SIMON BOLIVAR UNIVERSITY

DECANATE OF PROFESSIONAL STUDIES

ENGINEERING COORDINATION OF TELECOMMUNICATIONS

AUTOMATED DRIVE TEST THROUGH ESP8266 AND THE GSM MODULE

By:

Br. Luis Alfredo Muñoz Molina

GRADUATION PROJECT

Presented at the Illustrious Simón Bolívar University

as a partial requirement to qualify for the title of

Telecommunication engineer

Sartenejas, July 2017

SIMON BOLIVAR UNIVERSITY

DECANATE OF PROFESSIONAL STUDIES
ENGINEERING COORDINATION OF TELECOMMUNICATIONS

AUTOMATED DRIVE TEST THROUGH ESP8266 AND THE GSM MODULE

By:

Br. Luis Alfredo Muñoz Molina

Made with the advice of:

Prof. Miguel Díaz

GRADUATION PROJECT
Presented at the Illustrious Simón Bolívar University
as a partial requirement to qualify for the title of
Telecommunication engineer

Sartenejas, July 2017

SIMON BOLIVAR UNIVERSITY
DECANATE OF PROFESSIONAL STUDIES
ENGINEERING COORDINATION OF TELECOMMUNICATIONS

AUTOMATED DRIVE TEST THROUGH ESP8266 AND THE GSM MODULE

GRADUATION PROJECT
Made by: Br. Luis Alfredo Muñoz Molina
With the advice of: Prof. Miguel Díaz

RESUME

A Drive Test system is one that is responsible for carrying out radio frequency measurements to verify the status of a cellular network. In the present work a Drive Test system is implemented that is based on the GSM SIM900 module and the XM37-1612 GPS, where the D1 mini development board is used for its control and configuration. To do this, the characteristics of these devices are studied, the different AT commands that the GSM module has are compared and finally a system proposal based on previous studies is presented. A C ++ library was created to implement the proposed system, and a test run was carried out that measured signal level, the quantity and location of missed calls, the quality of the calls made, the absolute radio frequency number of the channel. diffusion control, the country code, the network code, the identity code of the base station, the identifier of the cell, the minimum level of access to the cell, the maximum transmission power of the mobile, the location area code, timing advance and bit error rate. The results obtained from the tour showed that with the proposed system it is feasible to implement a simple Drive Test system, due to the lack of functionalities of a complete Drive Test.

Keywords: Drive Test, Base Station, SIM900, ESP8266, GPS.

GENERAL INDEX

LIST OF FIGURES

LIST OF TABLES

List of abbreviations

API	Application programming interface.
ARFCN	Absolute radio-frequency channel number Absolute radio-frequency channel numbers.
AT	ATtention Attention.
AuC	Authentication Center Authentication center.
BCCH	Broadcast Control Channel Broadcast Control Channel.
BER	Bit Error Rate Bit error rate.
BLER	Block Error Rate Block error rate.
BSC	Base Station Controller Base Station Controller.
BSIC	Base Station Identity Code Base station identity code.
BSS	Base Station Subsystem Base Station Subsystem.
Bts	Base Transceiver Station Base station.
CRC	Cyclic Redundancy Check Cyclic redundancy check.
CS	Coding Schemes Coding schemes.
DTX	Discontinuous Transmission Discontinuous transmission.
EFR	Enhanced Full Rate Improved full rate.
AND GO	Equipment Identity Register Registration of the identity of the team.

ETSI	European Telecommunications Standards Institute European Telecommunications Standards Institute.
FR	Full rate Full rate.
GMSC	Gateway Mobile Switching Center Gateway Mobile Switching Center.
GPIO	General Purpose Input / Output General purpose entry and exit.
GPRS	General Packet Radio Service General packet service via radio.
GPS	Global Positioning System Global Position System.
GSM	Global System for Mobile communications Global System for mobile communications.
HLR	Home Location Register Base location record.
HR	Half rate Average rate.
IMSI	International Mobile Subscriber Identity International Mobile Subscriber Identity.
ISDN	Integrated Services Digital Network Integrated services digital network.
LMSI	Local Mobile Subscriber Identity Local mobile station identifier.
M2M	Machine to Machine Machine to machine.
MCC	Mobile Country Code Mobile Country Code.
MNC	Mobile Network Code Mobile Operator Code.
MOC	Mobile-Originated Call Call originated by mobile.
MSC	Mobile Switching Center Central Mobile Switching.

MSIC	International Mobile Subscriber Identity International mobile subscriber identity.
MSRN	Mobile Station Roaming Number Roaming Number of the Mobile Station.
TCM	Mobile-Terminated Call Call completed on a mobile.
PLMN	Public Land Mobile Network Public land mobile network.
PSTN	Public Switched Telephone Network Public switched telephone network.
RF	Radiofrequency.
RMC	Recommended Minimim Speci fi c GNSS Data Minimum speci fi c GNSS data recommended.
RSSI	Received Signal Strength Indicator Indicator of received signal strength.
RTC	Real Time Clock Real time clock.
RXQ	Quality of the received signal.
RXQUAL	Receiver Quality Quality received.
SDK	Software Development Kit Software development kit.
SIM	Subscriber Identity Module Subscriber Identity Module.
SMT	Surface Mount Technology Surface mount technology.
SoC	System on a Chip System on a chip.
SRAM	Static Random Access Memory Static Random Access Memory.
TCP / IP	Transmission Control Protocol / Internet Protocol Transmission Control Protocol / Internet Protocol.
TMSI	Temporary Mobile Subscriber Identity Temporary mobile station identifier.

VLR	Visitor location register Visitor Location Record.
WDT	Watch Dog Timer Guard dog timer.
WPA	WiFi Protected Access WiFi protected access.

INTRODUCTION

The planning of a cellular system is carried out taking into account the capacity and coverage required to provide users with a good service. The process of establishing a cellular network involves much more than the implementation and commissioning of the site, since a process of continuous measurements and adjustments is necessary to optimize the performance of the network and allow users the best possible quality; These measurements are made in road tests over a certain area, better known as Drive Test.

The Drive Test seeks to collect information about the already established cellular network, among these data we have, for example, the power level of the received signal, quality of the signal, cell identifier, bit error rate, etc. In addition to the network parameters, measurements must also be carried out in relation to the calls, observing if the network mechanisms, for these, work correctly.

The equipment used during a Drive Test can be very expensive, however, on the market there are equipment capable of performing similar tasks, among these equipment there is the GSM module (from the English *Global System for Mobile communications)* SIM900 which can be con fi gured by AT commands (from English *ATtention),* This module allows you to make calls, send messages, use data and observe the state of the network, these qualities make it a candidate to be part of the Drive Test system.

The GSM module needs a controller element capable of receiving, processing and sending the information. In a Drive Test tour, several modules are usually used, one for cellular technology (2G, 3G, 4G), or perhaps also one for each operator, it is considered to use an element with WIFI technology as the controlling element of the GSM module, because All the interconnections in this system can become very complicated.

Problem Statement

A Drive Test system must be able to report the coverage, the quality of the service, the rate of successful calls and indicate if failures were found in the relay mechanisms in the system in a given area, however, the complexity of the wiring of the The system increases as the technologies to be measured (2G, 3G, 4G) increase. On the other hand, the ESP8266 is a system on a chip that integrates the TCP / IP protocol stack and is capable of connecting to WiFi networks, allowing the device interact with other elements wirelessly.

Faced with this scenario, is it possible to implement an automated Drive Test system that meets these characteristics and also does not need to be wired, using the GSM SIM900 and ESP8266 module?

Justification

Drive Test systems are of great interest to cell phone companies, due to because it is always sought to optimize and verify the network already deployed. Due to the high cost of specialized equipment to carry out a Drive Test, it is desired to implement a low-cost application to carry out said tests using the ESP8266 module and the GSM module. It is of interest to use a module with wireless technology such as ESP8266 to facilitate the sending of the data obtained by the GSM module to a server, which stores and allows the presentation of the data. Wireless connection is required to reduce the use of connection cables.

Overall objective

Implement an automated drive test based on ESP8266 with the GSM module.

Specific objectives

- Study the fundamentals of a Drive Test, and the parameters of interest must be measured
- Study the characteristics of the ESP8266 and its programming in Arduino format, in C ++.

- Study the fundamentals of GPS and its location and time parameters. Study the
- GSM Modules, and their AT commands, required for the Drive Test. Carry out
- communication routines between the GSM module and the ESP8266. Carry out
- signal quality reporting routines and cell parameters. Make routine voice calls with
- the GSM module.
- Make data call routines with the GSM Module (via GPRS)
- Send the results of the measurements to a server through the WI-FI module.
- Implement the Storage and Presentation of data on a Server.

CHAPTER 1

THEORETICAL FRAMEWORK

1.1. What is a cellular network?

A cellular network is a communication system made up of cells, each with its own transmitter, known as base stations (BTS). *base transceiver station).*
These cells are used to provide radio coverage over an area larger than that of a cell. The BTS allows the user to access the network and use its services (voice and data).

1.1.1. Cell structure

To cover large coverage areas, a large number of cells is needed, each cell uses a set of frequencies, which will serve as physical channels for communication between the BTS and the mobile, however, due to the scarcity of the radio spectrum , a large frequency range cannot be assigned to a cellular operator. Due to the need to cover these coverage areas, the concept of frequency reuse was created.

The total set of frequency channels is divided into N approximately equal groups, and each one of these groups is assigned to a different cell looking for adjacent cells to use different frequency channel groups, this is done to avoid interference. [one]

Each group of N contiguous cells that together use all the available frequencies constitutes a group (from the English *cluster).* The group pattern is repeated periodically over the geographic area to be covered, allowing each channel to be used several times but preventing adjacent cells from using the same channel, as seen in Figure 1.1.

1.1.2. Network planning

If you think of a country with a varied population density such as Venezuela, it is easy to understand why cells of the same size cannot be used throughout the network. It is not the same for an operator to supply coverage to a city with high population density, such as Caracas, for example, to supply an island with low population density, such as Margarita.

Cell division or micro cell applications. Due to the increase in the number of subscribers, the density of users per cell also increases. A fairly basic idea is to divide the existing space into smaller portions, thus multiplying the number of users that can be served with the same set of frequencies (Figure 1.2). Also with this simple scheme the level of power consumed in the cell is reduced.

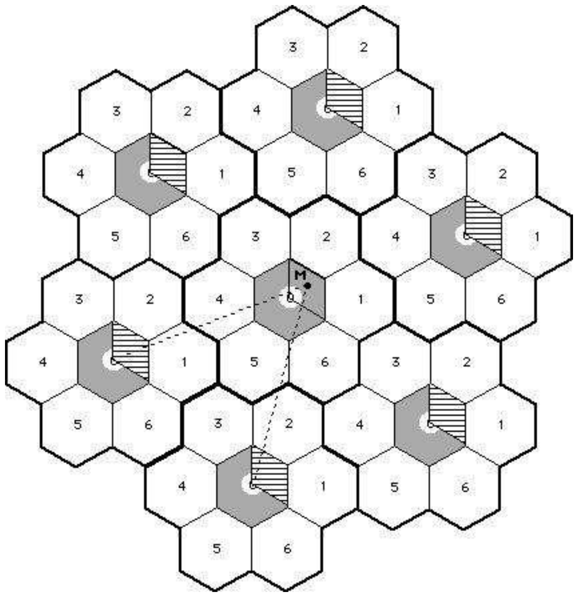

Figure 1.1. Cell structure [2]

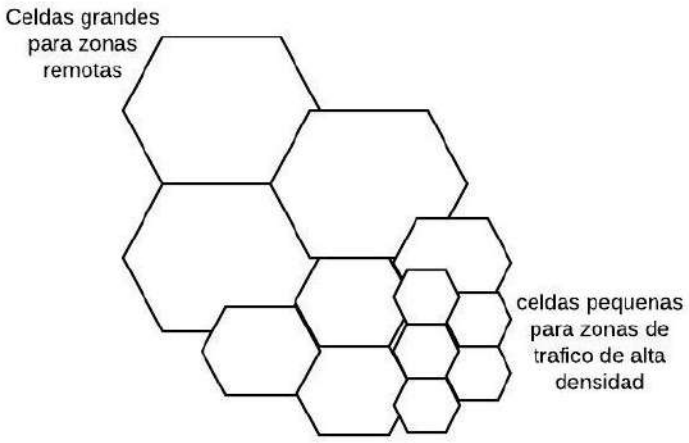
Celdas grandes para zonas remotas
celdas pequenas para zonas de trafico de alta densidad

Figure 1.2. Cell division

Selective or sectorized cells. It is a technique to improve the management of self-interference in a cellular system. In a sectorized cell, the frequency channels assigned to that cell are divided into M (typically 3) approximately equal groups, and each one of them is used on an angular sector of the cell, through directional antennas as seen in the figure 1.3.

Umbrella cells. The first time that cell division was applied, operators realized that on high-speed mobiles a large number of relays were carried out between the smaller cells. Since to carry out each relay more work is done per

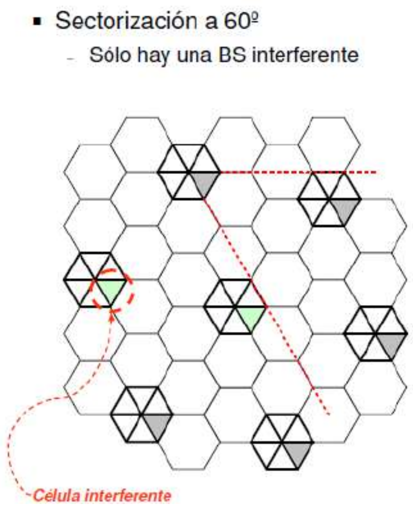

Figure 1.3. Sectorization [3]

part of the network, it is not desirable to increase the number of these events. Umbrella cells (Figure 1.4) were introduced to solve this problem. An umbrella cell transmits at a higher power than micro-cells that already contain a different frequency. In this way, when the network detects a mobile traveling at high speed, it handles it with the umbrella cells instead of the micro-cells. It is detected that the mobile is going at high speed through its propagation characteristics or due to its excessive relays. In this cell the mobile can spend a longer amount of time thus lowering the network load.

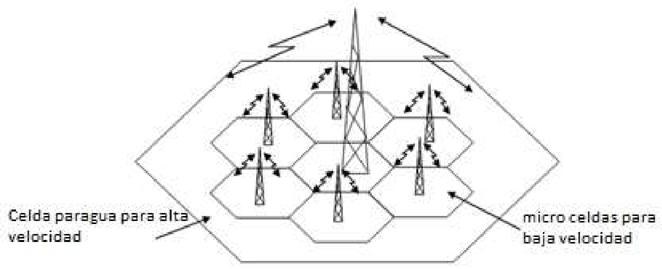
Celda paragua para alta velocidad
micro celdas para baja velocidad

Figure 1.4. Umbrella cell [4]

1.2. GSM, a digital cellular mobile radio system

The global system for GSM mobile communications *Global System Mobile Communications)* is a standard developed by the ETSI (from the English *European Telecommunications Standards Institute)* which is used to describe the protocols of the second generation (2G) digital cellular network. The standard was delivered in phases, the first being a subset of the network characteristics, compatible with updates and aggregation.

of services. The necessary plug-ins for the full implementation of all the services and features of the network were found in the second phase.

1.3. Services offered in a GSM system

The characteristics and benefits of the system were: superior voice quality (equal to or better than analog cellular technology), low cost in the mobile terminal, operations and services, a high level of security (confidentiality and fraud prevention), roaming international (with a single subscriber number), low-power mobile terminal support, and a variety of new network services and facilities.

1.4. System architecture [1]

To provide cellular service to users, cellular operators have to install a complete infrastructure, which at some point has to communicate with the public switched telephone network (PSTN). *Public Switched Telephone Network).* In addition to the standard national roaming feature, the GSM system decided to provide international roaming. This implies that users have the option of using their mobile phone abroad and therefore in an external GSM system.

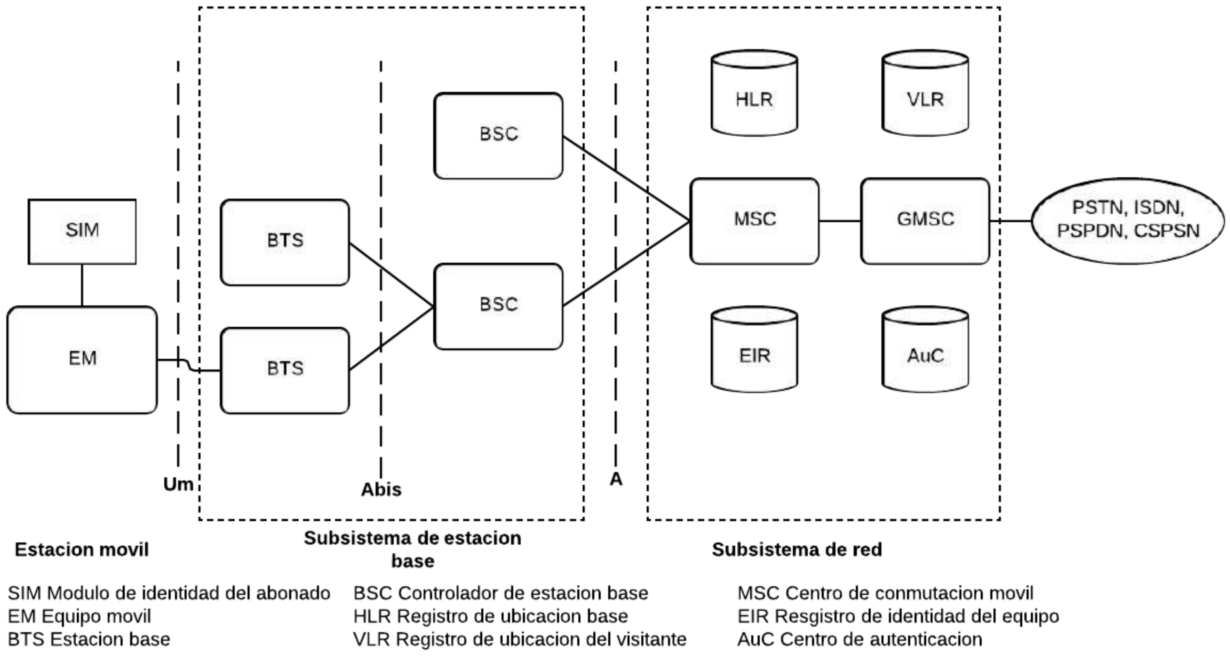
HLR
VLR
BSC
MSC
GMSC
PSTN, ISDN, PSPDN, CSPSN
SIM
BTS
BSC
EM
BTS
EIR
AuC
Um
Abis
A
Estacion movil
Subsistema de estacion base
Subsistema de red
SIM Modulo de identidad del abonado
EM Equipo movil
BTS Estacion base
BSC Controlador de estacion base
HLR Registro de ubicacion base
VLR Registro de ubicacion del visitante
MSC Centro de conmutacion movil
EIR Resgistro de identidad del equipo
AuC Centro de autenticacion

Figure 1.5. GSM architecture

1.4.1. Subscriber or subscriber identity module

The subscriber identity module (SIM in English *subscriber identity module)* provides the mobile with an identification. It is a removable smart card used in mobile phones and GSM modems.

SIM cards securely store the subscriber's service key used to authenticate to the network so that it is possible to change the line from one terminal to another simply by changing the card. Its use is mandatory in GSM networks (except for emergency calls).

1.4.2. Base station

The counter part of the mobile station in a cellular network is the base station (BTS), which is the interface of the mobile with the network. The BTS is usually located in the center of the cell, its transmitting power and antennas define the size of the cell. A base station has a large number of transceivers, each representing a separate RF channel. Part of the intelligence possessed by analog base stations, such as the measurement of RF channels as a criterion of the relay, was moved to mobile stations, to improve the standard.

1.4.3. Mobile switching center

The mobile switching center (MSC) *Mobile Switching Center)* It is a telephone exchange that performs the switching of calls and messages within the network, in case of needing to make a connection outside the network, it is sent to the mobile switching center of
gateway (GMSC from English *Gateway Mobile Switching Center)*

The MSC also manages relays to neighboring base stations, keeps track of the location of mobile subscribers, is responsible for subscriber services and billing.

1.4.4. Base station controller

The Base Station Controller (BSC) *Base Station Control ler)* monitors and controls several BTS, the amount of BTS it can control depends on the manufacturer, it varies from tens to hundreds of BTS. The main task of the BSC is frequency management, BTS control and exchange functions. The BSC may be located on the same site as the BTS, on its own independent site, or with the MSC. The BSC and BTS together form a functional entity sometimes called a base station subsystem (BSS of the
English *Base Station Subsystem).*

1.4.5. Gateway Mobile Switching Center

The gateway mobile switching center (GMSC) is the interface that communicates the network with external networks. When a call comes in from an external network or a call goes out to an external network, the GMSC takes care of routing it.

