UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA DE LOCALIZACIÓN Y APLICACIÓN MÓVIL PARA VEHÍCULOS EN APARCAMIENTOS

AUTOR: HÉCTOR SANCHO CHILET

TUTORA: LARISA DUNAI DUNAI

Curso Académico: 2016-17

RESUMEN

El presente trabajo consiste en el desarrollo de un sistema de geolocalización de vehículos u objetos sirviéndose de un sistema de posicionamiento global mediante satélites. Se ha proyectado y construido un sistema formado por un dispositivo dotado de la capacidad de realizar la geolocalización obteniendo las coordenadas precisas de su posición y comunicarlas a una aplicación para teléfonos móviles, desarrollada en este mismo trabajo, como canal de comunicación del sistema con el usuario.

En el mercado se pueden encontrar gran cantidad de localizadores de vehículos, la gran mayoría orientados a empresas. Cada vez existen más opciones de localizadores orientados al uso particular, ya sea para localizar vehículos, objetos, personas, mascotas, etc. La gran mayoría son apoyados por una aplicación desde la cual se gestionan estos localizadores, muchos de los cuales no incorporan la geolocalización en el propio dispositivo.

Para la construcción del dispositivo se ha utilizado la tarjeta Raspberry Pi 3 como unidad de procesamiento comunicada con un módulo de posicionamiento mediante satélites (U-blox NEO-6M) y un módulo el cual hace uso del sistema global de comunicaciones móviles para la comunicación a distancia (SIM800L). La programación del dispositivo se ha realizado en lenguaje Python.

La aplicación para teléfonos móviles Android se ha desarrollado en lenguaje Java utilizando el software oficial y gratuito ofrecido por Google, Android Studio.

Los datos muestran una precisión muy buena en la localización con un error que ronda el metro en exteriores y los tres metros en interiores.

Como resultado se ha obtenido un sistema de fácil utilización para el usuario con una aplicación con interfaz sencilla y con una curva de aprendizaje corta y un dispositivo que solo se necesita de encender un interruptor para que funcione.

Palabras Clave: localizador vehículo, sistema de posicionamiento global, aplicación móvil, Android, Raspberry Pi, Java, Python, geolocalización, satélites.

RESUM

El present treball consisteix en el desenvolupament d'un sistema de geolocalització de vehicles o objectes servint-se d'un sistema de posicionament global per mitjà de satèl·lits. S'ha projectat i construït un sistema format per un dispositiu dotat de la capacitat de realitzar la geolocalització obtenint les coordenades precises de la seua posició i comunicar-les a una aplicació per a telèfons mòbils, desenrotllada en aquest mateix treball, com a canal de comunicació del sistema amb l'usuari.

En el mercat es poden trobar gran quantitat de localitzadors de vehicles, la gran majoria orientats a empreses. Cada vegada existixen més opcions de localitzadors orientats a l'ús particular, ja siga per a localitzar vehicles, objectes, persones, mascotes, etc. La gran majoria són recolzats per una aplicació des de la qual es gestionen estos localitzadors, molts dels quals no incorporen la geolocalització en el propi dispositiu.

Per a la construcció del dispositiu s'ha utilitzat la targeta Raspberry Pi 3 com a unitat de processament comunicada amb un mòdul de posicionament per mitjà de satèl·lits (U- blox NEO-6M) i un mòdul el qual fa ús del sistema global de comunicacions mòbils per a la comunicació a distància (SIM800L). La programació del dispositiu s'ha realitzat en llenguatge Python.

L'aplicació per a telèfons mòbils Android s'ha desenrotllat en llenguatge Java utilitzant el programa oficial i gratuït oferit per Google, Android Studio.

Les dades mostren una precisió molt bona en la localització amb un error que ronda el metre en exteriors i els tres metres en interiors.

Com resultat s'ha obtingut un sistema de fàcil utilització per a l'usuari amb una aplicació amb interfície senzilla i amb una corba d'aprenentatge curta i un dispositiu que només es necessita d'encendre un interruptor perquè funcione.

Paraules clau: localitzador vehicle, sistema de posicionament global, aplicació mòbil, Android, Raspberry Pi, Java, Python, geolocalització, satèl·lits.

ABSTRACT

This project explains the development of a vehicles and objects geolocation system via global positioning system using satellites. The system has been designed and built during this project and it consists on a device equipped with the ability to perform geolocation to obtain the precise coordinates and send them to the created smartphone app, which also has been developed in this project and serves as interface between the system and the user.

Different types of vehicle locators can be found in the current market but most of them are business-oriented. Even though more locators are appearing in the market and are aimed for particular users in order to locate vehicles, objects, people, pets etc, most of them do not have the geolocation performance. Usually, they have an app to manage their operation.

For the device construction a Raspberry Pi 3 has been used as a processing core, communicated with a satellites positioning module (U-blox NEO-6M) and with another module (SIM800L) which uses the global system of mobile communications to send a long range message. The code has been written in Python.

The Android app has been developed using Java in the Google official software, Android Studio.

The data shows a really good geolocation accuracy with an approximate outdoor error of one meter and three meters indoor error.

As a result is has been obtained a friendly-user system with a simple interface application and a device that just need to be powered on to start using it.

Keywords: vehicle locator, Global Positioning System, smartphone application, Android, Raspberry Pi, Java, Python, geolocation, satellites.

ÍNDICE

FIGURAS

TABLAS

CAPÍTULO 1. INTRODUCCIÓN

Los dispositivos de localización basados en el sistema de posicionamiento global, más conocido como GPS (*Global Positioning System*), son imprescindibles en muchas aplicaciones hoy en día. La gran mayoría de personas lo utiliza todos los días sin ni siquiera saberlo a través de su teléfono móvil o su vehículo. Esta tecnología tiene infinidad de aplicaciones en nuestro día a día cotidiano a pesar de ser de origen militar.

Este sistema de geolocalización surgió durante la guerra fría, en plena carrera de armamento la Unión Soviética lanzó al espacio el primer satélite artificial, el Sputnik I, en octubre de 1957. A raíz de este lanzamiento Estados Unidos se planteó que al igual que se podía obtener la localización del satélite Sputnik I desde la Tierra, también se podría realizar la operación contraria, obtener la localización del receptor en la Tierra usando los satélites. Esta tecnología sería realmente útil para la defensa ante la amenaza de los misiles nucleares intercontinentales que empezaban a surgir en la época.

Debido a esto, tras investigaciones independientes de la Marina y las Fuerzas Aereas, en 1968 el Departamento de Defensa creó el comité NAVSEG (Navigation Satellite Executive Committee) encargado de la investigación de posicionamiento mediante satélites. La investigación de este organismo dio como resultado el NAVSTAR-GPS (Navigation System Timing and Ranging-Global Positioning System), un sistema de posicionamiento mediante satélites formado por un total de 24 satélites en 6 órbitas a 20.200 km de altura. El 14 de julio de 1974 fue lanzado el primer satélite de la serie GPS. El funcionamiento detallado del sistema se verá en el capítulo 2 - Sistema GPS.

El sistema GPS no estuvo disponible para uso civil hasta 1983. Durante la guerra fría la Unión Soviética derribó un avión civil de Corea del Sur pensando que se trataba de un avión espía de EEUU. Esto sucedió porque el avión se había desviado de su trayectoria por un error de posicionamiento e invadió su espacio aéreo. Murieron 269 pasajeros entre los que se encontraba un congresista estadounidense, a causa de este incidente el presidente Ronald Reagan sugirió habilitar el sistema GPS para usos en aviación civil, lo cual ocurrió en 1993 (Ortega & Moya, 2013).

Actualmente el sistema GPS es propiedad de los EEUU y aunque permiten su uso gratuito a todo el mundo, todo su funcionamiento sigue estando bajo el control del Departamento de Defensa de EEUU.

Para poder usar este sistema de posicionamiento no es necesario ningún permiso especial, tan solo es necesaria una antena capaz de comunicarse con los satélites y un módulo de procesamiento para interpretar esta información, medir la distancia a los satélites y obtener las coordenadas de su posición como se explica en el apartado 2.2. Obtención de coordenadas.

1.1. OBJETIVO

El objetivo de este proyecto es la construcción y programación de un sistema de localización de vehículos mediante GPS basado en un dispositivo, al que se ha nombrado UBIC, y una aplicación Android, a la que se ha llamado UBICCAPP.

Este sistema permite al usuario saber dónde está estacionado el vehículo en el cual se encuentre el dispositivo UBIC a través de un teléfono móvil, usando la aplicación UBICAPP será capaz de ver en el mapamundi la localización exacta y recibir las indicaciones para llegar hasta este.

Principalmente está dirigido a la localización de vehículos como coches, motos y camiones, pero puede ser perfectamente útil para no perder objetos como mochilas o maletas u otros vehículos no motorizados como bicicletas.

Puede resultar muy útil como servicio opcional que pueden ofrecer, por ejemplo, empresas de alquiler de vehículos. El mercado objetivo de estas empresas suelen ser turistas que se encuentran en localizaciones desconocidas para ellos, el sistema de localización de vehículos puede ser un gran complemento al alquiler de un vehículo.

En el proyecto se pueden diferenciar 2 partes:

1. El dispositivo GPS UBIC: compuesto por una unidad de procesamiento, un módulo GPS, un módulo GSM y una batería de litio. Su función es recibir la señal de los satélites y obtener las coordenadas geográficas mediante trilateración usando el módulo GPS y la antena cerámica (véase 2.2. Obtención de coordenadas). La unidad de procesamiento lee los datos, procesa, ordena (véase 3.2. Programa Python) y comparte mediante Wifi creando un punto de acceso con el fin de comunicarse con el usuario a través de la aplicación para teléfonos móviles inteligentes.
2. La aplicación para Android UBICAPP: esta aplicación se desarrolla con Android Studio 2.3 en lenguaje de programación Java. La aplicación se conecta mediante Wifi al punto de acceso creado por el dispositivo UBIC para guardar en el teléfono móvil la última posición conocida del dispositivo y poder realizar la navegación hasta este punto. En el caso de querer comprobar la posición estando a larga distancia o no encontrar el dispositivo donde marcaban la posición, la aplicación incluye la opción de obtener las coordenadas actualizadas a través de un mensaje de texto.

La figura 1.1 es un esquema simplificado de la comunicación mediante Wifi del sistema, desde la obtención de las coordenadas haciendo uso del sistema GPS hasta que esta información llega al usuario. Se espera que el uso del dispositivo sea posible tanto en exteriores como en interiores de edificios aunque se descarta el posicionamiento y comunicación en sitios subterráneos.

Figura 1.1 – Flujo de información mediante Wifi en el sistema UBIC/UBICAPP.

Como puntos a favor respecto a otros localizadores con GPS como los sistemas de control de flotas, UBIC es completamente independiente del vehículo, es decir, no necesita instalación pudiendo así utilizarlo en cualquier vehículo u objeto. Dispone de una batería de Litio que puede ser recargada en el propio vehículo y lo dota de una buena autonomía.

Es un dispositivo de dimensiones reducidas con el fin de maximizar su portabilidad y que no sea un impedimento su utilización en aquellos vehículos que no dispongan de mucho lugar de almacenamiento, como puedan ser las motocicletas.

Todo esto permite al usuario no tener que preocuparse a la hora de estacionar su vehículo en lugares desconocidos, grandes ciudades, centros comerciales o durante un viaje.

1.2. ESTADO DE ARTE

La tecnología en cuanto a localizadores GPS para vehículos que se utiliza hoy en día está en un punto muy avanzado, con estos localizadores se puede llegar a alcanzar una gran precisión.

La vertiente más avanzada de esta tecnología se encuentra en sectores como aviación y militar.

Por ejemplo, en aviación se utilizan localizadores mucho más sofisticados y caros con relojes más precisos y antenas más potentes que los que podemos encontrar en dispositivos y aplicaciones más comunes. Con esta tecnología tanto los controladores aéreos como los pilotos pueden determinar la posición casi exacta de cualquier avión en todo instante, incluyendo la altitud.

Durante el vuelo, unos pocos metros de error no suponen un problema si se adoptan márgenes de seguridad en el diseño de las rutas, estos márgenes prácticamente desaparecen durante los momentos críticos del vuelo, el despegue y el aterrizaje. Para estos momentos críticos los localizadores instalados en los aviones utilizan el GPS diferencial, este tipo de localización se utiliza en aplicaciones donde se necesita una precisión muy elevada, llegando a conseguir la posición con un error de unos 2 centímetros en posicionamientos dinámicos o incluso milímetros en posicionamientos estáticos.

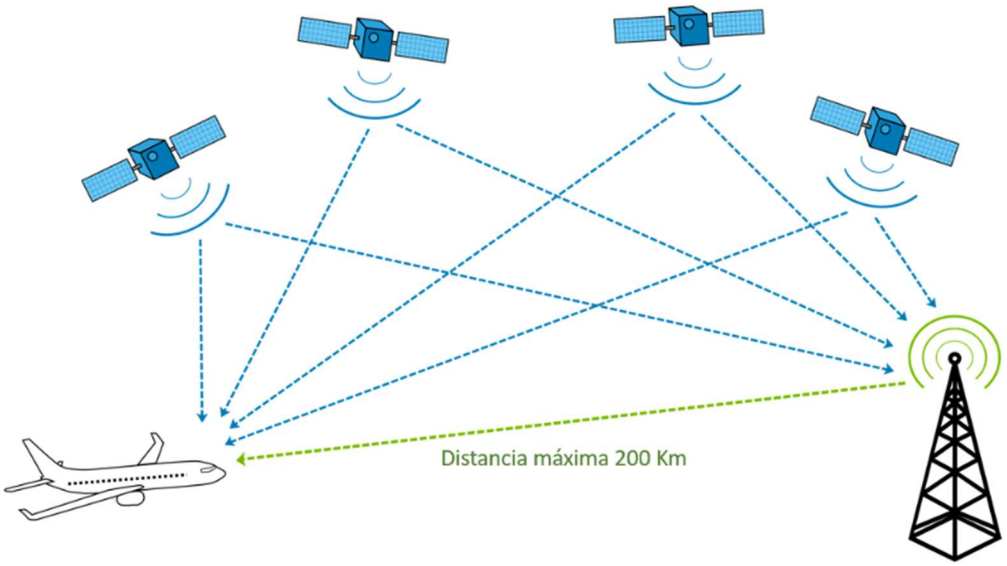


Figura 1.2- Principio de funcionamiento del GPS diferencial.

Para conseguir esta precisión los localizadores se hacen servir de una base en tierra con una posición conocida, cancelando de esta manera el error que pudiera haber en la medida de la posición (véase 2.4.Fuentes de error del GPS). Esto es posible dentro del rango de señal de la base el cual es de hasta un máximo de 200 km aproximadamente (véase figura 1.2) (Trimble Navigation Ltd., s.f.).

Otros dispositivos de localización menos sofisticados son los utilizados en sistemas de seguimiento, control y gestión de flotas. Estos dispositivos se suelen instalar de forma permanente en el vehículo y cuentan con conexión a internet para enviar los datos a un servidor y poder ser consultados desde la empresa para controlar y llevar a cabo el seguimiento de los vehículos.

En la mayoría de los casos utilizan la tecnología GSM (del inglés Global System for Mobile communications) para transmitir la información a larga distancia al tener acceso a internet. El dispositivo instalado en el vehículo habitualmente puede acceder a datos del ordenador de abordo como velocidad, temperatura, revoluciones, etc., también dispone de un receptor GPS. Todo esto suele utilizar la energía proporcionada por el vehículo.

El funcionamiento del sistema es el siguiente: en primer lugar realiza la localización GPS, lee los datos relevantes del vehículo y los envía a un servidor siempre y cuando haya cobertura disponible. El servidor recibe y organiza los datos. Por otra parte el sistema dispone de una aplicación ya sea para móvil, ordenador o web desde donde se puede administrar toda la información recibida en el servidor (véase figura 1.3).

Estos dispositivos están ampliamente extendidos en las empresas de transportes, las cuales pueden obtener datos a tiempo real sobre la posición y velocidad de cada uno de los componentes de su flota y controlar si su vehículo o mercancía se sale de una ruta programada e incluso recibir alertas por fallos, accidentes o cualquier tipo de contratiempo de cualquiera de sus vehículos.

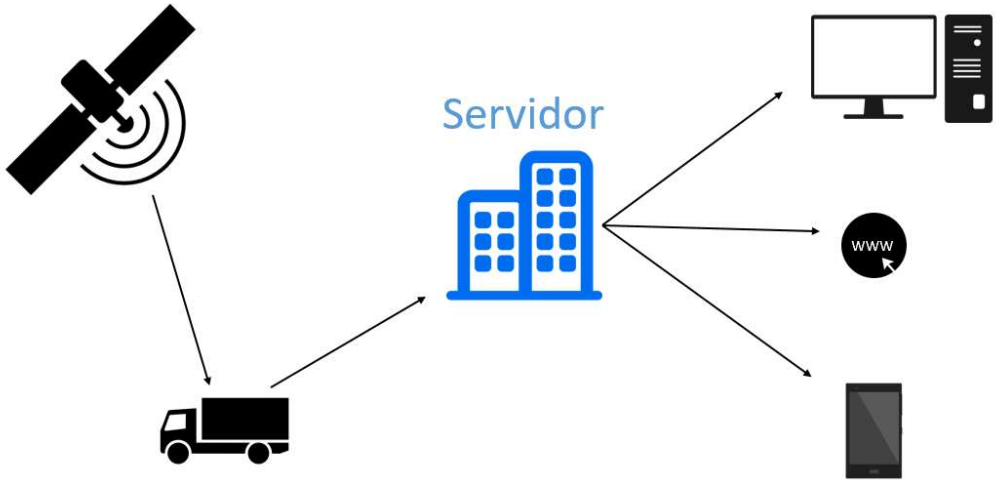


Figura 1.3- Esquema de funcionamiento de un sistema de seguimiento de flotas.

Los principales objetivos de estos sistemas de seguimiento son:

- Mejora de la productividad de la flota
- Aumento del rendimiento de los conductores
- Reducción de tiempos muertos
- Reducción del consumo de combustible
- Mayor seguridad sobre su activo principal

Los tipos de localizadores vistos hasta este punto corresponden a un ámbito más profesional y no a aplicaciones dirigidas a usuarios particulares.

En el mercado también encontramos los localizadores más enfocados a usos más cotidianos, pensados para no perder objetos como llaves, carteras, mochilas y que también pueden ser utilizados para guardarla posición de vehículos estacionados de una forma no muy precisa ya que en la gran mayoría de casos necesitan apoyarse por completo en la aplicación móvil para su funcionamiento ya que no disponen de un receptor GPS.

