UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE TELECOMUNICACIONES

DRIVE TEST AUTOMATIZADO MEDIANTE ESP8266 Y EL MODULO GSM

Por:

Br. Luis Alfredo Muñoz Molina

PROYECTO DE GRADO

Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Telecomunicaciones

Sartenejas, Julio de 2017

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE TELECOMUNICACIONES

DRIVE TEST AUTOMATIZADO MEDIANTE ESP8266 Y EL MODULO GSM

Por:

Br. Luis Alfredo Muñoz Molina

Realizado con la asesoría de:

Prof. Miguel Díaz

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Telecomunicaciones

Sartenejas, Julio de 2017

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE TELECOMUNICACIONES

DRIVE TEST AUTOMATIZADO MEDIANTE ESP8266 Y EL MODULO GSM

PROYECTO DE GRADO

Realizado por: Br. Luis Alfredo Muñoz Molina
Con la asesoría de: Prof. Miguel Díaz

RESUMEN

Un sistema de Drive Test es aquel que se encarga de realizar medidas de radiofrecuencia para verificar el estado de una red celular. En el presente trabajo se implementa un sistema de Drive Test que se basa en el modulo GSM SIM900 y en el GPS XM37-1612, donde se hace uso de la placa de desarrollo D1 mini para su control y configuración. Para ello se estudian las características de estos dispositivos, se comparan los distintos comandos AT que posee el módulo GSM y por último se presenta una propuesta del sistema basado en los estudios anteriores. Se creó una librería en C++ para implementar el sistema propuesto, y se realizó un recorrido de prueba que medía nivel de la señal, la cantidad y la ubicación de las llamadas perdidas, la calidad de las llamadas realizadas, el número de radiofrecuencia absoluta del canal de control de difusión, el código de país, el código de la red, el código de identidad de la estación base, el identificador de la celda, el nivel mínimo de acceso a la celda, la potencia de transmisión máxima del móvil, el código de área de ubicación, el avance temporal y la tasa de error de bit. Los resultados obtenidos del recorrido demostraron que con el sistema propuesto es factible implementar un sistema de Drive Test sencillo, debido a que aun faltan funcionalidades de un Drive Test completo.

Palabras clave: Drive Test, Estación Base, SIM900, ESP8266, GPS.

ÍNDICE GENERAL

INTRODUCCIÓN **1**

CAPÍTULO 1 MARCO TEÓRICO **3**

1.1 ¿Qué es una red celular? . . . 3
1.1.1 Estructura celular . . . 3
1.1.2 Planificación de la red . . . 3
1.2 GSM, un sistema digital celular de radio móvil . . . 5
1.3 Servicios ofrecidos en un sistema GSM . . . 6
1.4 Arquitectura del sistema [1] . . . 6
1.4.1 Modulo de identidad del abonado ó suscriptor . . . 7
1.4.2 Estación base . . . 7
1.4.3 Central de conmutación móvil . . . 7
1.4.4 Controlador de estación base . . . 7
1.4.5 Centro de conmutación móvil de pasarela . . . 7
1.4.6 Identidad internacional del abonado a un móvil . . . 8
1.4.7 Registro de ubicación . . . 8
1.4.8 Registro de ubicación del visitante . . . 8
1.4.9 Centro de autenticación . . . 9
1.4.10 Registro de la identidad del equipo . . . 9
1.5 Establecimiento de la llamada . . . 10
1.6 Relevo . . . 10
1.7 Voceo (paging) . . . 11
1.8 Drive Test . . . 12
1.8.1 Tipos de Drive Test . . . 14
1.8.2 Criterio de muestreo . . . 14

CAPÍTULO 2 PROPUESTA DEL SISTEMA **16**

2.1 SIM900 . . . 16

2.1.1 Descripción de los pines 17
2.1.2 Modos de operación 17
2.1.3 Comandos AT 17
2.1.3.1 Comandos AT utilizados 20
2.2 GPS XM37-1612 22
2.2.1 Características del módulo 23
2.2.2 Especificaciones del módulo 23
2.2.3 Descripción de pines 23
2.2.4 Interfaz de software 23
2.3 Módulo ESP8266 24
2.3.1 Características del ESP8266 26
2.3.2 Descripción de pines 27

CAPÍTULO 3 IMPLEMENTACIÓN DEL SISTEMA 28
3.1 Sistema de Drive Test 28
3.2 Servidor 33
3.3 Problemas presentados en el desarrollo 33
3.3.1 Temporizador de perro guardián 34
3.3.2 Memoria estática de acceso aleatorio [18] 34
3.3.3 Velocidad de recepción de datos 35

CAPÍTULO 4 RESULTADOS 36

CONCLUSIONES Y RECOMENDACIONES 40

REFERENCIAS 41

APENDICE A 43

LISTA DE FIGURAS

1.1 Estructura celular [2] 4
1.2 División de celdas 4
1.3 Sectorización [3] 5
1.4 Celda paragua [4] 5
1.5 Arquitectura GSM 6
1.6 Relevo simple intra BSC 12
1.7 Relevo entre MSCs 12

2.1 Propuesta de diseño del sistema 16
2.2 Placa del SIM900A V1 SCH [9] 17
2.3 Placa de desarrollo del GPS XM37-1612 [12] 22
2.4 Placa de desarrollo D1 mini [14] 26

3.1 Conexiones módulo GSM, GPS y ESP8266 28
3.2 Diagrama de estado general del sistema 29
3.3 Diagrama de estado para la verificación del GPS 30
3.4 Diagrama de estado para la adquisición de datos del GPS 31
3.5 Diagrama de estado de enviar comandos AT 32
3.6 Diagrama de estado para la obtención de los datos de ingeniería 33
3.7 Diagrama de estado del servidor 34
3.8 Estados de la memoria SRAM [19] 35

4.1 Recorrido del Drive Test a 5 Km/h 36
4.2 Marcador 37
4.3 Filtrado de celda 12481, en el recorrido de 5Km/h 37
4.4 Recorrido del Drive Test a 20 Km/h 38
4.5 Recorrido del Drive Test a 40 Km/h 39
4.6 Recorrido del Drive Test a 40 Km/h externo 39

LISTA DE TABLAS

1.1 Formato del IMSI 8
1.2 Procedimiento de establecimiento de llamada [1] 11
1.3 Niveles de RxLev [6] 13
1.4 Niveles de RXQUAL [6] 14

2.1 Pines de la placa de desarrollo SIM900A V1 SCH 18
2.2 Resumen de los modos de operación [8] 19
2.3 Especificaciones del receptor GPS [11] 24
2.4 Descripción de pines 25
2.5 Descripción del mensaje GPRMC [11] 25
2.6 Descripción de pines D1Mini [14] 27

Lista de abreviaturas

API Interfaz de programación de aplicaciones.

ARFCN Absolute radio-frequency channel number
Números de canal de radiofrecuencia absoluta.

AT ATtention
Atención.

AuC Authentication Center
Centro de autenticación.

BCCH Broadcast Control Channel
Canal de Control de Difusión.

BER Bit Error Rate
Tasa de error de bit.

BLER Block Error Rate
Tasa de error de bloque.

BSC Base Station Controller
Controlador de Estaciones Base.

BSIC Base Station Identity Code
Código de identidad de la estacion base.

BSS Base Station Subsystem
Subsistema de Estación Base.

BTS Base Transceiver Station
Estacion Base.

CRC Cyclic Redundancy Check
Comprobacion de redundancia cíclica.

CS Coding Schemes
Esquemas de codificación.

DTX Discontinuous Transmission
Transmisión discontinua.

EFR Enhanced Full Rate
Tasa completa mejorada.

EIR Equipment Identity Register
Registro de la identidad del equipo.

ETSI European Telecommunications Standards Institute
Instituto Europeo de Normas de Telecomunicaciones.

FR Full Rate
Tasa completa.

GMSC Gateway Mobile Switching Center
Centro de Conmutación Móvil de Pasarela.

GPIO General Purpose Input/Output
Entrada y salida de propósito general.

GPRS General Packet Radio Service
Servicio general de paquetes vía radio.

GPS Global Positioning System
Sistema de Posición Global.

GSM Global System for Mobile communications
Sistema Global para las comunicaciones moviles.

HLR Home Location Register
Registro de ubicación base.

HR Half Rate
Tasa media.

IMSI International Mobile Subscriber Identity
Identidad Internacional del Abonado a un Móvil.

ISDN Integrated Services Digital Network
Red digital de servicios integrados.

LMSI Local Mobile Subscriber Identity
Identificador local de estación móvil.

M2M Machine to Machine
Máquina a máquina.

MCC Mobile Country Code
Código de País de Móvil.

MNC Mobile Network Code
Código de Operador de Móvil.

MOC Mobile-Originated Call
Llamada originada por el móvil.

MSC Mobile Switching Center
Central de Conmutación Móvil.

MSIC International Mobile Subscriber Identity
Identidad internacional del abonado a un móvil.

MSRN Mobile Station Roaming Number
Número de Itinerancia de la Estación Móvil.

MTC Mobile-Terminated Call
Llamada culminada en un móvil.

PLMN Public Land Mobile Network
Red móvil terrestre publica.

PSTN Public Switched Telephone Network
Red telefónica publica conmutada.

RF Radiofrecuencia.

RMC Recommended Minimim Specific GNSS Data
Datos GNSS específicos mínimos recomendados.

RSSI Received Signal Strength Indicator
Indicador de fuerza de la señal recibida.

RTC Real Time Clock
Reloj en tiempo real.

RXQ Calidad de la señal recibida.

RXQUAL Receiver Quality
Calidad recibida.

SDK Software Development Kit
Kit de desarrollo de software.

SIM Subscriber Identity Module
Módulo de Identidad de Abonado.

SMT Surface Mount Technology
Tecnología de montaje superficial.

SoC System on a Chip
Sistema en un chip.

SRAM Static Random Access Memory
Memoria estática de acceso aleatorio.

TCP/IP Transmission Control Protocol/Internet Protocol
Protocolo de control de Transmisión/ Protocolo de Internet.

TMSI Temporary Mobile Subscriber Identity
Identificador temporal de estación móvil.

VLR Visitor location register
Registro de Ubicación del Visitante.

WDT Watch Dog Timer
Temporizador de perro guardian.

WPA WiFi Protected Access
Acceso protegido WiFi.

INTRODUCCIÓN

La planificación de un sistema celular se realiza teniendo en cuenta la capacidad y cobertura requerida para proveer a los usuarios con un buen servicio. El proceso de establecer una red celular conlleva mucho más que la implantación y puesta en servicio del sitio, ya que es necesario un proceso de mediciones y ajustes continuos que permitan optimizar el desempeño de la red y permita a los usuarios la mejor calidad posible; dichas mediciones son realizadas en pruebas de recorrido sobre un área determinada, mejor conocido como Drive Test.

El Drive Test busca recolectar información sobre la red celular ya establecida, entre estos datos tenemos, por ejemplo, nivel de potencia de la señal recibida, calidad de la señal, identificador de la celda, tasa de error de bits, etc. Además de los parámetros de la red también se deben realizar medidas en relación a las llamadas, observar si los mecanismo de la red, para estas, funcionan correctamente.

Los equipos utilizados durante un Drive Test pueden llegar a ser muy costosos, sin embargo, en el mercado existen equipos capaces de realizar tareas parecidas, entre estos equipos se tiene al módulo GSM (del ingles *Global System for Mobile communications*) SIM900 el cual se puede configurar mediante comandos AT (del ingles *ATtention*), este modulo permite realizar llamadas, enviar mensajes, utilizar datos y observar el estado de la red, estas cualidades lo convierten en un candidato para formar parte del sistema del Drive Test.

El módulo GSM necesita un elemento controlador capaz de recibir, procesar y enviar la información. En un recorrido de Drive Test se suelen utilizar varios módulos, uno por tecnología celular (2G, 3G, 4G), o quizás también uno por operadora, se considera utilizar un elemento con tecnología WIFI como elemento controlador del módulo GSM, debido a que realizar todas las interconexiones de este sistema puede llegar a ser muy complicado.

Planteamiento del problema

Un sistema de Drive Test debe ser capaz de reportar la cobertura, la calidad del servicio, la tasa de llamadas exitosas e indicar si se encontraron fallas en los mecanismos de relevo en el sistema en una zona determinada, sin embargo, la complejidad del cableado del sistema incrementa a medida que aumentan las tecnologías que se desean medir (2G,3G,4G).Por otro lado el ESP8266 es un sistema en un chip que integra la pila de protocolos TCP/IP y es capaz de conectarse a redes WiFi, permitiendo al dispositivo interactuar con otros elementos de manera inalámbrica.

Ante este escenario, ¿es posible implementar un sistema automatizado de Drive Test que cumpla con estas características y además, no tenga la necesidad de estar cableado, mediante el módulo GSM SIM900 y el ESP8266?.

Justificación

Los sistemas de Drive Test son de gran interés para las empresas de telefonía celular, debido

a que siempre se busca optimizar y verificar la red ya desplegada. Debido al alto costo de los equipos especializados para realizar un Drive Test se desea implementar una aplicación de bajo costo para realizar dichas pruebas mediante el módulo ESP8266 y el módulo GSM. Es de interés utilizar un módulo con tecnología inalámbrica como el ESP8266 para facilitar el envío de los datos obtenidos por el módulo GSM a un servidor, que almacene y permita la presentación de los datos. Se requiere la conexión inalámbrica para disminuir el uso de cables de la conexión.

Objetivo general

Implementar un drive test automatizado basado en el ESP8266 con el módulo GSM.

Objetivos específicos

- Estudiar los fundamentos de un Drive Test, y los parámetros de interés se deben medir
- Estudiar las características del ESP8266 y su programación en formato Arduino, en C++.
- Estudiar los fundamentos de GPS y sus parámetros de ubicación y tiempo.
- Estudiar los Módulos GSM, y sus comandos AT, requeridos para el Drive Test.
- Realizar rutinas de comunicación entre el módulo GSM y el ESP8266.
- Realizar rutinas de reporte de calidad de la señal y parámetros de la celda.
- Realizar rutinas llamadas de voz con el módulo GSM.
- Realizar rutinas de llamadas de Datos con el Modulo GSM (vía GPRS)
- Enviar los resultados de las medidas a un servidor mediante el módulo de WI-FI.
- Implementar sobre un Servidor el Almacenamiento y Presentación de los datos.

CAPÍTULO 1

MARCO TEÓRICO

1.1. ¿Qué es una red celular?

Una red celular es un sistema de comunicaciones formada por celdas, cada una con su propio transmisor, conocidas como estaciones base (BTS del ingles *base transceiver station*). Estas celdas son utilizadas para brindar cobertura de radio sobre un área mayor a la de una celda. La BTS le permite al usuario acceder a la red y utilizar sus servicios (voz y datos).

1.1.1. Estructura celular

Para cubrir grandes áreas de cobertura se necesita de un gran número de celdas, cada celda utiliza un conjunto de frecuencias, que serviran como canales físicos para la comunicación entre la BTS y el móvil, sin embargo, debido a lo escaso que es el espectro radioeléctrico, no se le puede asignar un gran rango de frecuencias a un operador celular. Debido a la necesidad de cubrir estas áreas de cobertura se creo el concepto de reuso frecuencial.

El conjunto total de canales de frecuencia se divide en N grupos aproximadamente iguales, y cada uno de esos grupos se asigna a una celda distinta buscando que celdas adyacentes utilicen grupos de canales de frecuencias distintos, esto se hace para evitar interferencias. [1]

Cada grupo de N celdas contiguas que en su conjunto utilizan todas las frecuencias disponibles constituyen un grupo (del ingles *cluster*). El patrón de grupos se repite periódicamente sobre el área geográfica a cubrir, permitiendo que cada canal se utilice varias veces pero evitando que celdas adyacente usen el mismo canal, como se ve en la figura 1.1.

1.1.2. Planificación de la red

Si se piensa en un país de densidad poblacional variada como lo es Venezuela es sencillo entender el porque no se pueden utilizar celdas del mismo tamaño en toda la red. No es lo mismo para un operador suplir con cobertura a una ciudad de alta densidad poblacional, como lo es por ejemplo caracas, a suplir a una isla de baja densidad poblacional, como lo es por ejemplo Margarita.

División de celda o aplicaciones de micro celda. Debido al incremento del número de suscriptores, la densidad de usuarios por celda también aumenta. Una idea bastante básica es dividir el espacio existente en porciones mas pequeñas, multiplicando así la cantidad de usuarios a los que se les puede servir con el mismo conjunto de frecuencias (figura 1.2). También con este simple esquema se reduce el nivel de potencia consumido en la celda.

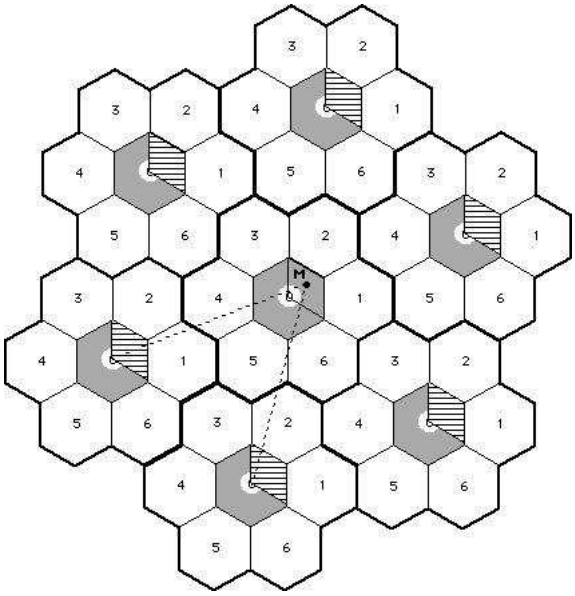

Figura 1.1. Estructura celular [2]

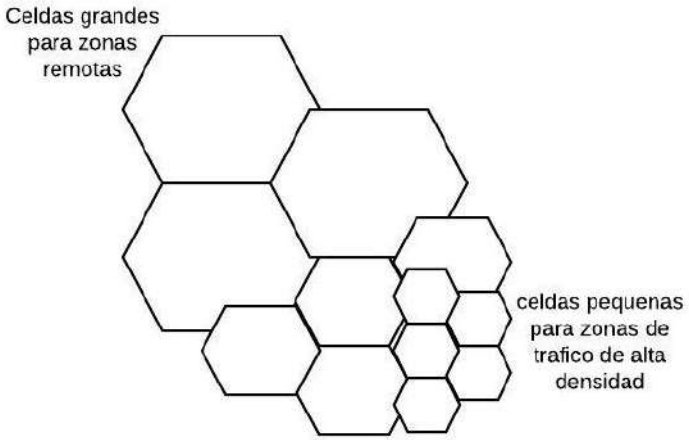

Figura 1.2. División de celdas

Celdas selectivas o sectorizadas. Es una técnica para mejorar la gestión de la autointerferencia en un sistema celular. En una celda sectorizada los canales de frecuencia asignados a esa celda se dividen en M (típicamente 3) grupos aproximadamente iguales, y se utiliza cada uno de ellos sobre un sector angular de la celda, a través de antenas direccionales como se ve en la figura 1.3.