1.4.6. International identity of the subscriber to a mobile

The international mobile subscriber identity (IMSI) is permanently stored on the SIM card. It is a unique identification code, of fifteen digits, for each mobile phone device in the GSM system. The first three digits of the IMSI (from the English *International Mobile Subscriber Identity)* correspond to the mobile country code (MCC). *mobile country code),* the next two digits correspond to the mobile network code (MNC). *mobile network code).* The remaining ten digits correspond to the Mobile Subscriber Identification Number (MSIC). *mobile subscriber identification number).*
Table 1.1 shows the IMSI format.

Table 1.1. IMSI format

MCC	MNC	MSIC

1.4.7. Location record

The base location record (HLR) *Home Location register)* It is a functional entity in charge of the storage of mobile subscriber data. A Public Land Mobile Network (PLMN) *Public Land Mobile Network)* it can have one or more HLRs, depending on the number of subscribers, the capacity of the equipment and the organization of the network. 2 types of information are stored:

- The subscription information.

- Location information that allows calls to be routed to the MSC where the mobile is located.

Two types of numbers related to the mobile subscription are stored in the HLR:

- International Mobile Station Identity (IMSI).

- One or more ISDN number (s) (from English *Integrated Services Digital Network)* International Mobile Station (MSISDN)

The IMSI or MSISDN will be used as a key to access the information in the database for a mobile subscription.

1.4.8. Visitor location record

Mobile phones are controlled by the visitor's location register (VLR). *Visitor Location Register)* corresponc to the area in which they are located. When a mobile device enters a new location, a registration procedure is initiated. The MSC in charge of the area detects this record and transfers the identity of the location area (where the mobile is located) to the VLR.

The visitor location record (VLR) contains the information necessary to manage the calls established or received by the mobile devices registered in its database, which contains the following elements:

- International Mobile Station Identity (IMSI).
- One or more mobile station international ISDN number (s) (MSISDN).
- The mobile station roaming number (MSRN).
- The temporary mobile station identifier (TMSI). *Temporary Mobile Subscriber Identity).*
- The local mobile station identifier (LMSI). *Local Mobile Subscriber Identity).*
- The location area where the mobile was registered. This data will be used to call the mobile.

1.4.9. Authentication Center

The Authentication Center (AuC) *Authentication Center)* is associated with an HLR, and stores an identity key for each mobile subscriber registered with the associated HLR. The key is used to generate:

- Data used to authenticate the IMSI.
- A key used to encrypt the radio link communication between the mobile and the network.

1.4.10. Team identity registration

The team identity record (EIR) *Equipment Identity Register)* It is a database in which there is information on the status of mobile phones. Within this database there are three IMEI lists: the white, the gray and the black.

- The white list identifies the teams that are authorized to receive and make calls. This list must always exist in the EIR, even if it is the only one; the other two are optional
- The gray list identifies the computers that can make and receive calls, but that can be monitored to discover the identity of the user using the information stored on the SIM card.
- The blacklist identifies computers that are prevented from connecting to the network. Therefore, they cannot make or receive calls.

1.5. Call establishment

Before making a call, the mobile must be registered in the system. There are two types of procedures, one is for the mobile originated call (MOC).
Mobile-Originated Call), and the other is for the call ended on a mobile (MTC from English *Mobile-Terminat* *Call).* The MOC will be described to give an idea of the message exchange used in the GSM system.

Equivalent to the location update procedure, the mobile initiates the procedure with a channel request, which is answered by the system with a channel assignment. The mobile indicates to the system what will be the use of said channel (in this case, establish a call). Before the procedure continues, the mobile must authenticate itself again. To protect any additional signaling messages from a third party, the network can now, with the following message, instruct the mobile to start encrypting the data. Encrypting means that messages are transmitted in a way that only the mobile and BTS understand. In the configuration message, the mobile sends the number you want to call. While the call is being made, the BSC through the BTS assigns a traffic channel, in which the user data exchange takes place. Different types of message and user data move in different types of channels. The different types of channels will be described in the next chapter.

The opposite MTC procedure is almost identical to the MOC.

1.6. Relief

The relief procedure (*Handover)* it is a means to continue a call when the mobile changes cells. Before the introduction of this mechanism, the call was lost when changing cells or when the distance between the mobile and a particular base station was very long.

In a cellular network, a cell has a set of neighboring cells, the mobile continuously monitors the power level received from said cells. To do this, the BTS gives the mobile a list of BTSs (channels) in which to perform power measurements. The list is transmitted on the control channel (again, system information), which is the first channel that the mobile tunes when it is turned on. The mobile makes continuous measurements on the quality and power level of its cell and adjacent cells. The results of these measurements are put into a *measurement report,* which is periodically sent to the BTS. The BTS itself may also be making measurements on the quality and power of the link with the mobile. If these measurements indicate the need for a relay, it can be done without delay, since the BTS for your relay is already known. Measurements arrive periodically and reflect the mobile's point of view. It is up to the operator to act on different levels of quality or power, and relay restrictions or thresholds can be adjusted based on the changing environment and operating conditions. [one]

The GSM system differentiates between different types of relay. Depending on what type of border the mobile is crossing, a different entity can take control of the relay to ensure that a channel is available in the new cell. If a relay takes place within the area of the same BSC, the relay will be controlled by the BSC without consulting the MSC, which at any time

Table 1.2. Call establishment procedure [1]

MS		Bts	Action
	- - - - →		Channel request.
	← - ——–		Channel assignment.
	- - - - →		Channel assignment request
	← - ——–		Authentication request.
	- - - - →		Authentication response.
	← - ——–		Encryption command.
	- - - - →		Encryption completed, from now on messages are encrypted. Con fi
	- - - - →		guration message, indicating the desired number.
	← - ——–		Call procedure, the network routes the call with the desired number. Traffic
	← - ——–		Channel Assignment for Use Designated "User Data Exchange"
	- - - - →		Assignment completed, from now on all messages are exchanged through the traffic channel
	← - ——–		Alerting, the requested number is not busy and the phone is ringing.
	← - ——–		Connection, the destination number accepted the call.
	- - - - →		Connection recognition, now the call is operational and both can talk to the other.
	← - - →		Voice data exchange.

case, it must be at least notified. This relay is known as a simple intra-BTS relay (Figure 1.6).

If, instead, the mobile is crossing the border of the BSC, then the MSC has to handle the relay to ensure the smooth transition of the conversation (Figure 1.7). This also applies to relays between MSC. The only difference, in the latter case, is that although the mobile is eventually handled by the second MSC, the first MSC has to maintain control of the call management.

1.7. Paging

Operation by which the entities belonging to the fixed network of a cellular system (BTSs, BSCs) seek to locate a specific mobile belonging to it. It is mainly used to locate a mobile before the start of a call that ends on the mobile.

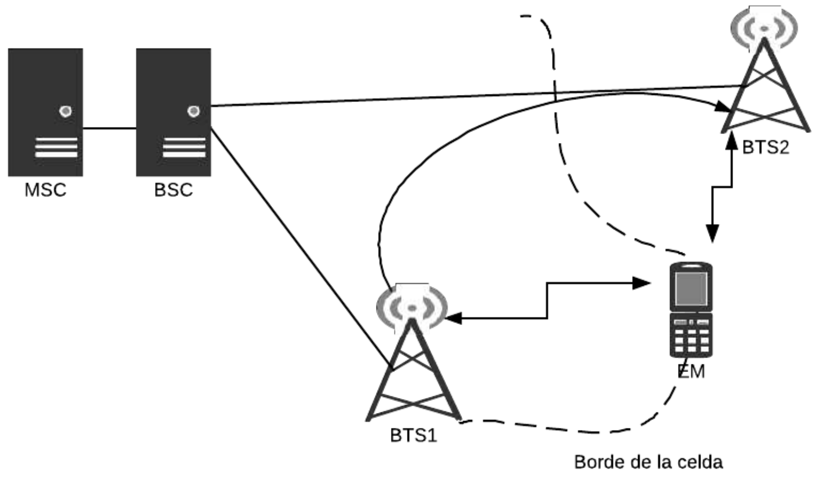

Figure 1.6. Simple relay intra BSC

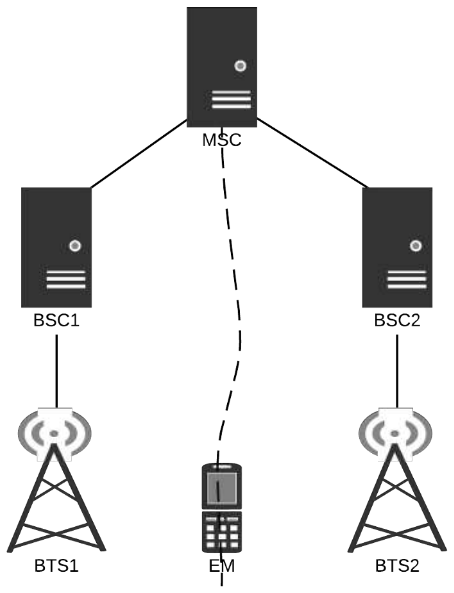

Figure 1.7. Relay between MSCs

1.8. Drive Test

It is a technique widely used to verify the quality of cellular service, which consists of collecting data from a cellular mobile system, through a tour over the coverage area. These data allow determining coverage and performance failures in urban, rural and even perimeter areas between buildings. The Drive Test also helps determine the performance and functionality of terminals in your relay process. Is

The test is carried out using software installed on a computer that is dedicated to analyzing the data it receives from a GPS, a mobile phone in normal mode and another in engineering mode, which is responsible for obtaining the most relevant data from the channels and the events generated by the network; additionally, a vehicle is used to travel a certain area.

Mobile terminals are used as follows. The first cell phone in normal mode is used to make calls, while the second phone is used in engineering mode to measure various parameters and events that occur when the cell phone is idle and during a call [5].

Among the parameters to be measured are:

- Received signal strength (RSSI received signal strength indicator) *Received Signal Strength Indicator),* RxLev (indicates the average strength of the received signal) which is governed by table 1.3).
- Signal-to-interference ratio (Carrier-to-interference ratio $\frac{C}{I}$ (of English *Carrier to interference ratio)).*
- Signal quality (Bit error rate BER) *Bit Error Rate),* BLER block error rate *Block Error Rate),* Quality received RXQUAL (From English *Receiver Quality)* which is governed by table 1.4).
- Cell with the best signal.
- Relay zones (*Handover).*
- Out of coverage areas.

Table 1.3. RxLev levels [6]

RXLEV	Range in dBm
0	less than -110
one	- 110 to -109
two	- 109 to -108
3	- 108 to -107
4	- 107 to -106
...	...
61	- 50 to -49
62	- 49 to -48
63	greater than -48

The BER is de fi ned as the ratio between the erroneously received bits and all the bits sent over the convolutional encoder. BLER is de fi ned as the ratio of the number of received erroneous bit blocks to the total number of sent blocks, a block is de fi ned as erroneous when the cyclic redundancy check (CRC) fails.

Table 1.4. RXQUAL levels [6]

RXQUAL	BER range
0	less than 0.1%
one	0.26% to 0.3%
two	0.51% to 0.64%
3	1.0% to 1.3%
4	1.9% to 2.7%
5	3.8% to 5.4%
6	7.6% to 11.0%
7	greater than 15.0%

1.8.1. Types of Drive Test

To determine the different parameters of the network, two types of drive tests are used:

- Before network design:
  - It is done to characterize the propagation pattern and the effects of fading.
  - The data collected will allow to adjust the propagation prediction model for the simulations.
- Post network design:
  - It is carried out to verify the configuration of neighboring cells and network optimization.
  - Mainly to verify coverage objectives.
  - Verify the overlap of neighboring cells, which allow the transfer to be carried out.
  - Verify the quality of the service.
  - Periodic maintenance.
  - Market change.
  - For customer complaints

1.8.2. Sampling criteria

The signal received from the cellular network is sampled and then several samples are averaged, over a spatial window (xy), called "Bin". The details about sampling must be taken care of.

When RF (Radio Frequency) measurements are made, the sampling criterion seeks to eliminate the small-scale fading component, in order to obtain the large-scale behavior. RF signal measurements should be averaged over intervals of at least 40 λ. If the distance is shorter the behavior of the small-scale fading, It cannot be eliminated and could affect the local average; if the distance is greater, the value obtained does not represent a local average of the signal, since it has different bin points.

The number of RF samples taken over 40 λ it must be equal to or greater than 50 samples. This is known as Richard's rule [7]. Depending on the frequency of operation and the speed of the vehicle, the sampling frequency will be obtained. For example, with a frequency f = 1900 MHz:

$$\lambda = \frac{C}{F} = \frac{3 * 10^8}{1900 * 10^6} = 0.158\ m \tag{1.1}$$

$$40\ \lambda = 40 * 0.158\ m = 6.32\ m$$

As we can see in equation 1.1, at least 50 samples must be averaged over 6.32 meters (for this example) or 1 sample every 0.1264m (6.32m / 50). We rewrite Rs as seen in 1.2:

$$R_s = \frac{Samples}{Second} = \frac{Speed}{\frac{Distance}{Samples}} \tag{1.2}$$

If we put for example 50 Km / h and the values used previously we have:

$$R_s = \frac{\frac{\text{fifty} * 10^3\ m}{3600\ s}}{0.1264\ m\ /\ sample} = 110\ \frac{samples}{second}.$$

This means that if you go in a vehicle at 50 km / h, the system must take 110 samples / second. However, in practice, the equipment reports a fixed rate Rs, which can be slow, so in this case, the vehicle speed must be modified during the drive test. Rewriting equation 1.2 we are left with 1.3:

$$V_{max} = 0.8 * \lambda * R_s \tag{1.3}$$

with which we can obtain the maximum speed at which a car must go based on the signal and sampling frequency.

As mentioned above it is necessary to define a Bin, the size of this averaging window should be small enough to capture slow variations due to shadowing and large enough to average fast variations due to multipaths. Size is typically selected from 40 λ at 1500m, that is, all measurements in this table are averaged to give a single value.

Not all bins within the coverage area can be tested, so a large number of samples must be considered. With a greater number of bins, the level of con fi dence in the results is higher. In general, to have an acceptable con fi dence, between 300 and 400 bins are necessary.

EPISODE 2

SYSTEM PROPOSAL

This chapter presents the system proposal, which is composed of a communication element, a processing element, a location element, and a data storage and presentation element. The devices that were used for this purpose are the ESP8266, the SIM900 module, the XM37-1612 GPS chip and a computer respectively. As can be seen in figure 2.1.

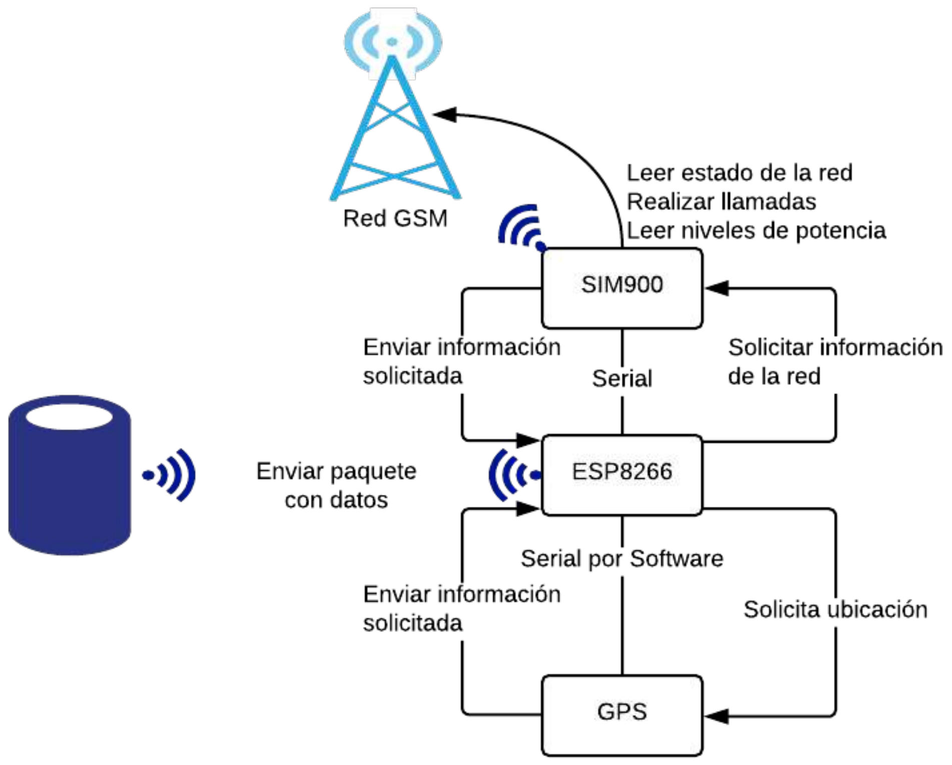

Figure 2.1. System design proposal

2.1. SIM900

The SIM900 is a GSM / GPRS module (*Global System for Mobile / General Packet Radio Service)* operating on the frequencies GSM 850 MHz, EGSM 900 MHz, DCS 1800 MHz and

PCS 1900 MHz, uses multiple class 10, class 8 slots and is compatible with the GPRS CS-1, CS-2, CS-3 and CS-4 coding schemes. *Coding Schemes).* Thanks to its small size of 24 * 24 * 3 millimeters, the SIM900 can meet almost all (physical) space requirements in user applications such as M2M (machine to machine), smartphones, etc.

The SIM900 also has 68 SMT surface mount connections. *Surface Mount Technology)* and provides all the hardware interfaces between the module and the board (in this case the Wemos D1 mini board (ESP8266)). The module is designed with energy saving technology so that the current consumption is 1.0mA in standby mode. It also integrates the TCP / IP protocol (*Transmission Control Protocol / Internet Protocol)* using ATs commands (from English *ATtention)* *[* 8].

For this design, the SIM900A V1 SCH development board was used, which can be seen in figure 2.2.

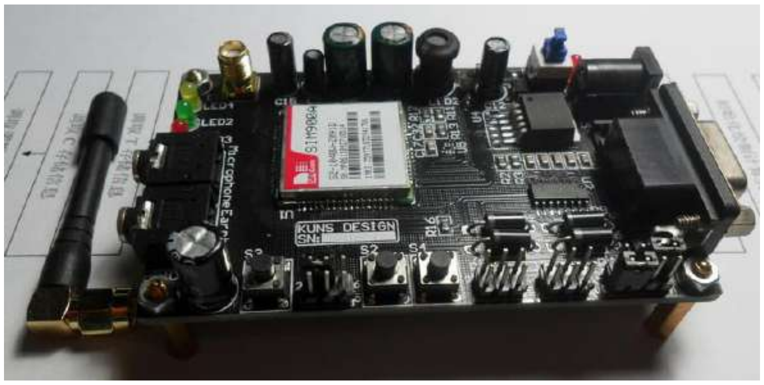

Figure 2.2. SIM900A V1 SCH board [9]

2.1.1. Pin description

This section details the pins on the SIM900A V1 SCH development board with their functions. Table 2.1 describes the function of each pin.

2.1.2. Operating modes

In table 2.2 you can see a summary of the operating modes of the SIM900 module.

2.1.3. AT commands

The SIM900 module is controlled by AT commands on its serial interface. The AT commands implemented in the SIM900 are a combination of GSM07.05, GSM07.07, the ITU-T V.25ter recommendation and the AT commands developed by SIMCom [10].

Table 2.1. SIM900A V1 SCH development board pins

Pin name	Functionality
Switched on	Turn on or turn off the module Port
Microphone	that allows you to connect a microphone for communication.
Headset	Port that allows you to connect a headset for communication.
Source of ali- mentoring	Connector to power the 5 volt ta, you need a power outlet or card source.
Q4	Con fi guration pins for TTL or Finish the serial communication DB9 front (DB9 join pi and 4-6). menu 1-3 and 6-8. TTL join pins 3-5
Port of ante- na	Allows you to connect an antenna to the chip.
P2	TTL serial communication pins (2 Tx, 4 Rx, 6 ground)
DB9 connector	Serial communication through co- nector DB9.

AT commands can be divided, according to their syntax, into three categories: "basic", "S parameter", "extended".

Basic syntax: These AT commands have the following formats "AT <x> <n>" or "AT & <x> <n>", where "<x>" is the command and "<n> is / are the argument / s for that command.

S parameter syntax: These AT commands have the format “ATS <n> = <m>”, where “<n>” is the index of the S-register to be modified, and “<m>” is the value that will be assigned to it. "<m>" is optional, otherwise it will be assigned a default value.

Extended syntax: These commands can operate in four modes, as shown in the following list.