Estos sistemas son pequeños y cuidados estéticamente, pensados para tener una gran portabilidad y poder guardarlos en cualquier objeto que queramos tener localizado. El hecho de que no dispongan de antena GPS propia hace posible conseguir una autonomía energética muy elevada pudiendo durar hasta 1 año sin cambiar la batería, esto es gracias a su simplicidad. El dispositivo no realiza tareas de procesamiento, es simplemente un controlador que almacena información y se conecta a la aplicación móvil mediante Bluetooth de bajo consumo energético.

Su principal función es la de encontrar objetos cercanos. La gran mayoría disponen de una alarma sonora y vibración que permiten localizar el objeto cuando el usuario lo selecciona en su móvil. Esto es posible siempre que el dispositivo este dentro del rango de conexión Bluetooth que suele rondar los 30 metros, algunos llegan hasta 50 metros.

También permiten la localización GPS aunque como se ha descrito anteriormente los dispositivos no disponen de antena GPS y no tienen capacidad de obtener sus propias coordenadas. Todo el trabajo recae en la aplicación móvil la cual detecta cuando se está alejando del localizador y guarda las coordenadas usando el propio sistema de localización del teléfono móvil. Debido a que el rango de conexión del que se ha hablado anteriormente ronda los 30 metros el posicionamiento es bastante impreciso, pudiendo cometer errores de localización muy grandes ya que las coordenadas marcadas indican la posición en la que se encontraba el teléfono antes de desconectarse del localizador y no la posición exacta de este.

Una característica muy interesante de estos localizadores es su funcionamiento para conseguir saber la posición de objetos o vehículos que están fuera del rango de conexión. Para esto las empresas que los ofrecen se basan en conseguir una red de usuarios bastante grande de forma se pueda conseguir que un gran número de personas tengan instalada la aplicación correspondiente en sus teléfonos móviles.

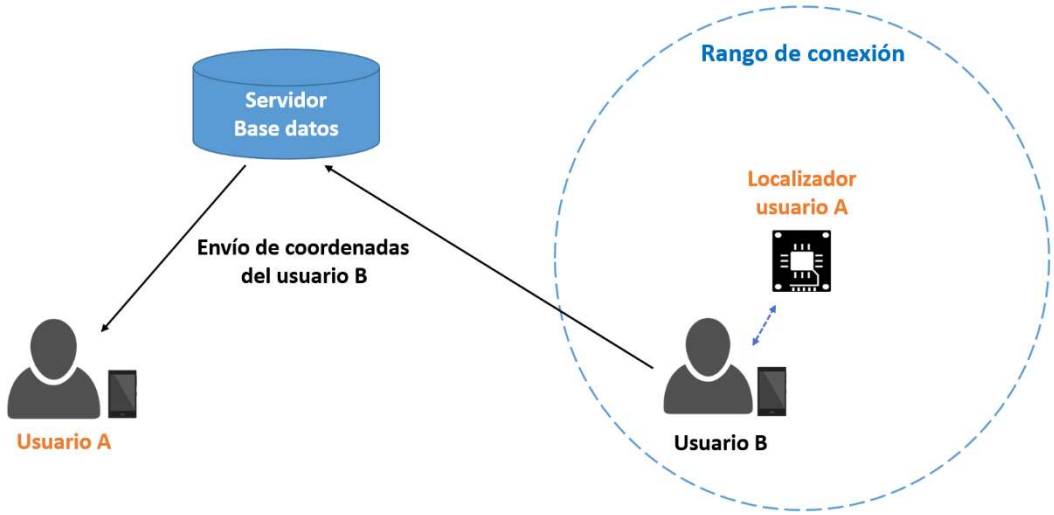


Figura 1.4 – Obtención de coordenadas por el usuario A fuera del rango de conexión.

Cada usuario tiene sus localizadores asignados con su identificación en una base de datos, cuando el usuario no está utilizando la aplicación esta busca conexiones Bluetooth con localizadores periódicamente en segundo plano. Suponiendo dos usuarios A y B, cuando la aplicación del usuario B encuentra un localizador del usuario A envía la localización a un centro de control, servidor o base de datos, y desde este punto se informa mediante un mensaje privado al usuario (A) al que estaba asignado el localizador para actualizar su posición (véase figura 1.4). Suponiendo que se tiene una red de usuarios de estos localizadores lo bastante grande, en una ciudad siempre se podrá saber dónde se encuentra el vehículo u objeto.

El precio de estos localizadores personales suele ser bastante más bajo que los anteriormente nombrados, su precio se encuentra rondando los 30 euros por localizador.

Se incluyen estos dispositivos porque aunque técnicamente no son un localizador GPS como tal sí que hacen esta función cuando están conectados a la aplicación móvil.

CAPÍTULO 2. SISTEMA GPS

2.1. SEGMENTOS DEL SISTEMA GPS

2.1.1. Segmento espacial

Este segmento abarca toda la constelación de satélites que se encuentran en órbita. La constelación principal del GPS está formada por 24 satélites distribuidos en 6 planos orbitales con 4 satélites cada uno volando aproximadamente a 20.200 Km de altura. Los planos orbitales están distribuidas uniformemente alrededor del ecuador y forman un ángulo de 55º con este. Aunque habitualmente las fuerzas aéreas tienen más satélites volando para mejorar la cobertura GPS en algunas zonas, estos satélites no cuentan como parte de la constelación principal.

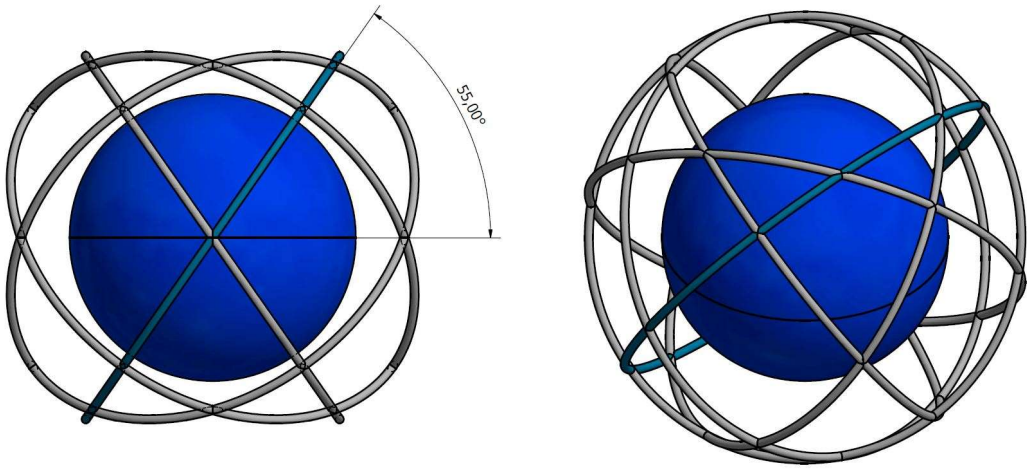


Figura 2.1 – Representación órbitas GPS posicionadas respecto al ecuador.

Siguiendo la ley de movimiento orbital de Kepler, toda órbita de un objeto alrededor de un cuerpo adquiere forma de elipse, donde uno de los focos de la elipse es dicho objeto.

Se puede ubicar con precisión cada satélite en cada instante sabiendo los parámetros keplerianos que definen su órbita: excentricidad y movimiento medio.

La **excentricidad** se puede calcular como el cociente entre la semidistancia entre los focos de la elipse (c) y el semieje mayor (a):

$$\varepsilon = \frac{c}{a} = \sqrt{1 - \left(\frac{b}{a}\right)^2} \quad (1)$$

Donde (b) es el semieje menor.

En el caso de las órbitas de los satélites GPS la excentricidad es muy pequeña ($\varepsilon = 0{,}02$), por lo que son prácticamente circulares. Si la excentricidad fuera cero, significaría que la distancia entre los focos es nula y por lo tanto solo existe un centro siendo la órbita completamente circular.

Dentro de la órbita se pueden encontrar dos puntos a destacar, el apogeo es donde el satélite se encuentra en el punto más alejado de la Tierra y en el que más superficie abarca y el perigeo representa el punto de la órbita más cercano a la Tierra y donde se recibe la señal del satélite más clara y en un tiempo más corto (Molino).

El **movimiento medio** muestra el número de vueltas que da un satélite a su órbita en 24 horas. El valor de este parámetro es directamente la inversa del periodo, pero se suele utilizar el movimiento medio por facilidad a la hora de realizar cálculos o comparativas ya que el periodo, al venir expresado en horas, minutos y segundos, es necesario utilizar conversiones para los cálculos.

Se puede calcular la posición de un satélite en cualquier momento con el movimiento medio y la hora a la que el satélite ha pasado por el perigeo, multiplicando el tiempo transcurrido desde que pasó por este punto y el movimiento medio se obtiene la posición que ocupará en la órbita en ese instante. El movimiento medio de los satélites GPS es 2,07 aproximadamente (Molino).

Este diseño de las orbitas tiene como consecuencia que en cualquier superficie del planeta siempre hayan al menos 4 satélites visibles para realizar el posicionamiento. Además, los satélites en sus respectivas orbitas giran en el mismo sentido que la rotación de la tierra, a causa de esto, la velocidad de 4 Km/s de los satélites respecto al centro de la tierra se reduce a aproximadamente 1 Km/s respecto a los usuarios en la superficie terrestre reduciendo problemas de comunicación (Blewitt, 1997).

2.1.2. Segmento de control

En este segmento se encuentra todo aquello relacionado con mantener el sistema GPS actualizado y operativo. Debido a las órbitas que hemos visto anteriormente, no todos los satélites son visibles desde un punto de la superficie terrestre, es por esto por lo que el segmento control está formado por instalaciones ubicadas alrededor del planeta.

El segmento de control está formado por:

Una **estación de control maestra** situada en Colorado que realiza las funciones de control de la constelación de satélites. Esta estación genera y actualiza los mensajes sobre la navegación de los satélites para mantener la integridad y precisión de estos. Se puede entender como el cerebro del sistema GPS. Es aquí donde se reciben todas las señales de los centros de monitorización y se procesan para actualizar constantemente las posiciones de los satélites y si

es necesario, corregir cualquier posible error y transmitir estas correcciones a los satélites. En caso de algún tipo de fallo en la constelación la estación de control maestra puede reposicionar todos los satélites para mantener el correcto funcionamiento del sistema.

Una **estación de control alternativa** para apoyar a la principal o sustituirla si fuera necesario, garantizando así el funcionamiento del GPS.

Las **estaciones de monitoreo** se encuentran alrededor del mundo (véase figura 2.2) estratégicamente situadas para tener visibilidad de todos los satélites. Estas estaciones recogen señales de los satélites y datos de la atmósfera y las transmiten a la estación de control maestra. Existen 16 estaciones de monitoreo de las cuales 6 pertenecen a las Fuerzas Aéreas de los Estados Unidos y las 10 restantes a la Agencia Nacional de Inteligencia Geoespacial o NGA (National Geospatial-Intelligence Agency).

Por último, las **antenas de tierra** son las que hacen posible la comunicación de las anteriores estaciones con los satélites GPS. Cuatro de estas antenas están situadas en las estaciones de monitoreo de Cabo cañaveral, isla de Ascensión, Diego García y Kwajalein. Otras siete se controlan de manera remota y conectadas a AFSCN (Air Force Satellite Control Network) permiten incrementar la visibilidad con los satélites consiguiendo un sistema de comunicación directa con la constelación mucho más robusto.

Figura 2.2 - Distribución geográfica del segmento de control GPS. Imagen modificada de Co per ni co (Template: Mapamundi), vía Wikimedia Commons.

La ubicación de estas instalaciones las podemos ver en el siguiente mapa:

Debido a que el periodo de la órbita es de unas 12 horas, cada satélite es visible por la antena correspondiente 2 veces por día. Es en estos instantes en los que se realiza la conexión con los

satélites, se actualiza su posición y se envían los datos de la estación de control maestra hacia la constelación.

En un futuro próximo se prevé un sistema de control de nueva generación para unificar todas las áreas de control del GPS en un solo centro con el lanzamiento de los nuevos satélites GPS III, aumentando en gran medida el rendimiento y la seguridad (The National Coordination Office for Space-Based Positioning, 2017).

2.1.3. Segmento de usuario

En este segmento se encuentran los equipos que reciben la señal de los satélites y la utilizan para obtener la posición tridimensional y la hora del usuario.

En este segmento entran todos los receptores, desde móviles o navegadores de coche hasta receptores de aviación o militar, aunque utilicen diferentes frecuencias de comunicación con los satélites.

2.2. OBTENCIÓN DE COORDENADAS

El cálculo de la posición se lleva a cabo midiendo la distancia del receptor con cada uno de los satélites visibles, con estas distancias se generan esferas donde el centro de cada una es el respectivo satélite y el radio es la distancia hasta este. El punto de intersección de las esferas es donde se encuentra el receptor.

Para la obtención de las coordenadas son necesarios 4 satélites para realizar la trilateración con una buena precisión. Como su propio nombre indica, la trilateración se puede realizar con tan solo 3 satélites visibles pero sería necesario una gran precisión en los relojes de los emisores y receptores, por parte de los satélites esto no es un problema ya que cuentan con un reloj atómico de gran precisión y estabilidad, no ocurre lo mismo con los receptores ya que de equiparlos con relojes atómicos su precio sería desorbitado. Es por esto que añadiendo una cuarta esfera/ecuación al sistema se reduce este error prácticamente en su totalidad y se minimiza el error causado por las variaciones que puedan tener los relojes en el tiempo.

Como Geoffrey Blewit explica en su libro “Geodetic Applications of GPS”, para medir la distancia a cada satélite el receptor genera una réplica de la señal transmitida por el satélite en el mismo instante que este y al recibir la señal proveniente del espacio las compara y mide el desfase entre ambas, este desfase es el tiempo que la señal ha estado viajando más los posibles errores de reloj del satélite y el receptor (véase Figura 2.3). Realmente con este tiempo no se miden distancias, sino pseudodistáncias debido a los errores de los relojes.

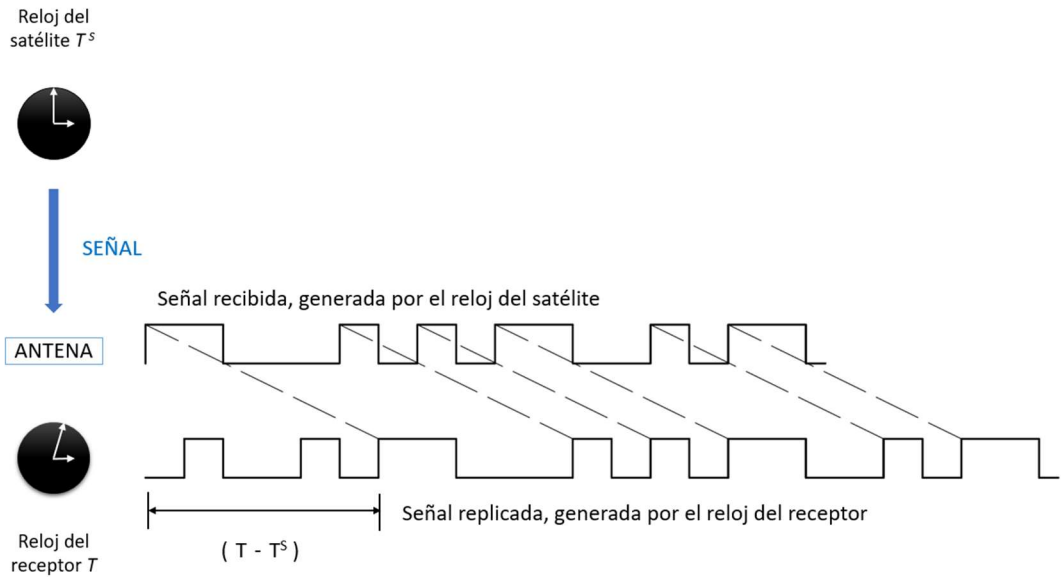


Figura 2.3 – Diagrama esquemático de la medida del tiempo que tarda la señal en viajar del satélite al receptor.

$$R = (T - T^S) \cdot c = \Delta T_{medido} \cdot c \tag{2}$$

Donde T es la lectura de la hora del receptor al recibir la señal, T^S es la lectura de hora del satélite cuando la señal fue enviada y *c* (velocidad de la luz en el vacío) = 299792458 m/s

Se puede obtener la distancia real sustituyendo el tiempo medido *T* por el tiempo real *t* más un error τ debido al desfase de ambos relojes:

$$T = t + \tau \tag{3}$$

$$T^s = t^s + \tau^s \tag{4}$$

Y sustituyendo en la ecuación (1) se obtiene la ecuación para el cálculo de la distancia real:

$$R = \left((t + \tau) - (t^s + \tau^s)\right) \cdot c = \Delta T \cdot c \tag{5}$$

Agrupando términos:

$$R = (t - t^s) \cdot c + \tau \cdot c + \tau^s \cdot c = P + \tau \cdot c + \tau^s \cdot c \tag{6}$$

Donde *R'* es la pseudodistancia definida en la ecuación (1) entre el receptor y el satélite. Aplicando el teorema de Pitágoras:

$$P(t, t^s) = \sqrt{\left(x^s(t^s) - x(t)\right)^2 + \left(y^s(t^s) - y(t)\right)^2 + (z^s(t^s) - z(t))^2} \tag{7}$$

El mensaje de navegación recibido permite obtener la posición del satélite (x^S, y^S, z^S) y su desfase de reloj (τ^S), quedando de esta manera 4 incógnitas en la ecuación (5), la posición del receptor (x, y, z) y el desfase de reloj del mismo (τ).

Es importante que las coordenadas del satélite sean computadas con el tiempo del satélite t^S ya que en el intervalo de tiempo transcurrido entre la transmisión y recepción de la señal, unos 0,07 segundos, la posición del satélite puede haber variado hasta 60 metros.

Sustituyendo R en la ecuación (5) y suponiendo 4 satélites visibles se obtiene el siguiente sistema de 4 ecuaciones y 4 incógnitas:

$$R_1 = \sqrt{\left(x_1{}^s(t_1{}^s) - x(t)\right)^2 + \left(y_1{}^s(t_1{}^s) - y(t)\right)^2 + (z_1{}^s(t_1{}^s) - z(t))^2} + \tau \cdot c + \tau_1{}^s \cdot c \tag{8}$$

$$R_2 = \sqrt{\left(x_2{}^s(t_2{}^s) - x(t)\right)^2 + \left(y_2{}^s(t_2{}^s) - y(t)\right)^2 + (z_2{}^s(t_2{}^s) - z(t))^2} + \tau \cdot c + \tau_2{}^s \cdot c \tag{9}$$

$$R_3 = \sqrt{\left(x_3{}^s(t_3{}^s) - x(t)\right)^2 + \left(y_3{}^s(t_3{}^s) - y(t)\right)^2 + (z_3{}^s(t_3{}^s) - z(t))^2} + \tau \cdot c + \tau_3{}^s \cdot c \tag{10}$$

$$R_4 = \sqrt{\left(x_4{}^s(t_4{}^s) - x(t)\right)^2 + \left(y_4{}^s(t_4{}^s) - y(t)\right)^2 + (z_4{}^s(t_4{}^s) - z(t))^2} + \tau \cdot c + \tau_4{}^s \cdot c \tag{11}$$

Para la resolución de este sistema no es necesario la linealización de las ecuaciones anteriores, aun así, para generalizar la solución si el número de satélites visibles es mayor o igual a 5, se realizará la linealización del sistema y la resolución por mínimos cuadrados.