Celdas paraguas. La primera vez que se aplico la división de celdas, los operadores se dieron cuenta de que en móviles a alta velocidad se realizaban una gran cantidad de relevos entre las celdas de menor tamaño. Ya que para realizar cada relevo se realiza mas trabajo por

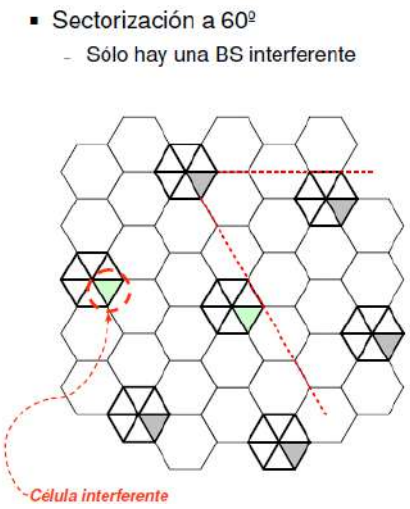

Figura 1.3. Sectorización [3]

parte de la red, no es deseable incrementar la cantidad de estos eventos. Las celdas paraguas (figura 1.4) se introdujeron para resolver este problema. Una celda paraguas transmite a mayor potencia que las micro-celdas que contiene y a diferente frecuencia. De esta manera cuando la red detecta un móvil viajando a alta velocidad lo maneja con la celdas paraguas en vez de las micro-celdas. Se detecta que el móvil va a alta velocidad mediante sus características de propagación o por sus relevos excesivos. En esta celda el móvil puede estar una mayor cantidad de tiempo bajando así la carga de la red.

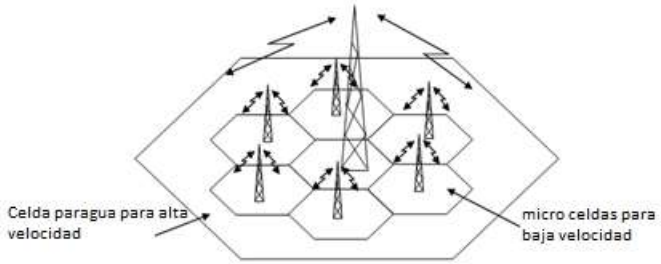

Figura 1.4. Celda paragua [4]

1.2. GSM, un sistema digital celular de radio móvil

El sistema global para comunicaciones móviles GSM (del ingles *Global System Mobile Communications*) es un estándar desarrollado por la ETSI (del ingles *European Telecommunications Standards Institute*) que sirve para describir los protocolos de la red celular digital de segunda generación (2G). El estándar se entrego por fases, siendo la primera un subconjunto de las características de la red, compatibles con actualizaciones y agregación

de servicios. Los complementos necesarios para la implementación completa de todos los servicios y características de la red se encontraron en la segunda fase.

1.3. Servicios ofrecidos en un sistema GSM

Las características y beneficios del sistema eran: calidad de voz superior (igual o mejor que la tecnología celular analógica), bajo costo en el terminal móvil, las operaciones y los servicios, un alto nivel de seguridad (confidencialidad y prevención de fraude), itinerancia internacional (con un solo número de suscriptor), soporte de terminales portátiles de bajo consumo y una variedad de nuevos servicios e instalaciones de red.

1.4. Arquitectura del sistema [1]

Para proveer de servicio celular a los usuarios, los operadores celulares tienen que instalar una infraestructura completa, que en algún punto, tiene que comunicarse con la red telefónica publica conmutada (PSTN del ingles *Public Switched Telephone Network*). Además de la característica de itinerancia nacional estándar, el sistema GSM decidio brindar itinerancia internacional. Esto implica que los usuarios tienen la opción de utilizar su móvil en el extranjero y por lo tanto en un sistema GSM externo.

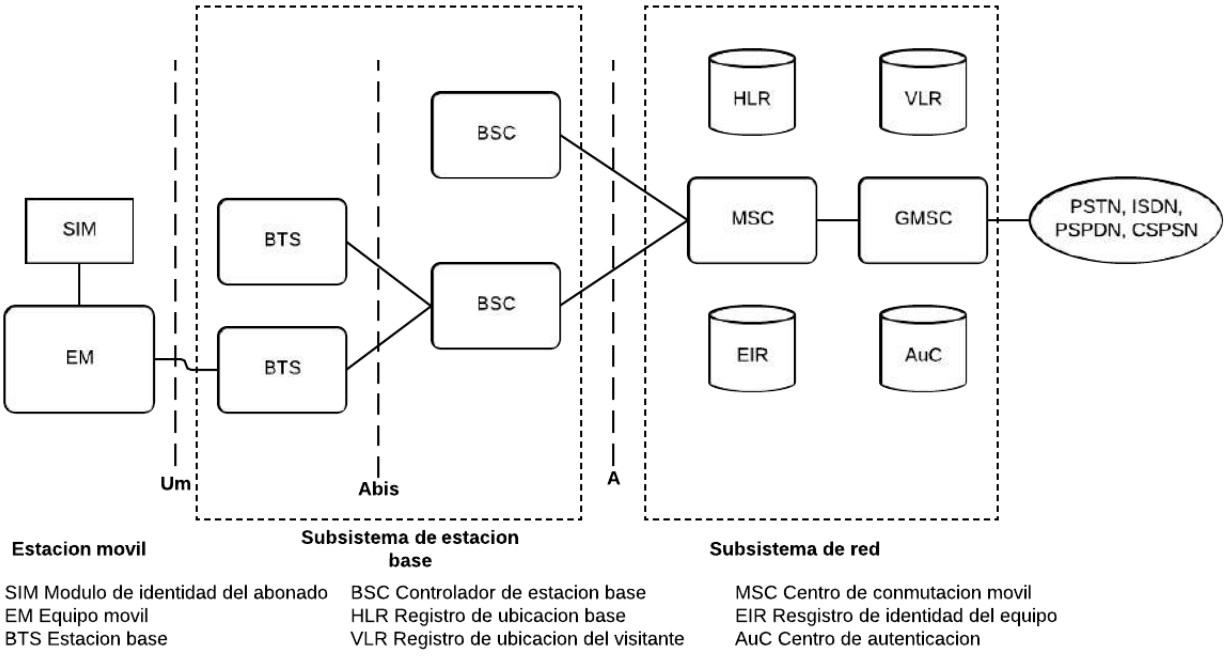

Figura 1.5. Arquitectura GSM

1.4.1. Modulo de identidad del abonado ó suscriptor

El modulo de identidad de abonado (SIM del ingles *subscriber identity module*) provee al móvil con una identificación. Es una tarjeta inteligente desmontable usada en teléfonos móviles y modems GSM.

Las tarjetas SIM almacenan de forma segura la clave de servicio del suscriptor usada para autenticarse ante la red de forma que sea posible cambiar la linea de un terminal a otro simplemente cambiando la tarjeta. Su uso es obligatorio en las redes GSM (excepto para llamadas de emergencia).

1.4.2. Estación base

La contra parte de la estación móvil en una red celular es la estación base (BTS), que es la interfaz del móvil con la red. La BTS se encuentra usualmente en el centro de la celda, su potencia de transmisión y antenas definen el tamaño de la celda. Una estación base tiene una gran cantidad de transceptores, cada uno representa un canal de RF separado. Parte de la inteligencia que poseían las estaciones base analógicas, como la medición de los canales de RF como criterio del relevo, se movieron a las estaciones móviles, para mejorar el estandar.

1.4.3. Central de conmutación móvil

El central de conmutación móvil (MSC del ingles *Mobile Switching Center*) es una central telefónica que realiza la conmutación de llamadas y mensajes dentro de la red, en caso de necesitar realizar una conexión fuera de la red se le envía al centro de conmutación móvil de pasarela (GMSC del ingles *Gateway Mobile Switching Center*)

El MSC también administra los relevos a estaciones base vecinas, mantiene un registro de la ubicación de los abonados móviles, es responsable de servicios de abonado y facturación.

1.4.4. Controlador de estación base

El controlador de estaciones base (BSC del ingles *Base Station Controller*) monitorea y controla varias BTS, la cantidad de BTS que puede controlar depende del fabricante, varia desde decenas hasta centenas de BTS. La tarea principal del BSC es la administración de frecuencias, el control de la BTS y funciones de intercambio. La BSC puede estar ubicada en el mismo sitio del BTS, en su propio sitio independiente o con el MSC. El BSC y BTS juntos forman un entidad funcional a veces llamada subsistema de estación base (BSS del ingles *Base Station Subsystem*).

1.4.5. Centro de conmutación móvil de pasarela

El centro de conmutación móvil de pasarela (GMSC) es la interfaz que comunica a la red con redes externas. Cuando entra una llamada desde una red externa o sale una llamada a una red externa el GMSC se encarga de enrutarla.

1.4.6. Identidad internacional del abonado a un móvil

La identidad internacional del abonado a un móvil (IMSI) se almacena de manera permanente en la tarjeta SIM. Es un código de identificación único, de quince dígitos, para cada dispositivo de telefonía móvil en el sistema GSM. Los primeros tres dígitos del IMSI (del ingles *International Mobile Subscriber Identity*) corresponden al código de país móvil (MCC del ingles *mobile country code*), los siguientes dos dígitos corresponden al código de red móvil (MNC del ingles *mobile network code*). Los diez dígitos restantes corresponden al número de identificación del abonado móvil (MSIC del ingles *mobile subscriber indentification number*). En la tabla 1.1 se puede observar el formato del IMSI.

Tabla 1.1. Formato del IMSI

MCC	MNC	MSIC

1.4.7. Registro de ubicación

El registro de ubicación base (HLR del ingles *Home Location register*) es una entidad funcional a cargo del almacenamiento de los datos de los abonados móviles. Una Red móvil terrestre publica (PLMN del ingles *Public Land Mobile Network*) puede tener una o varias HLRs, dependiendo del número de abonados, la capacidad del equipo y la organización de la red. Se almacenan 2 tipos de informaciones:

- La información de suscripción.
- Información de ubicación que permite enrutar las llamadas hacia el MSC donde e encuentra el móvil.

En el HLR se almacenan dos tipos de números relacionados a la suscripción móvil:

- Identidad internacional de estación móvil (IMSI).
- Uno o mas número(s) ISDN (del ingles *Integrated Services Digital Network*) internacional de estación móvil (MSISDN)

El IMSI o el MSISDN se usara como clave para acceder a la información en la base de datos para una suscripción móvil

1.4.8. Registro de ubicación del visitante

Los móviles son controlados por el registro de ubicación del visitante (VLR del ingles *Visitor Location Register*) correspondiente al área en el cual se encuentran. Cuando un equipo móvil entra a una nueva ubicación se inicia un procedimiento de registro. El MSC a cargo del área detecta este registro y transfiere la identidad del área de ubicación (donde el móvil se encuentra) al VLR.

El registro de ubicación del visitante (VLR) contiene la información necesaria para gestionar las llamadas establecidas o recibidas por los equipos móviles registrados en su base de datos, la cual contiene los siguientes elementos:

- Identidad internacional de estación móvil (IMSI).
- Uno o mas número(s) ISDN internacional de estación móvil (MSISDN).
- El número de itinerancia de la estación móvil (MSRN).
- El identificador temporal de estación móvil (TMSI del ingles *Temporary Mobile Subscriber Identity*).
- El identificador local de estación móvil (LMSI del ingles *Local Mobile Subscriber Identity*).
- El área de ubicación donde el móvil fue registrado. Esta data sera utilizada para llamar al móvil.

1.4.9. Centro de autenticación

El centro de autenticación (AuC del ingles *Authentication Center*) se encuentra asociado con una HLR, y almacena una clave de identidad para cada abonado móvil registrado con el HLR asociado. La clave se utiliza para generar:

- Datos que se utilizan para autenticar el IMSI.
- Una clave que se utiliza para cifrar la comunicación del radio enlace entre el móvil y la red.

1.4.10. Registro de la identidad del equipo

El registro de la identidad del equipo (EIR del ingles *Equipment Identity Register*) es una base de datos en la que existe información sobre el estado de los móviles. Dentro de esta base de datos existen tres listas de IMEI: la blanca, la gris y la negra.

- La lista blanca identifica a los equipos que están autorizados de recibir y realizar llamadas. esta lista debe existir siempre en el EIR, aun cuando sea la única; las otras dos son opcionales
- La lista gris identifica a los equipos que pueden hacer y recibir llamadas, pero que pueden ser monitoreados para descubrir la identidad del usuario utilizando la información almacenada en la tarjeta SIM.
- La lista negra identifica a los equipos a los que se les impide conectarse a la red. Por lo tanto, no pueden realizar ni recibir llamadas.

1.5. Establecimiento de la llamada

Antes de poder realizar una llamada el móvil debe estar registrado en el sistema. Existen dos tipos de procedimientos, uno es para la llamada originada por el móvil (MOC del ingles *Mobile-Originated Call*), y el otro es para la llamada culminada en un móvil (MTC del ingles *Mobile-Terminated Call*). Se describirá el MOC para dar una idea del intercambio de mensajes utilizado en el sistema GSM.

De manera equivalente al procedimiento de actualización de ubicación, el móvil inicia el procedimiento con una solicitud de canal, la cual es respondida por el sistema con una asignación de canal. El móvil le indica al sistema cual sera el uso de dicho canal (en este caso establecer una llamada). Antes de que el procedimiento continue el móvil debe autenticarse nuevamente. Para proteger cualquier mensaje de señalización adicional de un tercero, la red puede ahora, con el siguiente mensaje, indicarle al móvil que empiece a cifrar los datos. Cifrar quiere decir que los mensajes se transmiten de una manera que solo el móvil y la BTS entienden. En el mensaje de configuración, el móvil envía el número al que desea llamar. Mientras se realiza la llamada, el BSC mediante la BTS asigna un canal de trafico, en el que se realiza el intercambio de datos de usuario. Distintos tipos de mensaje y datos de usuarios se mueven en distintos tipos de canales. Los distintos tipos de canales serán descritos en el siguiente capitulo.

El procedimiento MTC opuesto es casi idéntico al MOC.

1.6. Relevo

El procedimiento de relevo (*Handover*) es un medio para continuar una llamada cuando el móvil cambia de celda. Antes de la introducción de este mecanismo, la llamada se perdía en el momento que cambiaba de celda o cuando la distancia entre el móvil y una estación base en particular era muy larga.

En una red celular, una celda tiene un conjunto de celdas vecinas, el móvil monitorea continuamente el nivel de potencia recibido de dichas celdas. Para realizar esto, la BTS le entrega al móvil una lista de BTSs (canales) en los cuales realizar las medidas de potencia. La lista es transmitida en el canal de control (otra vez, información del sistema), que es el primer canal que el móvil sintoniza cuando se enciende. El móvil realiza mediciones continuas en la calidad y el nivel de potencia de su celda y las celdas adyacentes. Los resultados de dichas medidas son puestos en un *reporte de medición*, el cual es enviado periódicamente a la BTS. La BTS misma puede también estar realizando mediciones en la calidad y potencia del enlace con el móvil. Si estas mediciones indican la necesidad de un relevo, este puede ser realizado sin demora, ya que la BTS para su relevo ya es conocida. Las mediciones llegan periódicamente y reflejan el punto de vista del móvil. Depende del operador actuar en base a diferentes niveles de calidad o de potencia, y las restricciones o umbrales del relevo pueden ajustarse en función del entorno cambiante y las condiciones de funcionamiento. [1]

El sistema GSM diferencia distintos tipos de relevo. Dependiendo de que tipo de frontera esta cruzando el móvil, una entidad distinta puede tomar el control del relevo para asegurar que un canal este disponible en la nueva celda. Si un relevo se realiza dentro del área de una misma BSC, el relevo sera controlado por la BSC sin consultar a la MSC, la cual en cualquier

Tabla 1.2. Procedimiento de establecimiento de llamada [1]

MS		BTS	Accion
	———⟶		Solicitud de canal.
	⟵———		Asignación de canal.
	———⟶		Solicitud de asignación de canal
	⟵———		Solicitud de autenticación.
	———⟶		Respuesta de autenticación.
	⟵———		Comando de cifrado.
	———⟶		Cifrado completado, de ahora en adelante los mensajes están cifrados.
	———⟶		Mensaje de configuración, indicando el número deseado.
	⟵———		Procedimiento de llamada, la red enruta la llamada con el número deseado.
	⟵———		Asignación del canal de tráfico para el uso designado "intercambio de datos de usuario"
	———⟶		Asignación completada, desde ahora todos los mensajes son intercambiados por el canal de tráfico
	⟵———		Alertando, el número solicitado no esta ocupado y el teléfono esta sonando.
	⟵———		Conexión, el número de destino acepto la llamada.
	———⟶		Reconocimiento de conexión, ahora la llamada esta operativa y ambos pueden hablar con el otro.
	⟵———⟶		Intercambio de datos de voz.

caso, debe ser al menos notificada. Este relevo se le conoce como relevo simple intra BTSs (Figura 1.6).

Si, en cambio, el móvil esta cruzando el borde de la BSC, entonces la MSC tiene que manejar el relevo para asegurar la transición suave de la conversación (Figura 1.7). Esto aplica también para relevos entre MSC. La única diferencia, en este ultimo caso, es que aunque el móvil sea manejado eventualmente por la segunda MSC, la primera MSC tiene que mantener el control de la gerencia de la llamada.

1.7. Voceo (paging)

Operación mediante la cual las entidades pertenecientes a la red fija de un sistema celular (BTSs, BSCs) buscan localizar a un móvil en especifico perteneciente a ella. Se utiliza principalmente para ubicar a un móvil antes del inicio de una llamada que culmina en el móvil.

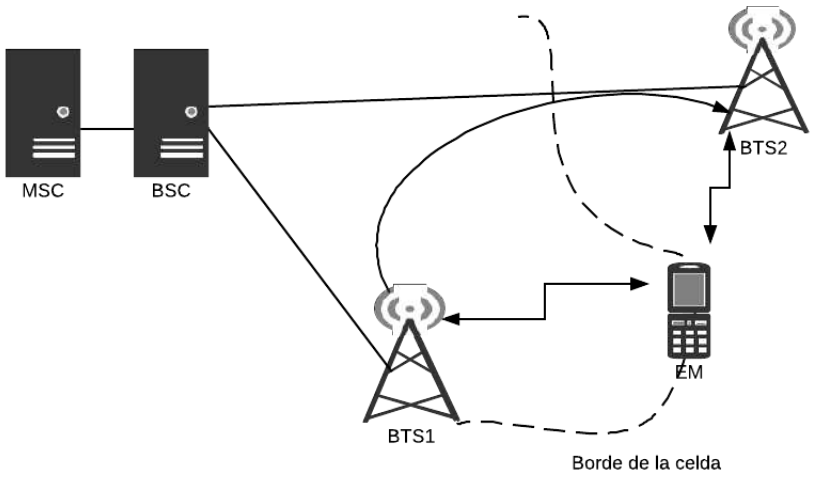

Figura 1.6. Relevo simple intra BSC

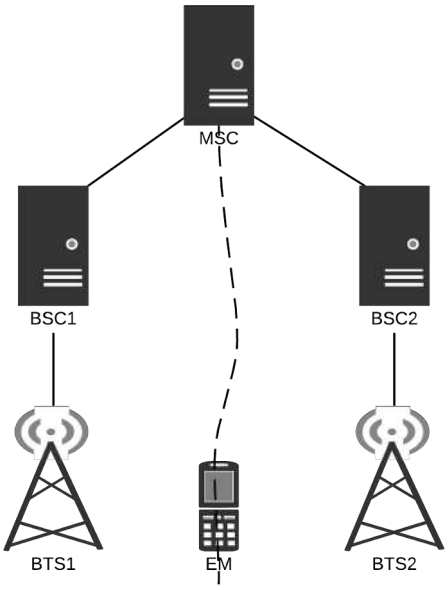

Figura 1.7. Relevo entre MSCs

1.8. Drive Test

Es una técnica muy utilizada para verificar la calidad del servicio celular, la cual consiste en la recolección de datos de un sistema móvil celular, a través de un recorrido sobre el área de cobertura. Estos datos permiten determinar fallas de cobertura y desempeño en zonas urbanas, rurales e incluso perimetral entre edificios. El Drive Test, también ayuda a determinar el rendimiento y la funcionalidad de terminales en su proceso de relevo. Esta

prueba se realiza mediante un software instalado en una computadora que se dedica a analizar los datos que recibe de un GPS, de un móvil en modo normal y otro en modo de ingeniería, que se encarga de obtener los datos mas relevantes de los canales y los eventos generados por la red; adicionalmente se utiliza un vehículo para recorrer una determinada zona.