- Test command: AT + <x> = ?, the device returns a list of parameters and ranges of values established with the corresponding write command or by internal processes.
- Read command: AT + <x>?, The device returns the current value of the parameter or parameters.
- Write command: AT + <x> = <...>, this command allows the user to set the value of a parameter.
- Execution command: AT + <x>, the execution command reads parameters that cannot be modified by the user, affected by internal processes in the GSM engine.

Table 2.2. Summary of operating modes [8]

Mode	Function	
Operation normal	GSM / GPRS SLEEP	The module will go into suspend mode automatically if there are no over-the-air transmissions and there are no hardware interrupts (such as GPIO interrupts or data on the serial port). In this mode, the current consumption will be reduced to the minimum level. In this mode, the module can receive paging messages (*paging)* and SMS.
	GSM IDLE	The module is registered in the GSM network, and is ready for communication.
	GSM TALK	The connection between two subscribers is in process. In this case, the power consumption depends on the network settings such as Transmission Discon- tinua or DTX (from English *Discontinuous Transmis- Zion)* on or off, Full rate, Enhanced full rate, or Average rate FR / EFR / HR (of English *Ful l Rate / Enhanced Ful l Rate / Half Ra- tea),* jump sequences, antennas.
	GPRS STANDBY	The module is ready to transmit GPRS data, but no data has been sent or received yet. In this mode the current consumption depends on the network parameters and the GPRS con fi guration. There is
	GPRS DATA	a GPRS data transfer in progress (PPP or TCP or UDP). In this mode, the current consumption is related to the network parameters (for example the power control level); upload and download data rates and GPRS settings (eg multiple slots).
Off	To enter this mode, use the AT command "AT + CPOWD = 1" or the PWRKEY. The power management unit cuts the power supply to the baseband section of the module, leaving only the power supply for the RTC. *Real Time Clock).* Both the software and the serial ports are not operational.	
Mode from it functioned nality minimal	The AT command “AT + CFUN” can be used to configure the module in this mode without removing the power supply. In this mode, the RF (radio frequency) section of the module will not work and the SIM card will not be accessible, however the serial ports will be accessible. Power consumption in this mode is lower than normal mode.	

2.1.3.1. AT commands used

The AT commands used were:

- AT + CREG: This command is used to register the module in the GSM network.
  - This command receives as a parameter a number between 0 and 2, being 0 to disable continuous messages from the network logs, 1 to enable continuous messages from the network only with the status and 2 to enable continuous messages from the network with all the status information. , area of location and identification of the cell.
  - The states are:

    0 Equipment not registered.

    1 Registered in the local network.

    2 Not registered, however the team is looking for a new operator to register.

    3 Registration denied.

    4 Unknown.

    5 Registered while roaming.

- AT + CENG: Allows you to configure the module in engineering mode, which allows you to observe the network parameters, such as: ARFCN (absolute radio frequency channel numbers) of the BCCH (Broadcast Control Channel), received power level, quality of the received signal, country code, network code, identity code of the base station BSIC (from English *Base Station Identity Code),* cell identi fi cation, minimum level of access to the cell, maximum transmission power of the mobile, location area code, time advance.

  - This command is of the extended type, it receives two parameters for its con fi guration, the first indicates the way in which the equipment will work and the second allows us to con fi gure whether we want the cell identification to appear.
  - Operation modes:

    0 Turn off engineering mode.

    1 Turn on engineering mode.

    2 Turn on engineering mode and activate continuous reporting messages about the net.

    3 Turn on engineering mode with short report messages.

  - Expected response: + CENG: <mode>, <Ncell>

    [+ CENG: <cell>, “<arfcn>, <rxl>, <rxq>, <mcc>, <mnc>, <bsic>, <cellid>, <rla>, <txp>, <lac>, < TA> ”<CR> <LF>

    + CENG: <cell>, "<arfcn>, <rxl>, <bsic>, [<cellid>,] <mcc>, <mnc>, <lac>" ...] OK

- ATD Allows you to make a call. This is a basic AT command, it receives as a parameter the number you want to call, ending in a semicolon.

- ATH Command that ends a call in progress. This is a basic AT command, it receives no parameters.
- AT + MORING: Command that enables or disables continuous call status messages.
  - This is an extended AT command, it receives a configuration parameter that indicates if this mode is active 1 or off 0. If active, it shows the status of the call originated in the equipment.
  - Team messages:
    - MO RING Message indicating that the destination number is being alerted.
    - MO CONNECTED Message indicating that the call was established.
    - NO CARRIER Message indicating that you cannot connect with that number.
- AT + CSQ: Command that allows observing the bit error rate BER (From English *Bit Error Rate)* and the received signal strength indicator RSSI (from English *Received Signal Strength Indicator).* It is an extended AT command, however it is not con fi gured, it is only executed. Your expected response is the RSSI and the BER in that order.
- AT + SAPBR: This command allows you to configure the modem for IP-based applications.
  - It receives two values and two character strings as a parameter.
    - Parameter 1: "cmd type" con fi gures what action the module will perform, it varies from 0-5:

      0 Close carrier.
      1 Open carrier.
      2 Check carrier.
      3 Set carrier parameters. 4 Get carrier
      parameters.
      5 Store the values in NVRAM.
    - Parameter 2: “cid” varies between 1-3 and identifies the connection.
    - Parameter 3: "ConParamTag" carrier parameter:
      - "CONTYPE" Type of internet connection.
      - "APN" Access point name, maximum 50 characters.
      - "USER" User name, maximum 50 characters.
      - "PWD" Password, maximum 50 characters.
      - "PHONENUM" Telephone number for the CSD call.
      - "RATE" connection rate for the CSD call.
    - Parameter 4: "ConParamValue" Carrier parameter value:
      - CONTYPE value: GPRS or CSD (from English *Circuit-switched data cal l).*
      - RATE value: 0 2400, 1 4800, 2 9600, 3 14400.

- AT + FTPCID: This command selects the previously configured pro fi le that will be used in FTP.
- AT + FTPSERV: This command sets the domain name or IP address of the server.
- AT + FTPUN: This command establishes the username that is registered in the FTP server.
- AT + FTPPW: This command sets the password for the session.
- AT + FTPGETNAME: This command sets the name of the file to download.
- AT + FTPGETPATH: This command sets the path where the file is located. AT + FTPGET:
- This command gets the file from the server.

To see all the AT commands and their parameters, access the AT command manual that can be found in [10].

2.2. GPS XM37-1612

The XM37-1612 module has a high sensitivity, low consumption and reduced size described later. Thanks to its small size, it is easy to integrate into portable devices such as mobile phones, cameras and vehicle locators.

The equipment uses a self-generated ephemeris prediction that does not require network or microcontroller assistance, this is valid for up to three days and is automatically updated when the module is on and the satellites are available. Ephemeris predictions are stored in memory *fl ash* integrated, which allows a cold start of less than 15 seconds [11]. The plate used can be seen in 2.3

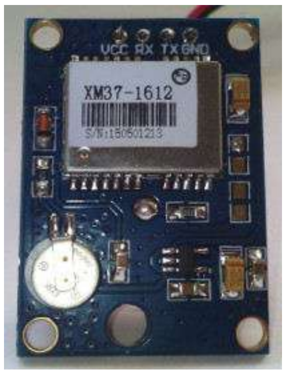

Figure 2.3. XM37-1612 GPS Development Board [12]

2.2.1. Module features

- High sensitivity MediaTek solution.
- Supports 66 channels of GPS
- Low consumption.
- Quick first location with low signal level.
- Built-in twelve multi-tone active interference canceller. Hybrid ephemeris
- prediction for faster cold start. Integrated data logging.
-
- Built-in DC / DC converter to save power. Up to 10 Hz
- refresh rate ± 11ns.
- SBAS (satellite-based argumentation system) capability. Japanese QZSS
- support.
- Indoor and outdoor multipath detection and compensation. Small
- size 16.0 * 12.2 * 2.2 mm.
- It has pins with SMD surface mounting technology.

2.2.2. Module speci fi cations

Table 2.3 shows a summary of the GPS receiver.

2.2.3. Pin description

This section details the pins on the development board of the XM37-1612 chip with their functions. Table 2.4 describes the function of each pin.

2.2.4. Software interface

The module delivers six types of messages, in different formats on positioning, due to the requirements of a drive test it was decided to work with the RMC message type. *Recommended Minimim Speci fi c GNSS Data).* Table 2.5 shows the format and parameters of the message.

Sample answer:

$ GPRMC, 213735.000, A, 1024.6604, N, 06652.9343, W, 2.00,250.24,191017 ,,,, A *

78. For more details on the types of messages, see [11].

Table 2.3. GPS receiver speci fi cations [11]

GPS receiver		.
Chip	MediaTek MY3337 (ROM version). L1	
Frequency	1575.42MHz, C / A code. Supports 66	
Channels	channels.	
Update rate	1Hz by default, up to 10Hz	
Sensitivity	Tracking	- 162dBm, up to -165dBm (With an external low noise ampli fi er).
	Cold start	- 143.5dBm, up to -148dBm (With an external low noise ampli fi er).
Location time tion	Hot start (open sky)	<1s typically.
	Hot start (indoors)	<30s
	Cold start (open sky)	32s (typically) without assisted GPS.
		<15s (typically) with assisted GPS (hybrid ephemeris prediction).
Location accuracy	Autonomous	3m (2D RMS (Double the distance of the root mean square)).
	SBAS	2.5m (depending on the accuracy of the correction data)
Maximum altitude	18000m	
Maximum speed	<515m / s	
Protocol support	NMEA 0183 ver 4.01	9600 bps (baud per second), 8 data bits, no parity, 1 stop bit (default) 1Hz: GGA, GLL, GSA, GSV, RMC, VTG
Physical characteristics		.
Guy	24 pin SMD	
Dimensions	16.0mm * 12.2mm * 2.2mm ± 0.2mm	

2.3. ESP8266 module

The ESP8266 is a system on a chip or SoC. *System on a Chip)* It integrates the TCP / IP protocol stack and is capable of connecting to WiFi networks, allowing the device to interact with other elements on the Internet. This device was developed by Espressif Systems and can be purchased integrated into a development board (as shown in Figure 2.4) at a low cost, compared to other WiFi systems [13].

At first, it was used as a WiFi adapter for other microcontrollers, which

Table 2.4. Pin description

Name	Function
GND	Land.
Tx	Serial transmission pin (3.3 V).
Rx	Serial receive pin (3.3 V). Power
VCC	pin 3.3 V.

Table 2.5. Description of the GPRMC message [11]

Name	Example	Unit	Description
Message id	$ GPRMC		RMC protocol header
UTC time	060406.000		hhmmmss.sss
Condition	TO		A = Valid data or V = invalid data
Latitude	2503.7148		ggmm.mmmm
Indicator N / S	N		N = north or S = south
Length	12138.7451		gggmm.mmmm
E / O indicator	AND		E = east or W = west
Speed over the land	0.01	Knots	True
Course on land	0.00	Degrees	
Date	180313		ddmmy
Variation mag- netic		Degrees	
Sense of it varies- tion			E = East or W = West
Mode	D		A = autonomous, D = DGPS, E = DR, N = invalid data, R = rough position, S = simulator
Checksum * 78 check			
<CR> <LF>			End of message

I would use AT commands to control the ESP8266, those commands were handled by a fi rmware which was by default in the module. However, the community realized that the device was programmable, allowing more freedom. Therefore, instead of using a set of AT commands, there was the possibility of developing an optimized serial protocol to interact with the device from another microcontroller, and better yet, it was possible to develop autonomous applications, since the ESP8266 integrates a 32-inch microcontroller. bits.

The device supports AT commands, Espressif has an SDK (software development kit) to program it and currently the device is supported in the arduino IDE, which facilitates

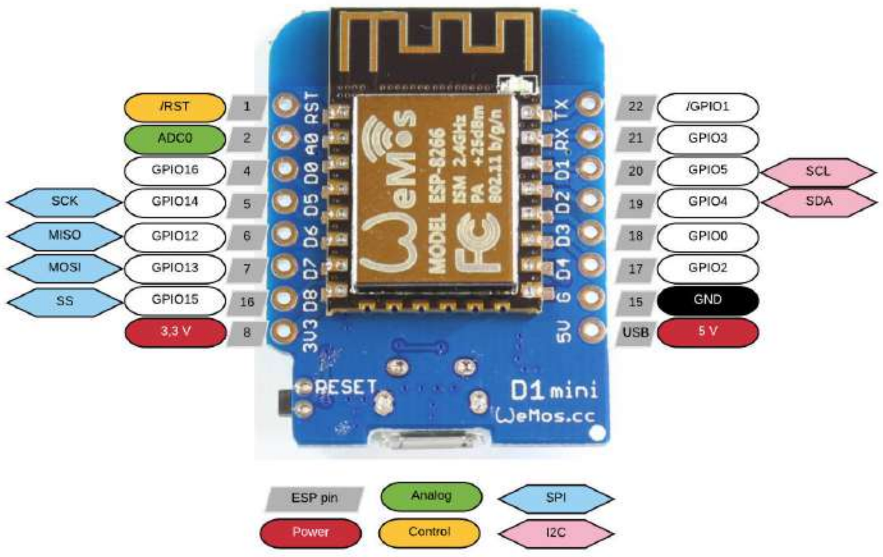

Figure 2.4. Development board D1 mini [14]

programming [13] [15].

2.3.1. ESP8266 Features

- Supports 802.11 b / g / n.
- Integrated 32-bit low-power microcontroller. Integrated
- 10-bit ADC.
- Integrated TCP / IP protocol stack.
- Supports 2.4 GHz WiFi and WPA / WPA2 encryption. *WiFi Protected Access).*
- Supports STA / AP / STA + AP modes of operation.
- 64k Static Random Access Memory (SRAM) *Static Random Access Memory)*
- STBC, 1X1 MIMO, 2X1 MIMO.
- Current consumption in deep sleep mode <10 *μA,* leakage current at shutdown <5 *μA.*
- Wake up and transmit packets in <2ms.
- Power consumption in idle mode <10mW (DTIM3). +
- 20dBm output power in 802.11b mode.

- Operating temperature range -40C 125C. Certi fi ed by FCC,
- CE, TELEC, WiFi Alliance and SRRC.

2.3.2. Pin description

This section details the pins on the D1mini development board with their functions. In figure 2.4 you can see the pin assignments. Table 2.6 describes the function of each pin.

Table 2.6. Pin description D1Mini [14]

Name	Function
TX	Serial transmission.
RX	Serial reception.
A0	Analog input, maximum 3.3V input. GPIO16.
D0	
D1	GPIO5, SCL.
D2	GPIO4, SDA.
D3	GPIO0, 10K Pull-up.
D4	GPIO2, 10K Pull-up, Integrated LED.
D5	GPIO14, SCK.
D6	GPIO12, MISO.
D7	GPIO13, MOSI.
D8	GPIO15, 10K Pull-down, SS. GND.
G	
5V	5V.
3V3	3.3V.
RST	Reset.

CHAPTER 3

SYSTEM IMPLEMENTATION

As mentioned in the previous chapter, the system is made up of a GSM, GPS and ESP8266 module. The latter being the controller, which is in charge of collecting, processing and retransmitting the data obtained. In figure 3.1 you can see the connections between the different modules through serial. Since the ESP8266 (D1 mini) board only has one serial port, it is necessary to use the library *SoftwareSerial.h* that allows to use the GPIO ports (from English *General Purpose Input / Output)* instead of the serial programming via USB.

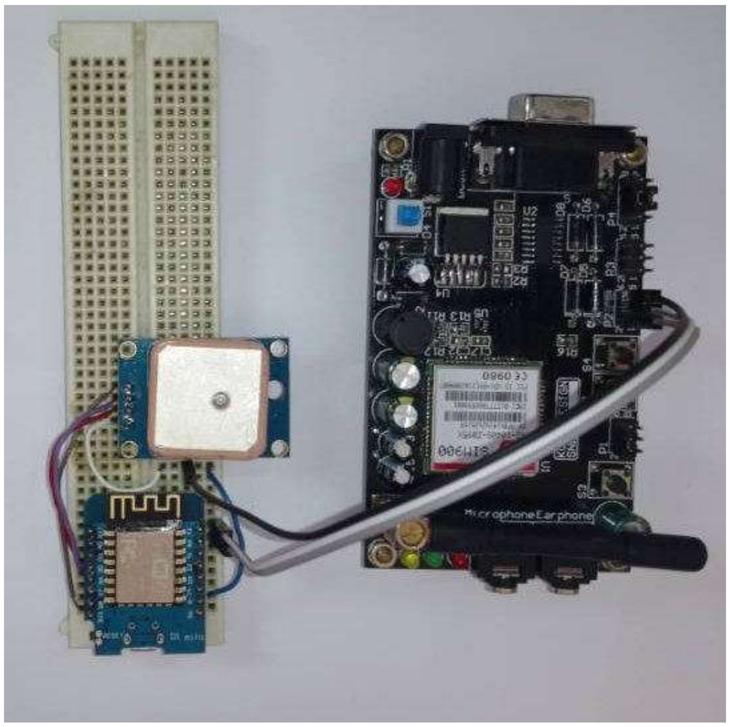

Figure 3.1. GSM, GPS and ESP8266 module connections

3.1. Drive Test System

The state diagram that governs the system algorithm can be seen in figure 3.2, it should be noted that the WDT timer must be disabled. *Watch Dog Timer).*

This verifies that the ESP8266 is not in a cycle without performing any operation and, if so, ends the execution of the program. However, the ESP8266 has two WTDs, one for software, which can be disabled; and another for hardware, which cannot be disabled, to prevent the hardware WDT from ending the program, the timer is restarted every time we are in a waiting cycle.

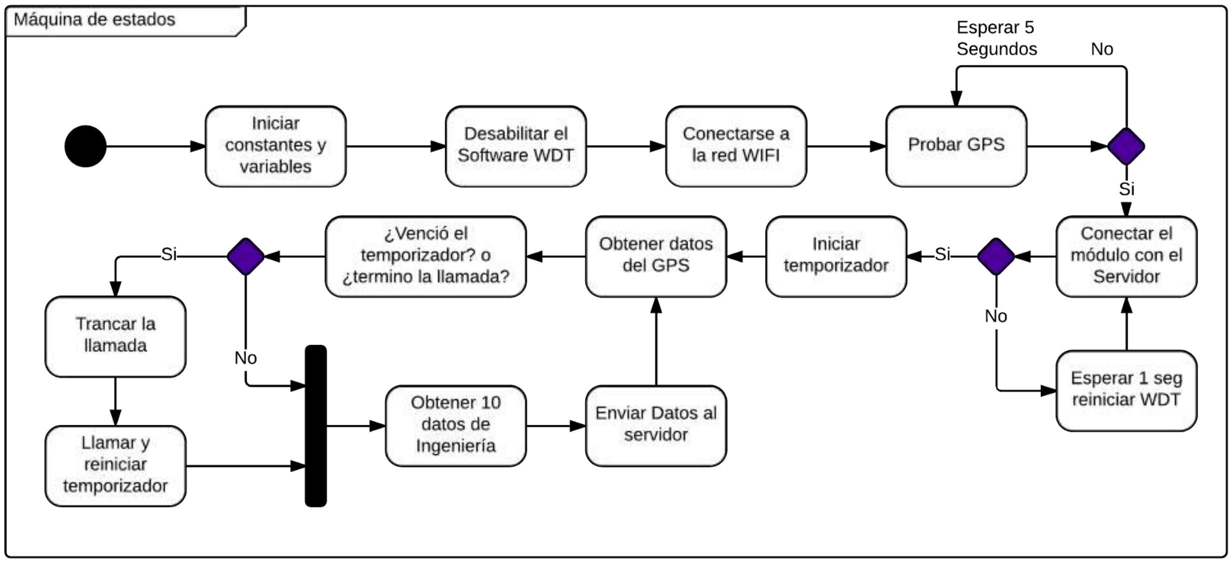
Máquina de estados
Esperar 5 Segundos
No
Iniciar constantes y variables
Desabilitar el Software WDT
Conectarse a la red WIFI
Probar GPS
Si
Conectar el módulo con el Servidor
Si
No
Iniciar temporizador
Obtener datos del GPS
¿Venció el temporizador? o ¿termino la llamada?
Si
No
Trancar la llamada
Llamar y reiniciar temporizador
Obtener 10 datos de Ingeniería
Enviar Datos al servidor
Esperar 1 seg reiniciar WDT

Figure 3.2. System general status diagram

After connecting to the WiFi network and initializing all the variables, it is verified that the GPS is working correctly. To do this, a function was created that reviews the GPS message described in the previous chapter and verifies if the data are valid or not through the "Status" field, the diagram that describes the function can be seen in figure 3.3. As previously mentioned, the GPS module communicates with the ESP8266 through serial communication, since the GPS sends continuous messages every second, it was decided to disable the interruptions generated by the serial message; In this way, if you want to obtain the GPS location, the interruption is activated, the message is saved and the interruption of the GPS message is disabled again. If this is not done the periodic GPS messages could fill up the reception buffer. If the GPS verification is unsuccessful, it will be tried again until it is successful.