En primer lugar se expresan las anteriores ecuaciones como:

$$R_{medido} = R_{teórico} + error = R(x, y, z, \tau) + v \tag{12}$$

A continuación se desarrolla en serie de Taylor respecto a un punto provisional (X_0, Y_0, Z_0) de una posición aproximada ignorando los términos mayores de segundo grado:

$$\begin{aligned} R(x, y, z, \tau) \cong R(x_0, y_0, z_0, \tau_0) + (x - x_0)\frac{\partial R}{\partial x} + (y - y_0)\frac{\partial R}{\partial y} + (z - z_0)\frac{\partial R}{\partial z} \\ + (\tau - \tau_0)\frac{\partial R}{\partial \tau} = R_{Calculado} + \frac{\partial R}{\partial x}\Delta x + \frac{\partial R}{\partial y}\Delta y + \frac{\partial R}{\partial z}\Delta z + \frac{\partial R}{\partial \tau}\Delta\tau \end{aligned} \tag{13}$$

La diferencia entre distancia medida y calculada queda de la siguiente forma:

$$\Delta R = R_{observado} - R_{calculado} = \frac{\partial R}{\partial x}\Delta x + \frac{\partial R}{\partial y}\Delta y + \frac{\partial R}{\partial z}\Delta z + \frac{\partial R}{\partial \tau}\Delta\tau \tag{14}$$

Representado en forma matricial:

$$\Delta R = \begin{pmatrix} \frac{\partial R}{\partial x} & \frac{\partial R}{\partial y} & \frac{\partial R}{\partial z} & \frac{\partial R}{\partial \tau} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \tau \end{pmatrix} + v \tag{15}$$

Se obtiene la ecuación (15) para cada satélite visible por el localizador. Suponiendo *n* satélites visibles:

$$\begin{pmatrix} \Delta R_1 \\ \Delta R_2 \\ \Delta R_3 \\ \vdots \\ \Delta R_n \end{pmatrix} = \begin{pmatrix} \frac{\Delta R_1}{\partial x} & \frac{\Delta R_1}{\partial y} & \frac{\Delta R_1}{\partial z} & \frac{\Delta R_1}{\partial \tau} \\ \frac{\Delta R_2}{\partial x} & \frac{\Delta R_2}{\partial y} & \frac{\Delta R_2}{\partial z} & \frac{\Delta R_2}{\partial \tau} \\ \frac{\Delta R_3}{\partial x} & \frac{\Delta R_3}{\partial y} & \frac{\Delta R_3}{\partial z} & \frac{\Delta R_3}{\partial \tau} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\Delta R_n}{\partial x} & \frac{\Delta R_n}{\partial y} & \frac{\Delta R_n}{\partial z} & \frac{\Delta R_n}{\partial \tau} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \tau \end{pmatrix} + \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \\ \vdots \end{pmatrix} \tag{16}$$

En símbolos:

$$b = Ax + v \tag{17}$$

En la cual se puede ver claramente la relación lineal entre la corrección de los parámetros *x* y las distancias *b*.

Realizando las derivadas parciales de la matriz *A* respecto a las ecuaciones de posición (8) obtenemos una matriz que muestra las funciones de la dirección a cada satélite desde la posición del receptor.

$$A = \begin{pmatrix} \frac{x_0 - x_1}{P} & \frac{y_0 - y_1}{P} & \frac{z_0 - z_1}{P} & c \\ \frac{x_0 - x_2}{P} & \frac{y_0 - y_2}{P} & \frac{z_0 - z_2}{P} & c \\ \frac{x_0 - x_3}{P} & \frac{y_0 - y_3}{P} & \frac{z_0 - z_3}{P} & c \\ \vdots & \vdots & \vdots & \vdots \\ \frac{x_0 - x_n}{P} & \frac{y_0 - y_n}{P} & \frac{z_0 - z_n}{P} & c \end{pmatrix} \tag{18}$$

Partiendo del modelo linealizado (17) se pueden aislar los errores:

$$v = b - Ax \tag{19}$$

Se minimiza la suma de los cuadrados de los errores (20) hasta que una variación de *x* no provoque apenas variación de *J(x).*

$$J(x) = \sum_{i=0}^{n} V_i^2 = v^T v = (b - Ax)^T (b - Ax) \tag{20}$$

Igualando la derivada de la función a cero obtenemos el punto mínimo:

$$
\begin{aligned}
\partial J(x) &= 0 \\
\partial\left\{\left(b-Ax\right)^T\left(b-Ax\right)\right\} &= 0 \\
\partial(\mathrm{b}-\mathrm{Ax})^T(b-Ax)+(b-Ax)^T\partial(b-Ax) &= 0 \\
(-A\partial x)^T(b-Ax)+(b-Ax)^T(-A\partial x) &= 0 \\
(-2\,\mathrm{A}\,\partial\,\mathrm{x})^T(b-Ax) &= 0 \\
(\partial x^T A^T)(\mathrm{b}-\mathrm{Ax}) &= 0 \\
\partial\,\mathrm{x}^T(A^Tb-A^TAx) &= 0
\end{aligned}
\tag{21}
$$

$$A^TAx = A^Tb \tag{22}$$

Quedando la solución al sistema de ecuaciones (22):

$$x = \left(A^TA\right)^{-1}A^Tb \tag{23}$$

En esta solución se asume que existe la inversa de *(A^TA),* para que esto sea cierto, que el número de satélites visibles sea n ≥ 4 no es una condición suficiente. Sí lo es cuando n ≥ 5 o fijando uno de los parámetros, por ejemplo fijando la altura a cero en el caso de embarcaciones (Blewitt, 1997).

2.3. FUENTES DE ERROR DEL GPS

Lo que se busca con el sistema GPS es conocer la posición tridimensional de un punto de la forma más precisa posible, en este proceso de localización influyen muchas variables y cada pequeño error en una de ellas conlleva una disminución de la precisión a la hora de la localización. Algunas de estas variables se pueden conocer con anterioridad y prevenir sus efectos mejorando de esta forma la precisión del sistema, de no ser así, el error de posición sería tan grande que haría del sistema GPS inútil para la gran mayoría de usos.

Hasta el año 2000, el sistema incorporaba un error intencionado introducido por el propio departamento de defensa de Estados Unidos para el uso civil y comercial, reservando el uso del sistema más preciso para el ejército y personal autorizado.

La gran mayoría de fuentes de error están relacionadas con el viaje de la señal desde los satélites situados a más de 20.000 kilómetros de altura y los usuarios, pero también hay errores causados por el hardware de los propios satélites y receptores.

Cada satélite está equipado con un reloj atómico, estos relojes son muy precisos y estables, pero no son perfectos y van acumulando error a lo largo de su órbita hasta que vuelven a pasar por el perigeo y se le envía una corrección de reloj desde la estación de monitorización correspondiente. En los cálculos de los desfases de los relojes se tienen en cuenta incluso leyes relativistas debido a la gran diferencia de velocidad entre los satélites y la superficie terrestre y la gran precisión necesaria.

Como se ha explicado anteriormente, el segmento de control es el responsable de mantener toda la constelación GPS con la menor deriva posible.

También influye la potencia de recepción del localizador, con un receptor potente se puede tener una precisión de centímetros en movimiento y de milímetros en el caso de una localización estática de larga duración, en el caso de receptores como los que puedan llevar los teléfonos inteligentes se puede lograr una precisión de entre 3 y 4 metros debido a la importancia de controlar el consumo energético de estos receptores.

La mayor concentración de errores ocurre durante el viaje y la recepción de la señal debido principalmente a obstáculos que dificultan la llegada de la señal de forma directa directa al receptor y distorsionan la distancia realmente recorrida por esta. Un ejemplo visual de esto es cuando un usuario se encuentra en una ciudad entre edificios altos y se reciben distintas señales que rebotan en estos.

Como hemos visto en el punto anterior, la distancia hasta los satélites se calcula midiendo el tiempo que tarda en llegar al dispositivo, si la señal no puede llegar de forma directa y llega de forma indirecta este tiempo será mayor, por lo que habrá un error en el cálculo de la distancia real al satélite.

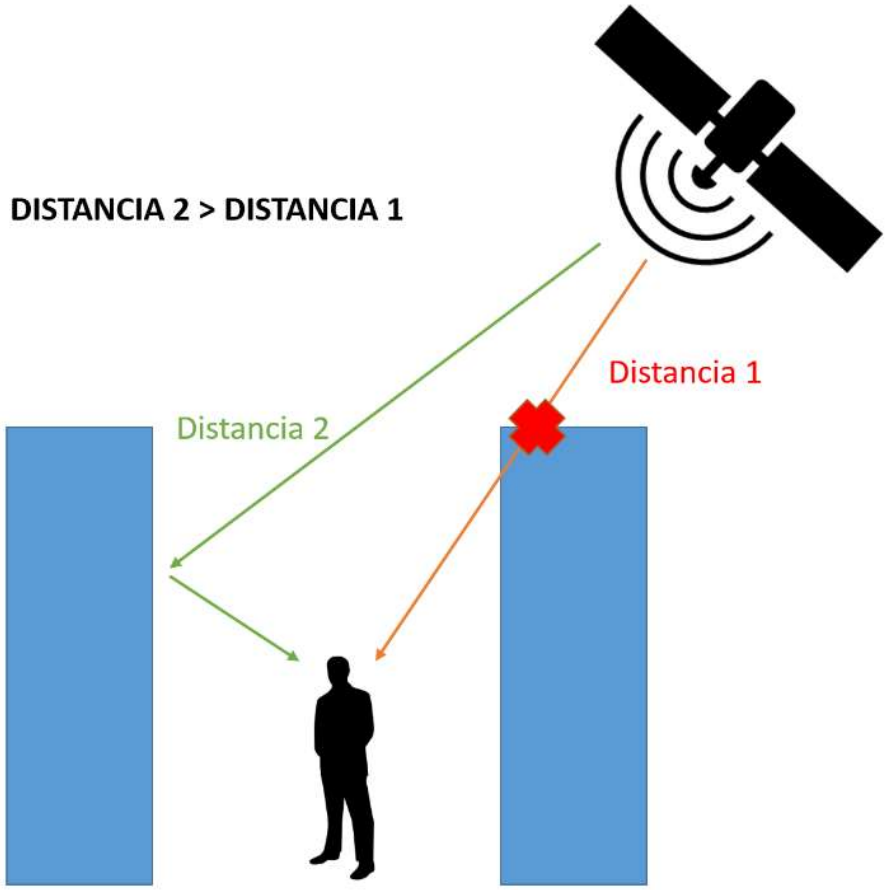


Figura 2.4 – Error de posicionamiento por bloqueo de señal.

Un factor también conocido es la distorsión o retraso que sufre la señal al atravesar una capa en particular de la atmósfera, la ionosfera. Esta capa es la responsable de la absorción de las radiaciones más dañinas para la vida en la tierra. Esta absorción se produce cuando las partículas presentes en esta capa, como por ejemplo el ozono, absorben esta radiación y se fragmentan emitiendo electrones e iones con sus respectivos campos eléctricos y magnéticos, responsables de dificultar el paso de la señal por esta capa atmosférica.

Otros factores no tan frecuentes que pueden afectar a la señal son posibles interferencias de radio ya que en el posicionamiento civil solo se utiliza una frecuencia y se pueden generar superposiciones, este error se evita en aplicaciones militares usando dos frecuencias distintas en el posicionamiento.

También pueden llegar a afectar factores como la radiación emitida a causa de erupciones solares o provenientes del espacio, pero estos factores raramente son determinantes ni los predominantes en un error de posición (The National Coordination Office for Space-Based Positioning, 2017).

CAPÍTULO 3. DISPOSITIVO GPS UBIC

3.1. METODOLOGÍA DE DESARROLLO DEL HARDWARE

El dispositivo UBIC es el encargado de realizar la localización GPS y obtener las coordenadas para posteriormente transferirlas a la aplicación móvil UBICAPP desde la cual el usuario puede realizar la navegación.

El dispositivo necesita de un núcleo de control y procesamiento desde el cual administre los datos y las conexiones, además también precisa de varios periféricos para realizar las funciones de obtención de coordenadas explicada anteriormente y la comunicación inalámbrica con la aplicación desarrollada para Android.

3.1.1. Conectividad y transferencia de datos

En primer lugar es necesario elegir la forma en la que se va a realizar la conexión entre dispositivo y aplicación ya que condicionará la elección de los componentes.

Las dos opciones que se han considerado han sido la conexión mediante Bluetooth y la conexión mediante Wifi. Cada una de las conexiones presenta unos pros y contras:

- La comunicación por Bluetooth presenta mayor facilidad de programación de la aplicación móvil y del dispositivo, facilidad de establecer conexión directa entre el localizador y el teléfono móvil y un menor consumo de energía. Por el contrario, el rango de conexión del Bluetooth es muy pobre, teniendo problemas de comunicación a partir de los 15/20 metros.
- La comunicación mediante Wifi consume más energía y presenta mayor dificultad de conexión y programación si se pretende transferir datos directamente sin un punto de acceso común, sin embargo el Wifi presenta una velocidad de transmisión mucho más rápida que el Bluetooth y lo que es más importante para el uso que se le quiere dar en este proyecto, la distancia máxima de conexión puede llegar a ser 5 veces mayor que el Bluetooth.

Estimando que la conectividad no va a suponer un gran ahorro de energía teniendo en cuenta el consumo del procesador y de los periféricos, no se ha considerado este punto como un factor determinante. No ocurre lo mismo con la distancia de conexión, la cual se ha valorado con una importancia elevada permitiendo de esta forma que el usuario pueda encender el dispositivo y no preocuparse por permanecer cerca el tiempo necesario para la puesta a punto y transferencia de datos.

Es por esto que finalmente se ha elegido la comunicación directa mediante Wifi para la comunicación UBIC-UBICAPP.

3.1.2. Elección de la unidad de procesamiento

Hoy en día existen muchas tarjetas ya sean micro controladores o micro procesadores preparadas para el prototipado rápido de proyectos, entre estas se encuentran conocidas marcas como Arduino, Intel, BQ o Raspberry Pi, entre otras. Estas tarjetas de desarrollo presentan conexiones rápidas y fáciles para crear los primeros prototipos o simplemente para aprender sobre el mundo de la programación, robótica y el cada vez más famoso internet de las cosas.

Entre todas estas tarjetas se ha tenido en cuenta en primer lugar las utilidades que incorpora (Wifi, Bluetooth, puertos de conexión, etc.), en segundo lugar el precio y en último lugar la potencia de procesamiento y dificultad de programación.

Se han barajado varias opciones como unidad de procesamiento del dispositivo. En un principio se estudió la posibilidad de utilizar una placa Arduino. En concreto o la Arduino Nano o la Arduino Uno Wifi.

Por un lado la placa Arduino Nano es una placa con la misma funcionalidad que la Arduino Uno, con el microcontrolador ATmega328 pero con un tamaño mucho más compacto. El gran punto a favor de esta placa es su reducido tamaño, sin embargo como punto negativo está su inexistente conectividad inalámbrica, no incluye Wifi ni Bluetooth por lo que precisa de un módulo externo Wifi para realizar la conexión con la aplicación. El módulo Wifi más utilizado con estas placas es el ESP8266 el cual según opiniones de usuarios no presenta una gran fiabilidad, fallando varias veces en el encendido y establecimiento de la conexión. Por estos puntos en contra se ha decidido desestimar esta opción.

La placa Arduino Uno Wifi es prácticamente igual que la Arduino Uno estándar pero con un módulo ESP8266 Wifi integrado en la propia placa conectado mediante I^2C (Bus serie de datos), este módulo al estar integrado directamente en la placa presenta una mayor estabilidad. El microcontrolador es, como en la mayoría de placas Arduino, un ATmega328. La principal traba es debida al soporte ofrecido para esta placa, debido a que es relativamente nueva en el mercado aún no tiene disponibles ciertas librerías que habilitan muchas de las funciones relacionadas con el Wifi.

Tras descartar esta última opción se optó por una tarjeta con un rango de precio similar a esta última, tras el estudio de varias opciones como la Orange Pi o la Intel Edison se ha seleccionado una tarjeta con más potencia que la Arduino y con conectividad Wifi y Bluetooth incorporadas en la misma placa, la Raspberry Pi 3 modelo B.

Raspberry Pi es una fundación que busca acercar la programación y promover la creación de sus propios proyectos a las personas, ofrecen una gama de tarjetas de software libre a un precio muy competitivo. Su filosofía es promover la creatividad de las personas para que creen sus propias ideas y que no se queden simplemente con lo que las grandes empresas les quieran vender.

La tarjeta Raspberry Pi 3 Model B es la unidad central de procesamiento del dispositivo en la cual se escriben los programas mediante los cuales se gestionan los demás módulos y la información recibida de estos.