Los terminales móviles se utilizan de la siguiente manera. El primer teléfono celular en modo normal, se utiliza para realizar las llamadas, mientras que el segundo teléfono se utiliza en modo de ingeniería para medir varios parámetros y eventos que ocurren cuando el celular esta en reposo y durante una llamada [5].

Entre los parámetros que se desean medir se encuentran:

- Intensidad de señal recibida (Indicador de fuerza de la señal recibida RSSI (del ingles *Received Signal Strength Indicator*), RxLev (indica la fuerza promedio de la señal recibida) que se rige por la tabla 1.3).
- Relación señal a interferencia (Relación portadora a interferencia $\frac{C}{I}$ (del ingles *Carrier to interference ratio*)).
- Calidad de la señal (Tasa de error de bit BER (Del ingles *Bit Error Rate*), Tasa de error de bloque BLER (Del ingles *Block Error Rate*), Calidad recibida RXQUAL (Del ingles *Receiver Quality*) que se rige por la tabla 1.4).
- Celda con mejor señal.
- Zonas de relevo (*Handover*).
- Zonas fuera de cobertura.

Tabla 1.3. Niveles de RxLev [6]

RXLEV	Rango en dBm
0	menor a -110
1	-110 a -109
2	-109 a -108
3	-108 a -107
4	-107 a -106
...	...
61	-50 a -49
62	-49 a -48
63	mayor a -48

El BER se define como la relación entre los bits recibidos erróneamente y todos los bits enviados por encima del codificador convolucional. El BLER se define como la relación entre el número de bloques de bits erróneos recibidos entre el número total de bloques enviados, un bloque se define como erróneo cuando la comprobación de redundancia cíclica (CRC) falla.

Tabla 1.4. Niveles de RXQUAL [6]

RXQUAL	Rango de BER
0	menor a 0.1 %
1	0.26 % a 0.3 %
2	0.51 % a 0.64 %
3	1.0 % a 1.3 %
4	1.9 % a 2.7 %
5	3.8 % a 5.4 %
6	7.6 % a 11.0 %
7	mayor a 15.0 %

1.8.1. Tipos de Drive Test

Para determinar los distintos parámetros de la red, se utilizan dos tipos de drive test:

- Antes del diseño de la red:
  - Se realiza para caracterizar el modelo de propagación y los efectos del desvanecimiento.
  - Los datos recolectados permitirán ajustar el modelo de predicción de propagación para las simulaciones.
- Posterior al diseño de la red:
  - Se realiza para verificar la configuración de las celdas vecinas y optimización de la red.
  - Principalmente para verificar los objetivos de cobertura.
  - Verificar el solapamiento de las celdas vecinas, que permitan realizar el traspaso.
  - Verificar la calidad del servicio.
  - Mantenimiento periódico.
  - Cambio de mercado.
  - Por quejas de cliente

1.8.2. Criterio de muestreo

La señal recibida de la red celular se muestrea y luego se promedian varias muestras, sobre una ventana espacial (x-y), llamada “Bin”. Se deben cuidar los detalles sobre el muestreo.

Cuando se realizan medidas de RF (Radiofrecuencia), el criterio de muestreo busca eliminar la componente de desvanecimiento a pequeña escala, de manera de obtener el comportamiento a gran escala. Las medidas de la señal de RF se debe promediar sobre intervalos de al menos 40λ. Si la distancia es mas corta el comportamiento del desvanecimiento a pequeña escala,

no podrá ser eliminado y podría afectar al promedio local; si la distancia es mayor, el valor obtenido no representa un promedio local de la señal, ya que posee puntos de bins distintos.

El número de muestras de RF tomadas sobre los 40λ debe ser igual o superior a 50 muestras. Esto se conoce como la regla de Richard [7]. Dependiendo de la frecuencia de operación y la velocidad del vehículo se obtendrá la frecuencia de muestreo. Por ejemplo, con una frecuencia f = 1900 MHz :

$$\lambda = \frac{C}{f} = \frac{3 * 10^8}{1900 * 10^6} = 0{,}158m \tag{1.1}$$

$$40\lambda = 40 * 0{,}158m = 6{,}32m$$

Como observamos en la ecuación 1.1 se deben promediar al menos 50 muestras en 6.32 metros (para este ejemplo) ó 1 muestra cada 0.1264m (6.32m/50). Reescribimos Rs como se ve en 1.2:

$$R_s = \frac{Muestras}{Segundo} = \frac{Velocidad}{\frac{Distancia}{Muestras}} \tag{1.2}$$

Si colocamos por ejemplo 50 Km/h y los valores utilizados anteriormente tenemos:

$$R_s = \frac{\frac{50*10^3 m}{3600 s}}{0{,}1264 m/muestra} = 110 \frac{muestras}{segundo}.$$

Esto quiere decir que si se va en un vehículo a 50 km/h, el sistema debe tomar 110 muestras/segundo. Sin embargo en la práctica los equipos reportan a una tasa Rs fija, y que puede ser lenta, por lo que en este caso, se debe modificar la velocidad del vehículo durante el drive test. Reescribiendo la ecuación 1.2 nos queda 1.3:

$$V_{max} = 0{,}8 * \lambda * R_s \tag{1.3}$$

con lo cual podremos obtener la velocidad máxima a la que debe ir un carro en base a la frecuencia de la señal y de muestreo.

Como se menciono anteriormente es necesario definir un Bin, el tamaño de esta ventana de promedio debe ser lo suficientemente pequeña para capturar variaciones lentas debido al ensombrecimiento y lo suficientemente grande para promediar las variaciones rápidas debido a los multitrayectos. El tamaño se selecciona típicamente entre 40λ a 1500m, es decir todas las medidas en este cuadro se promedian para entregar un solo valor.

No todos los bins dentro del área de cobertura pueden ser probados, así que se debe considerar un gran numero de muestras. Con un mayor numero de bins el nivel de confianza en los resultados es mayor, en general, para tener una confianza aceptable son necesarios entre 300 y 400 bins.

CAPÍTULO 2

PROPUESTA DEL SISTEMA

En este capítulo se presenta la propuesta del sistema, que esta compuesta por un elemento de comunicación, uno de procesamiento, uno de ubicación, uno para el almacenamiento y presentación de los datos. Los dispositivos que se utilizaron para dicho propósito son el ESP8266, el módulo SIM900, el chip de GPS XM37-1612 y un computador respectivamente. Como se puede observar en la figura 2.1.

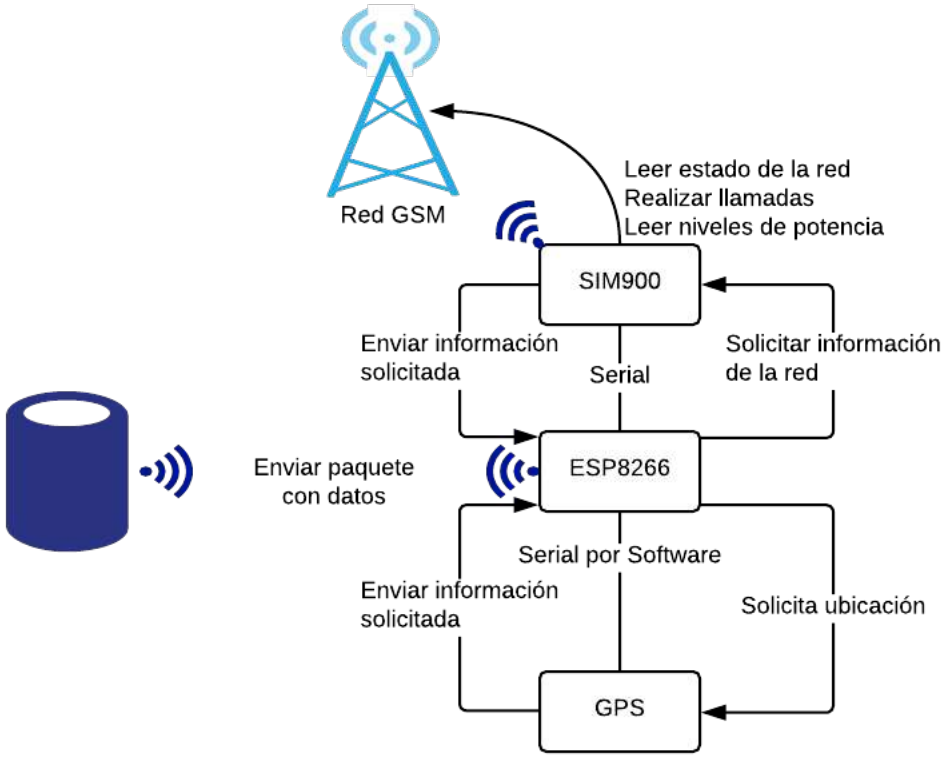

Figura 2.1. Propuesta de diseño del sistema

2.1. SIM900

El SIM900 es un módulo de GSM/GPRS (*Global System for Mobile/ General Packet Radio Service*) que funciona en las frecuencias GSM 850 MHz, EGSM 900 MHz, DCS 1800 MHz y

PCS 1900 MHz, utiliza ranuras múltiples de clase 10, clase 8 y es compatible con los esquemas de codificación de GPRS CS-1, CS-2, CS-3 y CS-4 (CS del ingles *Coding Schemes*). Gracias a su tamaño reducido de 24*24*3 milímetros, el SIM900 puede cumplir con casi todos los requerimientos de espacio (físico) en aplicaciones de usuarios tales como M2M (máquina a máquina), teléfonos inteligentes, etc.

El SIM900 además posee 68 conexiones de montaje superficial SMT (del ingles *Surface Mount Technology*) y proporciona todas las interfaces de hardware entre el módulo y la placa (en este caso la placa Wemos D1 mini (ESP8266)). El módulo esta diseñado con tecnología ahorradora de energía de manera que el consumo de corriente sea 1.0mA en el modo suspendido. Además integra el protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*) mediante comandos ATs (del ingles *ATtention*) [8].

Para este diseño se utilizo la placa de desarrollo SIM900A V1 SCH, que se puede observar en la figura 2.2.

Figura 2.2. Placa del SIM900A V1 SCH [9]

2.1.1. Descripción de los pines

En esta sección se detallan los pines que posee la placa de desarrollo SIM900A V1 SCH con sus funciones. En la tabla 2.1 se describe la función de cada pin.

2.1.2. Modos de operación

En la tabla 2.2 se puede observar un resumen de los modos de operación del módulo SIM900.

2.1.3. Comandos AT

El módulo SIM900 se controla mediante comandos AT en su interfaz serial. Los comandos AT implementados en el SIM900 son una combinación del GSM07.05, GSM07.07, la recomendación V.25ter de la ITU-T y los comandos AT desarrollados por SIMCom [10].

Tabla 2.1. Pines de la placa de desarrollo SIM900A V1 SCH

Nombre del pin	Funcionalidad
Encendido	Encender o apagar el modulo
Microfono	Puerto que permite conectar un microfono para comunicación.
Audifono	Puerto que permite conectar un audifono para la comunicación.
Fuente de alimentación	Conector para alimentar la tarjeta, es necesario toma corriente o fuente de 5 voltios.
P4	Pines de configuración para definir la comunicación serial mediante TTL ó DB9 (DB9 unir pines 1-3 y 6-8. TTL unir pines 3-5 y 4-6).
Puerto de antena	Permite conectar una antena al chip.
P2	Pines de comunicación serial TTL (2 Tx, 4 Rx, 6 tierra)
Conector DB9	Comunicación serial mediante conector DB9.

Los comandos AT pueden ser divididos, según su sintaxis, en tres categorías: “básica”, “parámetro S”, “extendida”.

Sintaxis básica: Estos comandos AT tienen los siguientes formatos “AT<x><n>” o “AT&<x><n>”, donde “<x>” es el comando y “<n>es/son el/los argumento/s para ese comando.

Sintaxis del parámetro S: Estos comandos AT tienen el formato “ATS<n>=<m>”, donde “<n>” es el índice del registro S a modificar, y “<m>” es el valor que se le asignara. “<m>” es opcional, si no esta se le asignara un valor por defecto.

Sintaxis extendida: Estos comandos pueden operar en cuatro modos, como se muestran en la siguiente lista.

- Comando de prueba: AT+<x>=?, el equipo devuelve una lista de parámetros y rangos de valores establecidos con el correspondiente comando de escritura o por procesos internos.
- Comando de lectura: AT+<x>?, el equipo devuelve el valor actual del parámetro o parámetros.
- Comando de escritura: AT+<x>=<...>, este comando le permite al usuario establecer el valor de un parámetro.
- Comando de ejecución: AT+<x>, el comando de ejecución lee parámetros no modificables por el usuario, afectados por procesos internos en el motor GSM.

Tabla 2.2. Resumen de los modos de operación [8]

Modo	Función	
Operación normal	GSM/GPRS SLEEP	El módulo pasara al modo suspendido de manera automática si no hay transmisiones en el aire y no hay interrupciones por hardware (como interrupciones del GPIO o datos en el puerto serial). En este modo, el consumo de corriente se reducirá al nivel mínimo. En este modo, el módulo puede recibir mensajes de voceo (*paging*) y SMS.
	GSM IDLE	El módulo se encuentra registrado en la red GSM, y esta listo para la comunicación.
	GSM TALK	La conexión entre dos abonados esta en proceso. En este caso, el consumo de energía depende de las configuraciones de la red como Transmisión discontinua o DTX (del ingles *Discontinuous Transmission*) encendida o apagada , Tasa completa, Tasa completa mejorada o Tasa media FR/EFR/HR (del ingles *Full Rate/Enhanced Full Rate/Half Rate*), secuencias de saltos, antenas.
	GPRS STANDBY	El módulo esta listo para transmitir datos GPRS, pero aun no se ha enviado ni recibido datos. En este modo el consumo de corriente depende de los parámetros de la red y la configuración GPRS.
	GPRS DATA	Hay una transferencia de datos GPRS en curso (PPP o TCP o UDP). En este modo, el consumo de corriente esta relacionado a los parámetros de la red (por ejemplo el nivel de control de potencia); tasas de datos de subida, de bajada y la configuración GPRS (por ejemplo múltiples ranuras).
Apagado	Para entrar a este modo se utiliza el comando AT "AT+CPOWD=1" o el PWRKEY. La unidad de manejo de energía corta el suministro de energía a la sección banda base del módulo, y solo queda la fuente de alimentación para el RTC (siglas del ingles *Real Time Clock*). Tanto el software como los puertos seriales no se encuentran operativos.	
Modo de funcionalidad mínima	El comando AT "AT+CFUN" se puede utilizar para configurar el módulo en este modo sin eliminar la fuente de alimentación. En este modo, la sección de RF (radiofrecuencia) del módulo no funcionara y no se podrá acceder a la tarjeta SIM, sin embargo los puertos seriales se encontraran accesibles. El consumo de energía en este modo es menor al modo normal.	

2.1.3.1. Comandos AT utilizados

Los comandos AT utilizados fueron:

- AT+CREG: Este comando se utiliza para registrar el módulo en la red GSM.
  - Este comando recibe como parámetro un numero entre 0 y 2, siendo 0 deshabilitar los mensajes continuos de registros en la red, 1 habilitar los mensajes continuos de la red solo con el estado y 2 habilitar los mensajes continuos de la red con toda la información estado, área de ubicación e identificación de la celda.
  - Los estados son:
    0 Equipo no registrado.
    1 Registrado en la red local.
    2 No registrado, sin embargo el equipo esta buscando un nuevo operador para el registro.
    3 Registro negado.
    4 Desconocido.
    5 Registrado en itinerancia.
- AT+CENG: Permite configurar el módulo en modo de ingeniería, el cual permite observar los parámetros de la red, tales como: ARFCN (números de canal de radiofrecuencia absoluta) del BCCH (Canal de control de difusión), nivel de potencia recibido, calidad de la señal recibida, código de país, código de la red, código de identidad de la estación base BSIC (del ingles *Base Station Identity Code*), identificación de la celda, nivel mínimo de acceso a la celda, potencia de transmisión máxima del móvil, código de área de ubicación, avance temporal.
  - Este comando es de tipo extendido, recibe dos parametros para su configuracion, el primero indica el modo en el que trabajara el equipo y el segundo nos permite configurar si deseamos que aparezca la identificacion de la celda.
  - Modos de operacion:
    0 Apagar modo de ingeniería.
    1 Encender modo de ingeniería.
    2 Encender modo de ingeniería y activar mensajes de reporte continuo sobre la red.
    3 Encender modo de ingeniería con mensajes de reporte corto.
  - Respuesta esperada: +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 Permite realizar una llamada. Este es un comando AT básico, recibe como parametro el número al cual se desea llamar, terminado en punto y coma.

- ATH Comando que termina una llamada en proceso. Este es un comando AT básico, recibe no recibe ningun parametro.
- AT+MORING: Comando que activa o desactiva mensajes continuos del estado de la llamada.
  - Este es un comando AT extendido, recibe un parametro de configuración que indica si este modo se encuentra activo 1 ó apagado 0. De encontrarse activo muestra el estado de la llamada originada en el equipo.
  - Mensajes del equipo:
    - MO RING Mensaje que indica que se esta alertando al número destino.
    - MO CONNECTED Mensaje que indica que la llamada se estableció.
    - NO CARRIER Mensaje que indica que no se puede conectar con dicho número.
- AT+CSQ: Comando que permite observar la tasa de error de bit BER (Del ingles *Bit Error Rate*) y el indicador de fuerza de la señal recibida RSSI (del ingles *Received Signal Strength Indicator*). Es un comando AT extendido, sin embargo no se configura, solo se ejecuta. Su respuesta esperada es el RSSI y el BER en ese orden.
- AT+SAPBR: Este comando permite configurar el modem para aplicaciones basadas en IP.
  - Recibe como parametro dos valores y dos cadenas de caracteres.
    - Parámetro 1: “cmd type” configura que acción realizara el módulo, varia de 0-5:
      0 Cerrar portadora.
      1 Abrir portadora.
      2 Consultar portadora.
      3 Configurar parametros de portadora.
      4 Obtener parametros de portadora.
      5 Almacenar los valores en la NVRAM.
    - Parámetro 2: “cid” varia entre 1-3 e identifica la conexión.
    - Parámetro 3: “ConParamTag” parámetro de la portadora:
      - “CONTYPE” Tipo de conexion a internet.
      - “APN” Nombre del punto de acceso, máximo 50 caracteres.
      - “USER” Nombre del usuario, máximo 50 caracteres.
      - “PWD” Contraseña, máximo 50 caracteres.
      - “PHONENUM” Número de telefono para la llamada CSD.
      - “RATE” tasa de conexión para la llamada CSD.
    - Parámetro 4: “ConParamValue” Valor del parametro de la portadora:
      - Valor del CONTYPE: GPRS ó CSD (del ingles *Circuit-switched data call*).
      - Valor del RATE: 0 2400, 1 4800, 2 9600, 3 14400.

- AT+FTPCID: Este comando selecciona el perfil previamente configurado que se utilizara en FTP.
- AT+FTPSERV: Este comando establece el nombre de dominio o dirección IP del servidor.
- AT+FTPUN: Este comando establece el nombre de usuario que esta registrado en el servidor FTP.
- AT+FTPPW: Este comando establece la contraseña para la sesión.
- AT+FTPGETNAME: Este comando establece el nombre del archivo que se desea descargar.
- AT+FTPGETPATH: Este comando establece la ruta en la que se encuentra el archivo.
- AT+FTPGET: Este comando obtiene el archivo del servidor.

Para ver todos los comandos AT y sus parámetros, acceder al manual de comandos AT que se puede encontrar en [10].

2.2. GPS XM37-1612

El módulo XM37-1612, posee una alta sensibilidad, bajo consumo y tamaño reducido descritos mas adelante. Gracias a su tamaño reducido, resulta fácil de integrar en dispositivos portátiles tales como teléfonos móviles, cámaras y localizadores de vehículos.