After the GPS is located, it proceeds to connect with the server to which the data will be sent, in case of failure, wait a second (the WDT is restarted) and try again. Once connected to the server, the GPS data is obtained, however, during each data collection it is verified that these are reliable (by means of the “Status” parameter of the message). If not, wait for the next message. If the data is valid, it is saved and then sent to the server. In the fi gure 3.4 we can see the diagram of the function in charge of obtaining the data.

Once the GPS data is available, the engineering measures are requested, first it is verified if the timer that was started has expired or if the call was lost, if so, the call in process is terminated (if any) , the timer is restarted, a number is called

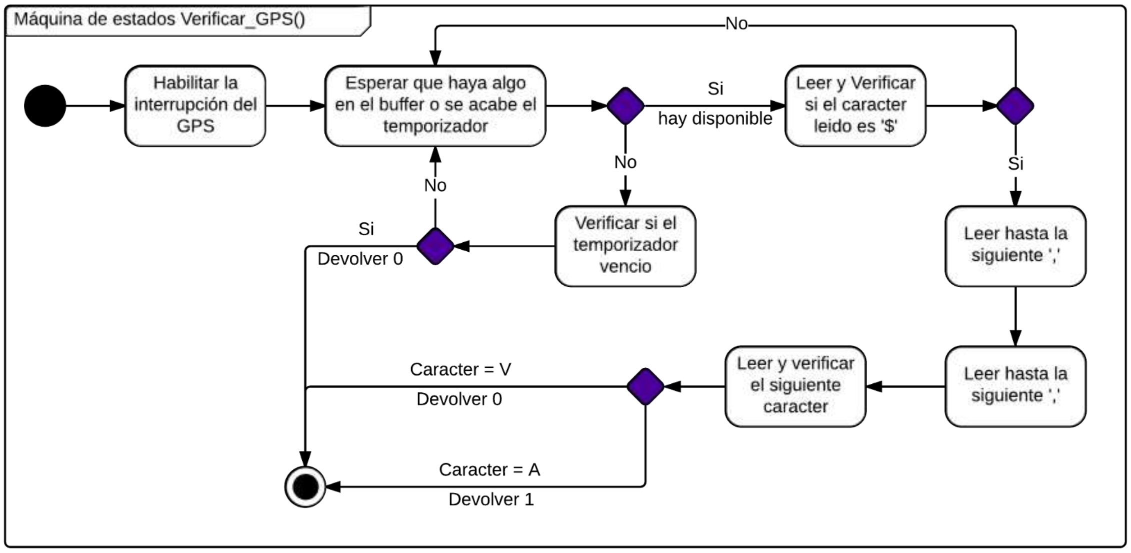
Máquina de estados Verificar_GPS()
Habilitar la interrupción del GPS
Esperar que haya algo en el buffer o se acabe el temporizador
Si
hay disponible
Leer y Verificar si el caracter leido es '$'
No
No
Si
Verificar si el temporizador vencio
No
Si
Devolver 0
Leer hasta la siguiente ','
Leer hasta la siguiente ','
Leer y verificar el siguiente caracter
Caracter = V
Devolver 0
Caracter = A
Devolver 1

Figure 3.3. Status diagram for GPS verification

In the test run, ten engineering data is obtained, the data is sent to the server, and the GPS data is obtained again. It was decided to choose ten data points because GPS time is very slow compared to taking engineering data.

To obtain the engineering data, call or end the call, it is necessary to send AT commands, with which the SIM900 is controlled, the diagram that describes the algorithm used to send and receive the response of the AT commands is shown in the figure
3.5. As can be seen, first you have to send the AT command (that you want to send), after this the module is expected to respond, this response is saved in an array and it is verified if it contains “NO CARRIER”, “MO CONNECTED” or the expected response, this is verified because when losing or establishing a call the module returns these messages respectively. Depending on the response, one of two flags is activated that allow us to know if the call was lost, to make another call, or if it was established, which allows activating the collection of call quality data. When the expected answer is received or when the timer expires, the function ends.

The engineering data that you want to obtain is relevant to know how the cellular network is, detect faults and report them. The SIM900 allows you to capture the following data:

- The absolute radio frequency channel number (ARFCN) *Absolute RadioFrequency Channel Number)* broadcast control channel. This number represents the frequency of the control channel of the cell that answers the phone when executing the command. (BCCH)

- Average received power level RxLEV. (RXL) Quality of
- the received signal. (RXQ)

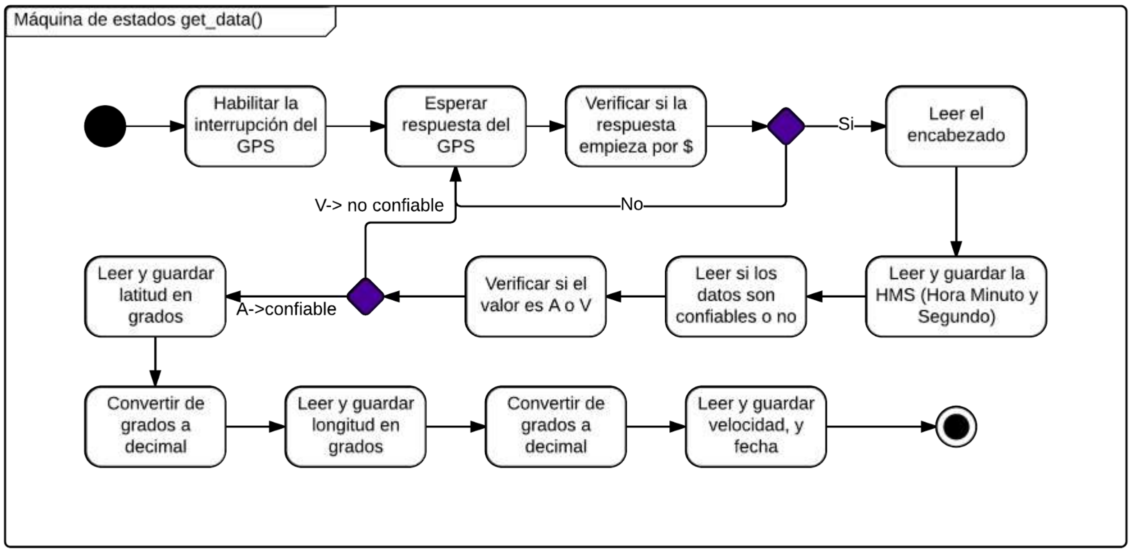
Máquina de estados get_data()
Habilitar la interrupción del GPS
Esperar respuesta del GPS
Verificar si la respuesta empieza por $
Si
Leer el encabezado
No
V-> no confiable
Leer y guardar la HMS (Hora Minuto y Segundo)
Leer si los datos son confiables o no
Verificar si el valor es A o V
A->confiable
Leer y guardar latitud en grados
Convertir de grados a decimal
Leer y guardar longitud en grados
Convertir de grados a decimal
Leer y guardar velocidad, y fecha

Figure 3.4. Status diagram for GPS data acquisition

- Country code. (MCC)
- Network code. (MCN)
- Identification code of the BTS. BSIC
- Cell identification in hexadecimal format. (CELLID) Minimum
- signal level to access the network. (RLA) Maximum transmission
- level on the CCCH. (TXP)
- Location area code, in hexadecimal format. (LAC) Temporary
- advance. (TA)
- Bit error rate. (BER)
- Received signal strength indicator. (RSSI)

Each of the parameters is saved in a structure and sent to the server. The last two parameters (BER and RSSI) are only measured when a call is in progress, this is done to estimate the quality of the call. Figure 3.6 describes how engineering data is obtained and stored.

Once the algorithm has been raised, it is necessary to highlight certain values of it. The algorithm manages to obtain the engineering data in approximately 0.5 seconds, if to this is added that the GPS takes 1 second to successfully deliver the message, this gives us a total of one and a half seconds per sample. Due to the slowness of the GPS to deliver the information

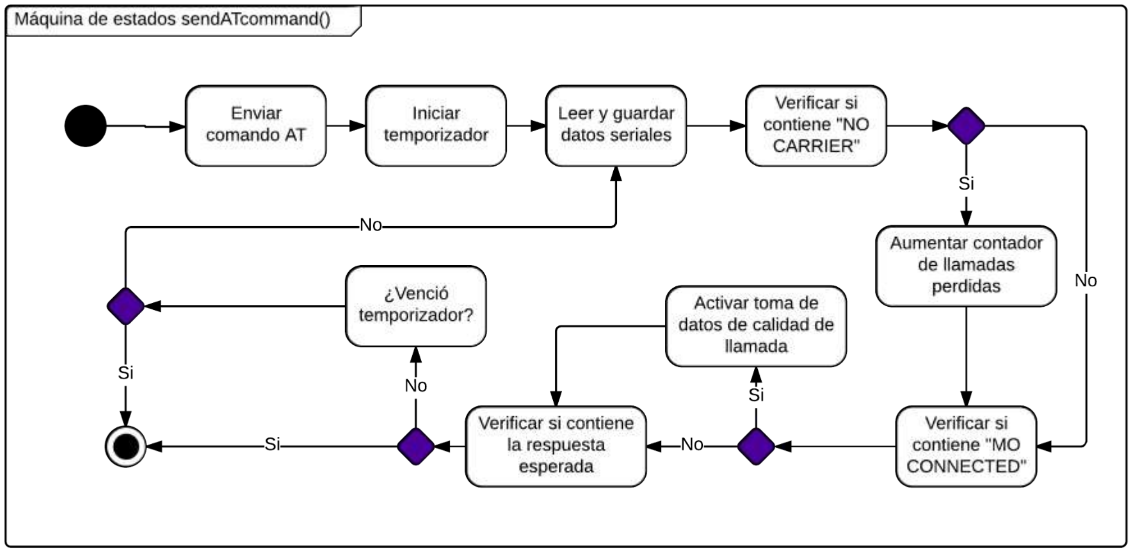
Máquina de estados sendATcommand()
Enviar comando AT
Iniciar temporizador
Leer y guardar datos seriales
Verificar si contiene "NO CARRIER"
Si
No
Aumentar contador de llamadas perdidas
No
¿Venció temporizador?
Activar toma de datos de calidad de llamada
Si
Si
No
Verificar si contiene la respuesta esperada
No
Verificar si contiene "MO CONNECTED"
Si

Figure 3.5. State diagram of sending AT commands

It was decided to take one GPS measurement for every ten engineering measurements, thus leaving a sampling rate:

$$R_{s} = \frac{10\ \textit{samples}}{\text{one } \textit{second (GPS)} + 5\ \textit{seconds (samples)}}$$

$$R_{s} = 1.66\ \textit{samples / second}$$

If we use the sampling rate obtained previously and assuming the 850 MHz band (Movistar Venezuela) we have to:

$$V_{max} = 0.8 * 0.3529 * 1.66$$

$$V_{max} = 0.4705 * \frac{m}{s} 3.6$$

$$V_{max} \approx 1.69\ \frac{Km}{h}$$

As can be seen, to meet Richard's criteria, the vehicle must go at a speed of 1.69 km / h. However, when studying the GSM SIM900 module it was observed that the module delivers the averaged power information, since the module measures it every 120 milliseconds at the end of each multiframe, which increases the number of samples that are had with the system as you can see the sample rate increases and therefore the speed also does:

$$R_{s} = \frac{4 * 10\ \textit{samples}}{\text{one } \textit{second (GPS)} + 5\ \textit{seconds (samples)}}$$

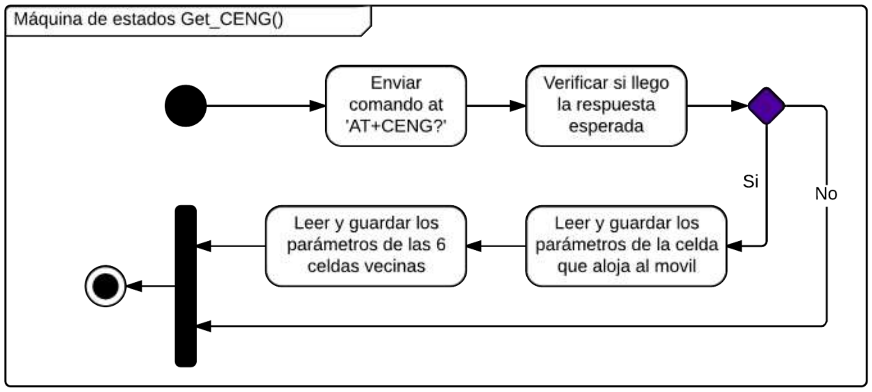
Máquina de estados Get_CENG()
Enviar comando at 'AT+CENG?'
Verificar si llego la respuesta esperada
Si
No
Leer y guardar los parámetros de la celda que aloja al movil
Leer y guardar los parámetros de las 6 celdas vecinas

Figure 3.6. State diagram for obtaining engineering data

$$R_s \approx 6.66$$

$$V_{max} \approx 1.88 * 3.6\frac{m}{s}$$

$$V_{max} \approx 6.77 \quad \frac{Km}{h}$$

3.2. Server

The server was developed in C language, it receives two input arguments: the port to use and the name of the tour database. The server communicates through the application programming interface (API of English *Application Programming Interface)* MySQL with the database and creates a new table with the name that was provided as a parameter. The server should create the *socket* and associate it to the listening port, which was provided as a parameter, after this the server remains waiting for connections from the D1 mini. Once the D1 mini connects to the server, it will receive the information sent by the D1 mini and will save it in a data structure that can be seen in appendix A, and will save said data structure in the table created by the server, in this way it completes the connection with the D1 mini and waits for connections. In the fi gure 3.7 you can see the status diagram of the server operation.

3.3. Problems presented in development

This section will explain the problems encountered during development, the three main problems were the ESP8266 watchdog timer (WDT). *WatchDog Timer),* Static Random Access Memory (SRAM) *Static Random Access Memory)* and the speed of data reception.

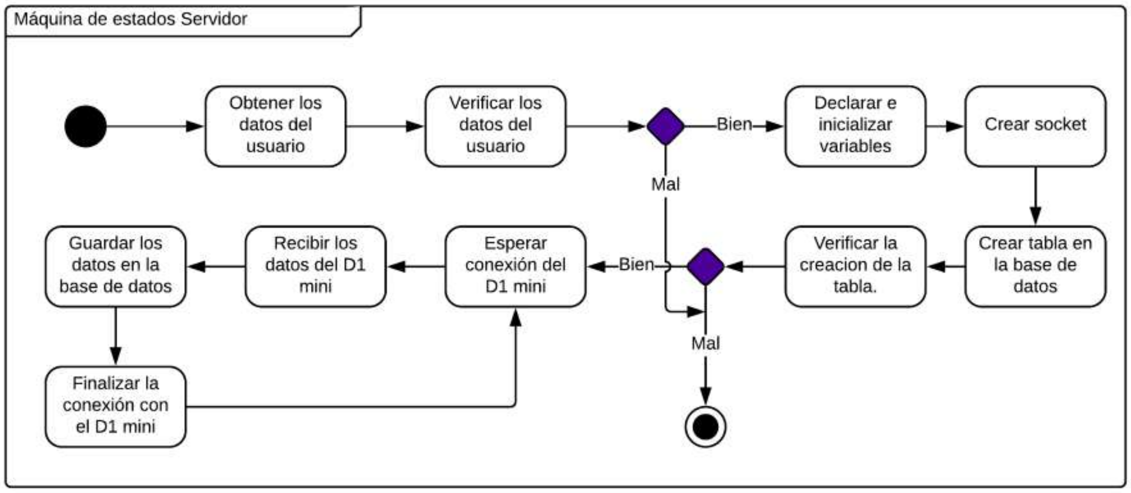
Máquina de estados Servidor
Obtener los datos del usuario
Verificar los datos del usuario
Bien
Mal
Declarar e inicializar variables
Crear socket
Crear tabla en la base de datos
Verificar la creacion de la tabla.
Bien
Mal
Esperar conexión del D1 mini
Recibir los datos del D1 mini
Guardar los datos en la base de datos
Finalizar la conexión con el D1 mini

Figure 3.7. Server health diagram

3.3.1. Watchdog timer

It is an electronic timer that is used to return the equipment to a safe state after a malfunction. The equipment restarts the timer continuously to prevent it from ending, in the event of a crash state in which the equipment fails to restart the timer, the equipment will restart to a safe state. [16]

In the case of the D1 mini, it consists of two WTDs, one for software and one for hardware, if the D1 mini is in a waiting cycle for a long time (approximately six seconds) it will restart. [17]

In the proposed algorithm there are waiting cycles, which have an expiration time greater than six seconds, this caused (in the first implementations) the D1 mini to restart. To prevent this from happening, the EspClass class was used, which has functionalities that allow manipulating the WDT timers. Software WDT is disabled first using the function *ESP.wdtDisable ()* and then in each long wait cycle it is placed *ESP.wdtFeed ()* which reboots the WDT by hardware.

3.3.2. Static Random Access Memory [18]

The D1 mini has 64 kilo Bytes of SRAM, this is used for different purposes:

- Static data - this memory block is reserved for all global and static variables in the program.
- Free storage *Heap)* - this block is used for elements that need to reserve memory dynamically. This block grows from the top of the static data area and in case of making a new reservation it will pile up until it reaches the stack block.

- Pila (from English *Stack)* - The stack is used to store local variables and to keep track of interrupts and function calls. The stack grows from the top of memory toward free storage. Every interrupt, function call, and / or local variable assignment causes the stack to grow. Returning from an interrupt or function call will reclaim all of the stack space used by that interrupt or function.

During the first implementations of the system an algorithm was carried out in which two large reception buffers were used, this caused the block of free storage memory and the stack to overlap, thus corrupting the system data, causing an error of battery as can be seen in the fi gure 3.8. The design was raised to use two serial communications through the library *SoftwareSerial.h,* to solve the memory problem a connection of *SoftwareSerial* to a connection through the hardware serial, thus eliminating the largest buffer and thus avoiding stack errors.

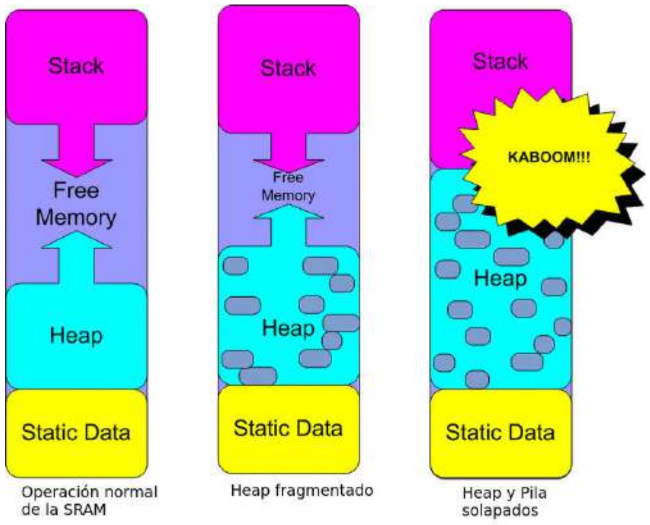

Figure 3.8. SRAM memory states [19]

3.3.3. Data reception speed

Due to the sampling requirements of a Drive Test system it is necessary to take data at a high speed, in the first implementations of the system functions were created for sending AT commands and receiving data sent by the GSM module, which had a fixed data reception time in the function. The problem is that for messages with few response characters and for long responses it took the same amount of time. To solve this, it was decided to set the minimum delay that is needed for a character and read until the reception buffer was empty, in this way it is ensured that the reading of the responses to the messages takes a time equivalent to its length.

CHAPTER 4

RESULTS

This chapter presents the results obtained from the implementation of the system proposed in the previous chapter. For this, a C ++ library was created that allows using the GSM module (from the English *Global System for Mobile Communications)* and the GPS (from English *Global Positioning System)* more freely, the most relevant functions of this library were described in the previous chapter. To carry out the tests, a vehicle with a power outlet was used to connect the GSM module, this due to the fact that the ESP8266 cannot supply the necessary current during the module's current peaks.

According to the calculations made in the previous chapter, the tests should be carried out at a maximum speed of approximately 6.7 km / h. Four runs were made at different speeds to test the system. The first test was carried out at a speed of 5 km / h, the route was approximately 1.2 km. Which can be seen in the figure
4.1.