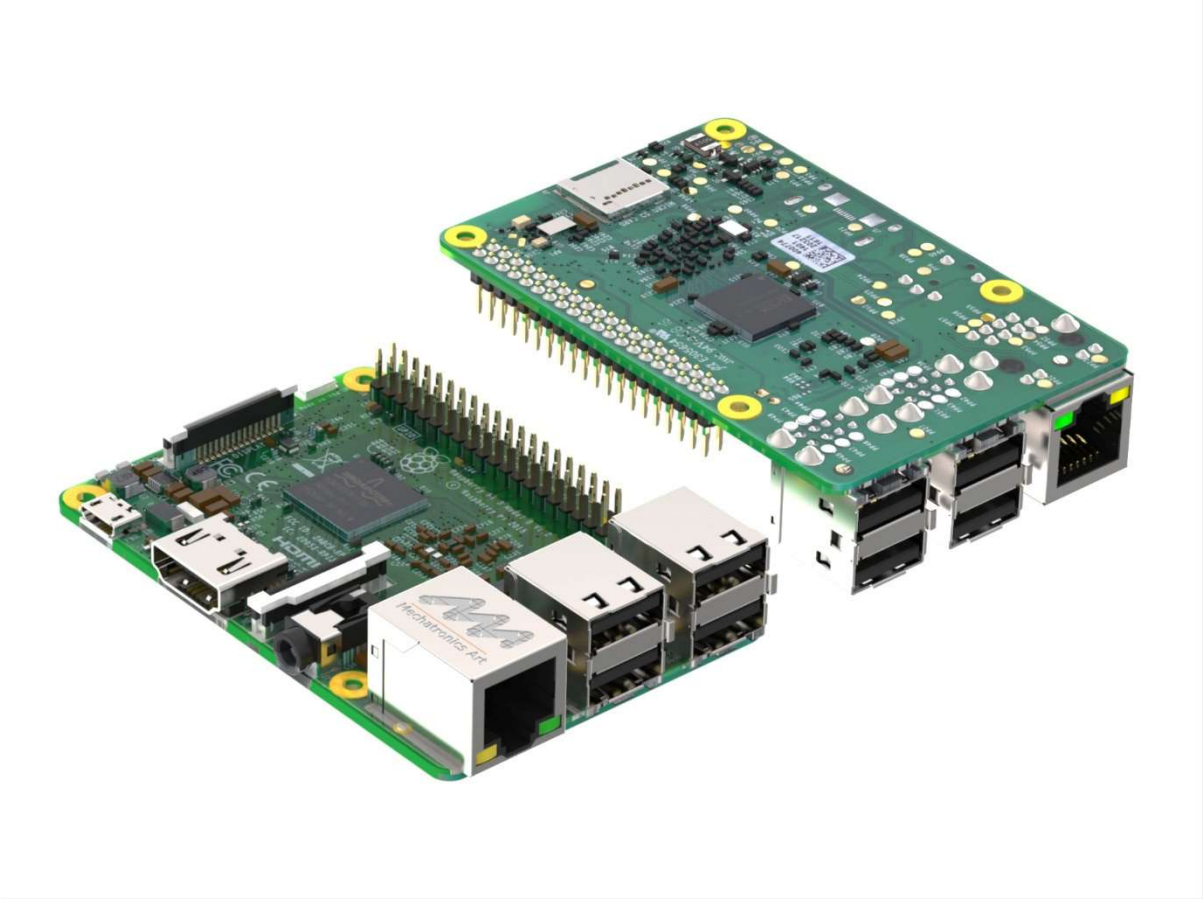


Figura 3.1 – Raspberry Pi 3 Modelo B. Imagen libre de www.grabcad.com.

Esta tarjeta dispone de:

- Microprocesador Quad Core 1.2GHz Broadcom BCM2837 de 64bit
- 1 GB de memoria RAM LPDDR2

Conectores

- Conectividad Wifi 802.11n y Bluetooth 4.1 integrados en la placa.
- Ethernet socket Ethernet 10/100 BaseT
- 40 GPIO (General Purpose Input/Output) pines entrada/salida.
- 4 puertos USB 2.0
- HDMI
- Puertos para conexión de cámara y pantalla táctil.
- Ranura micro SD para la instalación de sistema operativo y almacenamiento de datos.
- Micro USB de alimentación (recomendable 5V a 2.5 A CC)

Con unas dimensiones de 85x53mm.

La principal diferencia entre la Raspberry Pi y Arduino es el núcleo de cada placa, mientras que en Arduino se trata de un microcontrolador, la Raspberry Pi monta un microprocesador el cual necesita un sistema operativo para funcionar.

El sistema operativo se instala en una tarjeta de memoria micro SD que se introduce en la parte inferior, se ha decidido instalar el sistema operativo oficial que ofrece la fundación Raspberry Pi para todas sus tarjetas. Raspbian es un sistema de código libre basado en Debian y está optimizado para el hardware de las placas Raspberry Pi.

Como unidad de almacenamiento se ha seleccionado una tarjeta de almacenamiento flash micro SD Samsung Evo de 32 GB de capacidad.

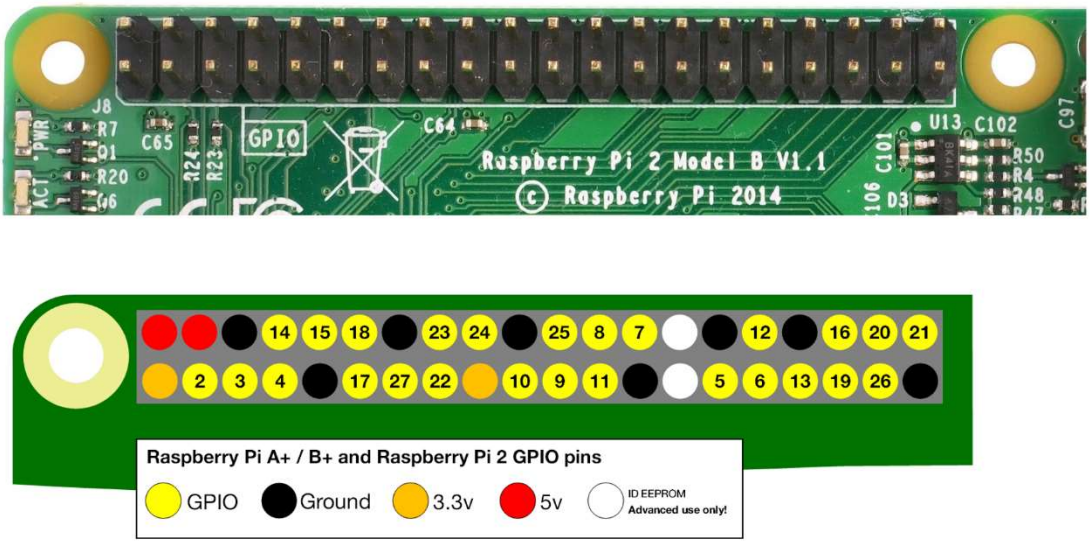


Figura 3.2 – Pines Raspberry Pi 1 B+, 2 y 3. Imagen por Clivebeale de https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/README.md

Todas las Raspberry Pi, excepto la primera, disponen de 40 pines (véase figura 3.2):

- 2 alimentación 5V
- 2 alimentación 3.3V
- 8 Ground, masa.
- 26 GPIO
- 2 ID EEPROM

Los pines GPIO son entrada/salida y se pueden usar tanto para señal analógica como digital, incluyendo PWM (Raspberry Pi Foundation, s.f.). Con estos pines se controlaran los módulos con los que se realizan las funciones de posicionamiento GPS y comunicación vía SMS.

3.1.3. Comunicación serial mediante UART

Antes de continuar con los componentes se introducirá el tipo de comunicación que existe entre los periféricos y la Raspberry Pi 3.

Para las conexiones con los módulos periféricos se utilizarán comunicaciones serie a través de un receptor/transmisor asíncrono universal, UART (*Universal Asynchronous Receiver/Transmitter*).

Para la comunicación serial es necesario un UART en el emisor y otro en el receptor. En rasgos generales estos transmisores/receptores consiguen transformar los bits de información de paralelo a serie, transmitirlos por un solo cable y volverlos a ensamblar en paralelo en el UART de destino para su utilización garantizando que llegan y se vuelven a ensamblar en el mismo orden que se enviaron.

La comunicación entre dos UART se realiza a través de las conexiones TxD (transmisor) y RxD (receptor). La conexión se realiza TxD_{Emisor} con $RxD_{Receptor}$ y RxD_{Emisor} con $TxD_{Receptor}$ (véase figura 3.3) (Nanda & Kumar Pattnaik, 2016).

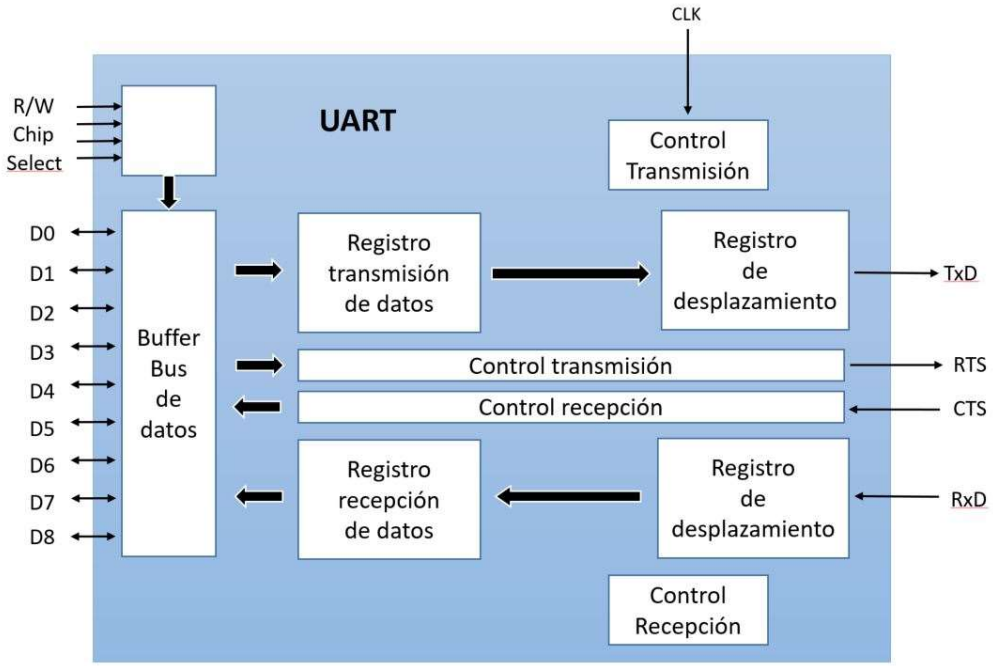


Figura 3.3 – Diagrama Transmisor/Receptor Asíncrono Universal (UART). (Departamento de Electrónica, Univesidad de Málaga)

Una de las conexiones serial se efectúa a través de los pines GPIO.

En la raspberry Pi el puerto UART se situa en los pines GPIO 14 y 15 los cuales son TxD y RxD respectivamente.

La otra conexión serial se realiza mediante un adaptador USB – TTL PL2303TA, el cual hace de puente entre la conexión USB de la Raspberry Pi y el UART del módulo.

3.1.4. Módulo GPS

En cuando a la localización GPS es necesario un módulo capaz de comunicarse con los satélites y realizar las mediciones explicadas en el apartado de obtención de coordenadas del capítulo 2.

El GPS tiene un protocolo de comunicación estandarizado por lo que para la elección del módulo solo habrá que tener en cuenta su precisión en la localización, número y tipo de conexiones de las que dispone y el precio.

Las primeras opciones de módulos GPS que se han considerado han sido de la marca de componentes electrónicos Adafruit, estos módulos cuentan con una buena precisión y fiabilidad. El único inconveniente de esta marca es el precio, el coste de los módulos GPS de Adafruit ronda los 40€, siendo incluso más caro que la Raspberry Pi 3, por lo que se han buscado alternativas.

Investigando sobre módulos de características similares se han hallado los módulos GPS de la marca U-blox, estos módulos presentan una gran precisión y fiabilidad a un precio muy reducido. Dentro de la amplia gama de U-blox se ha escogido el NEO-6M con un precio aproximado de 8€, viene montado en una placa con las salidas VCC, TxD, RxD y GND. Las dimensiones de este son de 25x35x3 mm (véase figura 3.4).

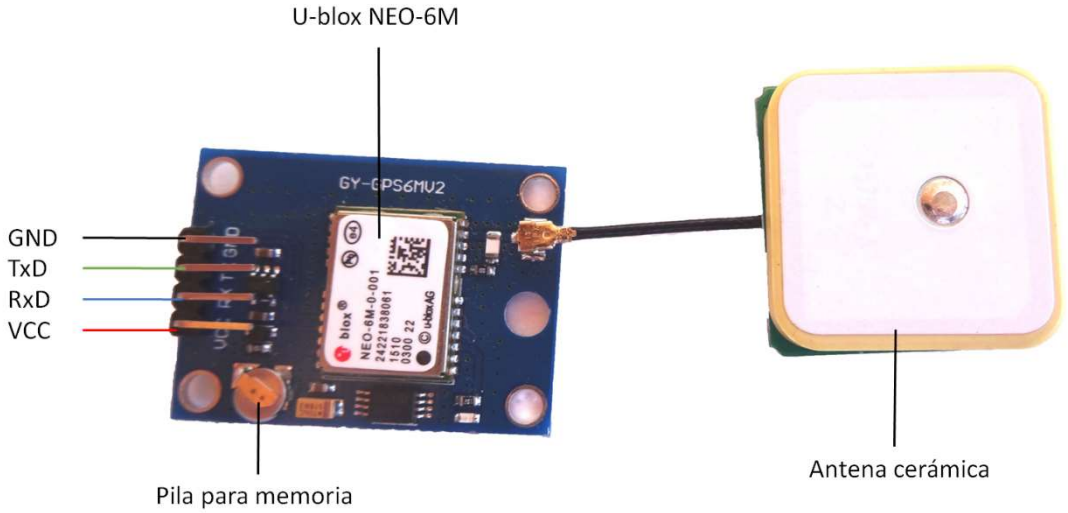


Figura 3.4 – Módulo GPS NEO-6M GY-GPS6MV2

El módulo está conectado a la Raspberry Pi a través del adaptador USB-TTL con dos cables de alimentación que proporcionan 3,3V y dos cables RxD y TxD que transfieren las coordenadas cada segundo aproximadamente de forma serial (véase figura 3.6). Éste módulo cuenta con una antena cerámica que se conecta a la placa mediante la cual efectúa la conexión con los satélites, las dimensiones de la antena son 25 x 25 x 8 mm.

Características U-blox Neo-6M:

- Tiempo de posicionamiento en frío de 27 segundos y en caliente de 1 segundo.
- Precisión horizontal GPS de 2.5m
- Rango de frecuencias configurables 0,25 Hz hasta 1 kHz.
- Precisión de pulso de la señal <60 nanosegundos.
- Precisión de velocidad 0.1 m/s
- Límites de operación de altitud = 50 km y velocidad = 500 m/s

EL módulo dispone de una pequeña pila para mantener la hora y de esta forma realizar el posicionamiento de forma más rápida después de la primera conexión (u-blox, 2011).

Es importante entender la diferencia entre arranque en frío, arranque templado y arranque en caliente. Los receptores guardan en sus memorias cierta información sobre la posición y el estado de los satélites para facilitar y agilizar la próxima conexión a estos.

Las tres informaciones utilizadas por el módulo receptor son el almanaque, las efemérides y el tiempo local. De estos tres conceptos hay dos que no se conocen aún, el almanaque es la información generada por los satélites sobre ellos mismos en conjunto, como por ejemplo el estado de salud individual y de la constelación o la posición de las órbitas, y las efemérides son la información que envía cada satélite sobre su posición en particular dentro de su órbita.

El más lento de todos es el arranque autónomo. Para que se de este arranque el receptor no debe tener actualizada ninguna de las 3 informaciones anteriores, teniendo en cuenta que el periodo de validez del almanaque son varios meses este arranque solo suele producirse la primera vez que se enciende el receptor o tras una parada muy prolongada. Este arranque puede durar desde minutos a horas dependiendo de la potencia de la antena y el receptor así como la buena visibilidad del cielo.

El arranque en frío se produce cuando el módulo solo tiene el almanaque válido para realizar la conexión con los satélites, el tiempo normal de conexión con este arranque ronda los 45 segundos.

Con el arranque en templado el tiempo de conexión se reduce hasta aproximadamente 15 segundos. Este arranque se produce cuando el almanaque y las efemérides son válidas, teniendo en cuenta que la validez de las efemérides es de unas 4 horas, tras este periodo de desconexión el siguiente arranque ya se producirá en frío.

El arranque más rápido de todos es el arranque en caliente. Este ocurre tras una breve desconexión del receptor GPS (por ejemplo, el paso por un túnel) y la reconexión ocurre en un periodo de 1 a 5 segundos aproximadamente. En este arranque el receptor tiene el almanaque y las efemérides actualizadas y además la hora local lo suficientemente precisa por lo que no es necesario volver a descargar la hora de los satélites (Mehaffey, 1999).

3.1.5. Módulo GSM

Una vez montado y funcionando el módulo GPS se ha decidido mejorar el sistema añadiendo comunicación mediante SMS, pudiendo de esta forma actualizar la posición del dispositivo incluso estando fuera del rango de conexión Wifi.

Para realizar el envío de coordenadas a larga distancia se necesita un módulo GSM (sistema global para comunicaciones móviles). En estos módulos se ha hallado el mismo problema que en el módulo GPS, la gran mayoría de módulos incluyen conectividad 3G o 4G con conexión a internet de alta velocidad, todas estas características aumentan el precio y no resultan útiles en el sistema ya que lo que se busca es dar la mayor cobertura posible.

Mediante la red 2G o GSM se pueden realizar y recibir llamadas y enviar SMS con información de las coordenadas siendo la red que más extensión cubre de modo que se ha buscado un módulo que pueda cumplir estas características sin añadir funcionalidades que no sean necesarias.

El módulo que se ha seleccionado finalmente es el **SIM800L** del fabricante SIMCom, por un precio aproximado de 5€, es un módulo GSM cuatribanda capaz de recibir y realizar llamadas o enviar y recibir SMS en todo el mundo, para esto es necesario una tarjeta SIM de un operador de telefonía móvil.

Tras un estudio sobre las tarjetas y ofertas de las principales compañías de telefonía, la mejor SIM encontrada para la finalidad del sistema es la tarjeta prepago DíaMóvil con un coste aproximado de 5 cent/SMS, sin embargo en todos los establecimientos en los que se ha intentado adquirir no se encontraba disponible por lo que se ha tenido que tomar como alternativa una SIM prepago Vodafone con un precio de 18 cent/SMS.

El módulo funciona en las frecuencias GSM850MHz, EGSM900MHz, DSC1800Mhz y PCS1900MHz, en España la banda GSM o 2G utiliza las frecuencias de 900 MHz y 1800 MHz.

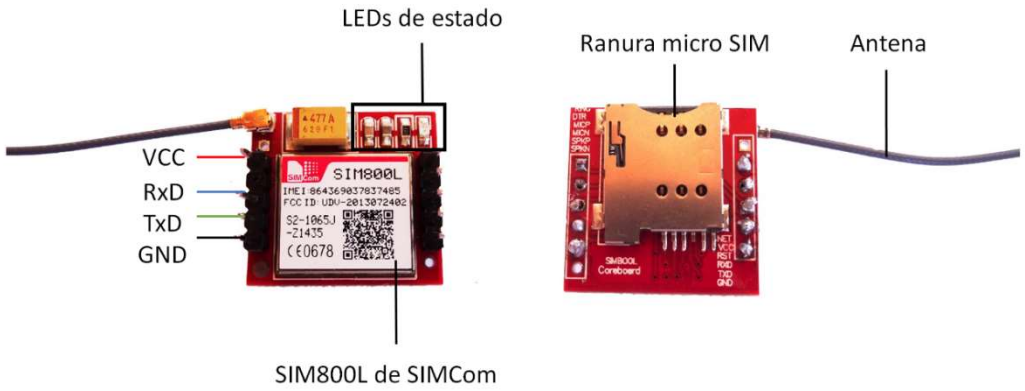


Figura 3.5 – Módulo GSM SIM800L coreboard.