El equipo utiliza una predicción de efemérides auto-generada que no necesita asistencia de red ni del microcontrolador, esto es válido hasta por tres días y se actualiza automáticamente cuando el módulo se encuentra encendido y los satélites están disponibles. Las predicciones de efemérides se almacenan en la memoria *flash* integrada, lo cual permite un arranque en frío inferior a los 15 segundos [11]. La placa utilizada se puede observar en 2.3

Figura 2.3. Placa de desarrollo del GPS XM37-1612 [12]

2.2.1. Características del módulo

- Solución MediaTek de alta sensibilidad.
- Soporta 66 canales de GPS
- Bajo consumo.
- Primera localización rápida con nivel bajo de señal.
- Cancelador activo de interferencias de doce tonos múltiples incorporado.
- Predicción de efemérides híbridas para lograr un arranque frío más rápido.
- Registro de datos integrado.
- Convertidor DC/DC incorporado para ahorrar energía.
- Velocidad de actualización de hasta 10 Hz ± 11ns.
- Capacidad de SBAS (sistema de argumentación basado en satélites).
- Soporte QZSS japones.
- Detección y compensación multitrayecto en interiores y exteriores.
- Tamaño reducido 16,0*12,2*2,2 mm.
- Posee pines con tecnología de montaje superficial SMD.

2.2.2. Especificaciones del módulo

En la tabla 2.3 se ve un resumen del receptor GPS.

2.2.3. Descripción de pines

En esta sección se detallan los pines que posee la placa de desarrollo del chip XM37-1612 con sus funciones. En la tabla 2.4 se describe la función de cada pin.

2.2.4. Interfaz de software

El módulo entrega seis tipos de mensajes, en diferentes formatos sobre el posicionamiento, debido a los requerimientos de un drive test se decidio trabajar con el tipo de mensaje RMC (del ingles *Recommended Minimim Specific GNSS Data*). En la tabla 2.5 se puede apreciar el formato y los parámetros del mensaje.

Ejemplo de respuesta:

$GPRMC,213735.000,A,1024.6604,N,06652.9343,W,2.00,250.24,191017„,A*78.

Para mayor detalle en los tipos de mensajes consultar [11].

Tabla 2.3. Especificaciones del receptor GPS [11]

Receptor GPS		
Chip	MediaTek MY3337 (versión ROM).	
Frecuencia	L1 1575.42MHz, código C/A.	
Canales	Soporta 66 canales.	
Tasa de actualización	1Hz por defecto, hasta 10Hz	
Sensibilidad	Rastreo	-162dBm, hasta -165dBm (Con un amplificador de bajo ruido externo).
	Arranque frío	-143.5dBm, hasta -148dBm (Con un amplificador de bajo ruido externo).
Tiempo de localización	Arranque en caliente (cielo abierto)	<1s típicamente.
	Arranque en caliente (en interior)	<30s
	Arranque en frío (cielo abierto)	32s(típicamente) sin GPS asistido.
		<15s (típicamente) con GPS asistido (predicción híbrida de efemérides).
Precisión de la localización	Autónoma	3m (2D RMS (El doble de la distancia de la raíz cuadrada media)).
	SBAS	2.5m (dependiendo de la exactitud de los datos de corrección)
Máxima altitud	18000m	
Máxima velocidad	<515m/s	
Soporte de protocolo	NMEA 0183 ver 4.01	9600 bps (baudios por segundo), 8 bits de datos, sin paridad, 1 bit de parada (por defecto) 1Hz: GGA, GLL, GSA, GSV, RMC, VTG
Características físicas		
Tipo	24 pines SMD	
Dimensiones	16.0mm*12.2mm*2.2mm±0.2mm	

2.3. Módulo ESP8266

El ESP8266 es un sistema en un chip o SoC (del ingles *System on a Chip*) que integra la pila de protocolos TCP/IP y es capaz de conectarse a redes WiFi, permitiendo al dispositivo interactuar con otros elementos en Internet. Este dispositivo fue desarrollado por Espressif Systems y se puede adquirir integrado en una placa de desarrollo (como se muestra en la figura 2.4) a un bajo costo, comparándolo con otros sistemas WiFi [13].

Al principio, se utilizo como un adaptador WiFi para otros microcontroladores, el cual

Tabla 2.4. Descripción de pines

Nombre	Función
GND	Tierra.
Tx	Pin de transmisión serial (3.3 V).
Rx	Pin de recepción serial (3.3 V).
VCC	Pin de alimentación 3.3 V.

Tabla 2.5. Descripción del mensaje GPRMC [11]

Nombre	Ejemplo	Unidad	Descripción
ID del mensaje	$GPRMC		Encabezado del protocolo RMC
Hora UTC	060406.000		hhmmmss.sss
Estado	A		A = Datos validos ó V = datos no validos
Latitud	2503.7148		ggmm.mmmm
Indicador N/S	N		N=norte o S=sur
Longitud	12138.7451		gggmm.mmmm
Indicador E/O	E		E = este o W = oeste
Velocidad sobre el terreno	0.01	Nudos	Verdadero
Curso sobre el terreno	0.00	Grados	
Fecha	180313		ddmmaa
Variación magnética		Grados	
Sentido de variación			E=Este o W=Oeste
Modo	D		A=autónomo, D=DGPS, E=DR, N=Datos no validos, R=Posición Tosca, S=Simulador
Suma de comprobación	*78		
<CR><LF>			Fin del mensaje

utilizaría comandos AT para controlar al ESP8266, esos comandos eran manejados mediante un firmware el cual se encontraba por defecto en el modulo. Sin embargo, la comunidad se dio cuenta de que el dispositivo era programable, lo que permitía mas libertad. por lo tanto en lugar de utilizar un conjunto de comandos AT, existía la posibilidad de desarrollar un protocolo serial optimizado para interactuar con el dispositivo desde otro microcontrolador, y mejor aun, era posible desarrollar aplicaciones autónomas, ya que el ESP8266 integra un microcontrolador de 32 bits.

El dispositivo soporta comandos AT, Espressif posee un SDK (kit de desarrollo de software) para programarlo y actualmente el dispositivo se soporta en el IDE de arduino, lo que facilita

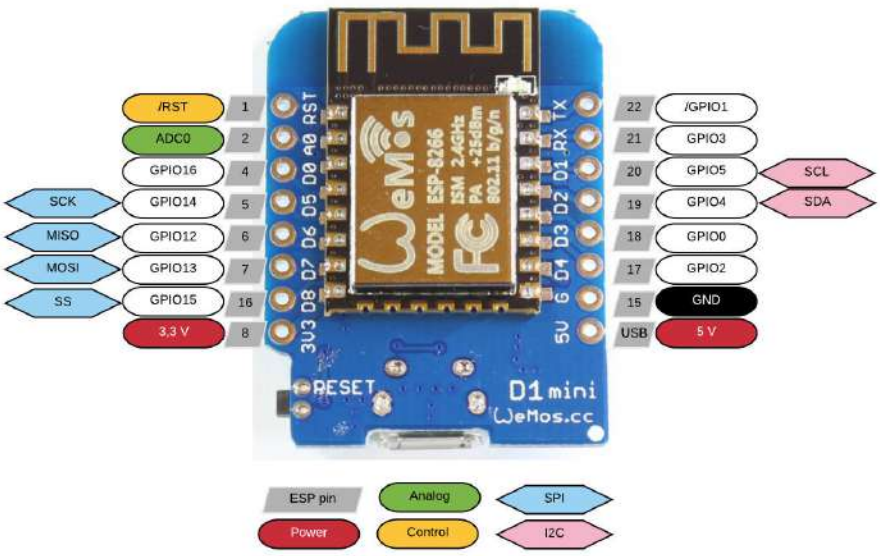

Figura 2.4. Placa de desarrollo D1 mini [14]

la programación [13] [15].

2.3.1. Características del ESP8266

- Soporta 802.11 b/g/n.
- Microcontrolador de 32 bits integrado de bajo consumo.
- ADC de 10 bits integrado.
- Pila de protocolos TCP/IP integrado.
- Soporta WiFi 2.4 GHz y encriptación WPA/WPA2 (del ingles *WiFi Protected Access*).
- Soporta los modos de operación STA/AP/STA+AP.
- 64k de memoria estática de acceso aleatorio (SRAM del ingles *Static Random Access Memory*)
- STBC, 1X1 MIMO, 2X1 MIMO.
- Consumo de corriente en el modo de reposo profundo $<10\mu A$, corriente de fuga al apagar $<5\mu A$.
- Despertar y transmitir paquetes en <2ms.
- Consumo de energía en modo ocioso <10mW (DTIM3).
- Potencia de salida de +20dBm en modo 802.11b.

- Rango de temperatura de funcionamiento -40C 125C.
- Certificado por FCC, CE, TELEC, WiFi Alliance y SRRC.

2.3.2. Descripción de pines

En esta sección se detallan los pines que posee la placa de desarrollo D1mini con sus funciones. En la figura 2.4 se puede observar la asignación de los pines. En la tabla 2.6 se describe la función de cada pin.

Tabla 2.6. Descripción de pines D1Mini [14]

Nombre	Función
TX	Transmision Serial.
RX	Recepcion Serial.
A0	Entrada analógica, máximo 3.3 V de entrada.
D0	GPIO16.
D1	GPIO5, SCL.
D2	GPIO4, SDA.
D3	GPIO0, Pull-up de 10K.
D4	GPIO2, Pull-up de 10K, LED integrado.
D5	GPIO14, SCK.
D6	GPIO12, MISO.
D7	GPIO13, MOSI.
D8	GPIO15, Pull-down de 10K, SS.
G	GND.
5V	5V.
3V3	3.3V.
RST	Reset.

CAPÍTULO 3

IMPLEMENTACIÓN DEL SISTEMA

Como se mencionó en el capítulo anterior el sistema está conformado por un módulo GSM, GPS y ESP8266. Siendo este último el controlador, el cual se encarga de recolectar, procesar y retransmitir los datos obtenidos. En la figura 3.1 se pueden apreciar las conexiones entre los distintos módulos mediante serial. Debido a que la placa del ESP8266 (D1 mini) solo posee un puerto serial, es necesario utilizar la biblioteca *SoftwareSerial.h* que permite utilizar los puertos GPIO (del ingles *General Purpose Input/Output*) en lugar del serial de programación vía USB.

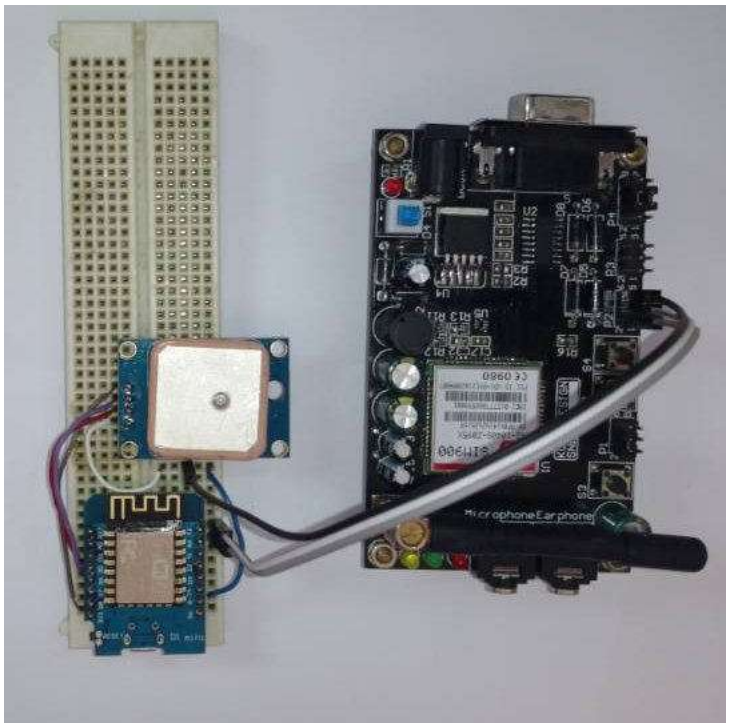

Figura 3.1. Conexiones módulo GSM, GPS y ESP8266

3.1. Sistema de Drive Test

El diagrama de estado que rige el algoritmo del sistema se puede observar en la figura 3.2, cabe destacar que se debe deshabilitar el temporizador WDT (del ingles *Watch Dog Timer*).

Este verifica que el ESP8266 no se encuentre en un ciclo sin realizar ninguna operación y de ser así finaliza la ejecución del programa. Sin embargo el ESP8266 posee dos WTD, uno por software, el cual se puede deshabilitar; y otro por hardware, el cual no se puede deshabilitar, para evitar que el WDT por hardware finalice el programa, se reinicia el temporizador cada vez que nos encontramos en un ciclo de espera.

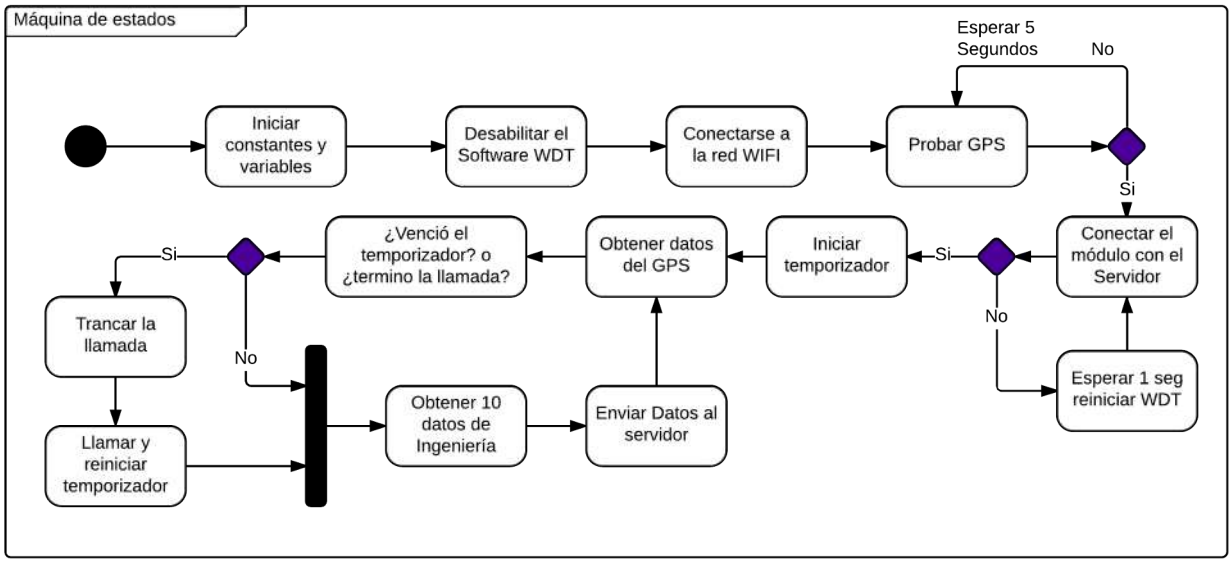

Figura 3.2. Diagrama de estado general del sistema

Luego de que se conecta a la red WiFi y se inicializan todas las variables, se verifica que el GPS se encuentre trabajando correctamente, para realizar esto se creo una función que revisa el mensaje del GPS descrito en el capitulo anterior y se verifica si los datos son validos o no mediante el campo “Estado”, el diagrama que describe a la función se puede observar en la figura 3.3. Como ya se menciono anteriormente el módulo de GPS se comunica con el ESP8266 mediante comunicación serial, debido a que el GPS envía mensajes continuos cada segundo, se decidió deshabilitar las interrupciones que generaba el mensaje serial; de esta manera si se desea obtener la localización del GPS se activa la interrupción, se guarda el mensaje y se vuelve a deshabilitar la interrupción del mensaje de GPS. Si no se realiza esto los mensajes periódicos del GPS podrían llenar el buffer de recepción. En caso de resultar no exitosa la verificación del GPS se volverá a intentar hasta ser exitoso.

Luego de que el GPS se encuentra ubicado se procede a conectarse con el servidor al cual se le enviaran los datos, en caso de fallar se espera un segundo (se reinicia el WDT) y se vuelve a intentar. Una vez conectados al servidor se obtienen los datos del GPS, sin embargo durante cada captación de los datos se verifica que estos sean confiables (mediante el parámetro “Estado” del mensaje). De no ser así se espera al próximo mensaje. Si los datos son validos se guardan para luego ser enviados al servidor. En la figura 3.4 se puede observar el diagrama de la función encargada de obtener los datos.

Una vez se tienen los datos del GPS se procede a solicitar las medidas de ingeniería, primero se verifica si el temporizador que se inició se venció o si se perdió la llamada, de ser así, se termina la llamada en proceso (si la hay), se reinicia el temporizador, se llama a un numero

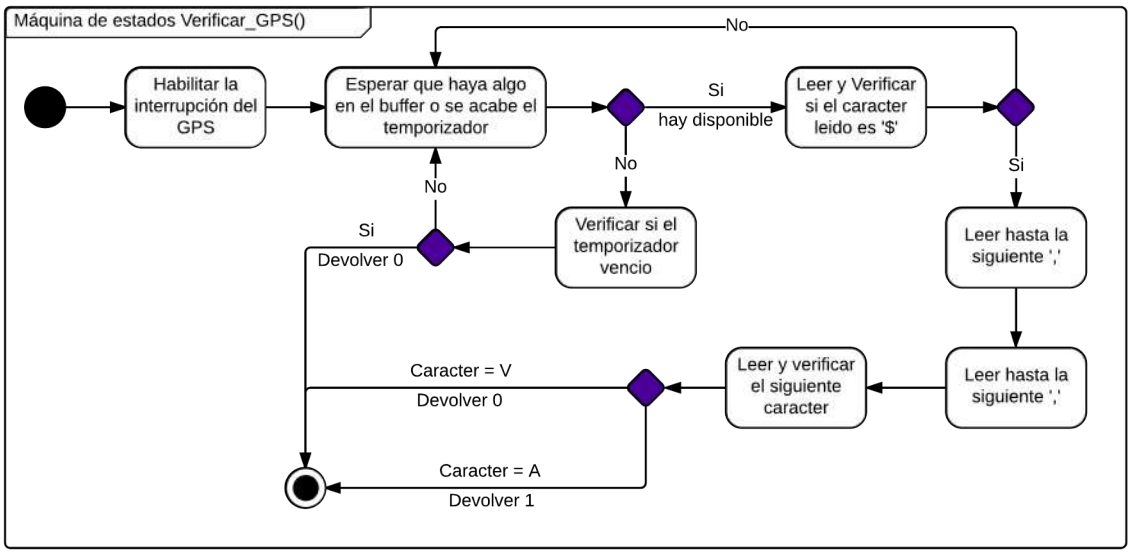

Figura 3.3. Diagrama de estado para la verificación del GPS

de prueba, se obtienen diez datos de ingeniería, se envían los datos al servidor y se vuelven a obtener los datos de GPS. Se decidió escoger diez datos debido a que el tiempo de GPS es muy lento comparado con la toma de datos de ingeniería.

Para obtener los datos de ingeniería, llamar o finalizar la llamada se necesitan enviar comandos AT, con los cuales se controla al SIM900, el diagrama que describe el algoritmo utilizado para enviar y recibir la respuesta de los comandos AT se muestra en la figura 3.5. Como se observa, primero se tiene que enviar el comando AT (que se desea enviar), luego de esto se espera que el modulo responda, se guarda esta respuesta en un arreglo y se verifica si contiene “NO CARRIER”, “MO CONNECTED” o la respuesta esperada, esto se verifica debido a que al perder o establecer una llamada el modulo devuelve estos mensajes respectivamente. Dependiendo de la respuesta se activa una de dos banderas que permiten saber si la llamada se perdió, para realizar otra llamada, o si se estableció, lo cual permite activar la toma de datos de calidad de la llamada. Al recibir la respuesta esperada o al terminar el temporizador finaliza la función.