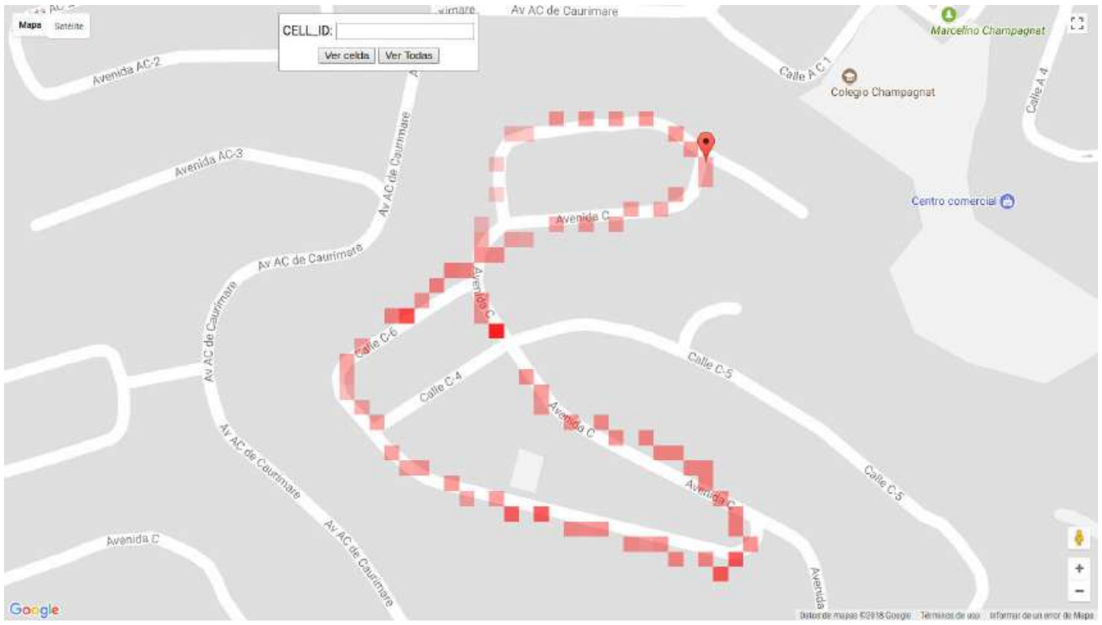

Figure 4.1. Route of the Drive Test at 5 Km / h

A 15x15 meter bin was used because for the ideal speed (approximately
6.7 km / h) the desired space is covered almost entirely. The intensity of the received power is represented by red squares, to have the real value of each

point must be obtained from the tour database. The markers on the map represent the missed calls that there were, clicking on the marker will show the information related to the missed call as shown in figure 4.2.

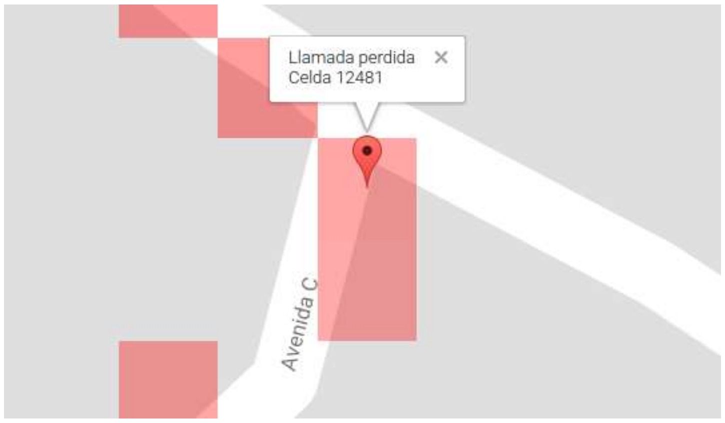

Figure 4.2. Marker

It can be seen that the missed call is in a low coverage area, however, since this was the only route with a missed call in this area, it can be said that the BTS did not have channels available and that is why it was not was able to deliver a channel to the modem. In the tool, coverage areas can be filtered using the cell identifier, using the box at the top of the page, this can be seen in figure 4.3.

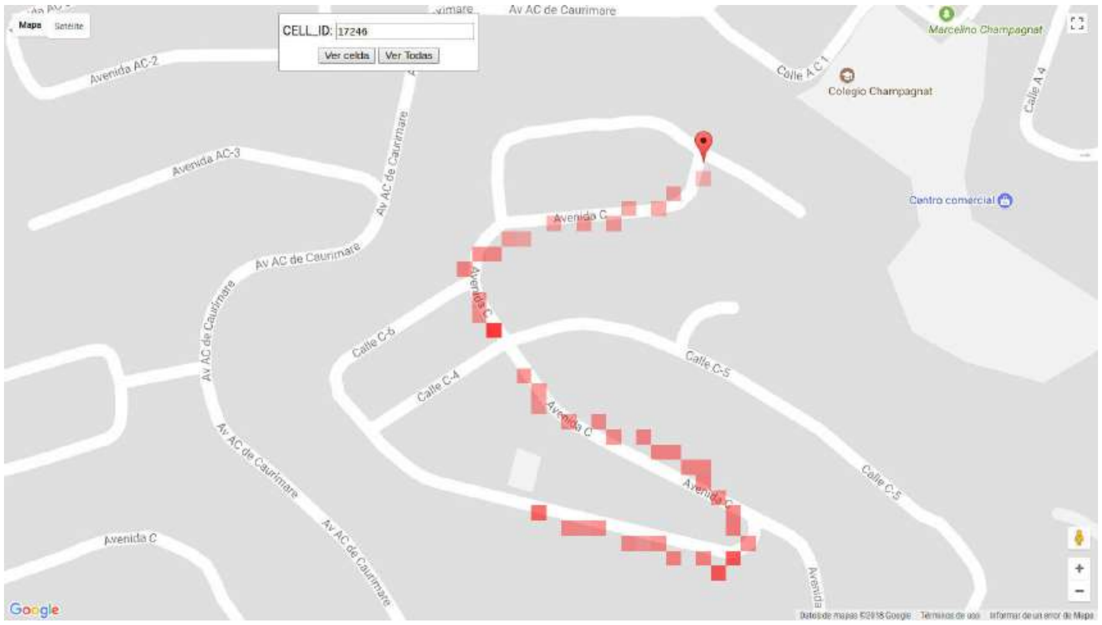

Figure 4.3. Filtering of cell 12481, in the route of 5Km / h

In this tour, six voice calls were made, of which only 1 call did not go through. In the corresponding database you have all the data related to the Drive

Test, in the tool you only have the coverage, the cell filter and the missed calls. distance and approximate speed of 1.05 km and 40 km / h respectively. The second route had an approximate distance and speed of 1.05 km and 20 km / h respectively, the result of the route can be seen in figure 4.4.

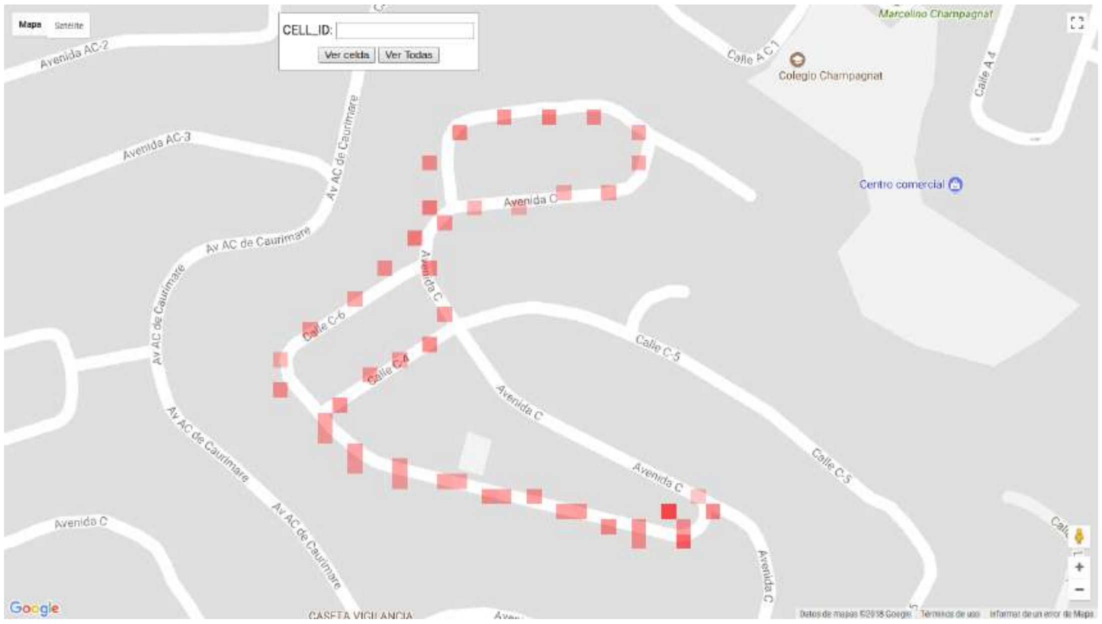

Figure 4.4. Route of the Drive Test at 20 Km / h

In this tour, 3 calls were made, all successful, however it can be seen that the points are separated by a greater distance than the previous tour, this is because the system is not capable of obtaining data at such a fast rate. However, it presents a good alternative of speed with respect to the 5 km / h that were used in the previous route.

The third trip had an approximate distance and speed of 1.05 km and 40 km / h respectively, the result of the trip can be seen in Figure 4.5.

In this route it can be observed that 40 km / h is not a desired speed for the system, because the points are widely spaced between them, as a consequence it is not possible to cover the route successfully. Something to highlight in this tour is that during this the same BTS was maintained, that is, no relay was carried out, unlike the other two tours. Two calls were made, both successful.

The fourth trip had an approximate distance and speed of 1.85 km and 40 km / h respectively, the result of the trip is observed in figure 4.6

This tour was made by passing twice on the same route, to have a greater number of samples. Four calls were made on this tour, all successful

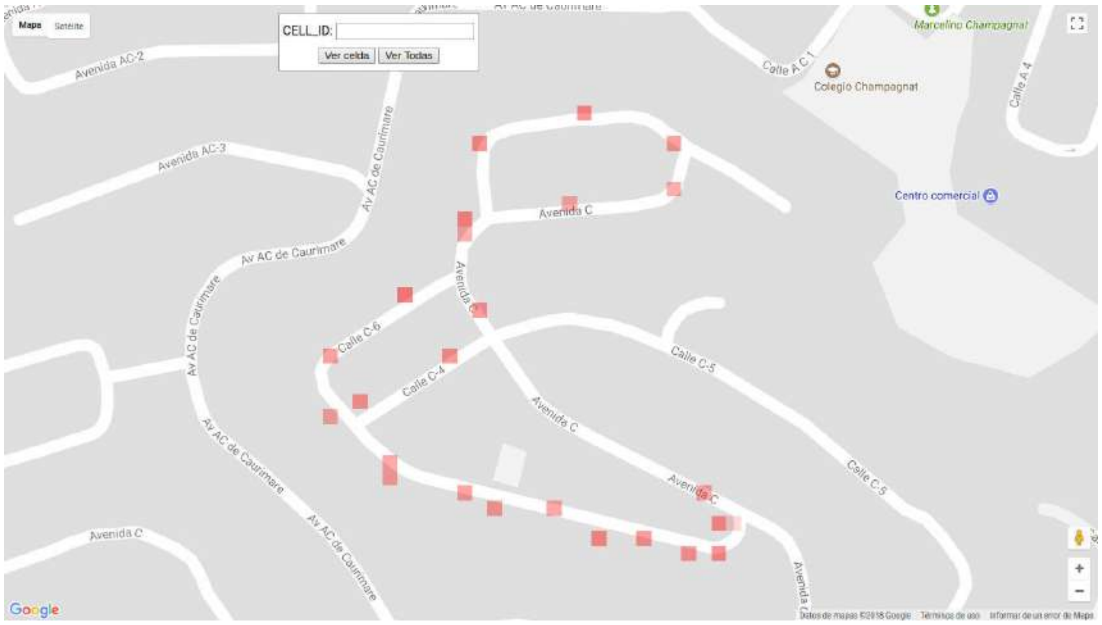

Figure 4.5. Route of the Drive Test at 40 Km / h

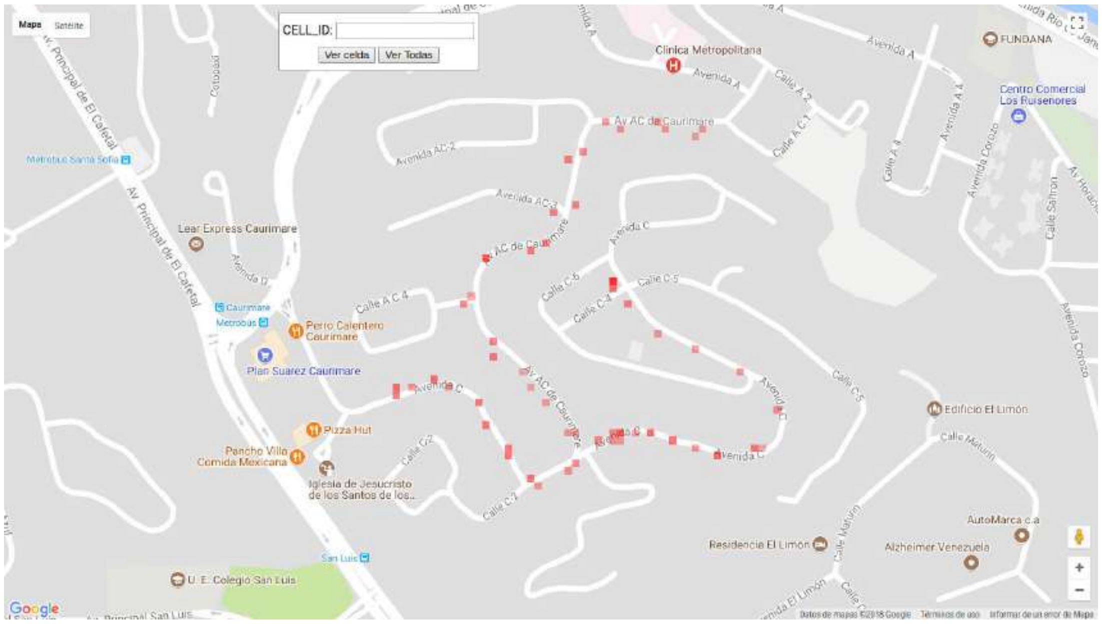

Figure 4.6. Route of the Drive Test at 40 Km / h external

CONCLUSIONS AND RECOMMENDATIONS

The main objective of this work was to implement a Drive Test system based on the GSM SIM900 module, the XM37-1612 GPS chip and the ESP8266 WIFI module, where not only the report that is generated is important, but also the time. that it takes to capture the data. From the work carried out and the results obtained, it can be concluded that:

- Thanks to the study carried out on the Drive Test, the GPS and the GSM module, it was possible to implement a C ++ library capable of obtaining the coverage data of a cellular operator through a route over a certain area. GPS data (location and time) during the tour.
- Thanks to the study carried out on the ESP8266, it was possible to implement a wireless solution of the Drive Test system, which allows adding a greater number of GSM modules without the need to use physical connections, thus increasing its scalability.

- The coverage area of each cell and the intensity of the signal along the route were estimated, thanks to the signal quality reporting routines and cell parameters performed.
- Routine voice calls were made with the GSM module. This made it possible to verify the relay mechanisms and the fault zones of the voice calls.
- A server capable of receiving, storing and presenting the data obtained from the Drive Test was developed.
- Yes, it is feasible to implement a Drive Test system, wirelessly, with limited functionalities with the GSM SIM900 module, the XM37-1612 GPS and the ESP8266 WIFI module.

For future works and extensions or improvements to the project carried out, the following recommendations are proposed:

- Use a GSM module capable of using the extended engineering mode, which includes a larger amount of cell data, such as the SIM808. This will allow the Drive test to be more reliable.
- Use a GPS chip that has a higher location update rate. Carry out the adaptation of the
- system for the third and fourth generation modules.
- Create a concurrent data server, so that it can serve several WiFi modules at the same time.

REFERENCES

[1] S. Redl, M. Weber, and M. Oliphant, *An Introduction to GSM.* ARTECH HOUSE, INC., 1995.

[2] “Mobile communication systems,” http://people.seas.harvard.edu/~jones/cscie129/nu_ lectures / lecture7 / cellular / cell_1.html, accessed: 2017-05-9.

[3] “Sistema celular,” https://infograph.venngage.com/p/124795/sistema-celular, accessed: 2017-05-15.

[4] "Question: Write short note on umbrella cell approach." http://www.ques10.com/p/ 11773 / write-short-note-on-umbrella-cell-approach-1 /, accessed: 2017-05-18.

[5] HE Moreno Trujillo, “PARTICIPATION IN THE OPTIMIZATION OF A NETWORK UMTS, ”2010.

[6] ETSI, “Digital cellular telecommunications system (Phase 2+); Radio subsystem link control (GSM 05.08 Version 5.1.0), ”pp. 16-17, 1996.

[7] M. Díaz, “Collection of coverage data,” 2015.

[8] Shanghai SIMCom Wireless Solutions Ltd, "SIM900 Hardware Design V2.0," p. 47, 2010. [Online]. Available: ftp://imall.iteadstudio.com/IM120417009íritu_samplesIComSat/ DOC {_} SIM900 {_} HardwareDesign {_} V2.00.pdf

[9] http://img01.cp.aliimg.com/imgextra/i3/38838966/T2BHJ3XhhXXXXXXXXX_ !! 38838966.jpg, accessed: 2017-12-9.

[10] SIM900A Datasheet, “SIM900_ AT Command,” 2009.

[11] L. Shenzhen Simplewe Technology Co., "Product speci fi cation XM37-1612," pp. 21–22, 1993. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/ 09613219308727250

[12] http://seta43.blogspot.com/2015/10/visualizar-datos-gps-xm37-1612-gambas_11. html, accessed: 2017-12-9.

[13] "ESP8266 | techtutorialsx. " [On-line]. Available: https://techtutorialsx.com/2016/02/ 14 / esp8266 /

[14] "D1 mini [WEMOS Electronics]." [On-line]. Available: https://wiki.wemos.cc/products: d1: d1 {_} mini

[15] E. Systems and IOT Team, "ESP8266EX Datasheet," 2015.

[16] "Introduction to Watchdog Timers | Embedded. " [On-line]. Available: https://www.embedded.com/electronics-blogs/beginner-s-corner/4023849/Introduction-to-Watchdog-Timers

[17] “ESP8266: Watchdog functions | techtutorialsx. " [On-line]. Available: https: //techtutorialsx.com/2017/01/21/esp8266-watchdog-functions/

[18] "Arduino Memories | Memories of an Arduino | Adafruit Learning System. " [On-line]. Available: https://learn.adafruit.com/memories-of-an-arduino/arduino-memories

[19] https://learn.adafruit.com/memories-of-an-arduino/measuring-free-memory, accessed: 2017-12-9.

APPENDIX A

Program code

The created library can be found at https://github.com/LuisAMM/Drive-test, where the codes of the tests carried out in this work are also found.

