: https://weinasa.en.alibaba.com/product/62165350293-814176540/Veinasa_TW_Temperature_Sensor_iot_Agriculture_Soil_Temperature_Sensor.html.