La alimentación del módulo se realiza conectando el positivo directamente a la alimentación de la batería a través de 2 diodos reguladores 1N4007 en paralelo, limitando de esta forma la

corriente a 2A y generando una caída de tensión de aproximadamente 0.7V, con esto se consigue que la alimentación del módulo sea de 4.3V y esté dentro del rango dado por el fabricante (3.4 V – 4.4 V). El negativo o masa está conectado a uno de los pines GROUND de la Raspberry Pi para que tengan una masa común y no haya problemas con los niveles lógicos en la comunicación (véase figura 3.6).

La alimentación se realiza de esta manera debido a los picos de demanda energética del módulo durante su funcionamiento. Mediante los pines de la Raspberry Pi no es posible suministrar estos picos energéticos pudiendo quedar el módulo bloqueado o resetearse de manera constante.

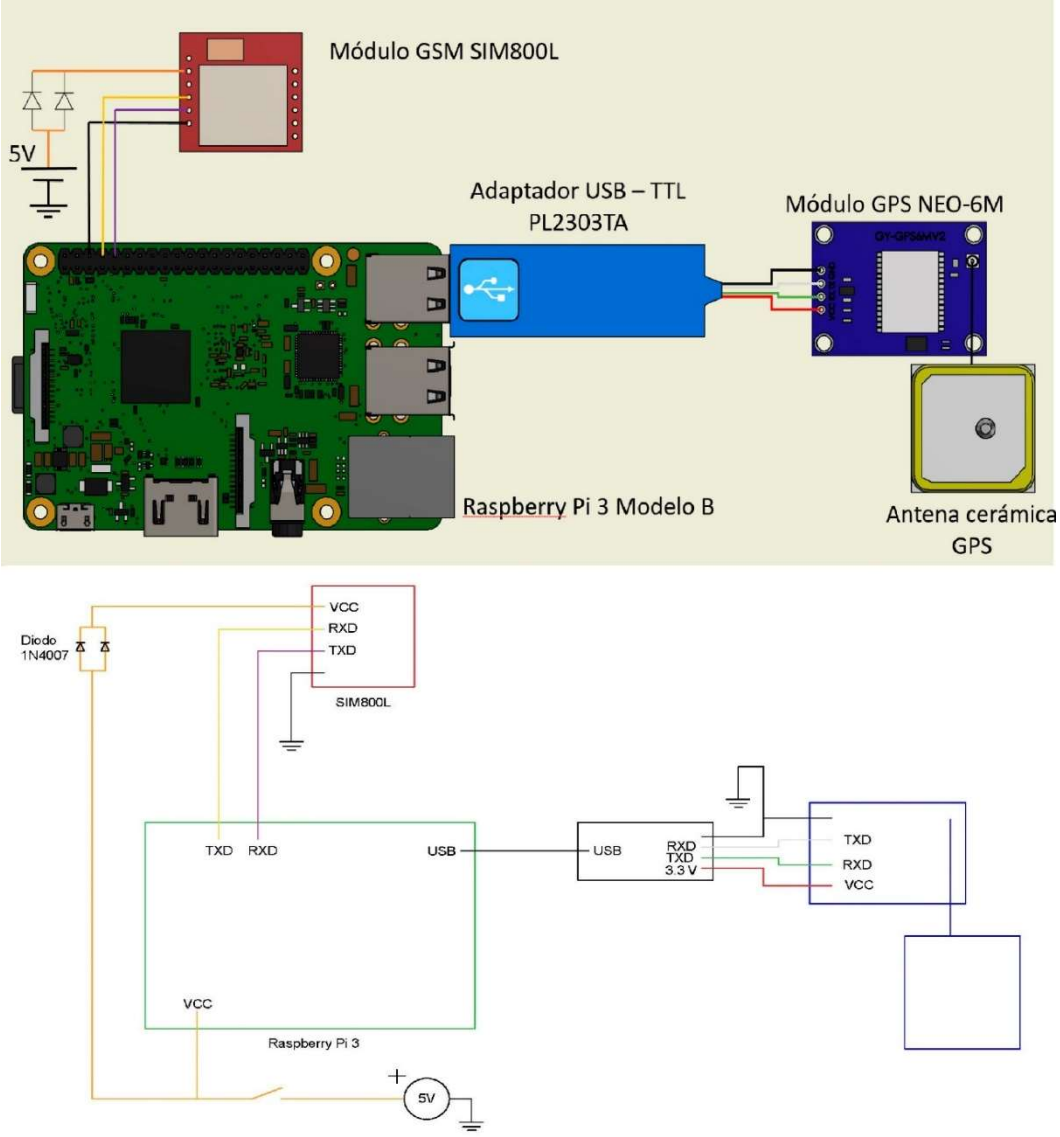


Figura 3.6 – Esquema conexiones dispositivo UBIC.

3.1.6. Fuente de alimentación

Todos los elementos del dispositivo están alimentados por una batería de iones de litio Poweradd Slim 2 de 5000 mAh a 3,7V de capacidad. Esta batería necesita una fuente de 5V y hasta 2A de corriente continua para la carga más rápida y durante su descarga ofrece una diferencia de potencial de 5,1 V y entrega hasta 2,1 A.

Para la elección de esta batería se ha tenido en cuenta las dimensiones de la misma así como su relación (capacidad en mAh)/precio.

Con esta batería se estima una autonomía del dispositivo de 8 horas aproximadamente, la carga se realiza a través de un conector micro USB.

3.1.7. Diseño de carcasa

Para facilitar el transporte de todos estos elementos e integrarlos en un único producto se ha diseñado una carcasa con el programa Inventor de Autodesk. Se ha modelado esta carcasa en 3D midiendo cada uno de los productos y ensamblándolos de la forma más adecuada para su correcta compactación y teniendo en cuenta la necesaria ventilación de los elementos electrónicos, para esto se ha dotado de rejillas en la carcasa para mejorar la disipación pasiva de calor de los elementos.

La carcasa se ha creado a través de impresión 3D en plástico ABS. Cuenta con un compartimento para la batería, un soporte para la Raspberry Pi 3 con 4 apoyos, otro soporte de 4 apoyos para el módulo GPS, un soporte para el módulo GSM y 2 aberturas preparadas para el interruptor de encendido/apagado y el micro USB de carga (véase figura 3.7).

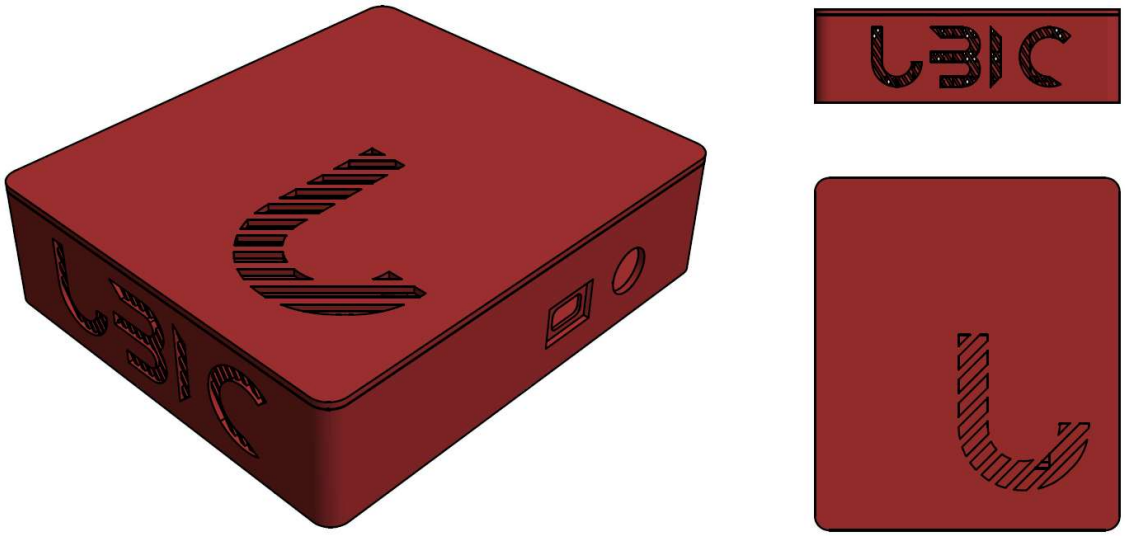


Figura 3.7 – Diseño 3D de la carcasa con el programa Autodesk Inventor Professional 2017.

Está formada por dos partes, una base donde se encuentran todos los soportes y la tapa para proteger los elementos electrónicos.

La principal función de la carcasa es la protección de todos los elementos ya que al ser componentes electrónicos con partes metálicas se pueden realizar contactos entre ellos produciendo cortocircuitos, fallos en los componentes o incluso causando la avería de estos.

Se adjuntan los planos en detalle de las dos partes que componen la carcasa (tapa y base) en el apartado de planos.

3.2. METODOLOGÍA DE DESARROLLO DEL SOFTWARE

3.2.1. Puesta en marcha de la Raspberry Pi 3

Como se ha visto anteriormente la Raspberry Pi necesita de un sistema operativo para funcionar el cual se instala en la tarjeta micro SD. El software necesario, llamado NOOBS, se descarga desde la página oficial de raspberrypi.org. En primer lugar se formatea la tarjeta SD y se copia el contenido descargado en esta, tras este paso se introduce la SD en la Raspberry Pi y se elige el sistema operativo deseado, en este caso Raspbian, y la instalación se realiza de forma automática tras seleccionar el idioma.

Para el primer arranque es necesario disponer de un teclado y ratón USB y un monitor con entrada HDMI para interactuar con la tarjeta. En este primer arranque se configura la conexión Wifi y la SSH, la cual facilitará el trabajo con la Raspberry Pi durante el desarrollo del software necesario.

Una conexión SSH permite controlar un dispositivo de forma remota mediante una conexión inalámbrica ya sea mediante Wifi directo entre PC/móvil/Tablet y Raspberry Pi o mediante una red Wifi externa común, esta conexión se crea de forma privada y codificada siendo completamente segura.

Para la conexión se necesita un cliente SSH desde el equipo desde el cual se va a controlar la Raspberry. En el caso del PC con sistema operativo Windows se ha utilizado el cliente Putty, mientras que en el caso del teléfono Android se ha utilizado la aplicación gratuita JuiceSSH, muy útil para modificar código desde el propio teléfono móvil y depurar los últimos detalles.

A partir de este momento todas las tareas de configuración y programación se han realizado mediante SSH con un PC o el teléfono móvil.

3.2.2. Comunicación con el módulo GPS

El módulo GPS adquirido dispone de 4 conexiones como se ha descrito anteriormente. En primer lugar se ha realizado la conexión de forma directa a los pines TxD y RxD de la Raspberry Pi. Para ello es necesario habilitar el UART de los pines GPIO que viene deshabilitado por defecto.

Esto ha requerido más tiempo del previsto debido a que la Raspberry Pi 3 no se configura de la misma manera que todas las demás. La causa de esto es que el Bluetooth incorporado utiliza el puerto serial /dev/ttyAMA0. Para la correcta comunicación con el GPS se debe elegir entre:

Desvincular el puerto serial /dev/ttyAMA0 del Bluetooth, es decir, dejar la Raspberry Pi 3 sin la funcionalidad de Bluetooth o utilizar el otro serial disponible /dev/ttyS0. Este puerto se rige por la frecuencia del núcleo y puede dar problemas de comunicación si el núcleo tiene mucha carga de trabajo. La forma de arreglar este problema es fijar la frecuencia de la CPU de manera que el puerto ttyS0 sea estable, sin embargo, esto afecta al rendimiento de la placa.

Debido a que no se quiere renunciar a la conectividad Bluetooth ya que puede resultar muy útil en mejoras posteriores del sistema y que para las funciones que se van a realizar no se necesita el 100% de la potencia de procesamiento se ha elegido la opción de limitar la frecuencia de la CPU y realizar la comunicación mediante el puerto ttyS0. Resumido:

/dev/ttyAMA0 → Bluetooth

/dev/ttyS0 → puerto serie pines GPIO → Módulo GPS

Limitando la velocidad del procesador no solo se obtiene el beneficio de disponer del puerto serie estable para la comunicación sino que además reducimos el consumo de la Raspberry Pi 3 mejorando la autonomía del dispositivo y alargando la vida útil de la batería.

En primer lugar se debe modificar el archivo *cmdline.txt* situado en el fichero *boot* y eliminar todo lo que haga referencia al ttyS0.

Tras guardar este archivo se deshabilita la consola serial que viene por defecto en el puerto ttyS0 con los comandos:

```
1. sudo systemctl stop serial-getty@ttyS0.service
2. sudo systemctl disable serial-getty@ttyS0.service
```

A continuación se activa el UART y se fija la frecuencia de la CPU. En el archivo config.txt se añade:

```
1. enable_uart = 1
2. core_freq = 250
```

Todos los GPS utilizan un lenguaje de comunicación común y estandarizado llamado NMEA. Para la lectura de este se necesita de un software capaz de entender el protocolo de comunicación, para ello se instala el software GPS Daemon, de ahora en adelante gpsd.

Se le indica el puerto a través del cual va a recibir los datos GPS con:

```
1. sudo gpsd /dev/ttyS0 -F /var/run/gpsd.sock
```

Se realiza un test para comprobar que la comunicación se ha realizado de forma correcta con:

```
1. sudo cat /dev/ttyS0
```

El cual muestra:

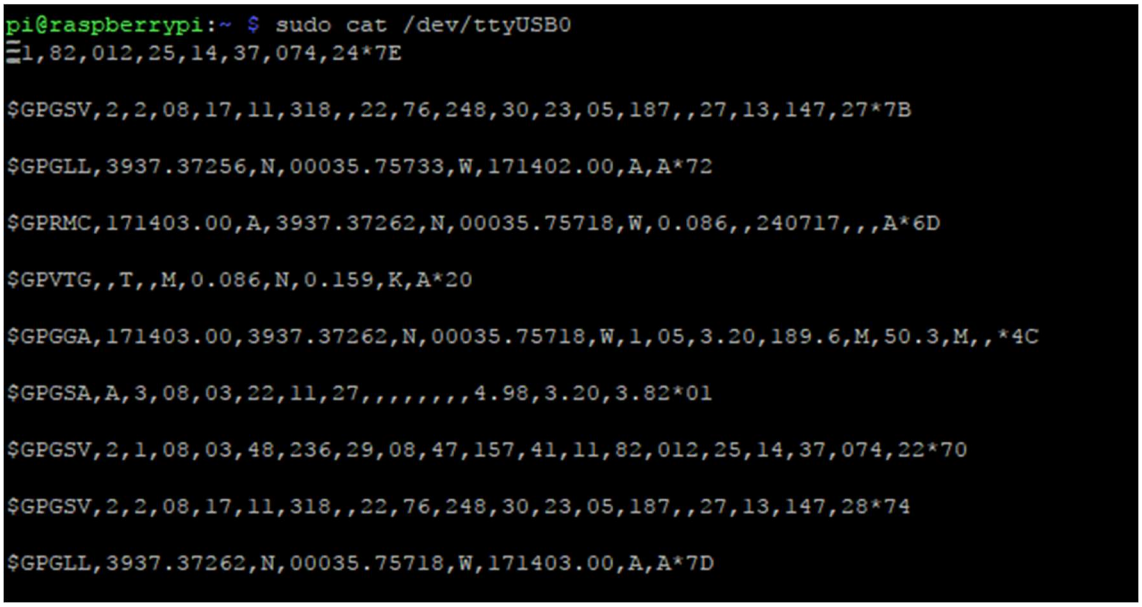


Figura 3.8 – Datos recibidos por el módulo GPS.

En la figura 3.8 se puede ver que llegan gran cantidad de datos periódicamente, con esto podemos comprobar que la conexión es correcta y que se están recibiendo 7 filas de datos cada segundo aproximadamente. Gracias al software gpsd instalado anteriormente y la librería Python-gps, se pueden aislar los datos que resulten útiles escribiendo el programa que se explicará más adelante.

Tras estos pasos ya se dispone del UART de los pines GPIO habilitado y configurado. Sin embargo, posteriormente se ha decidido cambiar la conexión del GPS de los pines GPIO al adaptador TTL-USB descrito anteriormente. La causa de esto es la incorporación del módulo GSM mencionado anteriormente y el cual irá conectado a los pines aprovechando que sirve la misma configuración ya realizada.

Esto afecta a la comunicación simplemente de una forma. El puerto de entrada de datos que debe escuchar ahora el gpsd pasa a ser /dev/ttyUSB0, el puerto al que está conectado el USB.

3.2.3. Conexión con la aplicación móvil.

Para realizar la comunicación con la aplicación móvil se ha optado por la conexión Wifi como se ha mencionado en el punto 3.1.1.

La transferencia de datos más común entre dispositivos a través de Wifi se suele realizar mediante una red Wifi externa a los dispositivos generada por un tercer elemento, el cual emite la señal tal y como se puede ver en el recuadro gris de la figura 3.9. Esto no es de utilidad en este caso debido a que está pensado para utilizarlo en exteriores donde lo más probable es que

no se disponga de un punto de acceso común abierto al que puedan conectarse ambos dispositivos y que funcione de canal de comunicación entre ellos.

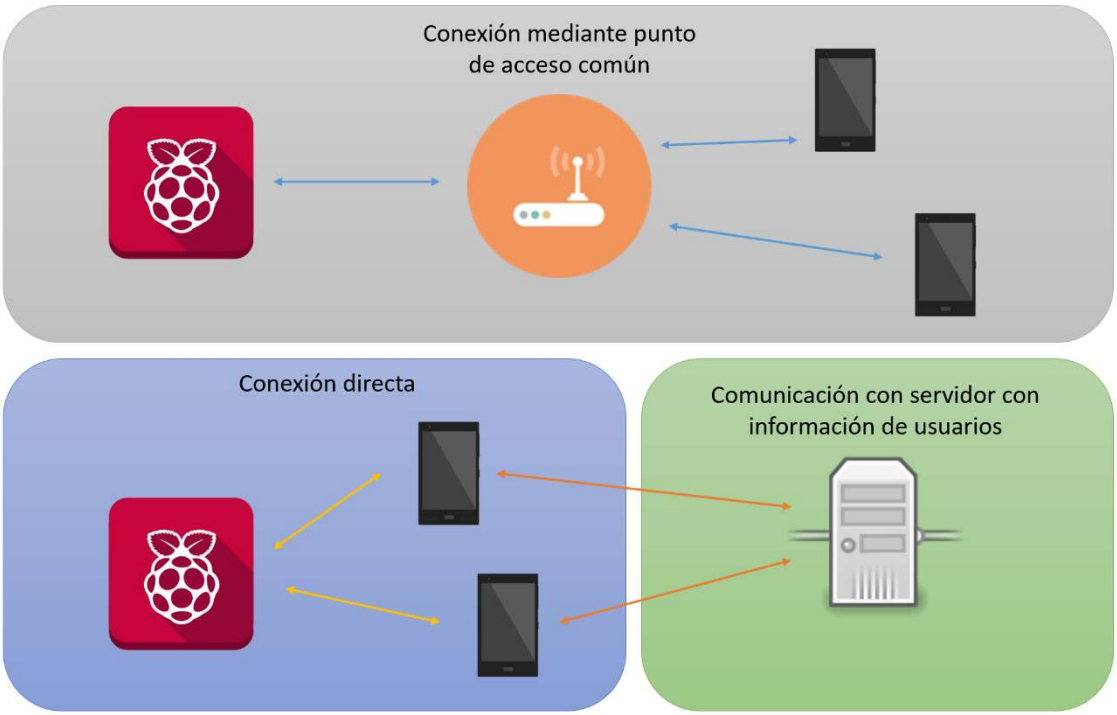


Figura 3.9 – Esquema conexiones inalámbricas.