Header file: AT library 1.h

```
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>

typedef struct Info_t
{
        uint16_t arfcn_bcch [ 7 ];          // Absolute frequency channel number
        of the BCCH
        uint8_t rxl [ 7 ];                  // Power level of the received signal // Quality of the
        uint8_t rqx;                        received signal
        uint16_t mcc [ 7 ];                 // Country code //
        uint8_t mnc [ 7 ];                  Network code
        uint8_t basic [ 7 ];                // Identity code of the base station // Cell identifier
        double cellid [ 7 ];
        uint8_t rla;                        // Minimum transmission level to access
        to network
        uint8_t txp;                        // Maximum transmission level in the CCCH //
        double      lac [ 7 ];              Location area
        uint8_t     TA;                     // Time advance

        uint8_t RSSI;                       // Signal level // Bit error
        uint8_t BER;                        rate

        unsigned int Lost calls;                    // counter for missed calls // counter for
        unsigned int Made_calls;                    calls made

        double latitude;                    // Longitude of the coordinate //
        double longitude;                   Latitude of the coordinate // hour
        double HMS;                         minute second
        uint DMA;                           //day, month and year
        double speedOTG;                    // speed over ground

} Info;
```

```
/*
 **
 ↪ ------------------------------------------------------------------
 **       Name            : Size
 **       Function        : Get the size of a character string
 ↪ generic.
 **       Parameters: Generic character string. Returns
 **                       : The size of the character string.
 **
 ↪ ------------------------------------------------------------------
*/

int size ( char * accountant);
/*
 **
 ↪ ------------------------------------------------------------------
 **       Name            : verify_call
 **       Function        : Check the value of the missed call variable
 ↪ located in the data structure.
 **       Parameters: -
 **       Returns         : The number of missed calls.
 **
 ↪ ------------------------------------------------------------------
*/

unsigned int verify_call ();
/*
 **
 ↪ ------------------------------------------------------------------
 **       Name            : Comparator
 **       Function        : Compare two character strings.
 **       Parameters: two character strings Returns
 **                       : NULL if one does not contain the other and a pointer to
 ↪ the expected response if it contains it
 **
 ↪ ------------------------------------------------------------------
*/

char * comparator ( char * answer, char * expected_response);
/*
 **
 ↪ ------------------------------------------------------------------
 **       Name            : Comparator
 **       Function        : Compare two character strings and return the
 ↪ position where the expected response ends
 * *        Parameters: two character strings and a pointer to a 16-bit unsigned integer
 ↪
 * *        Returns          : NULL if one does not contain the other and a pointer to
 ↪    the expected response if it contains it.
 * *
 ↪    -----------------------------------------------------------------
* /

char * comparator ( char * answer, char * expected_response, uint16_t * w);
/ *
 * *
 ↪    -----------------------------------------------------------------
 * *        Name             : Send_command
 * *        Function         : Send AT command to SIM900
 * *        Parameters: Character string that contains the command AT Returns
 * *                         : -
 * *
 ↪    -----------------------------------------------------------------
* /

void Send_command ( char * CommandAT);
/ *
 * *
 ↪    -----------------------------------------------------------------
 * *        Name             : Receive_Serial
 * *        Function         : Read from the serial buffer and store it in the
 ↪    response string
 * *        Parameters: character string used to store, timer, position in the array
 ↪
 * *        Returns          : -
 * *
 ↪    -----------------------------------------------------------------
* /

void Receive_Serial ( char * response, unsigned long timeout, uint16_t * x);
/ *
 * *
 ↪    -----------------------------------------------------------------
 * *        Name             : Search_inicio
 * *        Function         : Find the beginning of the message.
 * *        Parameters: Header: header of the message stored in response.
 ↪
 * *        Returns          : Position where the heading ends
 * *
 ↪    -----------------------------------------------------------------
* /
```

```
uint16_t Search_start ( char * header, char * response);
/*
  **
  ↪ -----------------------------------------------------------------
  **        Name             : get_CENG
  **        Function         : Obtain and store in the structure the data of
  ↪ system engineering
  **        Parameters: -
  **        Returns          : if successful 1, if not, the team responds
  ↪ returns 0, if it does not get the start -1
  **
  ↪ -----------------------------------------------------------------
*/

int8_t get_CENG ();
/*
  **
  ↪ -----------------------------------------------------------------
  **        Name             : separator
  **        Function         : Get the data from the engineering message to be
  ↪ stored in a 32-bit variable.
  **        Parameters: token (delimiter of the values), output (variable in which the value will
  ↪ be stored),
                                i (position being read), response (string
  ↪ character containing the whole answer)
  **        Returns          : Position of the next value.
  **
  ↪ -----------------------------------------------------------------
*/

uint16_t separator ( char token, uint * Exit, uint16_t i, char * response);
/*
  **
  ↪ -----------------------------------------------------------------
  **        Name             : separator
  **        Function         : Get the data from the engineering message to be
  ↪ stored in a 16-bit variable.
  **        Parameters: token (delimiter of the values), output (variable in which the value will
  ↪ be stored),
                                i (position being read), response (string
  ↪ character containing the whole answer)
  **        Returns          : Position of the next value.
  **
  ↪ -----------------------------------------------------------------
*/
```

```
uint16_t separator ( char token, uint16_t * Exit, uint16_t i, char *
    response);
/*
 **
    -----------------------------------------------------------------
 **        Name            : separator
 **        Function        : Get the data from the engineering message to be
    stored in an 8-bit variable.
 **        Parameters: token (delimiter of the values), output (variable in which the value will
    be stored),
                                  i (position being read), response (string
    character containing the whole answer)
 **        Returns         : Position of the next value.
 **
    -----------------------------------------------------------------
*/

uint16_t separator ( char token, uint8_t * Exit, uint16_t i, char *
    response);
/*
 **
    -----------------------------------------------------------------
 **        Name            : separator
 **        Function        : Get the data from the engineering message to be
    stored in a double.
 **        Parameters: token (delimiter of the values), output (variable in which the value will
    be stored),
                                  i (position being read), response (string
    character containing the whole answer)
 **        Returns         : Position of the next value.
 **
    -----------------------------------------------------------------
*/

uint16_t separator ( char token, double * Exit, uint16_t i, char * response);
/*
 **
    -----------------------------------------------------------------
 **        Name            : separatorHEX
 **        Function        : Get the data from the engineering message to be
    found in Hexadecimal.
 **        Parameters: token (delimiter of the values), output (variable in which the value will
    be stored),
                                  i (position being read), response (string
    character containing the whole answer)
 **        Returns         : Position of the next value.
 **
 ↪ ------------------------------------------------------------------------
*/

uint16_t HEX separator ( char token, double * Exit, uint16_t i, char *
 ↪ response);
/*
 **
 ↪ ------------------------------------------------------------------------
 **      Name        : separator
 **      Function    : Skip empty spaces
 **      Parameters: token (delimiter of the values), i (position that is being read),
 ↪
                              response (character string that contains the entire
 ↪ answer)
 **      Returns     : Position of the next value.
 **
 ↪ ------------------------------------------------------------------------
*/

uint16_t separator ( char token, uint16_t i, char * response);
/*
 **
 ↪ ------------------------------------------------------------------------
 **      Name        : separador_signo
 **      Function    : get a character from the message.
 **      Parameters: output (Variable in which the sign will be stored) i (position that is being
 ↪ read),
                              response (character string that contains the entire
 ↪ answer)
 **      Returns     : Position of the next value.
 **
 ↪ ------------------------------------------------------------------------
*/

uint16_t separator_sign ( char * Exit, uint16_t i, char * response);
/*
 **
 ↪ ------------------------------------------------------------------------
 **      Name        : Call
 **      Function    : Call a number
 **      Parameters: Character string (number) Returns
 **                  : 1 if the call is successful, 0 if it is not successful.
 **
 ↪ ------------------------------------------------------------------------
*/
```

```
int Call ( char * number);
/*
 **
 ↪ ------------------------------------------------------------
 **        Name         : Trancar
 **        Function     : End call
 **        Parameters: -
 **        Returns      : -
 **
 ↪ ------------------------------------------------------------
*/

void Lock ();
/*
 **
 ↪ ------------------------------------------------------------
 **        Name         : Verify_CSQ
 **        Function     : get the value of the flag flag_CSQ
 **        Parameters: -
 **        Returns      : flag value flag_CSQ
 **
 ↪ ------------------------------------------------------------
*/

uint8_t verify_CSQ ();
/*
 **
 ↪ ------------------------------------------------------------
 **        Name         : SendATcommand
 **        Function     : Send AT command and verify that the
 ↪ answer agrees with expectations
 **        Parameters: AT command, expected_response, timer, character string that will
 ↪ store the response
 **        Returns      : 1 if the expected response was received, 0 if not.
 **
 ↪ ------------------------------------------------------------
*/

uint8_t sendATcommand ( char * CommandAT, char * Expected_response, unsigned
 ↪ long timeout, char * response);
/*
 **
 ↪ ------------------------------------------------------------
 **        Name         : Start variables
 **        Function     : initialize variables and reserve memory
 ↪ the structure
 **        Parameters: -
 * *          Returns            : -
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void start_variables ();
/ *
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * *          Name               : restart_RSSI_BER
 * *          Function           : Reset the values   of BER, RSSI and of the
 ↪    flag flag_CSQ
 * *          Parameters: -
 * *          Returns            : -
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void restart_RSSI_BER ();
/ *
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * *          Name               : begin_myserial
 * *          Function           : Set the size of the serial reception buffer.
 * *          Parameters: Send configuration messages to both GPS and SIM900 for calls
 ↪
 * *          Returns            : -
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void begin_myserial ();
/ *
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * *          Name               : begin_myserial_Datos
 * *          Function           : Set the size of the serial reception buffer.
 * *          Parameters: Send configuration messages to both GPS and SIM900 for data
 ↪
 * *          Returns            : -
 * *
 ↪     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void begin_myserial_Data ();
/ *
```

```
 **
↪ ------------------------------------------------------------------
 **      Name          : ftp_get
 **      Function      : request a file from the FTP server
 **      Parameters: -
 **      Returns       : -
 **
↪ ------------------------------------------------------------------
*/

void ftp_get ();
/*
 **
↪ ------------------------------------------------------------------
 **      Name          : Send_WiFi
 **      Function      : Send structure via WiFi
 **      Parameters: Object WiFiClient client Returns
 **                    : -
 **
↪ ------------------------------------------------------------------
*/

void Send_WiFi (WiFiClient client);
/*
 **
↪ ------------------------------------------------------------------
 **      Name          : DegToDec
 **      Function      : Convert place value from degrees to decimal
 **      Parameters: sign and position in degrees Returns
 **                    : -
 **
↪ ------------------------------------------------------------------
*/

void DegToDec ( char sign, double * position);
/*
 **
↪ ------------------------------------------------------------------
 **      Name          : Verify_GPS
 **      Function      : Verify that the GPS is located and
↪ working
 **      Parameters: -
 **      Returns       : 1 if it is located, 0 if not
 **
↪ ------------------------------------------------------------------
*/
```

```
uint8_t Verify_Gps ();
/*
 **
 ↪ --------------------------------------------------------------
 **      Name           : Test_GPS
 **      Function       : Verify that the GPS works correctly and
 ↪ wait until it works
 **      Parameters: -
 **      Returns        : 1 if it works properly
 **
 ↪ --------------------------------------------------------------
*/

int8_t Test_GPS ();
/*
 **
 ↪ --------------------------------------------------------------
 **      Name           : get_data
 **      Function       : Get the relevant values  from the GPS and
 ↪ store them in the structure
 **      Parameters: -
 **      Returns        : -
 **
 ↪ --------------------------------------------------------------
*/

void get_data ();
/*
 **
 ↪ --------------------------------------------------------------
 **      Name           : get_CSQ
 **      Function       : Get and store in the structure the values  of
 ↪ RSSI and BER
 **      Parameters: -
 **      Returns        : -
 **
 ↪ --------------------------------------------------------------
*/

void get_CSQ ();
/*
 **
 ↪ --------------------------------------------------------------
 **      Name           : separator
 **      Function       : Get the data from the GPS message being
 ↪ stored in a 32-bit unsigned variable.
 * *         Parameters: token (delimiter of the values), output (variable in which the value will
 ↪   be stored),
                                  my_Serial (Object that receives the GPS information): -
 * *         Returns
 * *
 ↪   ------------------------------------------------------------------
*/

void separator ( char token, uint * output, SoftwareSerial * my_Serial);
/*
 * *
 ↪   ------------------------------------------------------------------
 * *         Name              : separator
 * *         Function          : Get the data from the GPS message being
 ↪   stored in a 16-bit unsigned variable.
 * *         Parameters: token (delimiter of the values), output (variable in which the value will
 ↪   be stored),
                                  my_Serial (Object that receives the GPS information): -
 * *         Returns
 * *
 ↪   ------------------------------------------------------------------
*/

void separator ( char token, uint16_t * output, SoftwareSerial * my_Serial);
/*
 * *
 ↪   ------------------------------------------------------------------
 * *         Name              : separator
 * *         Function          : Get the data from the GPS message being
 ↪   stored in an 8-bit unsigned variable.
 * *         Parameters: token (delimiter of the values), output (variable in which the value will
 ↪   be stored),
                                  my_Serial (Object that receives the GPS information): -
 * *         Returns
 * *
 ↪   ------------------------------------------------------------------
*/

void separator ( char token, uint8_t * output, SoftwareSerial * my_Serial);
/*
 * *
 ↪   ------------------------------------------------------------------
 * *         Name              : separator
 * *         Function          : Get the data from the GPS message being
 ↪   stored in a double.
 * *         Parameters: token (delimiter of the values), output (variable in which the value will
 ↪   be stored),
                                    my_Serial (Object that receives the GPS information): -
 * *        Returns
 * *
 ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void separator ( char token, double * output, SoftwareSerial * my_Serial);

/ *
 * *
 ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * *        Name            : separator
 * *        Function        : Skip empty spaces
 * *        Parameters: token (delimiter of the values), mySerial (Object that receives the
 ↪   information from the GPS)
 * *        Returns         : -
 * *
 ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void separator ( char token, SoftwareSerial * my_Serial);
/ *
 * *
 ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 * *        Name            : separador_signo
 * *        Function        : Store the value of the sign
 * *        Parameters: Output (variable in which the value will be stored),
                                    mySerial (Object that receives the GPS information): -
 * *        Returns
 * *
 ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void separator_sign ( char * output, SoftwareSerial * my_Serial);
```