Los datos de ingeniería que se desean obtener son de relevancia para conocer como se encuentra la red celular, detectar fallos y reportarlos. El SIM900 permite capturar los siguientes datos:

- El numero de canal de radiofrecuencia absoluta (ARFCN del ingles *Absolute Radio-Frequency Channel Number*) del canal de control de difusión. Este número representa en que frecuencia se encuentra el canal de control de la celda que atiende al teléfono al momento de ejecutar el comando. (BCCH)
- Nivel promedio de potencia recibido RxLEV. (RXL)
- Calidad de la señal recibida. (RXQ)

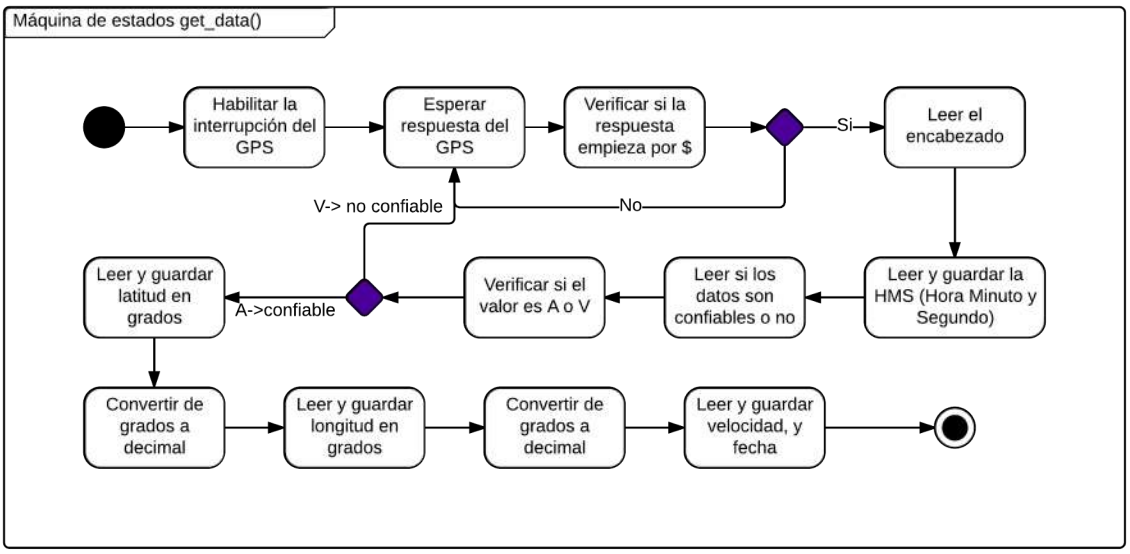

Figura 3.4. Diagrama de estado para la adquisición de datos del GPS

- Codigo de país. (MCC)
- Codigo de red. (MCN)
- Codigo de identificación de la BTS. BSIC
- Identificación de la celda en formato hexadecimal. (CELLID)
- Mínimo nivel de señal para acceder a la red. (RLA)
- Máximo nivel de transmisión en el CCCH. (TXP)
- Codigo de área de ubicación, en formato hexadecimal. (LAC)
- Avance temporal. (TA)
- Tasa de error de bit. (BER)
- Indicador de fuerza de la señal recibida. (RSSI)

Cada uno de los parámetros se guarda en una estructura y se envían al servidor. Los últimos dos parámetros (BER y RSSI) se miden únicamente cuando hay una llamada en proceso, esto se hace para estimar la calidad de la llamada. La figura 3.6 describe como se obtienen y almacenan los datos de ingeniería.

Ya planteado el algoritmo es necesario destacar ciertos valores del mismo. El algoritmo logra obtener los datos de ingeniería en aproximadamente 0.5 segundos, si a esto se le suma que el GPS tarda 1 segundo en entregar de manera exitosa el mensaje esto nos da un total de un segundo y medio por muestra. Debido a la lentitud del GPS para entregar la información

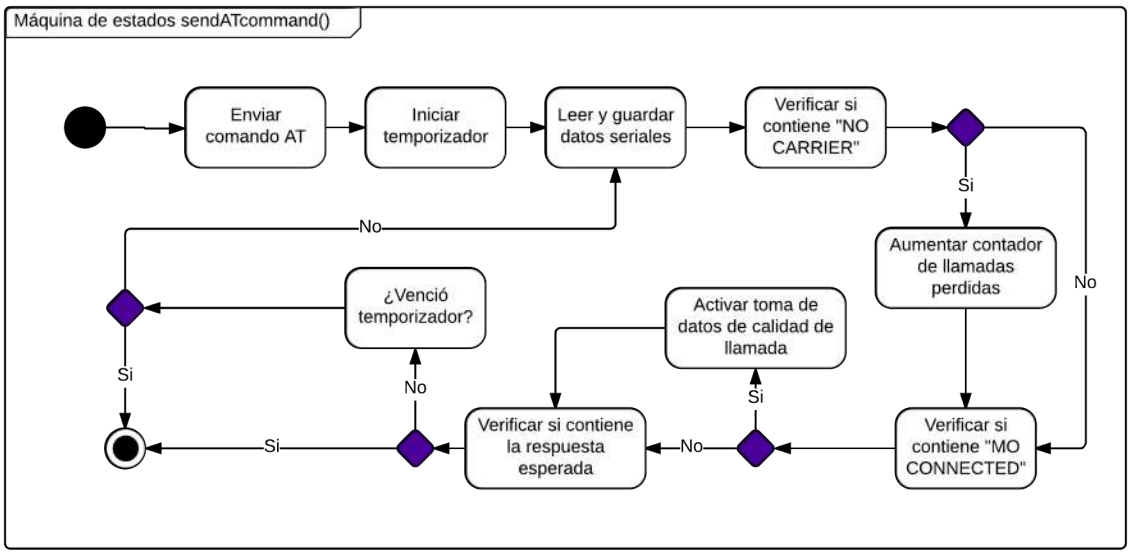

Figura 3.5. Diagrama de estado de enviar comandos AT

se decidio tomar una medida de GPS por cada diez medidas de ingeniería quedando de esta manera una tasa de muestreo:

$$R_s = \frac{10 muestras}{1 segundo(GPS) + 5 segundos(muestras)}$$
$$R_s = 1{,}66 muestras/segundo$$

Si utilizamos la tasa de muestreo obtenida anteriormente y asumiendo la banda de 850 MHz (Movistar Venezuela) tenemos que:

$$V_{max} = 0{,}8 * 0{,}3529 * 1{,}66$$
$$V_{max} = 0{,}4705 \frac{m}{s} * 3{,}6$$
$$V_{max} \approx 1{,}69 \frac{Km}{h}$$

Como se puede observar, para cumplir con el criterio de Richard el vehículo debe ir a una velocidad de 1.69 Km/h. Sin embargo al estudiar el módulo GSM SIM900 se pudo observar que el modulo entrega la información de potencia promediada, dado que el modulo la mide cada 120 milisegundos al final de cada multitrama, lo cual aumenta la cantidad de muestras que se tienen con el sistema como se puede ver la frecuencia de muestreo aumenta y por lo tanto la velocidad también lo hace:

$$R_s = \frac{4 * 10 muestras}{1 segundo(GPS) + 5 segundos(muestras)}$$

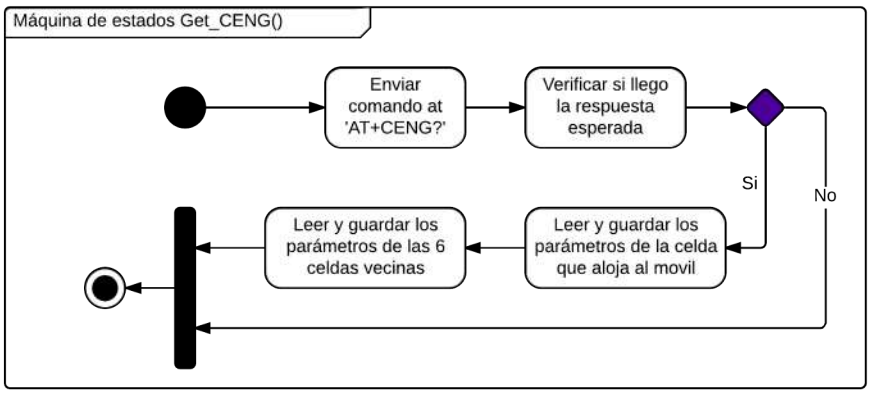

Figura 3.6. Diagrama de estado para la obtención de los datos de ingeniería

$$R_s \approx 6{,}66$$
$$V_{max} \approx 1{,}88\frac{m}{s} * 3{,}6$$
$$V_{max} \approx 6{,}77\frac{Km}{h}$$

3.2. Servidor

El servidor se desarrolló en lenguaje C, éste recibe dos argumentos de entrada: el puerto a utilizar y el nombre de la base de datos del recorrido. El servidor se comunica mediante la interfaz de programación de aplicaciones (API del ingles *Application Programming Interface*) de MySQL con la base de datos y crea una nueva tabla con el nombre que se le proporcionó como parámetro. El servidor debe crear el *socket* y asociarlo al puerto de escucha, que le fue proporcionado como parámetro, luego de esto el servidor permanece esperando conexiones por parte del D1 mini. Una vez el D1 mini se conecte al servidor, este recibirá la información enviada por el D1 mini y la guardará en una estructura de datos que se puede observar en el apéndice A, y guardará dicha estructura de datos en la tabla creada por el servidor, de esta manera completa la conexión con el D1 mini y permanece esperando conexiones. En la figura 3.7 se puede observar el diagrama de estado del funcionamiento del servidor.

3.3. Problemas presentados en el desarrollo

En esta sección se explicaran los problemas que se tuvieron durante el desarrollo, los tres principales problemas fueron el temporizador de perro guardián del ESP8266 (WDT del ingles *WatchDog Timer*), la memoria estática de acceso aleatorio (SRAM del ingles *Static Random Access Memory*) y la velocidad de recepción de datos.

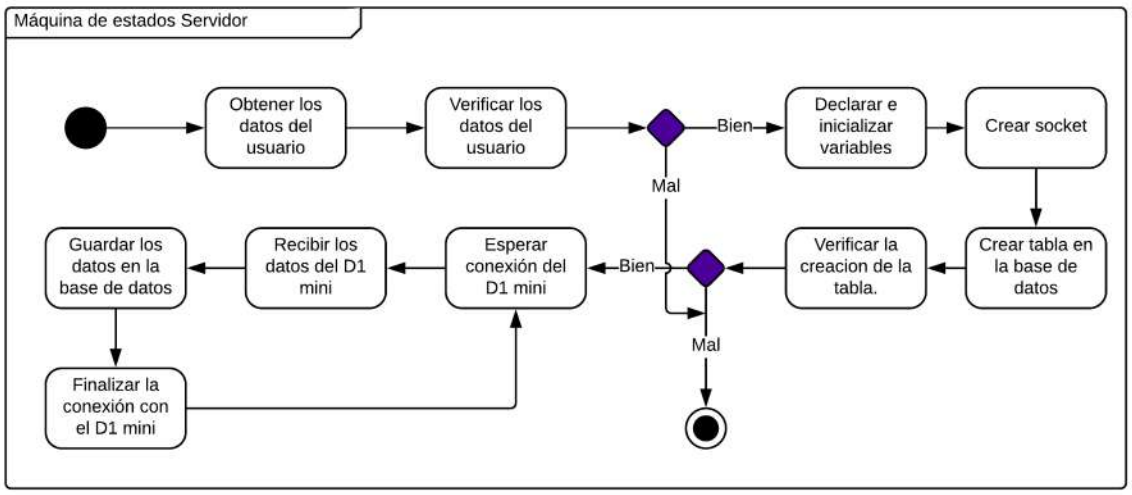

Figura 3.7. Diagrama de estado del servidor

3.3.1. Temporizador de perro guardián

Es un temporizador electrónico que se utiliza para regresar al equipo a un estado seguro luego de un mal funcionamiento. El equipo reinicia el temporizador continuamente para evitar que este termine, en caso de presentar un estado de bloqueo en el que el equipo no logre reiniciar el temporizador, el equipo se reiniciará a un estado seguro. [16]

En el caso del D1 mini, consta de dos WTD, uno por software y uno por hardware, si el D1 mini se encuentra en un ciclo de espera durante mucho tiempo (aproximadamente seis segundos) este se reiniciara. [17]

En el algoritmo planteado se tienen ciclos de espera, los cuales poseen un tiempo de vencimiento mayor a los seis segundos, esto ocasionaba (en las primeras implementaciones) que el D1 mini se reiniciara. Para que esto no sucediera se utilizó la clase EspClass la cual posee funcionalidades que permiten manipular los temporizadores WDT. Primero se deshabilita el WDT de software mediante la función *ESP.wdtDisable()* y luego en cada ciclo de espera largo se coloca *ESP.wdtFeed()* el cual reinicia el WDT por hardware.

3.3.2. Memoria estática de acceso aleatorio [18]

El D1 mini cuenta con 64 kilo Bytes de SRAM, esta se utiliza para distintos propósitos:

- Datos estáticos - este bloque de memoria se reserva para todas las variables globales y estáticas del programa.
- Almacenamiento libre (del ingles *Heap*) - este bloque se utiliza para los elementos que necesiten reservar memoria de manera dinámica. Este bloque crece desde la parte superior del área de datos estáticos y en caso de realizar una nueva reserva se amontonara hasta llegar al bloque de pila.

- Pila (del ingles *Stack*) - La pila se utiliza para almacenar variables locales y para mantener un registro de interrupciones y llamadas a función. La pila crece desde la parte superior de la memoria hacia el almacenamiento libre. Cada interrupción, llamada de función y/o asignación de variables locales hace que la pila crezca. Al regresar de una llamada de interrupción o función se recuperará todo el espacio de la pila utilizado por esa interrupción o función.

Durante las primeras implementaciones del sistema se realizó un algoritmo en el cual se utilizaron dos búfers de recepción de gran tamaño, esto ocasionaba que el bloque de memoria de almacenamiento libre y el de pila se solaparan, corrompiendo así los datos del sistema, ocasionando un error de pila como se puede ver en la figura 3.8. El diseño se planteó para utilizar dos comunicaciones seriales mediante la librería *SoftwareSerial.h*, para solucionar el problema de la memoria se cambió una conexión de *SoftwareSerial* a una conexión mediante el serial de hardware, eliminando así el búfer de mayor tamaño y de esta manera evitar los errores de pila.

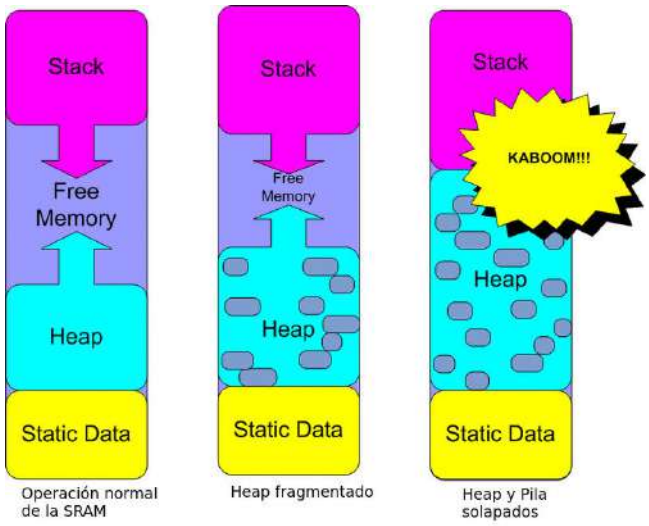

Figura 3.8. Estados de la memoria SRAM [19]

3.3.3. Velocidad de recepción de datos

Debido a los requerimientos de muestreo de un sistema de Drive Test es necesario tomar los datos a una alta velocidad, en las primeras implementaciones del sistema se crearon funciones de envío de comandos AT y recepción de los datos enviados por el modulo GSM, que contaban con un tiempo de recepción de datos fijo en la función. El problema es que para mensajes con pocos caracteres de respuesta y para respuestas largas tardaba la misma cantidad de tiempo. Para resolver esto se decidio colocar el retardo mínimo que se necesita para un caracter y leer hasta que el búfer de recepción se encontrara vacío, de esta manera se asegura que la lectura de las respuestas de los mensajes tarden un tiempo equivalente a su longitud.

CAPÍTULO 4

RESULTADOS

En este capitulo se presentan los resultados obtenidos a partir de la implementación del sistema propuesto en el capítulo anterior. Para esto se creo una biblioteca en C++ que permite utilizar el módulo GSM (del ingles *Global System for Mobile Communications*) y el GPS (del ingles *Global Positioning System*) con mayor libertad, las funciones mas relevantes de dicha biblioteca se describieron en el capitulo anterior. Para la realización de las pruebas, se utilizó un vehículo con toma corriente para poder conectar el módulo GSM, esto debido a que el ESP8266 no puede suministrar la corriente necesaria durante los picos de corriente del módulo.

De acuerdo a los cálculos realizados en el capitulo anterior, se deben realizar las pruebas a una velocidad máxima de aproximadamente 6,7 km/h. Se realizaron cuatro recorridos a distintas velocidades para probar el sistema. La primera prueba se realizó a una velocidad de 5 Km/h, el recorrido fue de aproximadamente 1,2 Km. El cual se puede observar en la figura 4.1.

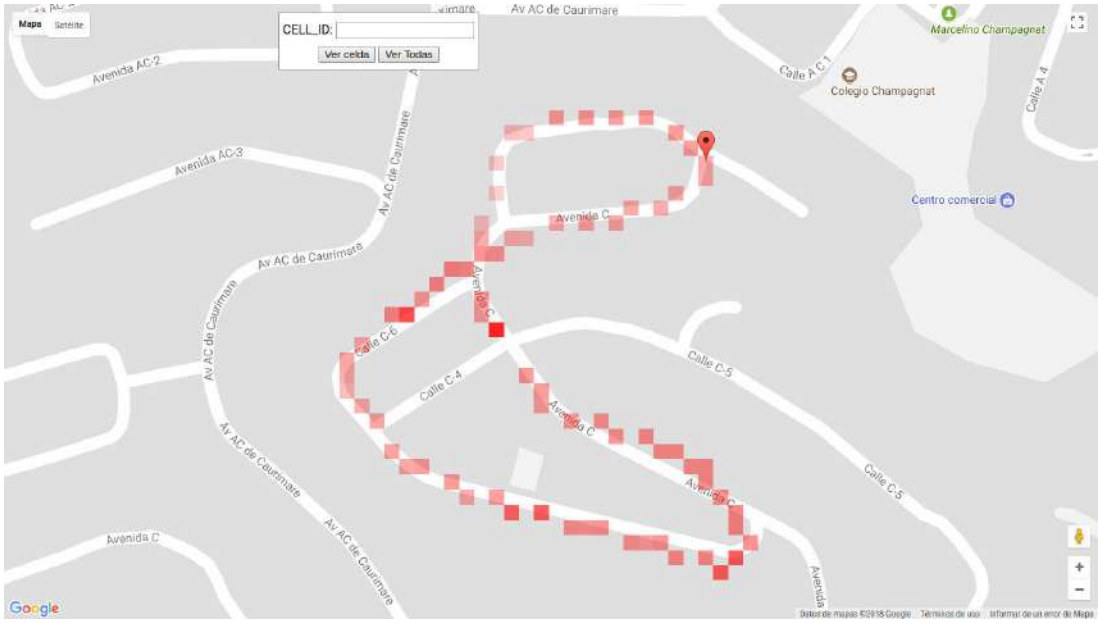

Figura 4.1. Recorrido del Drive Test a 5 Km/h

Se utilizó un bin de 15x15 metros debido a que para la velocidad ideal (aproximadamente 6,7 km/h) se cubre casi en su totalidad el espacio deseado. La intensidad de la potencia recibida es representada mediante cuadrados de color rojo, para tener el valor real de cada

punto se debe obtener de la base de datos del recorrido. Los marcadores en el mapa representan las llamadas perdidas que hubo, al hacer click en el marcador mostrara la información relacionada a la llamada perdida como se ve en la figura 4.2.