El sistema UBIC/UBICAPP se comunica mediante una red Wifi propia generada por la Raspberry Pi 3. En el momento en el que se cree una red de usuarios la aplicación móvil se comunicará con un servidor desde el cual se administrará la información de cada usuario en una base de datos.

Para disponer de una red Wifi propia es necesario cambiar la configuración de la tarjeta de red que se ocupa de la conexión Wifi. Primero se fija una IP estática (IP asignada a la Raspberry Pi: 192.168.42.1) que es su identidad dentro de la red Wifi.

Para utilizar esta utilidad con la Raspberry se debe instalar el software hostapd. Tras su instalación se configura la red creando el archivo hostapd.conf en el cual especificaremos las características de la red Wifi como su nombre (SSID: UBIC), seguridad (WPA: 2) y contraseña (WPA_PASSPHRASE: sistemaUBIC).

A continuación se configura la máscara DNS (sistema de nombres de dominio). En esta configuración se asigna el rango de direcciones IP que se asigna a los dispositivos que se conecten a la red. Puesto que se tiene pensado que al menos 4 dispositivos puedan conectarse a la vez a la red Wifi se ha designado un rango de direcciones IP muy acotado de 192.168.42.10 a 192.168.42.14. De esta forma si más dispositivos intentan conectarse no recibirán IP y por lo tanto no tendrá comunicación dentro de la red.

Para terminar con la configuración se reinicia la tarjeta wlan0 y se reinicia la Raspberry Pi 3, tras el encendido de esta ya está disponible la red Wifi UBIC.

3.2.4. Comunicación con el módulo GSM

Como ya se ha explicado anteriormente, el módulo GSM se ha incorporado al sistema en el último momento cuando ya se tenía todo conectado y funcionando. Esto ha causado algunos cambios como el de la conexión del módulo GPS.

Sin embargo la configuración de los puertos serie ya explicada es perfectamente válida para la comunicación con módulos GSM que funcionen por protocolo AT.

Los comandos AT reciben su nombre del prefijo que se utiliza en cada uno de ellos y que significa atención. Estos comandos fueron desarrollados por la compañía Hayes Communications y se convirtieron en el estándar de comunicación con los módems hasta que apareció el Windows 95, con la aparición de este sistema operativo se empezaron a desarrollar controladores específicos y los comandos AT perdieron importancia.

Para comprobar la comunicación con el módulo se utiliza una consola serial llamada Minicom, para ello se utilizan los comandos obtenidos de la hoja de datos del fabricante.

Con el comando:

```
1. minicom –D /dev/ttyS0 –b 9600
```

Se abre la consola para leer y escribir en el fichero *ttyS0* con una tasa de baudios de 9600.

Al escribir el comando AT el módulo contesta con OK, lo que significa que la conexión y el inicio del módulo se han realizado correctamente. Es necesario iniciar la comunicación con el comando AT cada vez que se reinicia el dispositivo para que el módulo este operativo y a la escucha de otros comandos.

Siguiendo con las comprobaciones, se introduce la tarjeta SIM y se comprueba que el módulo tiene señal y puede realizar y recibir llamadas y mandar mensajes de texto.

3.2.5. Programas Python

En el dispositivo se ejecutan 2 programas de forma automática cada vez que este se enciende.

Estos programas se han escrito en lenguaje Python debido a la simplicidad de comunicación con los periféricos gracias a las librerías disponibles, sobretodo en el caso del módulo GPS.

El programa que se ha desarrollado en primer lugar es el llamado UBIC.py. Este programa es el encargado de la obtención de las coordenadas desde el puerto serie al que está conectado el módulo GPS y el envío de estas coordenadas mediante Wifi a la aplicación móvil.

La primera idea que se tenía para realizar esta conexión y el envío de datos a la aplicación se basaba en una comunicación bidireccional entre dispositivo UBIC y aplicación UBICAPP, es decir, la aplicación solicita las coordenadas al dispositivo cuando el usuario lo desee y este las envía a través del Wifi con una conexión directa.

No obstante, el hecho de necesitar la atención del usuario cada vez para actualizar las coordenadas genera una pérdida de utilidad del sistema. Finalmente se ha decidido realizar la

comunicación de manera unidireccional desde la Raspberry Pi hacia las IP conectadas con el fin de automatizar el proceso.

UBIC.py es un programa que se ejecuta constantemente en bucle el cual comprueba la información recibida por el puerto ttyUSB0.

Se sirve de las librerías gps, socket, sys, time y os.

Cada pulso de información recibido por el módulo GPS lo guarda en una variable llamada *report*. Esta variable siempre contiene datos ya que aunque el GPS no tenga señal y no sea capaz de realizar el posicionamiento manda información al puerto constantemente.

Se comprueba si la información contenida en *report* es útil, o lo que es lo mismo, si el GPS está conectado a los satélites y ha realizado el posicionamiento. En caso afirmativo, de la variable *report* se extrae lat y lon, que son latitud y longitud respectivamente, y se guardan en una cadena de texto (string) separados por el signo '|'.

Con esta información se realizan dos acciones: por una parte se guarda en un fichero de texto llamado coordenadas.txt en la memoria de la Raspberry Pi el cual siempre contiene la última posición actualizada, por otra parte se envía al rango de direcciones IP: [192.168.42.10 , 192.168.42.14].

El envío de esta información ha resultado algo compleja ya que la Raspberry Pi no dispone de conexión a internet. Se ha intentado exponer la información como http en una dirección web local para que posteriormente la aplicación Android accediera a esta dirección y descargara la información, sin embargo, no se ha tenido éxito en el desarrollo de esta idea.

Como alternativa descubierta posteriormente y que resulta ser más práctica que la inicial, la información se envía como cadena de texto (string) en un socket a las direcciones IP del rango configurado en la máscara DNS. Desde la aplicación móvil se está escuchando para recibir los sockets con la información.

Cada pulso de información recibido del GPS se comprueba si sigue conectado a los satélites, de ser así se sigue mandando sockets. En caso de que se perdiera la señal, el envío de sockets se detiene hasta que el posicionamiento sea posible de nuevo (véase figura 3.10). El código completo está disponible en el anexo 1.

Siguiendo el esquema de la figura 3.10 se puede ver el itinerario que sigue la información de las coordenadas, desde que se lee del serial conectado al GPS hasta que le resultan útiles al usuario.

El otro programa Python al que se le ha nombrado como UBICSMS.py es el administrador del módulo GSM. El programa se inicia con 30 segundos de desfase después del encendido de la Raspberry Pi para que el módulo pueda arrancar y establecer la conexión con el operador, tras este tiempo de iniciación se activa el módulo escribiendo en el serial el comando AT.

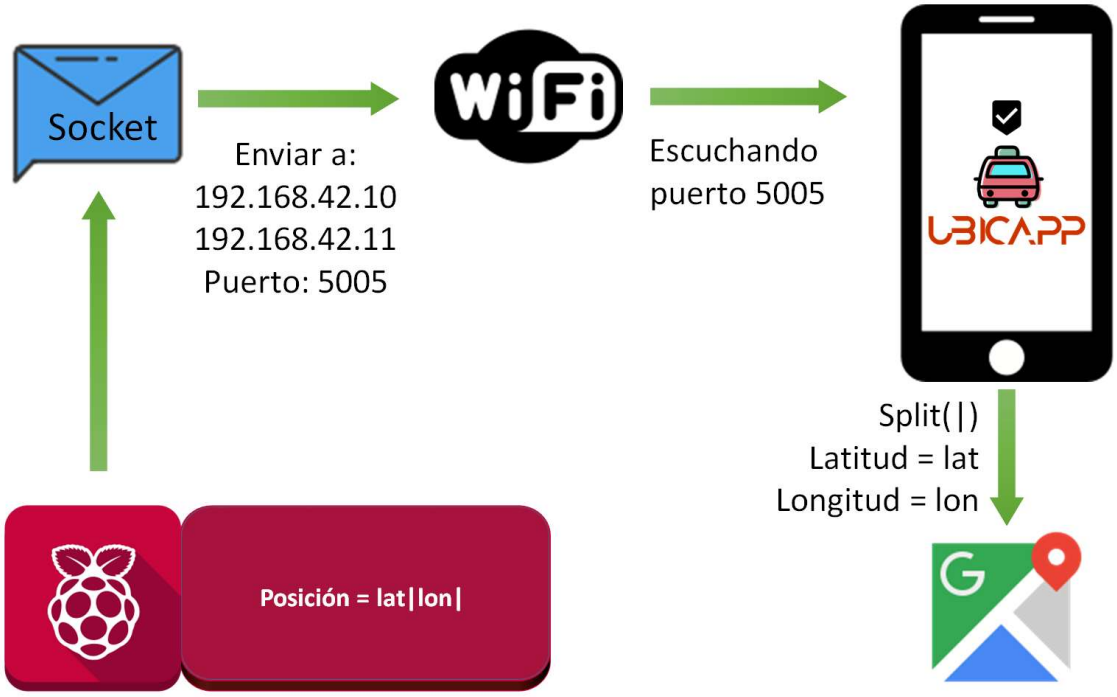


Figura 3.10 – Esquema comunicación mediante sockets.

El funcionamiento del módulo es igual al de un teléfono móvil, este se mantiene a la espera de recibir llamadas, SMS o alguna orden por parte del serial al que está conectado. La función del módulo es recibir una llamada, cortar la conexión y devolver las coordenadas que en ese momento se encuentren en el archivo coordenadas.txt mediante un SMS al teléfono que ha efectuado la llamada. El código completo se puede consultar en el Anexo 2.

Cuando el módulo recibe una llamada en el serial se puede leer:

RING

+CLIP: "695240363",161,"",0,"",0

El módulo avisa con una cadena de texto que empieza con la palabra RING seguida de saltos de línea y otra parte de la cadena que contiene el número que ha llamado.

El código que se ha programado se mantiene a la espera de que haya información en el serial correspondiente, cuando detecta información la separa por cadenas buscando el elemento “\n” y comprueba si la primera de ellas es igual a la palabra RING. En caso afirmativo cuelga la llamada y realiza un Split (corte) de la segunda cadena para aislar el número de teléfono al que tendrá que responder con un SMS.

Una vez se ha obtenido el número de teléfono se escriben en el serial los comandos AT necesarios para enviar el SMS con un intervalo de tiempo de 1 segundo entre ellos y se introduce en el cuerpo del mensaje las últimas coordenadas obtenidas que siempre se van a encontrar en el archivo coordenadas.txt (véase figura 3.11)

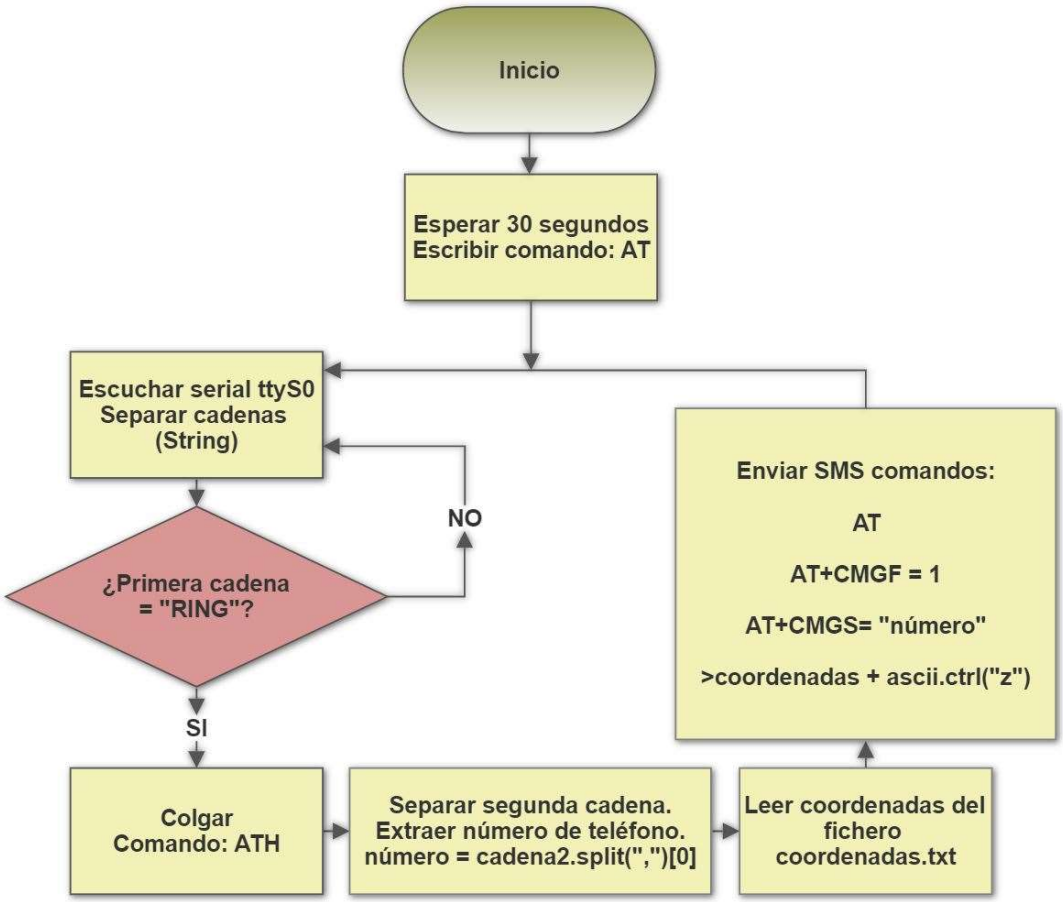


Figura 3.11 – Secuencia del programa UBICSMS.py

Como se puede ver en la figura 3.11 este bucle se ejecuta de manera continua, desde que se enciende el dispositivo empieza la comprobación y no se detiene hasta que se apaga UBIC. Es importante el hecho de determinar un periodo para el bucle, ya que de no ser así el programa realizaría el bucle a la máxima velocidad posible lo que provocaría un gran calentamiento de la CPU y un consumo de energía innecesario. Se ha fijado un periodo de 500 milisegundos para la ejecución del bucle para reducir este efecto sin que afecte a la detección de llamadas entrantes.

CAPÍTULO 4. APLICACIÓN ANDROID UBICAPP

4.1. SISTEMA OPERATIVO ANDROID Y LENGUAJE DE PROGRAMACIÓN

El sistema operativo Android, basado en un núcleo Linux, fue creado por la empresa Android Inc. con el respaldo económico de Google, que compró la empresa en 2005. El sistema operativo Android es el gran predominante en el mercado comparado don sus competidores directos: IOS y Windows.

Recientemente Android acaba de presentar su última versión, Android 7.0 Nougat. Sin embargo, el porcentaje de dispositivos con esta última versión de Android es muy pequeño con respecto el total, es por esto que al desarrollar una aplicación es aconsejable la compatibilidad con una o dos versiones anteriores de Android.

La gran mayoría de las aplicaciones de Android están escritas en Java. Este lenguaje de programación está orientado a objetos, es decir, los datos y código se combinan en objetos sobre los cuales se pueden realizar acciones, contienen información o métodos y pueden interaccionar entre ellos. El principal propósito por el que Java fue creado es el llamado WORA (en inglés: *write once, run anywhere*), lo que significa que puede ser ejecutado en cualquier dispositivo sin necesidad de recompilar el código.

4.2. ELEMENTOS DE UNA APLICACIÓN EN ANDROID

En el código de una aplicación Android se pueden distinguir varios elementos. Aquellos utilizados en la aplicación se definen a continuación.

El manifiesto de la aplicación es un archivo XML que debe estar presente en todas las aplicaciones con el nombre exacto de AndroidManifest.XML. Este archivo contiene la información esencial sobre la aplicación para el sistema operativo del dispositivo. En este archivo se especifica información como el nombre de la aplicación, el estilo, los servicios, recibidores de anuncios y permisos entre otras cosas.

La Activity es el componente principal de la interfaz gráfica, cada ventana de la aplicación es una actividad (activity). Está compuesta por dos archivos: Un archivo XML con el layout y los elementos con los que interactúa el usuario como texto, botones, imágenes, etc.. Por otra parte, el archivo Java, en el cual se encuentran las clases Java y se programa la funcionalidad de la aplicación. Desde el archivo Java también se puede gestionar los elementos del XML, cada uno de los cuales es un objeto en el archivo Java, por ejemplo, se puede ocultar y mostrar elementos o cambiar de color y tamaño el texto entre otras cosas.

Las vistas o views son los elementos básicos de la interfaz gráfica como texto, listas, botones, imágenes, etc.

Las vistas que se han utilizado en este proyecto son:

- TextView: un elemento que muestra información al usuario a través de texto.
- ImageView: un contenedor donde se puede mostrar una imagen.
- ImageButton: al igual que el ImageView permite mostrar una imagen y además permite realizar acciones con la pulsación de la misma.
- MapView: Permite visualizar los mapas de Google.

En esta aplicación se ha utilizado un Intent Service. Este elemento no dispone de interfaz gráfica, por lo que se ejecutan en segundo plano sin que el usuario vea nada. Se utilizan para actualizar datos, crear, leer o borrar archivos o lanzar notificaciones entre otros usos en los cuales el usuario no tiene que intervenir.

El Broadcast Receiver o recibidor de anuncios es un receptor que reacciona ante cierta actividad que no se puede predecir, como por ejemplo cuando se inserta una tarjeta de memoria SD, cuando se detecta el nivel de batería bajo o como en este caso, cuando se recibe un SMS.

Un Intent se utiliza para pasar información entre los diferentes elementos de la aplicación o incluso externos a esta. Por ejemplo para cambiar de actividad o realizar una llamada se debe utilizar un Intent.