Function file: AT library 1.cpp

```
#include "AT_libreria_1.h"

# define TIME 1
# define BUFFER_RX 400

SoftwareSerial mySerial_GPS (D5, D6, false , 100 );              // RX, TX

char response [BUFFER_RX] = {};           // Buffer that stores the response of the
 ↪   SIM900 in engineering mode
```

```
char response_CSQ [ fifty ] = {};            // Buffer that stores the response of the
    ↪ get_CSQ function
Info * Dtest;                                // Pointer to main structure // Flag that tells the
uint8_t flag_CSQ = 0 ;                       program that
    ↪ must take the signal quality measurements

int size ( char * accountant)
{
        int i = 0 ;
        while ( counter [i] ! = '\ 0' )
        {
                i ++ ;
        }
        return i;
}

unsigned int verify_call ()
{
        return Dtest -> Lost calls;
}

char * comparator ( char * answer, char * expected_response) {

      int i = 0 , k;
      int j = 0 , l;
      char aux3 = 'z' ;
      k = size (answer);
      l = size (expected_response);

      if ( k < l)
        return Null ;

      while ( aux3 ! = '\ 0' && expected_response [j] ! = '\ 0' )
      {
            aux3 = answer [i];
            if ( aux3 == expected_response [j])
            {
                  i ++ ;
                  j ++ ;
            }
            else
            {
                  i ++ ;
                  j = 0 ;
            }
      }
      if ( j == l)
```

```
        return expected_response;
      else
        return Null ;
}

char * comparator ( char * answer, char * expected_response, uint16_t * w) {

      int i = 0 , k;
      int j = 0 , l;
      char aux3 = 'z' ;
      k = size (answer);
      l = size (expected_response);

      if ( k < l)
        return Null ;

      while ( aux3 ! = '\ 0' && expected_response [j] ! = '\ 0' )
      {
            aux3 = answer [i];
            if ( aux3 == expected_response [j])
            {
                  i ++ ;
                  j ++ ;
            }
            else
            {
                  i ++ ;
                  j = 0 ;
            }
            ESP.wdtFeed ();
      }
      *w = i;
      if ( j == l)
        return expected_response;
      else
        return Null ;
}

void Send_command ( char * CommandAT)
{
                Serial.println (CommandAT);
}

void Receive_Serial ( char * response, unsigned long timeout, uint16_t * x) {

                unsigned long before = millis ();
```

```
    // Waiting cycle while a value arrives by serial or the
 ↪  timer.
        while ( Serial.available () <= 0 && (before - millis ()) <= timeout)
                ESP.wdtFeed ();

        while ( Serial.available () > 0 && * x < BUFFER_RX) {
                response [ * x] = Serial.read ();
                ( * x) ++ ;
                delay (TIME);
        }
}

uint16_t Search_start ( char * header, char * response) {

                uint16_t x = 0 ;
                if ( comparator (response, header, & x) ! = Null )
                                return x;
                else
        return - one ;
}

int8_t get_CENG ()
{
        uint16_t j = 1 ;
        uint16_t i = 0 , aux;
        // Is the command AT + CENG sent? to have the engineering parameters // If OK is not
        received, discard the measurement
        if ( sendATcommand ( "AT + CENG?" , "OKAY" , 1000 , response) ! = 1 )
                        return 0 ;
        // Find the header + CENG: 0, "
        i = Search_start ( "+ CENG: 0, \ " " , response);
        // If position 0 or 1 discard measurement
        aux = i;
        if ( aux == 0 || aux == - 1 )
        {
                return - one ;
        }
        // if position is buffer size, discard measurement
        if ( i == BUFFER_RX -one )
                        return - one ;

        // the values   of the cell that serves the mobile are stored in the structure
 ↪
        i = separator( ',' , & (Dtest -> arfcn_bcch [ 0 ]), i, response); i = separator( ',' , & (Dtest
        -> rxl [ 0 ]), i, response); i = separator( ',' , & (Dtest -> rqx), i, response); i = separator(
        ',' , & (Dtest -> mcc [ 0 ]), i, response);
        i = separator( ',' , & (Dtest -> mnc [ 0 ]), i, response); i = separator( ',' , & (Dtest
        -> basic [ 0 ]), i, response); i = separatorHEX ( ',' , & (Dtest -> cellid [ 0 ]), i,
        response); i = separator( ',' , & (Dtest -> rla), i, response); i = separator( ',' , &
        (Dtest -> txp), i, response); i = separatorHEX ( ',' , & (Dtest -> lac [ 0 ]), i,
        response); i = separator( '\ "' , & (Dtest -> TA), i, response);



        while ( j <= 6 )
        {       // values   of neighboring cells
                i = separator( '\ "' , i, response); i = separator( ',' , & (Dtest -> arfcn_bcch [j]), i,
                response); i = separator( ',' , & (Dtest -> rxl [j]), i, response); i = separator( ',' , &
                (Dtest -> bsic [j]), i, response); i = separatorHEX ( ',' , & (Dtest -> cellid [j]), i,
                response); i = separator( ',' , & (Dtest -> mcc [j]), i, response); i = separator( ',' ,
                & (Dtest -> mnc [j]), i, response); i = separatorHEX ( '\ "' , & (Dtest -> lac [j]), i,
                response); j ++ ;



        }
        return one ;
}

                        / * ENGINEERING SEPARATORS * /

uint16_t separator ( char token, uint * Exit, uint16_t i, char * response) {

        int j = 0 ;
        char aux [ 25 ];
        char aux2;
        do
        {
                aux2 = response [i];
                aux [j] = aux2;
                j ++ ;
                i ++ ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;

        *Exit = atoi (aux);
        return i;
}

uint16_t separator ( char token, uint16_t * Exit, uint16_t i, char *
        response)
{
        int j = 0 ;
        char aux [ 25 ];
        char aux2;
        do
        {
                aux2 = response [i];
                aux [j] = aux2;
                j ++ ;
                i ++ ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;
        *Exit = atoi (aux);
        return i;
}

uint16_t separator ( char token, uint8_t * Exit, uint16_t i, char * response) {

        int j = 0 ;
        char aux [ 25 ];
        char aux2;
        do
        {
                aux2 = response [i];
                aux [j] = aux2;
                j ++ ;
                i ++ ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;

        *Exit = atoi (aux);
        return i;
}

uint16_t separator ( char token, double * Exit, uint16_t i, char * response) {

        int j = 0 ;
        char aux [ 25 ];
        char aux2;
        do
        {
                aux2 = response [i];
                aux [j] = aux2;
                j ++ ;
                i ++ ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;

        *Exit = atof (aux);
        return i;
}

uint16_t HEX separator ( char token, double * Exit, uint16_t i, char *
        response)
{
        int j = 0 ;
        char aux [ 25 ];
        char aux2;
        do
        {
                aux2 = response [i];
                aux [j] = aux2;
                j ++ ;
                i ++ ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;

        *Exit = ( double ) strtol (aux, Null , 16 );
        return i;
}

uint16_t separator ( char token, uint16_t i, char * response) {

        char aux;
        int j = 0 ;
        char aux2;
        do
        {
                aux2 = response [i];
                i ++ ;
        }
        while ( aux2 ! = token);
        return i;
}

uint16_t separator_sign ( char * Exit, uint16_t i, char * response) {

        *Exit = response [i];
        i ++ ;
        return i;
}
```

```
int Call ( char * number)
{
        char command[ 40 ] = {};
        uint8_t i = 0 ;
        sprintf (command, "ATD% s;" ,number);
        if ( sendATcommand (command, "OKAY" , 1000 , response) == 1 )
                {
                                i = 1 ;
                }
        (Dtest -> Calls_made) ++ ;
        return i;
}

void Lock ()
{
                sendATcommand ( "ATH" , "OKAY" , 1000 , response);
}

uint8_t verify_CSQ ()
{
                return flag_CSQ;
}

uint8_t sendATcommand ( char * CommandAT, char * Expected_response, unsigned
  ↪     long timeout, char * response)
{
    // The AT command is sent
                Send_command (CommandAT);
    // Start the timer
                long unsigned before = millis ();
                uint8_t answer = 0 , flag = 0 ;
                uint16_t x = 0 ;
    // the storage buffer is emptied
                memset (response, '\ 0' , BUFFER_RX);

                do
                {
                                Receive_Serial (response, 1000 , & x);
                                // we compare if the received response contains NO
  ↪     CARRIER
        if ( comparator (response, "NO CARRIER" ) ! = Null && flag == 0 )
                                {
            flag = 1 ;
                                                (Dtest -> Lost calls) ++ ;
                                }
        // we compare if the received response contains MO CONNECTED
```

```
                              if ( comparator (response, "MO CONNECTED" ) ! = Null )
                              {
                                      flag_CSQ = 1 ;
                              }
          // we compare if the received response contains the expected response
                              if ( comparator (response, expected_response) ! = Null )
                              {
                                      answer = 1 ;
                              }
          ESP.wdtFeed ();
              } while ( answer == 0 && ((millis () - before) <= timeout));
        // place delimiter \ 0
              if ( x ! = 400 )
                      response [x] = 0 ;
              return answer;
      }

      void start_variables ()
      {
          Dtest = (Info * ) malloc ( sizeof ( Info)); Dtest -> HMS
          = 0 ;
          Dtest -> latitude = 0 ;
          Dtest -> length = 0 ;
          Dtest -> speedOTG = 0 ;
          Dtest -> DMA = 0 ;
          Dtest -> rqx = 0 ;
          Dtest -> rla = 0 ;
          Dtest -> txp = 0 ;
          Dtest -> TA = 0 ;

          Dtest -> RSSI = 0 ;
          Dtest -> BER = 0 ;

          Dtest -> Lost calls = 0 ;
          Dtest -> Calls_made = 0 ;
          for ( int i = 0 ; i <7 ; i ++ ) {

              Dtest -> arfcn_bcch [i] = 0 ;
              Dtest -> rxl [i] = 0 ;
              Dtest -> basic [i] = 0 ;
              Dtest -> cellid [i] = 0 ;
              Dtest -> mcc [i] = 0 ;
              Dtest -> mnc [i] = 0 ;
              Dtest -> the CI] = 0 ;
          }
      }
```

```
void restart_RSSI_BER ()
{
          flag_CSQ = 0 ;
     Dtest -> RSSI = 0 ;
     Dtest -> BER = 0 ;
}

void begin_myserial ()
{
     Serial.setRxBufferSize (BUFFER_RX);
     mySerial_GPS.begin ( 9600 );
     delay ( 10 );
     // GPS command to use $ GPGGA messages

     mySerial_GPS.print (F ( "$ PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0 * 29 \ r \ n " ));
     // GPS command to set its transmission rate to 19200 baud

     mySerial_GPS.print (F ( "$ PMTK251,19200 * 22 \ r \ n " ));
     mySerial_GPS.begin ( 19200 );
     mySerial_GPS.enableRx ( false );
     delay ( 10 );
     while ( mySerial_GPS.available () > 0 ) mySerial_GPS.read ();

     // we activate the engineering mode
     sendATcommand ( "AT + CENG = 1,1" , "OKAY" , 1000 , response);
     // we activate the connection messages of the call
     sendATcommand ( "AT + MORING = 1" , "OKAY" , 1000 , response);
}

void begin_myserial_Data ()
{
     Serial.setRxBufferSize (BUFFER_RX);
     mySerial_GPS.begin ( 9600 );
     delay ( 10 );
     // GPS command to use $ GPGGA messages

     mySerial_GPS.print (F ( "$ PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0 * 29 \ r \ n " ));
     // GPS command to set its transmission rate to 19200 baud

     mySerial_GPS.print (F ( "$ PMTK251,19200 * 22 \ r \ n " ));
     mySerial_GPS.begin ( 19200 );
     mySerial_GPS.enableRx ( false );
     delay ( 10 );
     while ( mySerial_GPS.available () > 0 ) mySerial_GPS.read ();
     // The FTP connections are configured
     sendATcommand ( "AT + CENG = 1,1" , "OKAY" , 1000 , response);
     sendATcommand ( "AT + SAPBR = 3.1, \ " Contype \ " , \ " GPRS \ " " , "OKAY" , 1000 , response);
```

```
        sendATcommand ( "AT + SAPBR = 3.1, \ " APN \ " , \ " internet.movistar.ve \ " " , "OKAY" ,
        1000 , response);
        sendATcommand ( "AT + SAPBR = 1,1" , "OKAY" , 1000 , response);
        sendATcommand ( "AT + FTPCID = 1" , "OKAY" , 1000 , response);
        sendATcommand ( "AT + FTPSERV = \ " speedtest.tele2.net \ " " , "OKAY" , 1000 , response);
        sendATcommand ( "AT + FTPUN = \ " anonymous \ " " , "OKAY" , 1000 , response); sendATcommand
        ( "AT + FTPPW = \ " 1414425 \ " " , "OKAY" , 1000 , response); sendATcommand ( "AT + FTPGETNAME
        = \ " 1KB.zip \ " " , "OKAY" , 1000 , response); sendATcommand ( "AT + FTPGETPATH   = \ " / \ " " , "OKAY" ,
        1000 , response);
}

void ftp_get ()
{
        int j = 1 ;
        sendATcommand ( "AT + FTPGET = 1" , "+ FTPGET: 1,1" , 20000 , response);
        if ( sendATcommand ( "AT + FTPGET = 2,1024" , "+ FTPGET = 2,1024" , 5000 , response) == 0 ) {

                while ( Serial.available ())
                {
/ *                             response [0] = Serial.read ();
                        if (response [0] == 'O')
                                response [1] = Serial.read ();
                                        if (response [1] = 'K')
                                        {
                                                Serial.println (F ("fine"));
                                                break;
                                        }
* /                     Serial.print (( char ) Serial.read ());
                }
        }
}

void Send_WiFi (WiFiClient client)
{
        client.write (( uint8_t * ) Dtest, sizeof ( * Dtest));
}

void DegToDec ( char sign, double * position) {

        double aux, aux2;
        int aux3;
        aux3 = * position / 100 ;
        aux = * position - aux3 * 100 ;

        aux2 = ( double ) aux3 + aux /60.0 ;

        if ( sign == 'W' | | sign == 'S' )
        *position = - aux2;
    else
        *position = aux2;
}

uint8_t Verify_Gps ()
{
          mySerial_GPS.enableRx ( true );

          int x = millis ();
          char aux, aux2;
          // Waiting cycle while a serial value arrives or the timer expires.

          while ( mySerial_GPS.available () <= 0 && (millis () - x) <= 3000 ) {

            ESP.wdtFeed ();
          }
          delay ( 10 );

          x = millis ();
          do
          {
            if ( mySerial_GPS.available () ! = 0 )
            {
            aux = ( char ) mySerial_GPS.read ();
            // We look for the header
            if ( aux == '$' )
            {
              // we skip the first 2 values
                      separator( ',' , & mySerial_GPS);
                      separator( ',' , & mySerial_GPS);
              // We store the value of the GPS status in aux2
                      sign_separator ( & aux2, & mySerial_GPS);
                      break ;
            }
            }
            else
              break ;
          }
          while ( mySerial_GPS.available () && (millis () - x) <= 5000 );
          mySerial_GPS.enableRx ( false );
          while ( mySerial_GPS.available () > 0 ) mySerial_GPS.read ();
          if ( aux2 == 'TO' )
            return one ;
          else
            return 0 ;
}
```

```
int8_t Test_GPS ()
{
        while ( mySerial_GPS.available () > 0 ) mySerial_GPS.read ();
        uint8_t answer = 0 ;

        answer = Check_Gps ();

        while ( answer ! = 1 )
        {
                Serial.print (F ( "." ));
                answer = Check_Gps ();

        }
        return one ;
}

void get_data ()
{
    mySerial_GPS.enableRx ( true );

    int x = millis ();
    char aux, aux2;
    uint8_t i = 0 ;

    // Waiting cycle while a value arrives by serial or the
 ↳      timer.
    while ( mySerial_GPS.available () <= 0 && (millis () - x) <= 5000 ) {

        ESP.wdtFeed ();
    }
    delay ( 10 );
    x = millis ();
    do
    {
        if ( mySerial_GPS.available () ! = 0 )
        {
            aux = ( char ) mySerial_GPS.read ();
            if ( aux == '$' | | i ! = 0 ) // we look for the header
            {
                // Store the GPS values   in the data structure
                separator( ',' , & mySerial_GPS);
                separator( ',' , & (Dtest -> HMS), & mySerial_GPS);
                separator( ',' , & mySerial_GPS);
                separator( ',' , & (Dtest -> latitude), & mySerial_GPS);
                sign_separator ( & aux2, & mySerial_GPS);
                DegToDec (aux2, & (Dtest -> latitude));
                separator( ',' , & (Dtest -> longitude), & mySerial_GPS);
                sign_separator ( & aux2, & mySerial_GPS);
                DegToDec (aux2, & (Dtest -> longitude));
                separator( ',' , & (Dtest -> speedOTG), & mySerial_GPS);
                separator( ',' , & mySerial_GPS);
                separator( ',' , & (Dtest -> DMA), & mySerial_GPS);
                break ;
            }
        }
        else
            break ;
    }
    while ( mySerial_GPS.available () && (millis () - x) <= 5000 );
    mySerial_GPS.enableRx ( false );
    while ( mySerial_GPS.available ()) mySerial_GPS.read ();
}

void get_CSQ ()
{
                uint8_t i = 0 ;
                memset (response_CSQ, '\ 0' , fifty );

        sendATcommand ( "AT + CSQ" , "OKAY" , 1000 , response_CSQ);

        i = Search_start ( "+ CSQ:" , response_CSQ);
        i = separator( ',' , & (Dtest -> RSSI), i, response_CSQ);
        i = separator( '\ r' , & (Dtest -> BER), i, response_CSQ);
}

                / * GPS SEPARATORS * /

void separator ( char token, uint * output, SoftwareSerial * my_Serial) {

        int j = 0 ;
        char aux [ 25 ];
        char aux2;

        do
        {
                if ( my_Serial -> available () > 0 )
                {
                        delay (TIME);
                        aux2 = ( char ) my_Serial -> read ();
                        aux [j] = aux2;
                        j ++ ;
                }
                else
                break ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;

        *Exit = atoi (aux);
}

void separator ( char token, uint16_t * output, SoftwareSerial * my_Serial) {

        int j = 0 ;
        char aux [ 25 ];
        char aux2;

        do
        {
                if ( my_Serial -> available () > 0 )
                {
                        delay (TIME);
                        aux2 = ( char ) my_Serial -> read ();
                        aux [j] = aux2;
                        j ++ ;
                }
                else
                        break ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;
        *Exit = atoi (aux);
}

void separator ( char token, uint8_t * output, SoftwareSerial * my_Serial) {

      int j = 0 ;
        char aux [ 25 ];
        char aux2;

        do
        {
                if ( my_Serial -> available () > 0 )
                {
                        delay (TIME);
                        aux2 = ( char ) my_Serial -> read ();
                        aux [j] = aux2;
                        j ++ ;
                }
                else
                break ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;
        *Exit = atoi (aux);
}

void separator ( char token, double * output, SoftwareSerial * my_Serial) {

        int j = 0 ;
        char aux [ 25 ];
        char aux2;

        do
        {
                if ( my_Serial -> available () > 0 )
                {
                        delay (TIME);
                        aux2 = ( char ) my_Serial -> read ();
                        aux [j] = aux2;
                        j ++ ;
                }
                else
                        break ;
        }
        while ( aux2 ! = token);
        aux [j] = '\ 0' ;

        *Exit = atof (aux);
}
void separator ( char token, SoftwareSerial * my_Serial) {

        char aux;
        do
        {
                if ( my_Serial -> available () > 0 )
                {
                        delay (TIME);
                        aux = my_Serial -> read ();
                }
                else
                        break ;
        }
        while ( aux ! = token);
}

void separator_sign ( char * output, SoftwareSerial * my_Serial)
```

```
{
        if ( my_Serial -> available () > 0 )
        {
                delay (TIME);
                *Exit = ( char ) my_Serial -> read ();
        }
        if ( my_Serial -> available () > 0 )
        {
                delay (TIME);
                my_Serial -> read ();
        }
}
```

Main file running the Drive Test: ceng gps.ino

```
#include "AT_libreria_1.h"

const char * ssid               = "luis-pc" ;           // Network to which the D1mini will connect
const char * password = "W6T4fAkP" ; // Network password

const char * server_ip = "10.42.0.1" ; // IP address of the server

uint16_t Port = 51002 ;                                 // Server port

IPAddress staticIP ( 10 , 42 , 0 , two );               // Static IP address of the device // Network
IPAddress gateway ( 10 , 42 , 0 , one );                gateway
IPAddress subnet ( 255 , 255 , 255 , 0 );               // Network mask

unsigned long previous;                                 // Timer
unsigned long before;                                   // Timer

WiFiClient client;                                      // Class that allows to connect by
  ↳    WIFI with a server
int i;
unsigned int aux = 4 ;

void setup () {

    ESP.wdtDisable ();                                  // WDT is disabled

    Serial.begin ( 9600 );
    while ( ! Serial) ESP.wdtFeed ();

    Serial.print (F ( "Connecting to" ));
    Serial.println (ssid);

    // we connect to the network
    WiFi.config (staticIP, gateway, subnet);
    WiFi.begin (ssid, password);

    // we check the status of the connection
    while ( WiFi.status () ! = WL_CONNECTED) {
        delay ( 1000 );
        Serial.print (F ( "." ));
    }

    Serial.println (F ( "" ));
    Serial.println (F ( "WiFi connected" ));

    before = millis ();
    // wait time to turn on the SIM900 module
    while ( millis () - before <= 10000 )
    {
        ESP.wdtFeed ();
    }

    begin_myserial ();
    start_variables ();

    Test_GPS ();
    if ( Serial.available () > 0 )
        Serial.read ();
}

void loop () {

    // The connection with the server is made
    if ( ! client.connect (server_ip, Port)) {
        ESP.wdtFeed ();
        Serial.println (F ( "" ));
        delay ( 1000 );
        return ;
    }
    // The GPS data is obtained
    get_data ();

    // It is verified if the call time ended or if the call was dropped
    if ( millis () - previous > = 120000 | | (aux - verify_call ()) ! = 0 ) {

        Trancar ();                         // the call hangs
        restart_RSSI_BER (); // rssi and ber values   are reset
        Call( "* 627568" );                 // the test number is called
        aux = check_call (); // aux is assigned the value of missed calls
        previous = millis ();                    // the timer is restarted
    }

    // the 10 engineering measures are taken, those of // quality
    and the report is sent to the server
    for ( i = 0 ; i <= 9 ; i ++ )
    {
        if ( get_CENG () ! = 1 )
            continue ;
        ESP.wdtFeed ();
        if ( verify_CSQ () == 1 )
            get_CSQ ();
        Send_WiFi (client);
    }
    // the connection with the server is terminated
    client.stop ();

}
```