Figura 4.2. Marcador

Se puede observar que la llamada perdida se encuentra en una zona de baja cobertura, sin embargo debido a que este fue el único recorrido con una llamada perdida en esta zona, se puede afirmar que la BTS no tenía canales disponibles y por esto no se le pudo entregar un canal al modem. En la herramienta se pueden filtrar zonas de cobertura mediante el identificador de la celda, utilizando el recuadro que se encuentra en la parte superior de la pagina, esto se puede observar en la figura 4.3.

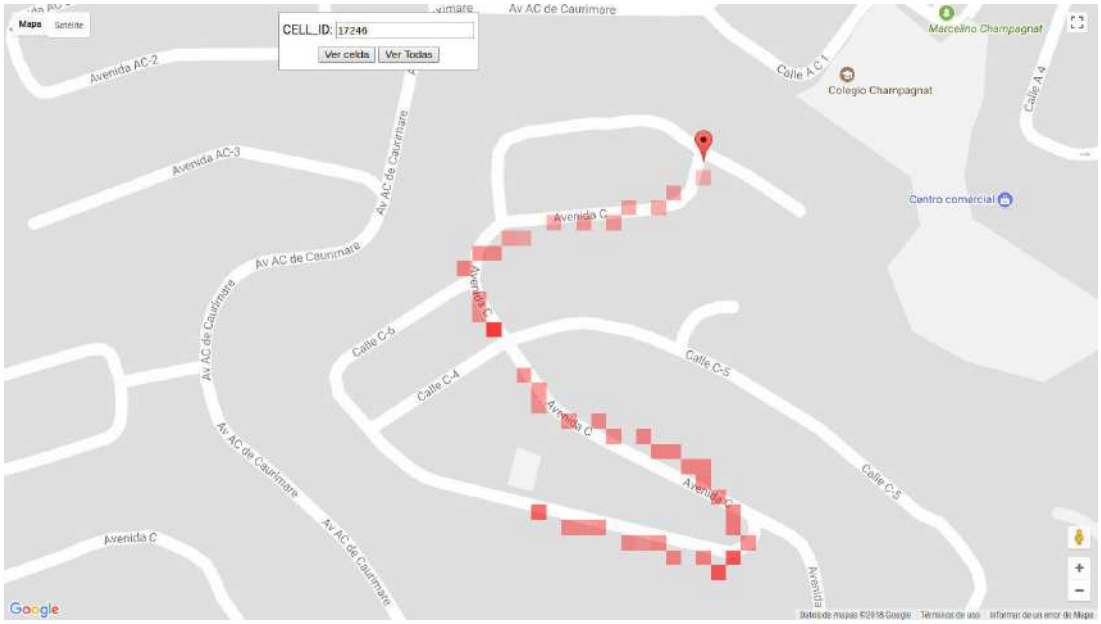

Figura 4.3. Filtrado de celda 12481, en el recorrido de 5Km/h

En este recorrido se realizaron seis llamadas de voz de las cuales solo 1 llamada no se concretó. En la base de datos correspondiente se tienen todos los datos relacionados al Drive

Test, en la herramienta solo se tiene la cobertura, el filtro de celdas y las llamadas perdidas. distancia y velocidad aproximada de 1.05 km y 40 km/h respectivamente El segundo recorrido tuvo una distancia y velocidad aproximada de 1.05 km y 20km/h respectivamente, el resultado del recorrido se puede observar en la figura 4.4.

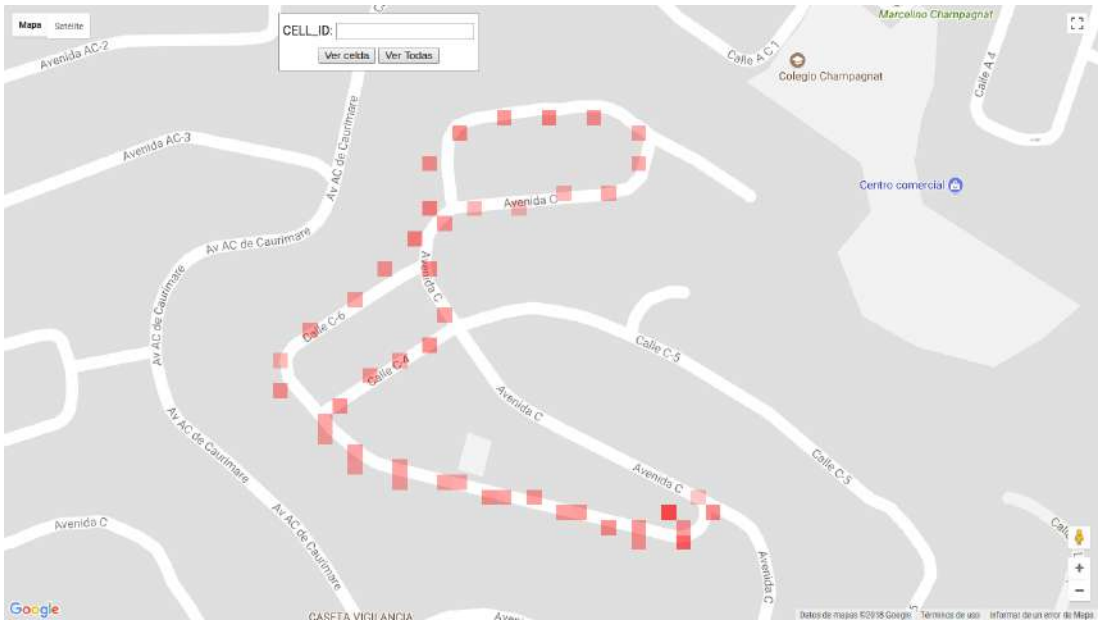

Figura 4.4. Recorrido del Drive Test a 20 Km/h

En este recorrido se realizaron 3 llamadas, todas exitosas, sin embargo se puede observar que los puntos se encuentran separados a una mayor distancia que el recorrido anterior, esto se debe a que el sistema no es capaz de obtener datos a una tasa tan rápida. Sin embargo presenta una buena alternativa de velocidad respecto a los 5 Km/h que se utilizaron en el recorrido anterior.

El tercer recorrido tuvo una distancia y velocidad aproximada de 1.05 km y 40 km/h respectivamente, el resultado del recorrido se puede observar en la figura 4.5.

En este recorrido se puede observar que 40 km/h no es una velocidad deseada para el sistema, debido a que los puntos se encuentran muy espaciados entre ellos, como consecuencia no se logra cubrir de manera exitosa la ruta. Algo a destacar en este recorrido es que durante este se mantuvo la misma BTS, es decir, no se realizo ningún relevo, a diferencia de los otros dos recorridos. Se realizaron dos llamadas, ambas exitosas.

El cuarto recorrido tuvo una distancia y velocidad aproximada de 1.85 km y 40 km/h respectivamente, el resultado del recorrido se observa en la figura 4.6

Este recorrido se realizó pasando dos veces sobre la misma ruta, para tener una mayor cantidad de muestras. En este recorrido se realizaron cuatro llamadas, todas exitosas

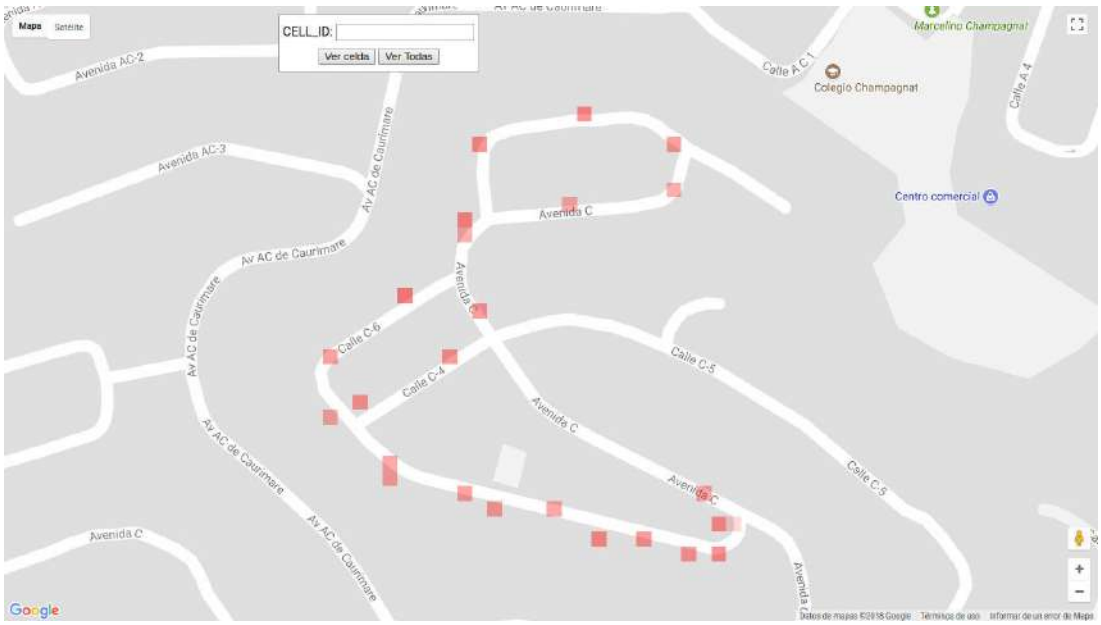

Figura 4.5. Recorrido del Drive Test a 40 Km/h

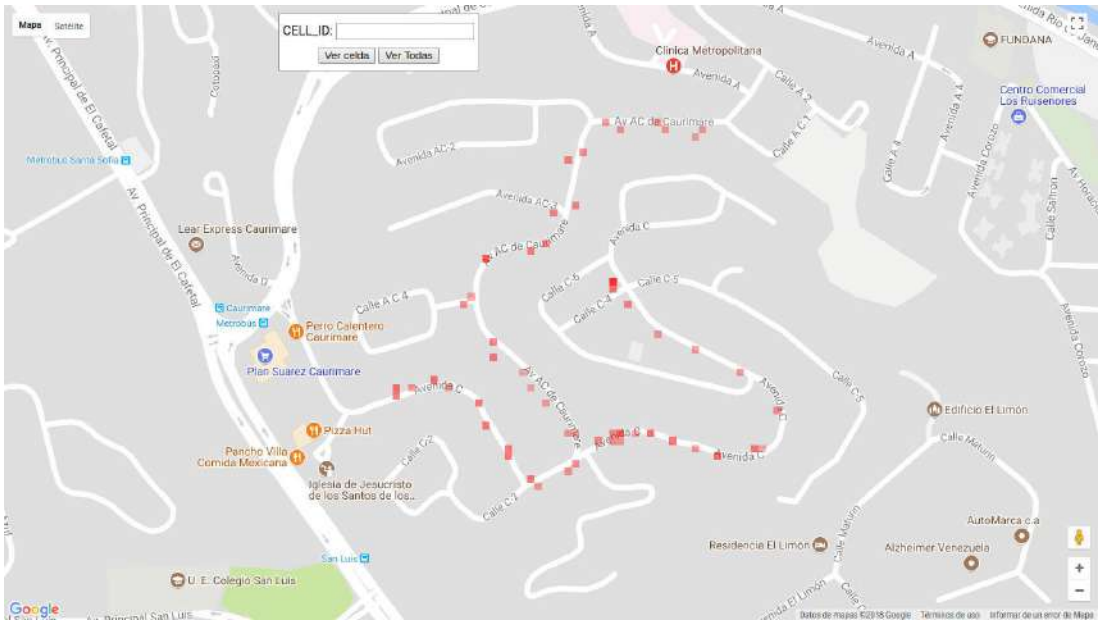

Figura 4.6. Recorrido del Drive Test a 40 Km/h externo

CONCLUSIONES Y RECOMENDACIONES

Este trabajo tuvo como objetivo principal la implementación de un sistema de Drive Test basado en el modulo GSM SIM900, el chip de GPS XM37-1612 y el modulo de WIFI ESP8266, donde no solo es importante el reporte que se genera, sino también el tiempo que se tarda en captar los datos. A partir del trabajo realizado y de los resultados obtenidos se puede concluir que:

- Gracias al estudio realizado sobre el Drive Test, el GPS y el módulo GSM se logró implementar una librería en C++ capaz de obtener los datos de cobertura de una operadora celular a través de un recorrido sobre un área determinada, además de esto también se lograron obtener los datos del GPS (ubicación y tiempo) durante el recorrido.
- Gracias al estudio realizado sobre el ESP8266 se logró implementar una solución inalámbrica del sistema de Drive Test, lo cual permite agregar una mayor cantidad de módulos GSM sin la necesidad de utilizar conexiones físicas, aumentando de esta manera la escalabilidad del mismo.
- Se estimó el área de cobertura de cada celda y la intensidad de la señal en el recorrido, gracias a las rutinas de reporte de calidad de la señal y parámetros de la celda realizados.
- Se realizaron rutinas llamadas de voz con el módulo GSM. Esto permitió verificar los mecanismos de relevo y las zonas de fallo de las llamadas de voz.
- Se desarrolló un servidor capaz de recibir, almacenar y presentar los datos obtenidos del Drive Test.
- Si, es factible implementar un sistema de Drive Test, de manera inalámbrica, con funcionalidades limitadas con el módulo GSM SIM900, el GPS XM37-1612 y el módulo de WIFI ESP8266.

Para futuros trabajos y ampliaciones o mejoras del proyecto realizado, se proponen las siguientes recomendaciones:

- Utilizar un módulo GSM capaz de utilizar el modo de ingeniería extendido, el cual incluye una mayor cantidad de datos de la celda, como por ejemplo, el SIM808. Esto permitirá que el Drive test tenga mayor confiabilidad.
- Utilizar un chip de GPS que posea una mayor tasa de actualización de ubicación.
- Realizar la adaptación del sistema para los módulos de de tercera y cuarta generación.
- Realizar un servidor de datos concurrente, de manera que permita atender a varios módulos de WiFi al mismo tiempo.

REFERENCIAS

[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] H. E. Moreno Trujillo, "PARTICIPACIÓN EN LA OPTIMIZACIÓN DE UNA RED 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, "Recolección de datos de cobertura," 2015.

[8] Shanghai SIMCom Wireless Solutions Ltd, "SIM900 Hardware Design V2.0," p. 47, 2010. [Online]. Available: ftp://imall.iteadstudio.com/IM120417009{_}IComSat/DOC{_}SIM900{_}HardwareDesign{_}V2.00.pdf

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

[10] Datasheet SIM900A, "SIM900_ AT Command," 2009.

[11] L. Shenzhen Simplewe Technology Co., "Product specification 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." [Online]. Available: https://techtutorialsx.com/2016/02/14/esp8266/

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

[15] E. Systems and I. O. T. Team, "ESP8266EX Datasheet," 2015.

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

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

[18] "Arduino Memories | Memories of an Arduino | Adafruit Learning System." [Online]. 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.

APENDICE A

Código del programa

La librería creada se puede conseguir en `https://github.com/LuisAMM/Drive-test`, donde también se encuentran los códigos de las pruebas realizadas en este trabajo.

Archivo de cabecera: AT libreria 1.h

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

typedef struct Info_t
{
    uint16_t arfcn_bcch[7];     // Número de canal de frecuencia absoluta
 ↪  del BCCH
    uint8_t  rxl[7];            // Nivel de potencia de la señal recibida
    uint8_t  rqx;               // Calidad de la señal recibida
    uint16_t mcc[7];            // Código de país
    uint8_t  mnc[7];            // Código de la red
    uint8_t bsic[7];            // Código de identidad de la estacion base
    double cellid[7];           // Identificador de la celda
    uint8_t  rla;               // Nivel mínimo de transmisión para acceder
 ↪  a la red
    uint8_t  txp;               // Máximo nivel de transmisión en el CCCH
    double    lac[7];           // Área de ubicación
    uint8_t   TA;               // Avance temporal

    uint8_t RSSI;               // Nivel de la señal
    uint8_t BER;                // Bit error rate

    unsigned int Llamadas_perdidas;     //contador de llamadas perdidas
    unsigned int Llamadas_realizadas;   //contador de llamadas realizadas

    double latitude;            //Longitud de la coordenada
    double longitude;           //Latitud de la coordenada
    double HMS;                 //hora minuto segundo
    uint DMA;                   //dia mes y año
    double speedOTG;            // speed over ground

}Info;
```

```
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Size
 **     Función      : Obtener el tamaño de una cadena de caracteres
 ↪  generico.
 **     Parámetros   : Cadena de caracteres generico.
 **     Retorna      : El tamaño de la cadena de caracteres.
 **
 ↪  ------------------------------------------------------------------------
*/

int size(char* contador);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : verificar_llamada
 **     Función      : Verificar el valor de la variable de llamada perdida
 ↪  ubicada en la estructura de datos.
 **     Parámetros   :  -
 **     Retorna      : El numero de llamadas perdidas.
 **
 ↪  ------------------------------------------------------------------------
*/

unsigned int verificar_llamada();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Comparador
 **     Función      : Comparar dos cadenas de caracteres.
 **     Parámetros   : dos cadenas de caracteres
 **     Retorna      : NULL si una no contiene a la otra y un apuntador a
 ↪  la respuesta esperada si la contiene
 **
 ↪  ------------------------------------------------------------------------
*/

char* comparador(char* respuesta, char* respuesta_esperada);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Comparador
 **     Función      : Comparar dos cadenas de caracteres y devolver la
 ↪  posicion donde termina la respuesta esperada
```

```
 **     Parámetros  : dos cadenas de caracteres y un apuntador a un entero
↪  sin signo de 16 bits
 **     Retorna     : NULL si una no contiene a la otra y un apuntador a
↪  la respuesta esperada si la contiene.
 **
↪  ------------------------------------------------------------------------
*/

char* comparador(char* respuesta, char* respuesta_esperada, uint16_t *w);
/*
 **
↪  ------------------------------------------------------------------------
 **     Nombre      : Enviar_comando
 **     Función     : Enviar comando AT al SIM900
 **     Parámetros  : Cadena de caracteres que contiene el comando AT
 **     Retorna     : -
 **
↪  ------------------------------------------------------------------------
*/

void Enviar_comando(char* ComandoAT);
/*
 **
↪  ------------------------------------------------------------------------
 **     Nombre      : Recibir_Serial
 **     Función     : Leer del buffer del serial y almacenarlo en la
↪  cadena de caracteres response
 **     Parámetros  : cadena de caracteres que sirve para almacenar,
↪  temporizador, posicion en el arreglo
 **     Retorna     : -
 **
↪  ------------------------------------------------------------------------
*/

void Recibir_Serial(char* response, unsigned long timeout, uint16_t *x);
/*
 **
↪  ------------------------------------------------------------------------
 **     Nombre      : Buscar_inicio
 **     Función     : Buscar el inicio del mensaje.
 **     Parámetros  : Header: encabezado del mensaje almacenado en
↪  response.
 **     Retorna     : Posicion en la que termina el encabezado
 **
↪  ------------------------------------------------------------------------
*/
```