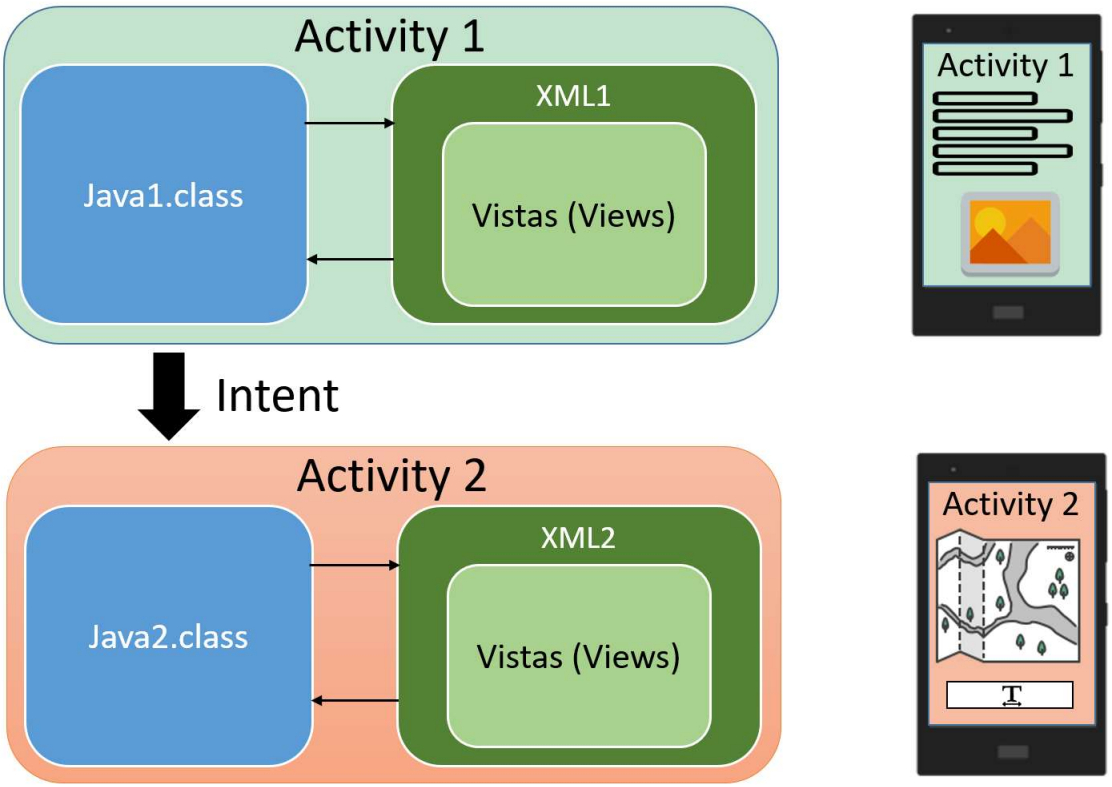


Figura 4.1 – Elementos de una actividad.

4.3. ENTORNO DE PROGRAMACIÓN

Existen varias plataformas para desarrollar aplicaciones Android en varios lenguajes de programación. Muchas de estas plataformas están dirigidas a desarrolladores principiantes facilitando la creación de las aplicaciones con lenguajes mucho más gráficos como VisualBasic sin necesidad de conocer Java.

Estas plataformas para programación sencilla aceleran mucho el trabajo pero también condicionan mucho la funcionalidad y apariencia de la aplicación.

Para la aplicación UBICAPP se ha utilizado el entorno de desarrollo integrado (IDE) oficial de Android, Android Studio. Este IDE fue lanzado en 2014 y su uso es completamente gratuito, se ha utilizado la versión Android Studio 2.3.1 ya que con la migración de la aplicación a la última versión 2.3.3 surgieron problemas que obligaron a rehacer parte del trabajo.

Android Studio dispone de varias actividades prediseñadas las cuales solo se deben modificar para el fin particular de cada aplicación, sin embargo esto solo se ha utilizado en el caso de la actividad Google Maps ya que son necesarias claves de Google para acceder a la API de Maps. Para el resto de actividades se ha partido de una “empty activity”, es decir, una actividad que solo incluye un texto: “Hello World” y en la que se ha de programar todo el código desde cero.

Este software permite el desarrollo de una misma aplicación para varios dispositivos con el sistema operativo Android. Separa por una parte los archivos de código Java comunes para los dispositivos y por otra los diferentes archivos XML que componen la interfaz de usuario pudiendo adaptar cada una a un dispositivo como por ejemplo relojes inteligentes, tabletas, automóviles o televisiones Android.

En la parte de los XML el programa permite dos formas de programación. Una mucho más visual en la que se arrastran elementos como widgets, botones, texto, etc., y otra en la que la programación del XML se realiza en texto. Pese a las continuas mejoras y actualizaciones, la opción más visual de programación aún es poco eficaz y para nada intuitiva, por lo que lo más recomendable es realizar la programación completa en texto o como alternativa, si no se tienen los conocimientos suficientes, combinar ambas.

4.4. DESARROLLO DE LA APLICACIÓN MÓVIL UBICAPP

Para el desarrollo de la aplicación se ha tenido en cuenta tanto la funcionalidad como la apariencia ya que es el único canal de comunicación del usuario con el sistema.

El funcionamiento de la aplicación ha variado desde la versión inicial hasta la actual debido a mejoras en la experiencia de usuario que se han ido implementando así como cambios en el funcionamiento del dispositivo UBIC.

Las primeras versiones de la aplicación necesitaban de la atención del usuario para realizar la actualización de coordenadas mediante un botón llamado Actualizar. Con la pulsación de este botón la aplicación actualizaba las coordenadas del dispositivo UBIC para su posterior

localización. Este es un método poco práctico y tedioso para el usuario, el cual puede olvidar realizarlo.

En versiones posteriores este proceso se automatizó mediante un Servicio, el cual se ejecuta de forma automática en segundo plano como se verá a continuación. Este método también tiene una pequeña desventaja, desde un servicio no se puede modificar la interfaz de usuario y por lo tanto no hay forma de hacer saber al usuario que la comunicación se está realizando de forma exitosa. Por otra parte, debido al gran uso que se le da a los teléfonos móviles hoy en día en los vehículos, tanto para escuchar música, realizar llamadas o navegar con el GPS, la opción del servicio permite usar todas estas funciones mientras UBICAPP funciona en segundo plano comunicándose con UBIC.

Otro gran cambio integrado en la última fase de desarrollo de la aplicación es la utilidad de localización a larga distancia mediante mensajes de texto. Esto se añadió pensando en un posible caso de sustracción del vehículo u objeto en el cual se encuentre el UBIC o en un posible fallo en la primera comunicación Wifi. La finalidad de esta mejora es que el usuario pueda tener la tranquilidad de saber dónde se encuentra el dispositivo en todo momento.

Iniciando el desarrollo de la aplicación en Android Studio se debe elegir en primer lugar el nombre del proyecto y ubicación del equipo donde se va a desarrollar, en este caso el nombre del proyecto es TFGAPP.

A continuación se deben elegir las plataformas para las que se quiere desarrollar: teléfono y tableta, reloj inteligente, Android televisión, Android Auto o Google Glass. En este caso se ha seleccionado tan solo la opción de teléfono y tableta. Dentro de esta selección se debe especificar la API a partir de la cual la aplicación será compatible. Debido a que el teléfono que se dispone para realizar las pruebas de la aplicación y su depuración dispone de Android 6.0 Marshmallow y el reciente lanzamiento de Android 7.0 Nougat, se ha decidido elegir la API 23 que corresponde a Android 6.0 ya que en un futuro próximo esta será la versión predominante en el mercado Android.

A continuación se elige el tipo de actividad con la que se creará el proyecto.

La actividad elegida para la pantalla principal (MainActivity en Android Studio) es una actividad vacía, tan solo contiene un TextView y la clase MainActivity básica con el método OnCreate(), el cual se ejecuta al lanzar la actividad.

Es sobre esta base que proporciona el programa sobre la que se empieza la programación de la aplicación.

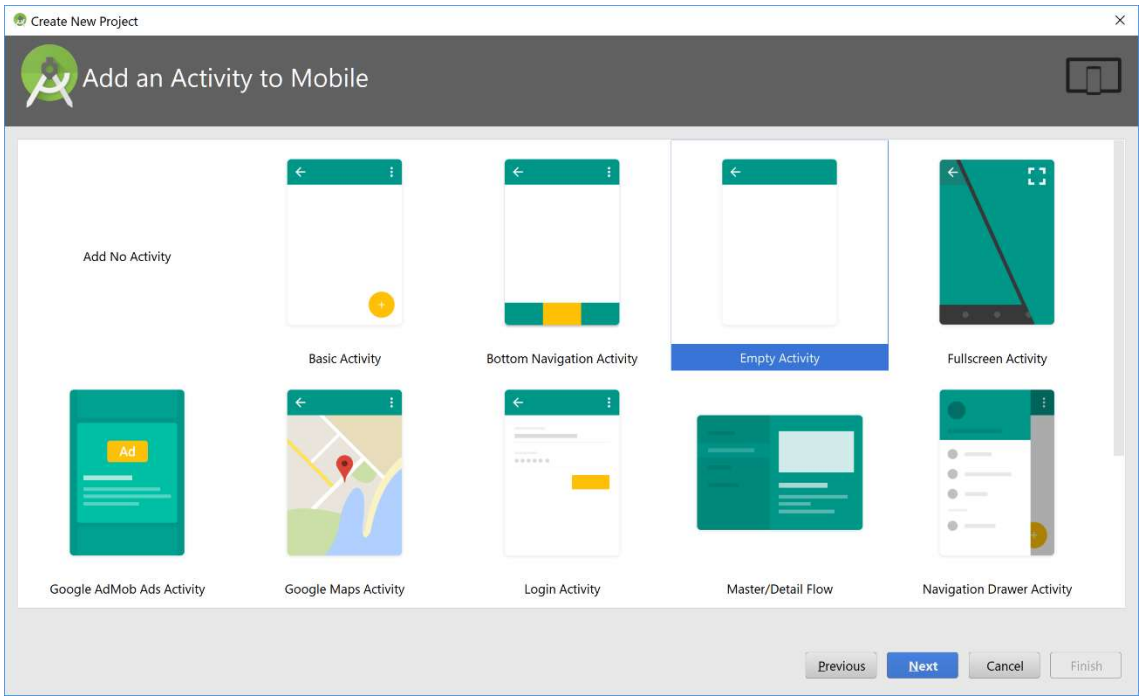


Figura 4.2 – Tipos de actividad, pantalla inicial selección de Android Studio.

4.4.1. Actividad principal

En la parte superior de la actividad principal se encuentra el logo de la aplicación, el cual se ha creado utilizando una versión de Photoshop Online gratuita, en la parte inferior del logo se encuentran 3 botones.

En primer lugar esta actividad comprueba si se han obtenido los permisos necesarios del usuario para realizar llamadas y recibir y leer SMS, de no ser así pregunta mediante una ventana emergente si concede estos permisos a la aplicación. Este paso solo ocurrirá la primera vez que se inicie la aplicación en un dispositivo ya que para los siguientes usos la aplicación ya cuenta con los permisos aceptados.

Cada elemento vista (View) dispone de una ID (identificación) a la cual se le hace referencia desde el archivo Java para controlar las acciones sobre este. Para utilizar un botón en el archivo Java de la actividad se crea un objeto tipo ImageButton y se refiere al ID del elemento View creado en el XML:

*AppCompatImageButton navegar = (AppCompatImageButton)findViewById(R.id.**navegar**);*

De esta forma se puede utilizar el método *.setOnClickListener()* dentro del cual se escribe el código que se requiere que se ejecute cuando se pulse el botón.

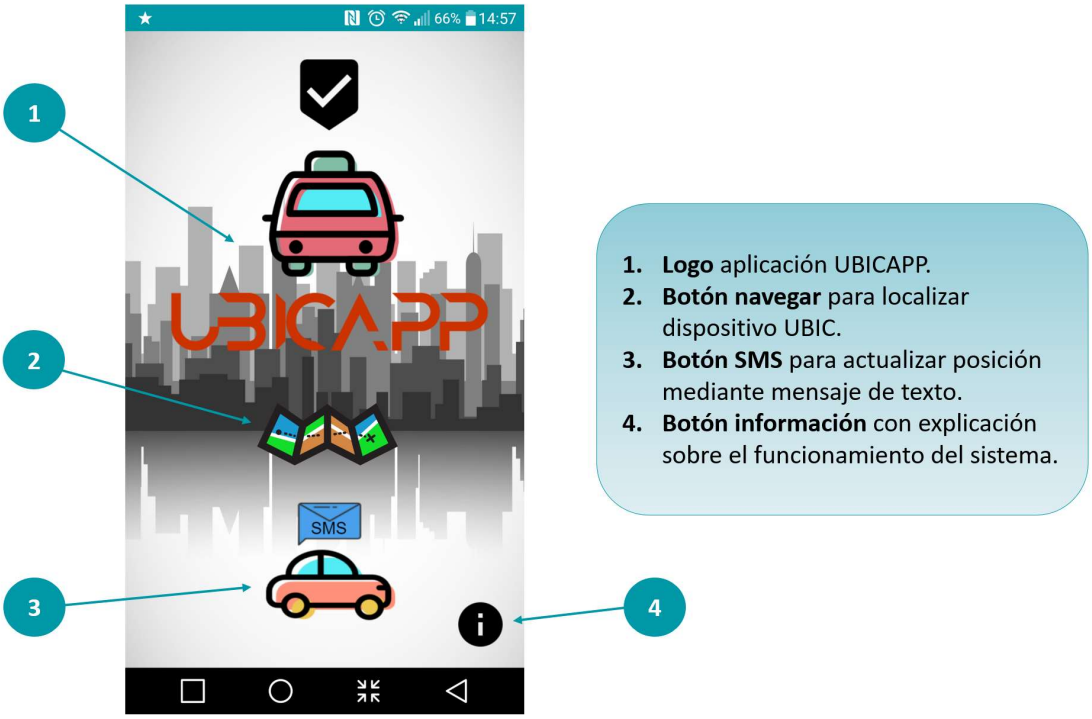


Figura 4.3 – Captura de pantalla de la actividad principal de la aplicación Ubicapp.

El primer botón (ID = navegar) es el botón principal para abrir el mapa y localizar el dispositivo UBIC. Este botón inicia la actividad MapsActivity.java mediante el uso de un Intent.

A continuación se encuentra el botón con el cual se realiza la localización del dispositivo UBIC cuando este esté fuera del alcance de la conexión Wifi (ID = sms). El funcionamiento de este botón es simple, mediante un Intent realiza una llamada de teléfono al número de teléfono asignado a la tarjeta SIM del dispositivo UBIC.

Por último, en la esquina inferior derecha de la pantalla, se encuentra el botón de información, el cual vuelve visible o invisible un objeto TextView donde se explica brevemente el funcionamiento de la aplicación.

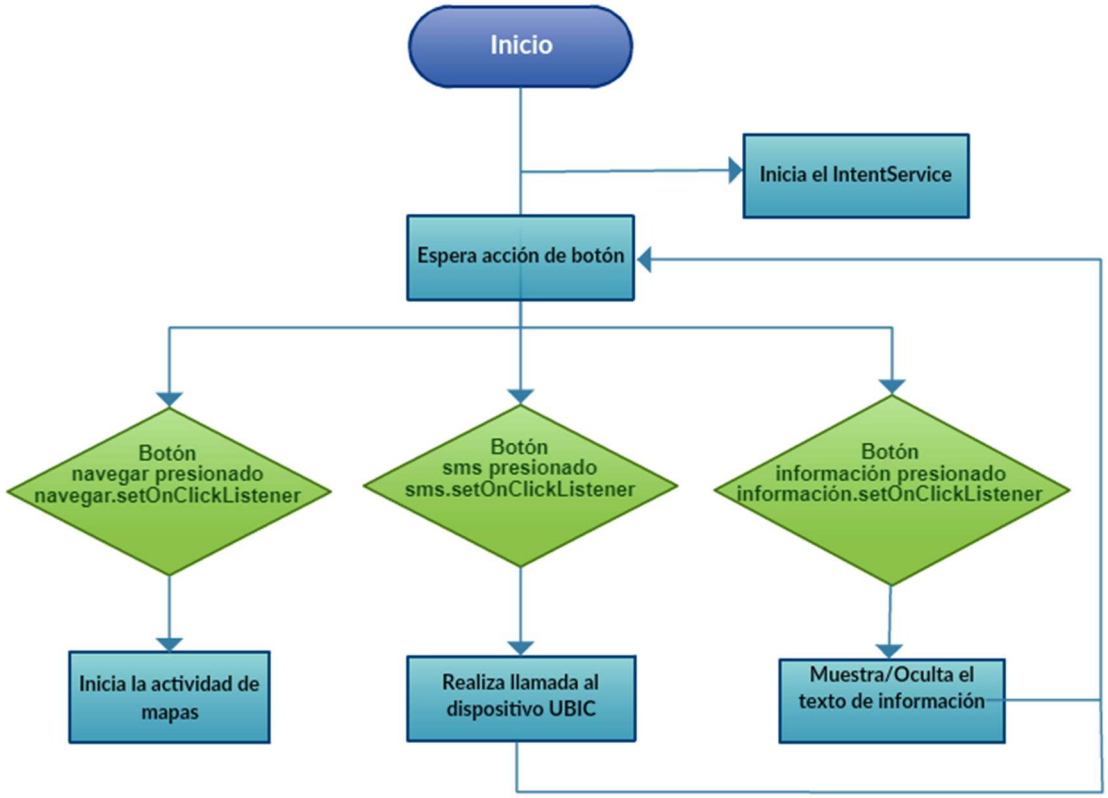


Figura 4.4 – Secuencia de la actividad principal.

4.4.2. IntentService (actualizar coordenadas)

Volviendo sobre lo que se ha hablado anteriormente, para automatizar la actualización de las coordenadas se ha utilizado un IntentService el cual es llamado al abrir la aplicación, desde la actividad principal.

El IntentService, llamado MiIntentService.java, se ejecuta y se mantiene en un bucle sin que el usuario perciba ningún cambio en la interfaz. El funcionamiento de este servicio es el siguiente:

Comprueba si el Wifi del dispositivo está encendido, de no ser así lo enciende y espera 5 segundos para que el Wifi se inicie correctamente y se conecte a alguna red próxima si le es posible.

Mediante un objeto WifiManager, con el cual se gestionan todas las funciones del Wifi, se comprueba la información de la red a la que se está conectado. Tras este punto existen dos posibilidades, que el móvil se haya conectado directamente al dispositivo UBIC si ya tenía la red guardada o que esté conectado a una red diferente, también existe la posibilidad que no esté conectado a ninguna red Wifi pero para este caso equivale a la segunda opción. Si la SSID (identificación de la red Wifi) de la red conectada es distinta de “UBIC” se realiza la desconexión de la actual y se intenta conectar a “UBIC”.