File running storage server: tcpechoserver pc

```
#include     <stdio.h>
#include     <stdlib.h>
#include     <unistd.h>
#include     <netdb.h>
#include     <strings.h>
#include     <harp / inet.h>
#include     <sys / types.h>
#include     <sys / socket.h>
#include     <netinet / in.h>
#include     <string.h>
#include     <mysql.h> // library that allows us to use <ctype.h> // connections
#include     and queries with MySQL <time.h>
#include
#include     <ctype.h>

# define BACKLOG 2 / * The number of connections allowed * /
# define MAXDATASIZE 1000
# define BACKUP 10

// Structure that stores all the data of the Drive Test

typedef struct Info_t
{
        uint16_t arfcn_bcch [ 7 ];               // Absolute frequency channel number
  ↪     of the BCCH
        uint8_t rxl [ 7 ];                       // Power level of the received signal
        uint8_t rqx;                          // Quality of the received signal //
        uint16_t mcc [ 7 ];                   Country code
        uint8_t mnc [ 7 ];                    // Network code
        uint8_t basic [ 7 ];                  // Identity code of the base station // Cell identifier
        double cellid [ 7 ];
        uint8_t rla;                          // Minimum transmission level to access
  ↪     to network
        uint8_t txp;                          // Maximum transmission level in the CCCH //
        double       lac [ 7 ];               Location area
        uint8_t      TA;                      // Time advance

        uint8_t RSSI;                         // Signal level // Bit error
        uint8_t BER;                          rate

        unsigned int Lost calls;                          // counter for missed calls // counter for
        unsigned int Made_calls;                          calls made

        double latitude;                      // Longitude of the coordinate //
        double longitude;                     Latitude of the coordinate // hour
        double HMS;                           minute second
        uint DMA;                             //day, month and year
        double speedOTG;                      // speed over ground

} Info;


/ *
  * *
  ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  * *       Name             : IsANumber
  * *       Function         : Check if a character is a number
  * *       Parameters: character string Returns
  * *                        : 1 if it is a number, -1 if not
  * *
  ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

int IsANumber ( const char * number) {

        int i = 0 ;

        while ( number [i] ! = 0 )
        {
                if ( ! isdigit (number [i]))
                {
                    return - one ;
                }
```

```
            i ++ ;
        }
        return one ;
}

/ *
  * *
  ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  * *       Name            : Start_Socket
  * *       Function        : Performs initial socket configurations
  * *       Parameters: fd (Socket descriptor), sin_size (Size of the sockaddr_in structure)
  ↪
                               argv (Contains the listening port): Server
  * *       Returns         structure
  * *
  ↪   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

struct sockaddr_in Start_Socket ( int * fd, int * sin_size, const char * argv) {

    struct sockaddr_in server;
    / * for the server address information * /

    / * Then the call to socket () * /
    if ((( * fd) = socket (AF_INET, SOCK_STREAM, 0 )) == -1 ) {perror ( "error
        in socket () \ n " );
        exit ( -one );
    }

    server.sin_family = AF_INET;

    server.sin_port = htons (atoi (argv));
    / * Remember htons () from the "Conversions" section? =) * /

    server.sin_addr.s_addr = INADDR_ANY;
    / * INADDR_ANY sets our IP address automatically * /

    bzero ( & (server.sin_zero), 8 );
    / * we write zeros in the structure challenge * /


    / * Next the call to bind () * /
    if ( bind (( * fd), ( struct sockaddr * ) & server,
                sizeof ( struct sockaddr)) == - 1 ) {
        dog "error in bind () \ n " );
        exit ( -one );
    }
    if ( listen (( * fd), BACKLOG) == -1 ) { / * call to listen () * /
        dog "error in listen () \ n " );
        exit ( -one );
    }

    *sin_size = sizeof ( struct sockaddr_in);

    return server;
}

/ *
  * *
 ↪  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  * *       Name            : finish_with_error
  * *       Function        : Save the generated error to a file and close the
 ↪  Connection
  * *       Parameters: Connection descriptor conn Returns
  * *                       : -
  * *
 ↪  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

void finish_with_error (MYSQL * conn)
{
   fprintf (stderr, "% s \ n " , mysql_error (conn));
   mysql_close (conn);
}

/ *
  * *
 ↪  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  * *       Name            : Save_DB
  * *       Function        : Store the values   of the Drive Test in the database
 ↪  data
  * *       Parameters: Drive Test structure and database name
 ↪
  * *       Returns         : 1 if it was successful, -1 if it failed
  * *
 ↪  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

int Save_DB (Info Dtest, const char * Table) {


     MYSQL * conn; / * connection variable for MySQL * /
```

```
char * server = "localhost" ; / * server address 127.0.0.1, localhost or ip address *
/
char * user = "root" ; / * user to query the database * /
char * password = "1414425" ; / * password for the user in question
* /
char * database = "thesis" ; / * name of the database to consult * /
conn = mysql_init ( Null ); / * initialization to null the connection * /
char query [ 1000 ];

if ( conn == Null )
{
        finish_with_error (conn);
        return - one ;
}

/ * connect to database * /
if ( ! mysql_real_connect (conn, server, user, password, database, 0 , Null ,
0 ))
{ / * define the connection parameters previously established * /
        finish_with_error (conn);
        return - one ;
}

/ * send SQL query * /
snprintf (query, 1000 , "INSERT INTO% s VALUES (% .6f,%. 6f,%. 4f,% u,%. 4f"
        ",% d,% d,% d,% d,% d,% d,% d,% d,% d,% d,%. 0lf,% d,% d,%. 0lf,% d,% d,% d,% d,
        ""% d,% d,%. 0f,%. 0f,% d,% d,% d,% d,% d,%. 0f,%. 0f,% d, % d,% d,% d,% d,%. 0f,
        ""% .0f,% d,% d,% d,% d,% d,%. 0f,%. 0f,% d,% d ,% d,% d,% d,%. 0f,%. 0f,% d,% d,
        ""% d,% d,% d,%. 0f,%. 0f) " , Table, Dtest.latitude, Dtest.longitude, Dtest.HMS,
        Dtest.DMA, Dtest.speedOTG, Dtest.Dialed_Calls, Dtest.Missed_Calls, Dtest.RSSI,
        Dtest.BER, Dtest.arfcn_bcch [ 0 ], Dtest.rxl [ 0 ], Dtest.rqx, Dtest.mcc [ 0 ], Dtest.mnc [ 0
        ], Dtest.bsic [ 0 ], Dtest.cellid [ 0 ], Dtest.rla, Dtest.txp, Dtest.lac [ 0 ], Dtest.TA,
        Dtest.arfcn_bcch [ one ], Dtest.rxl [ one ], Dtest.mcc [ one ], Dtest.mnc [ one ],
        Dtest.bsic [ one ], Dtest.cellid [ one ], Dtest.lac [ one ], Dtest.arfcn_bcch [ two ],


Dtest.rxl [ two ], Dtest.mcc [ two ], Dtest.mnc [ two ], Dtest.bsic [ two ], Dtest.cellid [ two ],
        Dtest.lac [ two ], Dtest.arfcn_bcch [ 3 ], Dtest.rxl [ 3 ], Dtest.mcc [ 3 ], Dtest.mnc [ 3 ],
        Dtest.bsic [ 3 ], Dtest.cellid [ 3 ], Dtest.lac [ 3 ], Dtest.arfcn_bcch [ 4 ], Dtest.rxl [ 4 ],
        Dtest.mcc [ 4 ], Dtest.mnc [ 4 ], Dtest.bsic [ 4 ], Dtest.cellid [ 4 ], Dtest.lac [ 4 ],
        Dtest.arfcn_bcch [ 5 ],

Dtest.rxl [ 5 ], Dtest.mcc [ 5 ], Dtest.mnc [ 5 ], Dtest.bsic [ 5 ], Dtest.cellid [ 5 ],
        Dtest.lac [ 5 ], Dtest.arfcn_bcch [ 6 ], Dtest.rxl [ 6 ], Dtest.mcc [ 6 ], Dtest.mnc [ 6 ],
        Dtest.bsic [ 6 ], Dtest.cellid [ 6 ], Dtest.lac [ 6 ]);

if ( mysql_query (conn, query))
```

```
        {
                finish_with_error (conn);
                return - one ;
        }

        / * frees the variable res and closes the connection * /
        mysql_close (conn);
        return one ;
}

/ *
  * *
↪    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  * *         Name              : Create_DB
  * *         Function          : Create table in database
  * *         Parameters: Name of the table in the database Returns
  * *                           : 1 if successful, 0 if not
  * *
↪    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* /

int Create_DB ( const char * Table) {

        MYSQL * conn; / * connection variable for MySQL * /
        char * server = "localhost" ; / * server address 127.0.0.1,
                                                    localhost or ip address * /
        char * user = "root" ; / * user to query the database * /
        char * password = "1414425" ; / * password for the user in question
↪       * /
        char * database = "thesis" ; / * name of the database to consult * /
        conn = mysql_init ( Null ); / * initialization to null the connection * /
        char query [ 1000 ];

        if ( conn == Null )
        {
                finish_with_error (conn);
                return - one ;
        }

        / * connect to database * /
        if ( ! mysql_real_connect (conn, server, user, password, database, 0 , Null ,
↪       0 ))
        { / * define the connection parameters previously established * /
                finish_with_error (conn);
                return - one ;
        }
        / * send SQL query * /
        snprintf (query, 1000 , "create table% s (LATITUDE FLOAT (9,4), LENGTH"
                "FLOAT (9,4), HMS INT, DMA INT, SPEEDOTG FLOAT (6,2), CALL_MADE INT,"
                "CALL_LOST INT, RSSI INT, BER INT, ARFCN_BCCH_0 INT, RXL_0 INT, RQX_0"
                "INT, MCC_0 INT, MNC_0 INT, BSIC_0 INT, CELL_ID_0 INT, RLA_0 INT, TXP_0" "INT,
               LAC_0 INT, TA_0 INT, ARFCN_BCCH_1 INT, RXL_1 INT, MCC_1 INT, MNC_1" "INT, BSIC_1
               INT, CELL_ID_1 INT, LAC_1 INT, ARFCN_BCCH_2 INT, RXL_2 INT, "

                "MCC_2 INT, MNC_2 INT, BSIC_2 INT, CELL_ID_2 INT, LAC_2 INT,
        ARFCN_BCCH_3"
                "INT, RXL_3 INT, MCC_3 INT, MNC_3 INT, BSIC_3 INT, CELL_ID_3 INT, LAC_3" "INT,
               ARFCN_BCCH_4 INT, RXL_4 INT, MCC_4 INT, MNC_4 INT, BSIC_4 INT" ", CELL_ID_4 INT,
               LAC_4 INT_, ARFCN_4 INT, LAC_4 INT_, ARFCN_4 INTXL INT, MCC_5 INT, "" MNC_5 INT,
               BSIC_5 INT, CELL_ID_5 INT, LAC_5 INT, ARFCN_BCCH_6 INT, "" RXL_6 INT, MCC_6 INT,
               MNC_6 INT, BSIC_6 INT, CELL_ID_6 INT, LAC_6 INT); " ,

                Table);

        if ( mysql_query (conn, query))
        {
                finish_with_error (conn);
                return - one ;
        }

        / * frees the variable res and closes the connection * /
        mysql_close (conn);
        return one ;
}

int main ( int argc, char const * argv []) {


        if ( argc ! = 3 )
        {
                printf ( "are two arguments: ./Program Port DataBase \ n " );
                return - one ;
        }
        if ( IsANumber (argv [ one ]) == - 1 )
        {
                printf ( "The second argument is a number \ n " );
                return - one ;
        }

        int k = 1 , accountant = 1 ; Info
        Dtest;
        int fd, fd2, j = 0 , numbytes; / * descriptor files * /
```

```
struct sockaddr_in server;
/ * for the server address information * /

struct sockaddr_in client;
/ * for customer address information * /

int sin_size;

server = Start_Socket ( & fd, & sin_size, argv [ one ]);

if ( Create_DB (argv [ two ]) ! = 1 )
{
        printf ( "Error creating database \ n " );
        return - one ;
}

/ * Then the call to accept () * /

 while ( one )
 {
        if (( fd2 = accept (fd, ( struct sockaddr * ) & client,
                                    & sin_size)) == - 1 ) {
              printf ( "With% d files \ t " , accountant ); dog "error
              in accept () \ n " );
              exit ( -one );
        }

        if ( k % BACK == 0 ) {

              printf ( "counter goes in% d \ n " , accountant );
        }

        / * which will show the client's IP * / // send (fd2,
        (void *) & msg, sizeof (Info), 0);
        / * which will send the welcome message to the client * /

        for ( int j = 0 ; j <= 9 ; j ++ ) {

                if (( numbytes = recv (fd2, ( void * ) & Dtest, sizeof ( Info), 0 )) == -1 ) {
                        / * call to recv () * /
                        dog "error Error \ n " );
                        exit ( -one );
                }

                if ( Save_DB (Dtest, argv [ two ]) ! = 1 )
                {
                        printf ( "Failed to save to database \ n " );
```

```
                    return - one ;
                }
            }

            accountant ++ ;
            k ++ ;
            close (fd2);
        }
}
```

File that generates the WEB page

```
<! DOCTYPE html>
< html >
    < head >
        < goal charset = "utf-8" >
        < title > 20km test </ title >
        < style >
            / * Map settings * /
            #map {
                height : 100% ;
            }
            html , body {
                height : 100% ;
                margin : 0 ;
                padding : 0 ;
            }
            # floating-panel { / * Panel settings * /
                position : absolute ;
                top : 10px ;
                left : 25% ;
                z-index : 5 ;
                background-color : #fff ;
                padding : 5px ;
                border : 1px solid # 999 ;
                text-align : center ;
                font-family : 'Roboto' , 'sans-serif' ;
                line-height : 30px ;
                padding-left : 10px ;
            }
            # floating-panel {
                background-color : #fff ;
                border : 1px solid # 999 ;
                left : 25% ;
                padding : 5px ;
                position : absolute ;
                top : 10px ;
            z-index : 5 ;
        }
    </ style >
</ head >

< body >
            < div go = "info" > < div class = "portlet-body"
               >
                           <? php
        // Add the PHP file that reads from the database
                           include ('test_20km.php');
                           ?>
               </ div >
           </ div >

<! - Panel containing the cell finder ->
        < div go = "floating-panel" >
        < form go = "frm1" action = "/action_page.php" >
              CELL_ID: < input type = "text" yam = "fname" > < br >
        </ form >
        < button onclick = "myFunction ()" > View cell </ button >
        < button onclick = "myFunction_1 ()" > See All </ button >
    </ div >

    < div go = "map" > </ div >

    < script >
       // Global variables that store the rectangles and the map
              var rectangle = [];
              var map;
        var aux;
              / * The map is initialized with the google maps API * /
              function initMap () {
                    map = new google.maps.Map ( document .getElementById ( 'map' ), {center :
                       new google.maps.LatLng ( 10.4725 , -66.8340 ), //
    coordinates in which the map will appear
                       zoom : 17 // zoom with which the map will appear
                    });
                    var infoWindow = new google.maps.InfoWindow;
                    var i = 0 ;
                       // Address of the XML file in which the
    data
                    downloadUrl ( './xml/test_20km.xml' , function ( data) {
                          var xml = data.responseXML;
                 // Search for the DT Tag created in the PHP file
                          var DT = xml.documentElement.getElementsByTagName ( 'DT' ); aux = 0 ;
```

```
                        Array .prototype.forEach.call (DT, function ( DTElem) {
                                                   var
↪ The t = parseFloat (DTElem.getAttribute ( 'LATITUDE' )); // latitude is saved
                                                   var
↪ Lng = parseFloat (DTElem.getAttribute ( 'LENGTH' )); // the length is saved
                        var Intensity =
↪ parseFloat (DTElem.getAttribute ( 'RXL_0' )); // intensity is saved
                        var CALL_LOST =
↪ parseInt (DTElem.getAttribute ( 'CALL_LOST' )); // calls are saved
↪ losses
                                                var CELL_ID =
↪ DTElem.getAttribute ( 'CELL_ID_0' ); // the identifier of the
↪ cell


                                    if ( aux ! = CALL_LOST)
                                    {
                                // If there is a missed call create a bookmark
↪ to identify
                                // in which coordinate it fell
                                    var infowincontent =
↪ document .createElement ( 'div' );
                                      var text = document .createElement ( 'text' );
                                      text.textContent = 'Lost call' ;
                                      infowincontent.appendChild (text);
                                infowincontent.appendChild ( document .createElement ( 'br' ));

                              var text_1 = document .createElement ( 'text' );
                              text_1.textContent = 'Cell' + CELL_ID;
                                infowincontent.appendChild (text_1);

                                                  var point = new google.maps.LatLng
↪ (The t -0.00005 , Lng -0.00005 );
                                                var marker = new google.maps.Marker ({
                                                    map : map,
                                                    position : point
                                                });
                                                marker.addListener ( 'click' , function () {

↪                                   infoWindow.setContent (infowincontent);
                                                    infoWindow.open (map, marker);
                                            });
                                                aux ++ ;
                                    }

                        // The rectangle is created in relation to the measured signal
↪
```

```
                                    rectangle [i] = new google.maps.Rectangle ({

                                          strokeColor : '# FF0000' ,
                                          strokeOpacity : 0 ,
                                          strokeWeight : 0 ,
                                          fillColor : '# FF0000' ,
                                          fillOpacity : Intensity / 500 ,
                                          map : map,
                                          bounds : {
                                                      north : The t,
                                                      south : The t -0.0001 ,
                                                      east : Lng,
                                                      west : Lng -0.0001
                                              }
                                       });
                               i ++ ;
                          });
                      });
                }

                // Function that is responsible for restoring the coverage map completely

                function myFunction_1 ()
                {
                      var i;
                      for ( i = 0 ; i < rectangle.length; i ++ ) {rectangle [i]
                            .setMap (map);
                      }
                }

                // Function that is responsible for finding and displaying only the selected
             cell
                function myFunction ()
                {
                      var x = document .getElementById ( "frm1" );
                      var text = "" ;
                         var i;
                         for ( i = 0 ; i < x.length; i ++ ) {text + = x.elements
                            [i] .value;
                         }
                         i = 0 ;
                      downloadUrl ( './xml/test_20km.xml' , function ( data) {
                   var xml = data.responseXML;
                   var DT = xml.documentElement.getElementsByTagName ( 'DT' );
                   Array .prototype.forEach.call (DT, function ( DTElem) {
```

```
                                    var
↳ The t = parseFloat (DTElem.getAttribute ( 'LATITUDE' ));
                                    var
↳ Lng = parseFloat (DTElem.getAttribute ( 'LENGTH' ));
                                    var cell =
↳ DTElem.getAttribute ( 'CELL_ID_0' );

                                    if ( text ! = cell)
                                rectangle [i] .setMap ( null );
                       else
                                rectangle [i] .setMap (map);
                       i ++ ;
                 });
               });
            }

                      function downloadUrl (url, callback) {
                     var request = window .ActiveXObject ?
                     new ActiveXObject ( 'Microsoft.XMLHTTP' ) :
                     new XMLHttpRequest;

                     request.onreadystatechange = function () {
                       if ( request.readyState == 4 )
                         {request.onreadystatechange = doNothing;
                         callback (request, request.status);
                       }
                     };

                     request.open ( 'GET' , url, true );
                     request.send ( null );
                   }

          function doNothing () {}
        </ script >
        < script async defer

↳ src = "https://maps.googleapis.com/maps/api/js?key=AIzaSyC6o2Hfl41F4pZSr2Ypa1eCNK6nmU2Q38Q
        </ script >
      </ body >
    </ html >
```

File that reads from the database: test 20km.php

```
<? php

// Add the file with the database credentials
include ( " database_info_20km.php " );
```

```
// The XML file is started and the parent node is created
$ sun = new DOMDocument ( '1.0' , 'UTF-8' );
$ sun -> formatOutput = true ;

$ root = $ sun -> createElement ( 'DTs' );
$ root = $ sun -> appendChild ( $ root );

// The connection to the database is opened

$ link = mysqli_connect ( "localhost" , $ username , $ password , $ database );

// We verify that the database exists
if ( ! $ link ) {
   threw out " Not connected: " . mysqli_connect_error ();
            exit ();
}

// Select all rows from the database
$ query = "SELECT * FROM $ table ; " ;

// Save the result in result
$ result = mysqli_query ( $ link , $ query );

// We verify that the result is not empty
if ( ! $ result ) {
   threw out " Invalid query: " . mysqli_error ( $ link );
            exit ();
}

// All the rows are read and the parameters corresponding to the
      xml file
while ( $ row = mysqli_fetch_assoc ( $ result )) {

              $ node = $ sun -> createElement ( 'DT' );
              $ node = $ root -> appendChild ( $ node );

            $ node -> setAttribute ( "LATITUDE" , $ row [ 'LATITUDE' ]);
            $ node -> setAttribute ( "LENGTH" , $ row [ 'LENGTH' ]);
            $ node -> setAttribute ( "HMS" , $ row [ 'HMS' ]);
            $ node -> setAttribute ( "DMA" , $ row [ 'DMA' ]);
            $ node -> setAttribute ( "SPEEDOTG" , $ row [ 'SPEEDOTG' ]);
            $ node -> setAttribute ( "CALL_MADE" , $ row [ 'CALL_MADE' ]);
            $ node -> setAttribute ( "CALL_LOST" , $ row [ 'CALL_LOST' ]);
            $ node -> setAttribute ( "RSSI" , $ row [ 'RSSI' ]);
            $ node -> setAttribute ( "BER" , $ row [ 'BER' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_0" , $ row [ 'ARFCN_BCCH_0' ]);
            $ node -> setAttribute ( "RXL_0" , $ row [ 'RXL_0' ]);
            $ node -> setAttribute ( "RQX_0" , $ row [ 'RQX_0' ]);
            $ node -> setAttribute ( "MCC_0" , $ row [ 'MCC_0' ]);
            $ node -> setAttribute ( "MNC_0" , $ row [ 'MNC_0' ]);
            $ node -> setAttribute ( "BSIC_0" , $ row [ 'BSIC_0' ]);
            $ node -> setAttribute ( "CELL_ID_0" , $ row [ 'CELL_ID_0' ]);
            $ node -> setAttribute ( "RLA_0" , $ row [ 'RLA_0' ]);
            $ node -> setAttribute ( "TXP_0" , $ row [ 'TXP_0' ]);
            $ node -> setAttribute ( "LAC_0" , $ row [ 'LAC_0' ]);
            $ node -> setAttribute ( "TA_0" , $ row [ 'TA_0' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_1" , $ row [ 'ARFCN_BCCH_1' ]);
            $ node -> setAttribute ( "RXL_1" , $ row [ 'RXL_1' ]);
            $ node -> setAttribute ( "MCC_1" , $ row [ 'MCC_1' ]);
            $ node -> setAttribute ( "MNC_1" , $ row [ 'MNC_1' ]);
            $ node -> setAttribute ( "BSIC_1" , $ row [ 'BSIC_1' ]);
            $ node -> setAttribute ( "CELL_ID_1" , $ row [ 'CELL_ID_1' ]);
            $ node -> setAttribute ( "LAC_1" , $ row [ 'LAC_1' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_2" , $ row [ 'ARFCN_BCCH_2' ]);
            $ node -> setAttribute ( "RXL_2" , $ row [ 'RXL_2' ]);
            $ node -> setAttribute ( "MCC_2" , $ row [ 'MCC_2' ]);
            $ node -> setAttribute ( "MNC_2" , $ row [ 'MNC_2' ]);
            $ node -> setAttribute ( "BSIC_2" , $ row [ 'BSIC_2' ]);
            $ node -> setAttribute ( "CELL_ID_2" , $ row [ 'CELL_ID_2' ]);
            $ node -> setAttribute ( "LAC_2" , $ row [ 'LAC_2' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_3" , $ row [ 'ARFCN_BCCH_3' ]);
            $ node -> setAttribute ( "RXL_3" , $ row [ 'RXL_3' ]);
            $ node -> setAttribute ( "MCC_3" , $ row [ 'MCC_3' ]);
            $ node -> setAttribute ( "MNC_3" , $ row [ 'MNC_3' ]);
            $ node -> setAttribute ( "BSIC_3" , $ row [ 'BSIC_3' ]);
            $ node -> setAttribute ( "CELL_ID_3" , $ row [ 'CELL_ID_3' ]);
            $ node -> setAttribute ( "LAC_3" , $ row [ 'LAC_3' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_4" , $ row [ 'ARFCN_BCCH_4' ]);
            $ node -> setAttribute ( "RXL_4" , $ row [ 'RXL_4' ]);
            $ node -> setAttribute ( "MCC_4" , $ row [ 'MCC_4' ]);
            $ node -> setAttribute ( "MNC_4" , $ row [ 'MNC_4' ]);
            $ node -> setAttribute ( "BSIC_4" , $ row [ 'BSIC_4' ]);
            $ node -> setAttribute ( "CELL_ID_4" , $ row [ 'CELL_ID_4' ]);
            $ node -> setAttribute ( "LAC_4" , $ row [ 'LAC_4' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_5" , $ row [ 'ARFCN_BCCH_5' ]);
            $ node -> setAttribute ( "RXL_5" , $ row [ 'RXL_5' ]);
            $ node -> setAttribute ( "MCC_5" , $ row [ 'MCC_5' ]);
            $ node -> setAttribute ( "MNC_5" , $ row [ 'MNC_5' ]);
            $ node -> setAttribute ( "BSIC_5" , $ row [ 'BSIC_5' ]);
            $ node -> setAttribute ( "CELL_ID_5" , $ row [ 'CELL_ID_5' ]);
            $ node -> setAttribute ( "LAC_5" , $ row [ 'LAC_5' ]);
            $ node -> setAttribute ( "ARFCN_BCCH_6" , $ row [ 'ARFCN_BCCH_6' ]);
            $ node -> setAttribute ( "RXL_6" , $ row [ 'RXL_6' ]);
```

```
            $ node -> setAttribute ( "MCC_6" , $ row [ 'MCC_6' ]);
            $ node -> setAttribute ( "MNC_6" , $ row [ 'MNC_6' ]);
            $ node -> setAttribute ( "BSIC_6" , $ row [ 'BSIC_6' ]);
            $ node -> setAttribute ( "CELL_ID_6" , $ row [ 'CELL_ID_6' ]);
            $ node -> setAttribute ( "LAC_6" , $ row [ 'LAC_6' ]);
}
// It is saved in the xml file
$ sun -> save ( "./xml/test_20km.xml" );
// The connection is closed
mysqli_close ( $ link );

?>
```