```
uint16_t Buscar_inicio(char* header, char* response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : get_CENG
 **     Función     : Obtener y almacenar en la estructura los datos de
 ↪  ingenieria del sistema
 **     Parámetros  : -
 **     Retorna     : si se realizó con exito 1, si no repsponde el equipo
 ↪  retorna 0, si no consigue el inicio -1
 **
 ↪  ------------------------------------------------------------------------
*/

int8_t get_CENG();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obtener los datos del mensaje de ingeniería que se
 ↪  almacenan en una variable de 32 bits.
 **     Parámetros  : token (delimitador de los valores), salida (variable
 ↪  en la que se almacenara el valor),
                      i (posicion que se esta leyendo), response (cadena
 ↪  de caracteres que contiene toda la respuesta)
 **     Retorna     : Posicion del proximo valor.
 **
 ↪  ------------------------------------------------------------------------
*/

uint16_t separador(char token, uint *salida, uint16_t i, char* response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obtener los datos del mensaje de ingeniería que se
 ↪  almacenan en una variable de 16 bits.
 **     Parámetros  : token (delimitador de los valores), salida (variable
 ↪  en la que se almacenara el valor),
                      i (posicion que se esta leyendo), response (cadena
 ↪  de caracteres que contiene toda la respuesta)
 **     Retorna     : Posicion del proximo valor.
 **
 ↪  ------------------------------------------------------------------------
*/
```

```
uint16_t separador(char token, uint16_t *salida, uint16_t i, char*
 ↪  response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : separador
 **     Función      : Obtener los datos del mensaje de ingeniería que se
 ↪  almacenan en una variable de 8 bits.
 **     Parámetros   : token (delimitador de los valores), salida (variable
 ↪  en la que se almacenara el valor),
                       i (posicion que se esta leyendo), response (cadena
 ↪  de caracteres que contiene toda la respuesta)
 **     Retorna      : Posicion del proximo valor.
 **
 ↪  ------------------------------------------------------------------------
*/

uint16_t separador(char token, uint8_t *salida, uint16_t i, char*
 ↪  response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : separador
 **     Función      : Obtener los datos del mensaje de ingeniería que se
 ↪  almacenan en un double.
 **     Parámetros   : token (delimitador de los valores), salida (variable
 ↪  en la que se almacenara el valor),
                       i (posicion que se esta leyendo), response (cadena
 ↪  de caracteres que contiene toda la respuesta)
 **     Retorna      : Posicion del proximo valor.
 **
 ↪  ------------------------------------------------------------------------
*/

uint16_t separador(char token, double *salida, uint16_t i, char* response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : separadorHEX
 **     Función      : Obtener los datos del mensaje de ingeniería que se
 ↪  encuentran en Hexadecimal.
 **     Parámetros   : token (delimitador de los valores), salida (variable
 ↪  en la que se almacenara el valor),
                       i (posicion que se esta leyendo), response (cadena
 ↪  de caracteres que contiene toda la respuesta)
 **     Retorna      : Posicion del proximo valor.
```

```
 **
 ↪  ------------------------------------------------------------------------
*/

uint16_t separadorHEX(char token, double *salida, uint16_t i, char*
 ↪  response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obviar espacios sin valor
 **     Parámetros  : token (delimitador de los valores), i (posicion que
 ↪  se esta leyendo),
                      response (cadena de caracteres que contiene toda la
 ↪  respuesta)
 **     Retorna     : Posicion del proximo valor.
 **
 ↪  ------------------------------------------------------------------------
*/

uint16_t separador(char token, uint16_t i, char* response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador_signo
 **     Función     : obtener un caracter del mensaje.
 **     Parámetros  : salida (Variable en la que se almacenara el signo) i
 ↪  (posicion que se esta leyendo),
                      response (cadena de caracteres que contiene toda la
 ↪  respuesta)
 **     Retorna     : Posicion del proximo valor.
 **
 ↪  ------------------------------------------------------------------------
*/

uint16_t separador_signo(char * salida, uint16_t i, char* response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : Llamar
 **     Función     : Llamar a un numero
 **     Parámetros  : Cadena de caracteres (numero)
 **     Retorna     : 1 si la llamada es exitosa, 0 si no es exitosa.
 **
 ↪  ------------------------------------------------------------------------
*/
```

```
int Llamar(char* numero);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Trancar
 **     Función      : Finalizar llamada
 **     Parámetros   : -
 **     Retorna      : -
 **
 ↪  ------------------------------------------------------------------------
*/

void Trancar();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Verificar_CSQ
 **     Función      : obtener el valor de la bandera flag_CSQ
 **     Parámetros   : -
 **     Retorna      : valor de la bandera flag_CSQ
 **
 ↪  ------------------------------------------------------------------------
*/

uint8_t verificar_CSQ();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : SendATcommand
 **     Función      : Enviar comando AT y verificar que el mensaje de
 ↪  respuesta concuerde con lo esperado
 **     Parámetros   : comandoAT, Respuesta_esperada, temporizador, cadena
 ↪  de caracteres que almacenará la respuesta
 **     Retorna      : 1 si se recibio la respuesta esperada, 0 si no.
 **
 ↪  ------------------------------------------------------------------------
*/

uint8_t sendATcommand(char* ComandoAT, char* Respuesta_esperada, unsigned
 ↪  long timeout, char* response);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Iniciar variables
 **     Función      : inicializar las variables y reservar la memoria de
 ↪  la estructura
 **     Parámetros   : -
```

```
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void iniciar_variables();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : reiniciar_RSSI_BER
 **     Función     : Reiniciar en 0 los valores de BER, RSSI y de la
 ↪  bandera flag_CSQ
 **     Parámetros  : -
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void reiniciar_RSSI_BER();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : begin_myserial
 **     Función     : Fijar tamaño del buffer de recepcion serial.
 **     Parámetros  : Enviar mensajes de configuracion tanto al GPS como
 ↪  al SIM900 para llamadas
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void begin_myserial();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : begin_myserial_Datos
 **     Función     : Fijar tamaño del buffer de recepcion serial.
 **     Parámetros  : Enviar mensajes de configuracion tanto al GPS como
 ↪  al SIM900 para datos
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void begin_myserial_Datos();
/*
```

```
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : ftp_get
 **     Función      : solicitar al servidor FTP un archivo
 **     Parámetros   : -
 **     Retorna      : -
 **
 ↪  ------------------------------------------------------------------------
*/

void ftp_get();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Send_WiFi
 **     Función      : Enviar estructura via WiFi
 **     Parámetros   : Objeto WiFiClient client
 **     Retorna      : -
 **
 ↪  ------------------------------------------------------------------------
*/

void Send_WiFi(WiFiClient client);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : DegToDec
 **     Función      : Convertir valor de posicion de grados a decimal
 **     Parámetros   : signo y posicion en grados
 **     Retorna      : -
 **
 ↪  ------------------------------------------------------------------------
*/

void DegToDec(char signo, double *posicion);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre       : Verificar_GPS
 **     Función      : Verificar que el GPS se encuentre ubicado y
 ↪  funcionando
 **     Parámetros   : -
 **     Retorna      : 1 si esta ubicado, 0 si no
 **
 ↪  ------------------------------------------------------------------------
*/
```

```
uint8_t Verificar_Gps();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : Probar_GPS
 **     Función     : Verificar que el GPS funcione correctamente y
 ↪  esperar hasta que funcione
 **     Parámetros  : -
 **     Retorna     : 1 si funciona correctamente
 **
 ↪  ------------------------------------------------------------------------
*/

int8_t Probar_GPS();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : get_data
 **     Función     : Obtener los valores relevantes del GPS y
 ↪  almacenarlos en la estructura
 **     Parámetros  : -
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void get_data();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : get_CSQ
 **     Función     : Obtener y almacenar en la estructura los valores de
 ↪  RSSI y BER
 **     Parámetros  : -
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void get_CSQ();
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obtener los datos del mensaje del GPS que se
 ↪  almacenan en una variable de 32 bits sin signo.
```

```
 **     Parámetros  : token (delimitador de los valores), salida (variable
↪  en la que se almacenara el valor),
                     my_Serial (Objeto que recibe la informacion del GPS)
 **     Retorna     : -
 **
↪  ------------------------------------------------------------------------
*/

void separador(char token, uint *salida, SoftwareSerial *my_Serial);
/*
 **
↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obtener los datos del mensaje del GPS que se
↪  almacenan en una variable de 16 bits sin signo.
 **     Parámetros  : token (delimitador de los valores), salida (variable
↪  en la que se almacenara el valor),
                     my_Serial (Objeto que recibe la informacion del GPS)
 **     Retorna     : -
 **
↪  ------------------------------------------------------------------------
*/

void separador(char token, uint16_t *salida, SoftwareSerial *my_Serial);
/*
 **
↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obtener los datos del mensaje del GPS que se
↪  almacenan en una variable de 8 bits sin signo.
 **     Parámetros  : token (delimitador de los valores), salida (variable
↪  en la que se almacenara el valor),
                     my_Serial (Objeto que recibe la informacion del GPS)
 **     Retorna     : -
 **
↪  ------------------------------------------------------------------------
*/

void separador(char token, uint8_t *salida, SoftwareSerial *my_Serial);
/*
 **
↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obtener los datos del mensaje del GPS que se
↪  almacenan en un double.
 **     Parámetros  : token (delimitador de los valores), salida (variable
↪  en la que se almacenara el valor),
```

```
                    my_Serial (Objeto que recibe la informacion del GPS)
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void separador(char token, double *salida, SoftwareSerial *my_Serial);

/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador
 **     Función     : Obviar espacios sin valor
 **     Parámetros  : token (delimitador de los valores), mySerial (Objeto
 ↪  que recibe la informacion del GPS)
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void separador(char token, SoftwareSerial *my_Serial);
/*
 **
 ↪  ------------------------------------------------------------------------
 **     Nombre      : separador_signo
 **     Función     : Almacenar el valor del signo
 **     Parámetros  : Salida (variable en la que se almacenara el valor),
                      mySerial (Objeto que recibe la informacion del GPS)
 **     Retorna     : -
 **
 ↪  ------------------------------------------------------------------------
*/

void separador_signo(char * salida, SoftwareSerial *my_Serial);
```

Archivo de funciones: AT libreria 1.cpp

```
#include "AT_libreria_1.h"

#define TIEMPO 1
#define BUFFER_RX 400

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

char response[BUFFER_RX]={};    //Buffer que almacena las respuesta del
 ↪  SIM900 en modo de ingenieria
```

```
char response_CSQ[50]={};       //Buffer que almacena la respuesta de la
 ↪  funcion get_CSQ
Info *Dtest;                    //Apuntador a estructura principal
uint8_t flag_CSQ = 0;           //Bandera que le indica al programa que
 ↪  debe tomar las medidas de calidad de la senal

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

unsigned int verificar_llamada()
{
    return Dtest->Llamadas_perdidas;
}

char* comparador(char* respuesta, char* respuesta_esperada)
{
   int i=0,k;
   int j=0,l;
   char aux3='z';
   k=size(respuesta);
   l=size(respuesta_esperada);

   if(k<l)
    return NULL;

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

```
    return respuesta_esperada;
   else
    return NULL;
}

char* comparador(char* respuesta, char* respuesta_esperada, uint16_t *w)
{
   int i=0,k;
   int j=0,l;
   char aux3='z';
   k=size(respuesta);
   l=size(respuesta_esperada);

   if(k<l)
    return NULL;

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

void Enviar_comando(char* ComandoAT)
{
        Serial.println(ComandoAT);
}

void Recibir_Serial(char* response, unsigned long timeout, uint16_t *x)
{
        unsigned long antes=millis();
```

```
  //Ciclo de espera mientras llega un valor por serial o se acaba el
↪  temporizador.
    while(Serial.available()<=0 && (antes-millis()) <= timeout)
        ESP.wdtFeed();

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

uint16_t Buscar_inicio(char* header, char* response)
{
        uint16_t x=0;
        if(comparador(response,header,&x)!=NULL)
                return x;
        else
    return -1;
}

int8_t get_CENG()
{
    uint16_t j=1;
    uint16_t i=0,aux;
    //Se envia el comando AT+CENG? para tener los parametros de ingeniería
    //Si no se recibe OK, descartar la medición
    if(sendATcommand("AT+CENG?","OK",1000,response) != 1)
            return 0;
    //Buscar el encabezado +CENG:0,"
    i = Buscar_inicio("+CENG:0,\"",response);
    //Si la posición 0 ó 1  descartar medición
    aux=i;
    if(aux==0 || aux==-1)
    {
        return -1;
    }
    //si la posicion es del tamaño del buffer, descartar medicion
    if(i==BUFFER_RX-1)
            return -1;

    //se almacenan los valores de la celda que sirve al movil en la
↪   estructura
    i=separador(',' , &(Dtest->arfcn_bcch[0]), i, response);
    i=separador(',' , &(Dtest->rxl[0]), i, response);
    i=separador(',' , &(Dtest->rqx), i, response);
    i=separador(',' , &(Dtest->mcc[0]), i, response);
```

```
    i=separador(',' , &(Dtest->mnc[0]), i, response);
    i=separador(',' , &(Dtest->bsic[0]), i, response);
    i=separadorHEX(',' , &(Dtest->cellid[0]), i, response);
    i=separador(',' , &(Dtest->rla), i, response);
    i=separador(',' , &(Dtest->txp), i, response);
    i=separadorHEX(',' , &(Dtest->lac[0]), i, response);
    i=separador('\"' , &(Dtest->TA), i, response);

    while(j<=6)
    {   //valores de las celdas vecinas
        i = separador('\"', i, response);
        i = separador(',' , &(Dtest->arfcn_bcch[j]), i, response);
        i = separador(',' , &(Dtest->rxl[j]), i, response);
        i = separador(',' , &(Dtest->bsic[j]), i, response);
        i = separadorHEX(',' , &(Dtest->cellid[j]), i, response);
        i = separador(',' , &(Dtest->mcc[j]), i, response);
        i = separador(',' , &(Dtest->mnc[j]), i, response);
        i = separadorHEX('\"' , &(Dtest->lac[j]), i, response);
        j++;
    }
    return 1;
}

                /*SEPARADORES DE INGENIERIA*/

uint16_t separador(char token, uint *salida, 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';

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

uint16_t separador(char token, uint16_t *salida, 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';
    *salida = atoi(aux);
    return i;
}

uint16_t separador(char token, uint8_t *salida, 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';

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

uint16_t separador(char token, double *salida, 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';

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

uint16_t separadorHEX(char token, double *salida, 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';

    *salida = (double)strtol(aux,NULL, 16);
    return i;
}

uint16_t separador(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 separador_signo(char * salida, uint16_t i, char* response)
{
    *salida=response[i];
    i++;
    return i;
}
```

```
int Llamar(char* numero)
{
    char comando[40]={};
    uint8_t i = 0;
    sprintf(comando,"ATD%s;",numero);
    if(sendATcommand(comando,"OK",1000,response)==1)
          {
                  i=1;
          }
    (Dtest->Llamadas_realizadas)++;
    return i;
}

void Trancar()
{
        sendATcommand("ATH","OK",1000, response);
}

uint8_t verificar_CSQ()
{
        return flag_CSQ;
}

uint8_t sendATcommand(char* ComandoAT, char* Respuesta_esperada, unsigned
 ↪  long timeout, char* response)
{
  //Se envía el comando AT
        Enviar_comando(ComandoAT);
  //Empieza el temporizador
        long unsigned antes=millis();
        uint8_t answer=0,flag=0;
        uint16_t x = 0;
  //se vacia el buffer de almacenamiento
        memset(response,'\0',BUFFER_RX);

        do
        {
                Recibir_Serial(response,1000,&x);
                //comparamos si la respuesta recibida contiene a NO
 ↪  CARRIER
    if(comparador(response,"NO CARRIER")!=NULL && flag==0)
                {
      flag=1;
                        (Dtest->Llamadas_perdidas)++;
                }
    //comparamos si la respuesta recibida contiene a MO CONNECTED
```

```
                if(comparador(response,"MO CONNECTED")!=NULL)
                {
                        flag_CSQ = 1;
                }
    //comparamos si la respuesta recibida contiene a la respuesta esperada
                if(comparador(response,Respuesta_esperada)!=NULL)
                {
                        answer=1;
                }
    ESP.wdtFeed();
        }while(answer==0 && ((millis()-antes)<=timeout));
  //se coloca delimitador \0
        if(x!=400)
                response[x]=0;
        return answer;
}

void iniciar_variables()
{
    Dtest = (Info *)malloc(sizeof(Info));
    Dtest->HMS = 0;
    Dtest->latitude = 0;
    Dtest->longitude = 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->Llamadas_perdidas = 0;
    Dtest->Llamadas_realizadas = 0;
    for (int i = 0; i < 7; i++)
    {
        Dtest->arfcn_bcch[i] = 0;
        Dtest->rxl[i] = 0;
        Dtest->bsic[i] = 0;
        Dtest->cellid[i] = 0;
        Dtest->mcc[i] = 0;
        Dtest->mnc[i] = 0;
        Dtest->lac[i] = 0;
    }
}
```

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

void begin_myserial()
{
    Serial.setRxBufferSize(BUFFER_RX);
    mySerial_GPS.begin(9600);
    delay(10);
    //comando del GPS para utilizar los mensajes $GPGGA

↪   mySerial_GPS.print(F("$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"));
    //comando del GPS para configurar su tasa de transmision a 19200
↪   baudios
    mySerial_GPS.print(F("$PMTK251,19200*22\r\n"));
    mySerial_GPS.begin(19200);
    mySerial_GPS.enableRx(false);
    delay(10);
    while(mySerial_GPS.available()>0) mySerial_GPS.read();

    //activamos el modo de ingenieria
    sendATcommand("AT+CENG=1,1","OK",1000, response);
    //activamos los mensajes de conexion de la llamada
    sendATcommand("AT+MORING=1","OK",1000, response);
}

void begin_myserial_Datos()
{
    Serial.setRxBufferSize(BUFFER_RX);
    mySerial_GPS.begin(9600);
    delay(10);
    //comando del GPS para utilizar los mensajes $GPGGA

↪   mySerial_GPS.print(F("$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"));
    //comando del GPS para configurar su tasa de transmision a 19200
↪   baudios
    mySerial_GPS.print(F("$PMTK251,19200*22\r\n"));
    mySerial_GPS.begin(19200);
    mySerial_GPS.enableRx(false);
    delay(10);
    while(mySerial_GPS.available()>0) mySerial_GPS.read();
    //Se configura las conexionex FTP
    sendATcommand("AT+CENG=1,1","OK",1000, response);
    sendATcommand("AT+SAPBR=3,1,\"Contype\",\"GPRS\"","OK", 1000,response);
```

```
    sendATcommand("AT+SAPBR=3,1,\"APN\",\"internet.movistar.ve\"","OK",
↪   1000,response);
    sendATcommand("AT+SAPBR=1,1","OK",1000,response);
    sendATcommand("AT+FTPCID=1","OK",1000,response);
    sendATcommand("AT+FTPSERV=\"speedtest.tele2.net\"","OK",1000,response);
    sendATcommand("AT+FTPUN=\"anonymous\"","OK",1000,response);
    sendATcommand("AT+FTPPW=\"1414425\"","OK",1000,response);
    sendATcommand("AT+FTPGETNAME=\"1KB.zip\"","OK",1000,response);
    sendATcommand("AT+FTPGETPATH=\"/\"","OK",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("bien"));
                        break;
                    }
*/          Serial.print((char)Serial.read());
        }
    }
}

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

void DegToDec(char signo, double *posicion)
{
    double aux,aux2;
    int aux3;
    aux3 = *posicion/100;
    aux= *posicion - aux3*100;

    aux2 = (double)aux3 + aux/60.0;

    if(signo == 'W' || signo == 'S')
```

```
        *posicion = -aux2;
    else
        *posicion = aux2;
}

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

          int x=millis();
          char aux,aux2;
          //Ciclo de espera mientras llega un valor por serial o se acaba
 ↪  el temporizador.
          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();
            //Buscamos el encabezado
            if( aux=='$' )
            {
              //saltamos los primeros 2 valores
                    separador(',',&mySerial_GPS);
                    separador(',',&mySerial_GPS);
              //Almacenamos el valor del estado del GPS en aux2
                    separador_signo(&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=='A')
            return 1;
          else
            return 0;
}
```

```
int8_t Probar_GPS()
{
    while(mySerial_GPS.available()>0) mySerial_GPS.read();
    uint8_t respuesta=0;

    respuesta = Verificar_Gps();

    while(respuesta != 1)
    {
        Serial.print(F("."));
        respuesta = Verificar_Gps();

    }
    return 1;
}

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

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

  //Ciclo de espera mientras llega un valor por serial o se acaba el
 ↪  temporizador.
  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 )//buscamos el encabezado
      {
        //Almacenar los valores del GPS en la estructura de datos
        separador(',',&mySerial_GPS);
        separador(',',&(Dtest->HMS),&mySerial_GPS);
        separador(',',&mySerial_GPS);
        separador(',',&(Dtest->latitude),&mySerial_GPS);
        separador_signo(&aux2,&mySerial_GPS);
        DegToDec(aux2,&(Dtest->latitude));
```

```
        separador(',',&(Dtest->longitude),&mySerial_GPS);
        separador_signo(&aux2,&mySerial_GPS);
        DegToDec(aux2,&(Dtest->longitude));
        separador(',',&(Dtest->speedOTG),&mySerial_GPS);
        separador(',',&mySerial_GPS);
        separador(',',&(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',50);

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

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

        /*SEPARADORES DE GPS*/

void separador(char token, uint *salida, SoftwareSerial *my_Serial)
{
    int j=0;
    char aux[25];
    char aux2;

    do
    {
        if(my_Serial->available()>0)
        {
            delay(TIEMPO);
            aux2=(char)my_Serial->read();
            aux[j]=aux2;
            j++;
        }
        else
```

```
            break;
    }
    while(aux2 != token);
    aux[j]='\0';

    *salida = atoi(aux);
}

void separador(char token, uint16_t *salida, SoftwareSerial *my_Serial)
{
    int j=0;
    char aux[25];
    char aux2;

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

void separador(char token, uint8_t *salida, SoftwareSerial *my_Serial)
{
   int j=0;
    char aux[25];
    char aux2;

    do
    {
        if(my_Serial->available()>0)
        {
            delay(TIEMPO);
            aux2=(char)my_Serial->read();
            aux[j]=aux2;
            j++;
        }
        else
```

```
            break;
    }
    while(aux2 != token);
    aux[j]='\0';
    *salida = atoi(aux);
}

void separador(char token, double *salida, SoftwareSerial *my_Serial)
{
    int j=0;
    char aux[25];
    char aux2;

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

    *salida = atof(aux);
}
void separador(char token, SoftwareSerial *my_Serial)
{
    char aux;
    do
    {
        if(my_Serial->available()>0)
        {
            delay(TIEMPO);
            aux = my_Serial->read();
        }
        else
            break;
    }
    while(aux != token);
}

void separador_signo(char * salida, SoftwareSerial *my_Serial)
```

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

Archivo principal que ejecuta el Drive Test: ceng gps.ino

```
#include "AT_libreria_1.h"

const char* ssid     = "luis-pc";   //Red a la que el D1mini se conectará
const char* password = "W6T4fAkP";  //Contraseña de la red

const char* ip_servidor = "10.42.0.1";  //dirección IP del servidor

uint16_t Port = 51002;                  //Puerto del servidor

IPAddress staticIP(10,42,0,2);          //Direccion IP estatica del equipo
IPAddress gateway(10,42,0,1);           //Gateway de la red
IPAddress subnet(255,255,255,0);        //Mascara de la red

unsigned long previous;                 //Temporizador
unsigned long antes;                    //Temporizador

WiFiClient client;                      //Clase que permite conectarse por
 ↪  WIFI con un servidor
int i;
unsigned int aux=4;

void setup() {

  ESP.wdtDisable();                     //Se desabilita el WDT

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

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

  //nos conectamos a la red
```

```
  WiFi.config(staticIP, gateway, subnet);
  WiFi.begin(ssid, password);

  //verificamos el estado de la conexion
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(F("."));
  }

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

  antes=millis();
  //tiempo de espera para encender el modulo SIM900
  while(millis()-antes<=10000)
  {
    ESP.wdtFeed();
  }

  begin_myserial();
  iniciar_variables();

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

void loop() {

  //Se realiza la conexion con el servidor
  if(!client.connect(ip_servidor, Port)){
    ESP.wdtFeed();
    Serial.println(F(""));
    delay(1000);
    return;
  }
  //Se obtienen los datos del GPS
  get_data();

  //Se verifica si el tiempo de la llamada termino ó si la llamada se cayo
  if(millis()-previous >= 120000 || (aux-verificar_llamada())!=0)
  {
    Trancar();            //se cuelga la llamada
    reiniciar_RSSI_BER(); //se reinician los valores de rssi y ber
    Llamar("*627568");    //se llama al numero de prueba
    aux=verificar_llamada();  //se le asigna a aux el valor de las llamadas
 ↪  perdidas
```

```
    previous=millis();        //se reinicia el temporizador
  }

  //se toman las 10 medidas de ingenieria, las de
  //calidad y se envia el reporte al servidor
  for(i=0;i<=9;i++)
  {
    if(get_CENG()!=1)
      continue;
    ESP.wdtFeed();
    if(verificar_CSQ()==1)
      get_CSQ();
    Send_WiFi(client);
  }
  //se termina la conexion con el servidor
  client.stop();

}
```

Archivo que ejecuta servidor de almacenamiento: tcpechoserver p.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <mysql.h> // libreria que nos permite hacer el uso de
#include <ctype.h> //las conexiones y consultas con MySQL
#include <time.h>
#include <ctype.h>

#define BACKLOG 2 /* El número de conexiones permitidas */
#define MAXDATASIZE 1000
#define RESPALDO 10

//Estructura que almacena todos los datos del Drive Test

typedef struct Info_t
{
    uint16_t arfcn_bcch[7];     // Número de canal de frecuencia absoluta
 ↪  del BCCH
    uint8_t  rxl[7];            // Nivel de potencia de la señal recibida
```

```
    uint8_t  rqx;               // Calidad de la señal recibida
    uint16_t mcc[7];            // Código de país
    uint8_t  mnc[7];            // Código de la red
    uint8_t bsic[7];            // Código de identidad de la estacion base
    double cellid[7];           // Identificador de la celda
    uint8_t  rla;               // Nivel mínimo de transmisión para acceder
↪   a la red
    uint8_t  txp;               // Máximo nivel de transmisión en el CCCH
    double    lac[7];           // Área de ubicación
    uint8_t   TA;               // Avance temporal

    uint8_t RSSI;               // Nivel de la señal
    uint8_t BER;                // Bit error rate

    unsigned int Llamadas_perdidas;     //contador de llamadas perdidas
    unsigned int Llamadas_realizadas;   //contador de llamadas realizadas

    double latitude;            //Longitud de la coordenada
    double longitude;           //Latitud de la coordenada
    double HMS;                 //hora minuto segundo
    uint DMA;                   //dia mes y año
    double speedOTG;            // speed over ground

}Info;


/*
 **
↪   ------------------------------------------------------------------------
 **     Nombre      : IsANumber
 **     Función     : Verificar si un caracter es un numero
 **     Parámetros  : cadena de caracteres
 **     Retorna     : 1 si es un numero, -1 si no
 **
↪   ------------------------------------------------------------------------
*/

int IsANumber(const char* numero)
{
    int i=0;

    while(numero[i] != 0)
    {
        if(!isdigit(numero[i]))
        {
          return -1;
        }
```

```
        i++;
    }
    return 1;
}

/*
 **
 ↪  --------------------------------------------------------------------------
 **     Nombre      : Iniciar_Socket
 **     Función     : Realiza configuraciones iniciales del socket
 **     Parámetros  : fd (Descriptor del socket), sin_size (Tamaño de la
 ↪  estructura sockaddr_in)
                      argv (Contiene el puerto de escucha)
 **     Retorna     : Estructura del servidor
 **
 ↪  --------------------------------------------------------------------------
*/

struct sockaddr_in Iniciar_Socket(int *fd, int *sin_size, const char* argv)
{
   struct sockaddr_in server;
   /* para la Información de la dirección del servidor */

   /* A continuación la llamada a socket() */
   if (((*fd)=socket(AF_INET, SOCK_STREAM, 0)) == -1 ) {
      perror("error en socket()\n");
      exit(-1);
   }

   server.sin_family = AF_INET;

   server.sin_port = htons(atoi(argv));
   /* ¿Recuerdas a htons() de la sección "Conversiones"? =) */

   server.sin_addr.s_addr = INADDR_ANY;
   /* INADDR_ANY coloca nuestra dirección IP automáticamente */

   bzero(&(server.sin_zero),8);
   /* escribimos ceros en el reto de la estructura */


   /* A continuación la llamada a bind() */
   if(bind((*fd),(struct sockaddr*)&server,
           sizeof(struct sockaddr))==-1) {
      perror("error en bind() \n");
      exit(-1);
   }
```

```
   if(listen((*fd),BACKLOG) == -1) {  /* llamada a listen() */
      perror("error en listen()\n");
      exit(-1);
   }

   *sin_size=sizeof(struct sockaddr_in);

   return server;
}

/*
 **
 ↪  --------------------------------------------------------------------------
 **     Nombre       : finish_with_error
 **     Función      : Guardar en un archivo el error generado y cerrar la
 ↪  conexion
 **     Parámetros   : Descriptor de la conexion conn
 **     Retorna      : -
 **
 ↪  --------------------------------------------------------------------------
*/

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

/*
 **
 ↪  --------------------------------------------------------------------------
 **     Nombre       : Guardar_DB
 **     Función      : Almacenar los valores del Drive Test en la base de
 ↪  datos
 **     Parámetros   : Estructura de Drive Test y nombre de la base de
 ↪  datos
 **     Retorna      : 1 si se realizo con exito, -1 si fallo
 **
 ↪  --------------------------------------------------------------------------
*/

int Guardar_DB(Info Dtest, const char* Tabla)
{

    MYSQL *conn; /* variable de conexión para MySQL */
```

```
    char *server = "localhost"; /*direccion del servidor 127.0.0.1,
↪   localhost o direccion ip */
    char *user = "root"; /*usuario para consultar la base de datos */
    char *password = "1414425"; /* contraseña para el usuario en cuestion
↪   */
    char *database = "tesis"; /*nombre de la base de datos a consultar */
    conn = mysql_init(NULL); /*inicializacion a nula la conexión */
    char query[1000];

    if(conn==NULL)
    {
        finish_with_error(conn);
        return -1;
    }

    /* conectar a la base de datos */
    if (!mysql_real_connect(conn, server, user, password, database, 0, NULL,
↪   0))
    { /* definir los parámetros de la conexión antes establecidos */
        finish_with_error(conn);
        return -1;
    }

    /* enviar consulta SQL */
    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)",Tabla,Dtest.latitude,Dtest.longitude,
        Dtest.HMS,Dtest.DMA,Dtest.speedOTG,Dtest.Llamadas_realizadas,
        Dtest.Llamadas_perdidas,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[1],Dtest.rxl[1],Dtest.mcc[1],Dtest.mnc[1],
        Dtest.bsic[1],Dtest.cellid[1],Dtest.lac[1],Dtest.arfcn_bcch[2],

↪   Dtest.rxl[2],Dtest.mcc[2],Dtest.mnc[2],Dtest.bsic[2],Dtest.cellid[2],
        Dtest.lac[2],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 -1;
    }

    /* se libera la variable res y se cierra la conexión */
    mysql_close(conn);
    return 1;
}

/*
 **
 ↪ ------------------------------------------------------------------------
 **     Nombre       : Crear_DB
 **     Función      : Crear tabla en la base de datos
 **     Parámetros   : Nombre de la tabla en la base de datos
 **     Retorna      : 1 si fue exitoso, 0 si no
 **
 ↪ ------------------------------------------------------------------------
*/

int Crear_DB(const char* Tabla)
{
    MYSQL *conn; /* variable de conexión para MySQL */
    char *server = "localhost"; /*direccion del servidor 127.0.0.1,
                                  localhost o direccion ip */
    char *user = "root"; /*usuario para consultar la base de datos */
    char *password = "1414425"; /* contraseña para el usuario en cuestion
 ↪  */
    char *database = "tesis"; /*nombre de la base de datos a consultar */
    conn = mysql_init(NULL); /*inicializacion a nula la conexión */
    char query[1000];

    if(conn==NULL)
    {
        finish_with_error(conn);
        return -1;
    }

    /* conectar a la base de datos */
    if (!mysql_real_connect(conn, server, user, password, database, 0, NULL,
 ↪  0))
    { /* definir los parámetros de la conexión antes establecidos */
        finish_with_error(conn);
        return -1;
    }
```

```
    /* enviar consulta SQL */
    snprintf(query, 1000, "create table %s (LATITUD FLOAT(9,4),LONGITUD"
        " 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_BCCH_5 INT,RXL_5 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);",
        Tabla);

    if (mysql_query(conn, query))
    {
        finish_with_error(conn);
        return -1;
    }

    /* se libera la variable res y se cierra la conexión */
    mysql_close(conn);
    return 1;
}

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

    if(argc != 3)
    {
        printf("son dos argumentos: ./Program Port DataBase\n");
        return -1;
    }
    if(IsANumber(argv[1])==-1)
    {
        printf("El segundo argumento es un numero\n");
        return -1;
    }

    int k = 1, contador = 1;
    Info Dtest;
    int fd, fd2 , j = 0 , numbytes; /* los ficheros descriptores */
```

```
struct sockaddr_in server;
/* para la Información de la dirección del servidor */

struct sockaddr_in client;
/* para la Información de la dirección del cliente */

int sin_size;

server = Iniciar_Socket(&fd, &sin_size, argv[1]);

if(Crear_DB(argv[2]) != 1)
{
    printf("Error al crear la base de datos\n");
    return -1;
}

/* A continuación la llamada a accept() */

while(1)
{
    if((fd2 = accept(fd,(struct sockaddr *)&client,
                     &sin_size))==-1) {
       printf("Con %d files\t",contador );
       perror("error en accept()\n");
       exit(-1);
    }

    if(k % RESPALDO == 0)
    {
       printf("contador va en %d\n",contador );
    }

    /* que mostrará la IP del cliente */
    //send(fd2,(void*)&msg,sizeof(Info),0);
    /* que enviará el mensaje de bienvenida al cliente */

    for (int j = 0; j <= 9; j++)
    {
        if ((numbytes=recv(fd2,(void*)&Dtest,sizeof(Info),0)) == -1){
            /* llamada a recv() */
            perror(" error Error \n");
            exit(-1);
        }

        if(Guardar_DB(Dtest,argv[2]) != 1)
        {
            printf("Error al guardar en la base de datos\n");
```

```
                return -1;
            }
        }

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

Archivo que genera la página WEB

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Prueba 20km</title>
    <style>
      /*Configuraciones del mapa*/
      #map {
        height: 100%;
      }
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
      }
      #floating-panel { /*Configuraciones del panel*/
        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 id = "info">
          <div class="portlet-body">
                  <?php
      //Se agrega el archivo PHP que lee de la base de datos
                  include('test_20km.php');
                  ?>
          </div>
        </div>

<!--Panel que contiene el buscador de celdas-->
      <div id="floating-panel">
      <form id="frm1" action="/action_page.php">
          CELL_ID: <input type="text" name="fname"><br>
      </form>
      <button onclick="myFunction()">Ver celda</button>
      <button onclick="myFunction_1()">Ver Todas</button>
  </div>

  <div id="map"></div>

  <script>
    //Variables globales que almacenan los rectangulos y el mapa
          var rectangle=[];
          var map;
      var aux;
          /*Se inicializa el mapa con la API de google maps*/
          function initMap() {
              map = new google.maps.Map(document.getElementById('map'), {
                center: new google.maps.LatLng(10.4725,-66.8340), //
↪  coordenadas en la que aparecera el mapa
                zoom: 17 //zoom con el que aparecera el mapa
              });
              var infoWindow = new google.maps.InfoWindow;
              var i=0;
                // Direccion del archivo XML en el que se encuentran los
↪  datos
              downloadUrl('./xml/test_20km.xml', function(data) {
                  var xml = data.responseXML;
            //Se busca por el Tag DT creado en el archivo PHP
                  var DT = xml.documentElement.getElementsByTagName('DT');
                  aux=0;
```

```
                    Array.prototype.forEach.call(DT, function(DTElem) {
                                        var
↪ Lat=parseFloat(DTElem.getAttribute('LATITUD')); //se guarda la latitud
                                        var
↪ Lng=parseFloat(DTElem.getAttribute('LONGITUD'));//se guarda la longitud
                    var Intensidad =
↪ parseFloat(DTElem.getAttribute('RXL_0'));//se guarda la intensidad
                    var CALL_LOST =
↪ parseInt(DTElem.getAttribute('CALL_LOST'));//se guardan las llamadas
↪ perdidas
                                    var CELL_ID =
↪ DTElem.getAttribute('CELL_ID_0');//se guarda el identificador de la
↪ celda


                            if(aux != CALL_LOST)
                            {
                        //Si existe una llamada perdida crear un marcador
↪ para identificar
                        //en que coordenada se cayo
                            var infowincontent =
↪ document.createElement('div');
                              var text = document.createElement('text');
                              text.textContent = 'Llamada perdida';
                              infowincontent.appendChild(text);
                        infowincontent.appendChild(document.createElement('br'));

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

                                      var point = new google.maps.LatLng
↪ (Lat-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++;
                            }

                    //Se crea el rectangulo con relacion a la señal medida
↪
```

```
                                rectangle[i] = new google.maps.Rectangle
                                ({
                                        strokeColor: '#FF0000',
                                        strokeOpacity: 0,
                                        strokeWeight: 0,
                                        fillColor: '#FF0000',
                                        fillOpacity: Intensidad/500,
                                        map: map,
                                        bounds: {
                                                        north: Lat,
                                                        south: Lat-0.0001,
                                                        east:  Lng,
                                                        west:  Lng-0.0001
                                                }
                                        });
                                i++;
                        });
                });
        }

        //Funcion que se encarga de reestablecer el mapa de cobertura por
↪   completo
        function myFunction_1()
        {
                var i;
                for (i = 0; i < rectangle.length; i++) {
                        rectangle[i].setMap(map);
                }
        }

        //Funcion que se encarga de buscar y mostrar solo la celda
↪   seleccionada
        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
↪ Lat=parseFloat(DTElem.getAttribute('LATITUD'));
                                var
↪ Lng=parseFloat(DTElem.getAttribute('LONGITUD'));
                                var celda =
↪ DTElem.getAttribute('CELL_ID_0');

                                    if(text != celda)
                                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=AIzaSyC6o2Hfl41F4pZSr2Ypa1eCNK6nm
    </script>
  </body>
</html>
```

Archivo que lee de la base de datos: test 20km.php

```
<?php

//Se agrega el archivo con las credenciales de la base de datos
include("database_info_20km.php");
```

```php
// Se inicia el archivo XML y se crea el nodo padre
$dom = new DOMDocument('1.0','UTF-8');
$dom->formatOutput = true;

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

// Se abre la conexión con la base de datos

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

//Verificamos que la base de datos exista
if (!$link) {
  echo "Not connected : " . mysqli_connect_error();
        exit();
}

// Selecciona todas las filas de la base de datos
$query = "SELECT * FROM $table;";

//Guarda el resultado en result
$result = mysqli_query($link , $query);

//Verificamos que el resultado no este vacio
if (!$result) {
  echo "Invalid query: " . mysqli_error($link);
        exit();
}

//Se leen todas las filas y se agregan los parametros correspondientes al
↪  archivo xml
while ($row = mysqli_fetch_assoc($result))
{
          $node = $dom->createElement('DT');
          $node = $root->appendChild($node);

        $node->setAttribute("LATITUD", $row['LATITUD']);
        $node->setAttribute("LONGITUD", $row['LONGITUD']);
        $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']);
}
//Se guarda en el archivo xml
$dom->save("./xml/test_20km.xml");
//Se cierra la conexion
mysqli_close($link);

?>
```