Se comprueba si se ha conectado correctamente:

Si no se ha conectado: se considera que el dispositivo está apagado o fuera de rango y el servicio espera 5 minutos y vuelve a intentarlo mientras no se cierre la aplicación desde el gestor de tareas.

Si se ha conectado: se empiezan a realizar intentos de recepción de los sockets enviados por el dispositivo UBIC y la información de estos se guarda en un archivo de texto llamado ultimaposicion.txt. Este archivo es reescrito con cada Socket recibido manteniendo actualizada la última posición recibida.

Se ha decidido guardar las coordenadas en un archivo para tenerlas disponibles en todo momento desde cualquier Actividad, cuando se necesita guardar o leer las coordenadas se puede hacer sin importar la actividad o servicio que se esté ejecutando, simplemente se obtiene de este archivo común.

Como se puede ver en la figura 4.5 mientras la aplicación está abierta sigue buscando el dispositivo UBIC cada 5 minutos, se especifica al usuario en el texto de información que cierre la aplicación (no minimizarla) si no se tiene pensado un uso del sistema en un periodo de tiempo corto, de esta manera se ahorra batería y recursos del sistema.

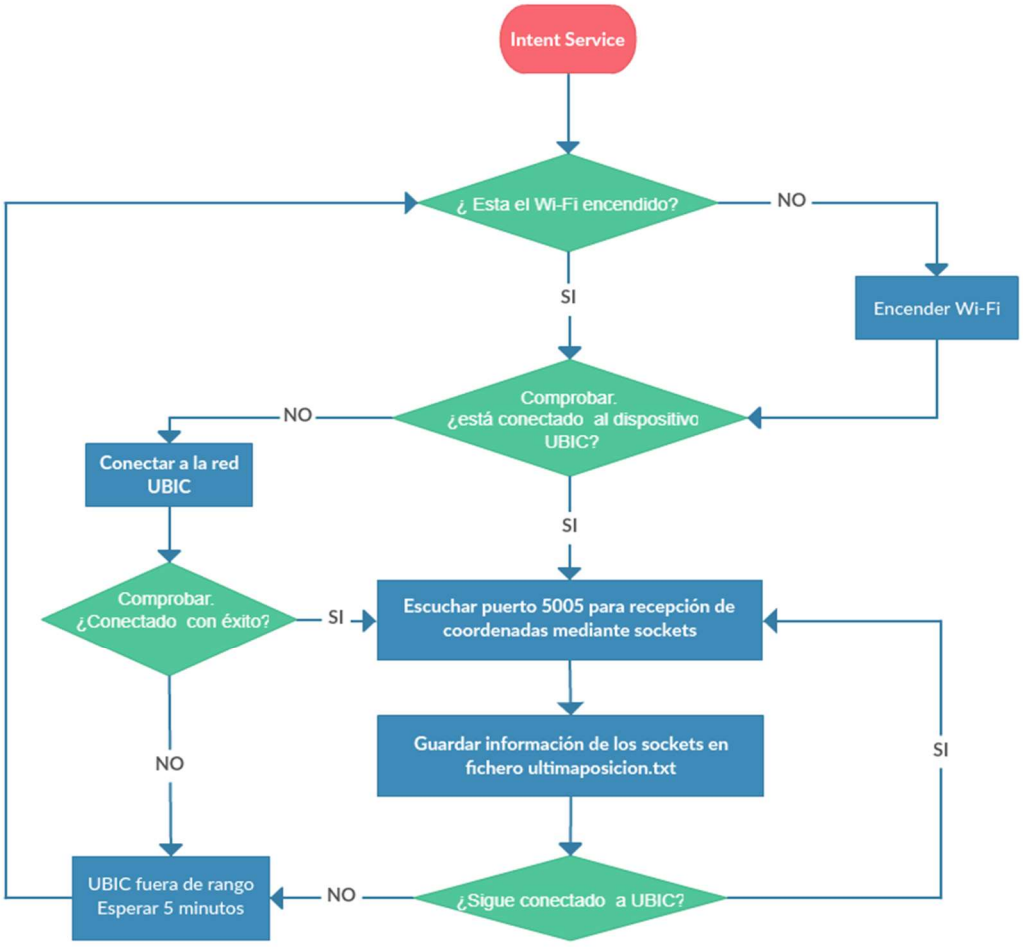


Figura 4.5 – Secuencia del IntentService.

Las últimas versiones de Android son capaces de detener todos los servicios que no se están utilizando de forma automática con el fin de mejorar el rendimiento y autonomía del teléfono, de esta forma no es un inconveniente si el usuario no recuerda detener la aplicación tras su uso.

4.4.3. Actividad de mapas

Esta actividad se crea desde Android Studio como una MapsActivity predeterminada sobre la cual se realizan las modificaciones para adaptarla al uso que se le va a dar en el sistema.

Para la utilización de los mapas de Google es necesario obtener una clave que permita el acceso a estos, para obtenerla se requiere de una cuenta en Google para acceder al enlace proporcionado por Android Studio tras crear la actividad de mapas. En la web se debe crear un proyecto y obtener una clave de API con restricción de Android, se copia la clave y se pega en el AndroidManifest.xml de Android Studio lo que permite tener acceso a los mapas de Google a través de la aplicación.

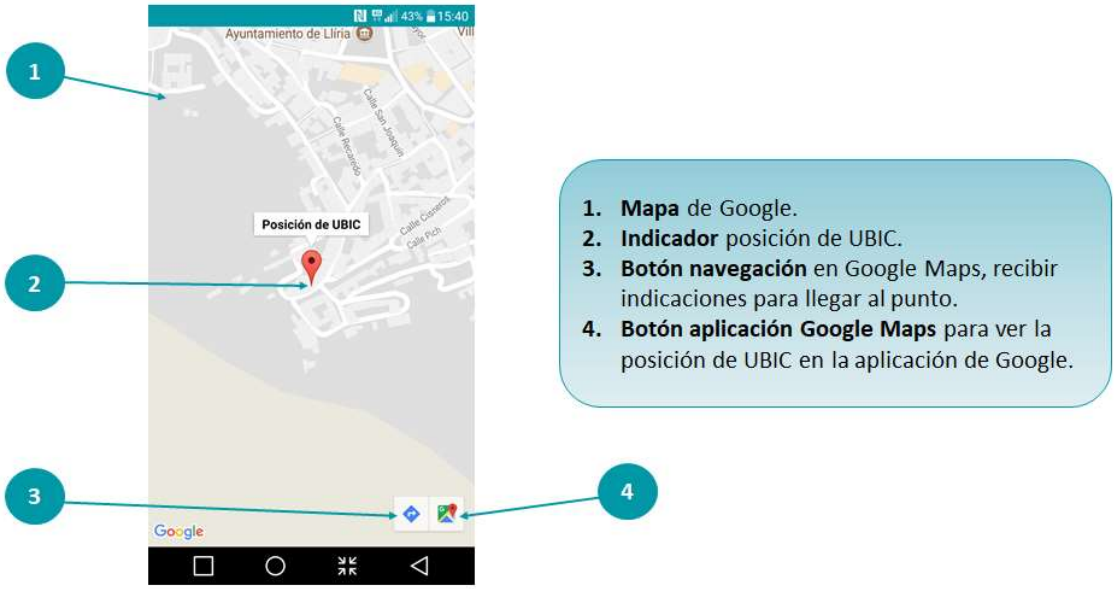


Figura 4.6 – Captura de pantalla de la actividad mapas de la aplicación Ubicapp.

Al lanzarse la actividad, mientras se carga el mapa, se abre el archivo ultimascoordenadas.txt y se guarda su contenido en una variable. La información viene en el formato "latitud|longitud|", por lo que mediante un .split("\\|",3) se fragmenta esta cadena en 3 partes: latitud, longitud y una vacía.

A continuación se crea un controlador para el mapa, igual que para controlar el Wifi se necesita un objeto WifiManager para el uso del mapa es necesario un objeto tipo GoogleMap con el cual se crea un marcador latitud-longitud usando las coordenadas. Solo resta fijar el zoom del mapa para ver un rango de calles aceptable para la ubicación del usuario (tras varias pruebas se ha fijado un zoom de 17 puntos) y centrar la cámara en el marcador (cameraUpdate).

Con esta actividad se puede ver la posición en la que se encuentra el dispositivo UBIC, si se presiona sobre el indicador aparecen en la esquina inferior derecha dos iconos para realizar la navegación hasta el punto o abrir la localización, ambos en la aplicación de Google Maps.

4.4.4. MessageReceiver (Broadcast Receiver)

MessageReceiver.java es el nombre que se le ha dado al Broadcast Receiver encargado de administrar la información recibida del dispositivo UBIC mediante mensajes de texto. Un BroadcastReceiver se utiliza para ejecutar cierto código ante el acontecimiento de alguna acción, en este caso cuando se recibe un SMS.

Al darse esta acción se ejecuta el método onReceive() del Broadcast Receiver donde se ha programado el código correspondiente.

Figura 4.7 – Secuencia Message Receiver.

Este proceso se ejecuta de principio a fin cada vez que se recibe un SMS al contrario de las otras actividades y servicios de los que se ha hablado, los cuales se ejecutan en bucle. El código comprueba si el SMS recibido proviene del número asignado al dispositivo UBIC, de ser así lee su contenido y lo guarda en el fichero ultimascoordenadas.txt. Tras guardarlo, el Broadcast Receiver lanza la actividad de mapas y termina, dejando en pantalla el mapa con la ubicación recibida por SMS.

En el caso que el número no coincida con el de UBIC el proceso termina sin hacer nada hasta que se vuelva a iniciar por la recepción de otro mensaje de texto.

En ningún momento se puede ver ninguna interfaz que muestre lo que está ocurriendo. Desde el punto de vista del usuario solo se ve la llamada realizada tras presionar el botón SMS de la actividad principal, la desconexión de la llamada y en un período de 5 a 10 segundos, si ambas partes tienen buena señal, se abre automáticamente el mapa marcando la posición del dispositivo UBIC.

En el caso de que el teléfono no tenga cobertura no será posible realizar la llamada por lo que el Message Receiver no se ejecutará, por el contrario, si es el dispositivo UBIC el que no dispone de cobertura, no podrá recibir la llamada por lo que el teléfono avisará al usuario que el dispositivo al que llama está apagado o fuera de cobertura.

4.4.5. Manifiesto de la aplicación (Android Manifest)

Con todas estas actividades y funciones de la aplicación se debe modificar su manifiesto para que funcione correctamente en el sistema operativo Android.

En primer lugar se deben declarar los permisos para poder utilizar ciertas funciones del teléfono, todo aquello que no depende de la propia aplicación pero que se va a hacer uso de ello.

Los permisos usados son:

- ACCESS_WIFI_STATE: el cual da permiso a la aplicación para comprobar el estado del Wifi, si esta encendido, conectado, a que red está conectado, intensidad de la señal…
- CHANGE_WIFI_STATE: con este permiso la aplicación es capaz de realizar acciones sobre el Wifi del dispositivo como encender/apagar o conectar a una red entre otras. Este permiso suele ir siempre acompañado del anterior ya que para realizar algún cambio primero se debe saber en el estado en el que se encuentra.
- INTERNET: este permiso habilita a la aplicación al uso de la conexión a internet del teléfono móvil. Aunque esta aplicación no utiliza la conexión a internet sí que utiliza la comunicación mediante Wifi en una red local.
- ACCESS_FINE_LOCATION: es necesaria para utilizar la localización GPS del teléfono móvil.
- CALL_PHONE: este permiso necesita de la aprobación del usuario tanto al instalar la aplicación como los casos anteriores y al abrir la aplicación por primera vez. Al usuario le aparece una ventana pidiendo permiso para utilizar el teléfono para realizar llamadas.

- RECEIVE_SMS: para utilizar este servicio se requiere la misma aprobación del usuario que en el anterior. Una vez aceptado la aplicación tendrá permitido recibir avisos cuando se reciban mensajes de texto.
- READ_SMS: comparándolo con el anterior y como su propio nombre indica este permiso es necesario para saber la información que contienen los mensajes, tanto el número del que provienen como el cuerpo del mensaje.

A parte de los permisos de la aplicación, en el manifiesto también viene la información básica como la referencia de la imagen que se utiliza como icono, que actividad es la principal y cual se debe lanzar cada vez que se inicie la aplicación.

Es en este archivo donde también se especifica la clave obtenida para el uso de los mapas de Google. Además se debe indicar el servicio que se ejecuta en segundo plano así como el recibidor de anuncios que detecta los SMS recibidos del dispositivo UBIC y a este último darle una prioridad alta para que reciba los mensajes antes que la aplicación nativa del teléfono.

CAPÍTULO 5. RESULTADOS

5.1. PRECISIÓN DEL SISTEMA DE POSICIONAMIENTO

Con el fin de cuantificar la precisión de la localización del sistema de posicionamiento se ha decidido realizar varias mediciones y tomas de datos en distintas condiciones.

Para ello se compararán mediciones después de un arranque y tras un periodo de tiempo X después de este arranque en un posicionamiento estacionario.

Se realiza la medición del tiempo que tarda el receptor en realizar la conexión y la dispersión de las mediciones en una posición descubierta con visión directa al cielo y en interior para comprobar el efecto en la precisión del sistema con el fin de comprobar si el sistema se puede utilizar tanto en interiores como en exteriores.

Para realizar la prueba se genera un programa en Python el cual va contando las mediciones y guardándolas en un archivo de texto el cual exportaremos más tarde para su estudio en una hoja de cálculo Excel.

Suponiendo el uso del sistema para la localización de un vehículo estacionado se ha realizado la estima de unos 5 minutos desde que se encuentra en el punto hasta que la aplicación sale fuera del rango de conexión del dispositivo. Siguiendo esta suposición y que el receptor realiza la adquisición de coordenadas en periodos de 1 segundo aproximadamente equivale a 300 mediciones.

Como coordenadas de referencia se han tomado las proporcionadas por el servicio de mapas de Google Maps donde se ha introducido la dirección Calle Pich, 40, 46160 Liria, Valencia, España, desde la que se realizan las pruebas y equivale a las coordenadas (latitud = 39,62285 longitud= = -0,59582).

En primer lugar se realiza la medición dentro de un edificio, en una habitación cerrada con una ventana de 1,15x1,15 metros. En este caso el dispositivo consigue realizar la comunicación con los satélites en un tiempo de arranque en frío de 2 minutos y 10 segundos situado a una distancia de 1,5 metros de la ventana.

Tras el arranque en frío se ejecuta el programa pruebaprecision.py el cual realiza la adquisición de las 300 coordenadas y las guarda en un fichero de texto.

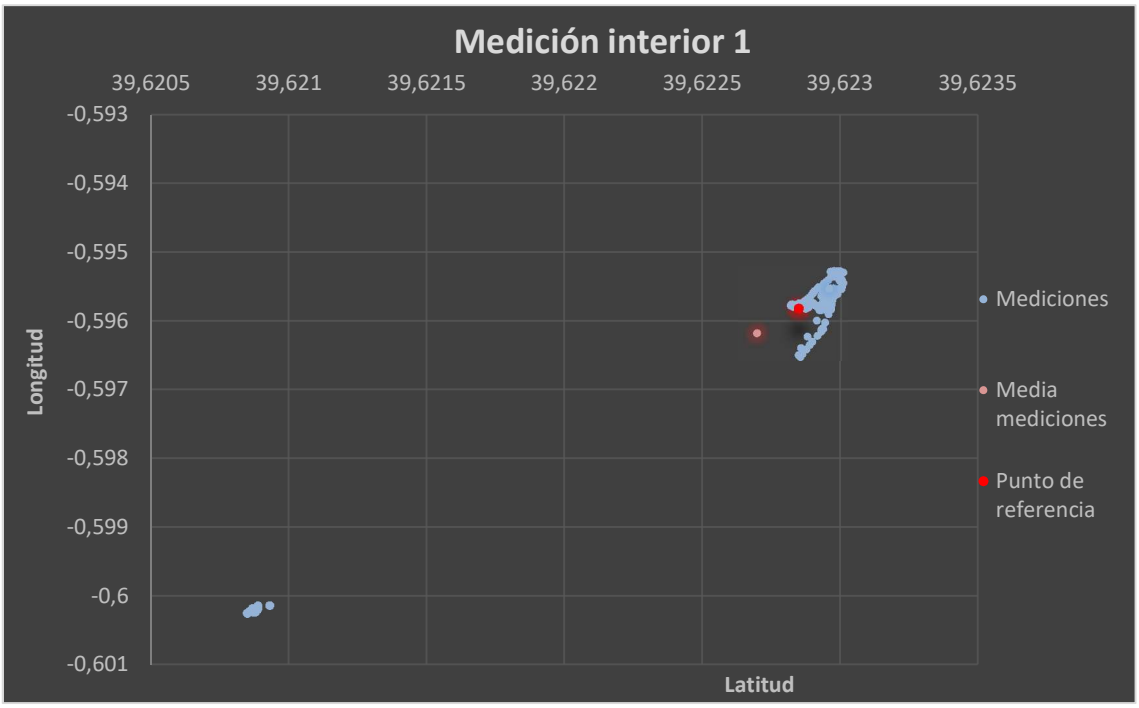


Figura 5.1 – Representación de las mediciones, el promedio de las mediciones y el punto de referencia en un gráfico de dispersión. Primera medición en interior.

Se realiza la representación de los datos y de su media aritmética y se comprueba el error en metros entre la media de los datos y las coordenadas obtenidas anteriormente.

En la figura 5.1 se pueden ver las coordenadas de la primera medición con su media aritmética y las coordenadas de referencia.

Se puede observar bastante dispersión en las coordenadas obtenidas, esto puede ser debido a la dificultad de recibir las coordenadas por la existencia de barreras físicas que influyen en la trayectoria de la señal. Los 30 primeros datos contienen un gran error debido a que la conexión acaba de establecerse, tras un periodo de tiempo de 30 segundos aproximadamente se produce una importante corrección y los puntos se encuentran mucho más cerca de la posición de referencia.

La diferencia entre la posición que ocupa la media de las mediciones y el punto de referencia que se aprecia en el gráfico 1 equivale a 36.5 metros de error, un error muy elevado debido a las primeras mediciones. Este error lo podemos ver sobre el mapa en la figura 5.2.

Figura 5.2 - Distancia entre el punto de referencia y el valor promedio de la primera toma de datos en interior.

Tras una hora con el dispositivo encendido y en posicionamiento estático (sin cambiarlo de ubicación) se ha realizado la siguiente medición. En este caso el dispositivo ha tenido tiempo de conectarse con más satélites e ir descargando información desde estos.

En la figura 5.3 se puede observar una mayor concentración de los puntos obtenidos de forma que el valor promedio de estas mediciones está más cerca del punto de referencia. A pesar de esto sigue habiendo un error en el posicionamiento como se puede apreciar en el gráfico.

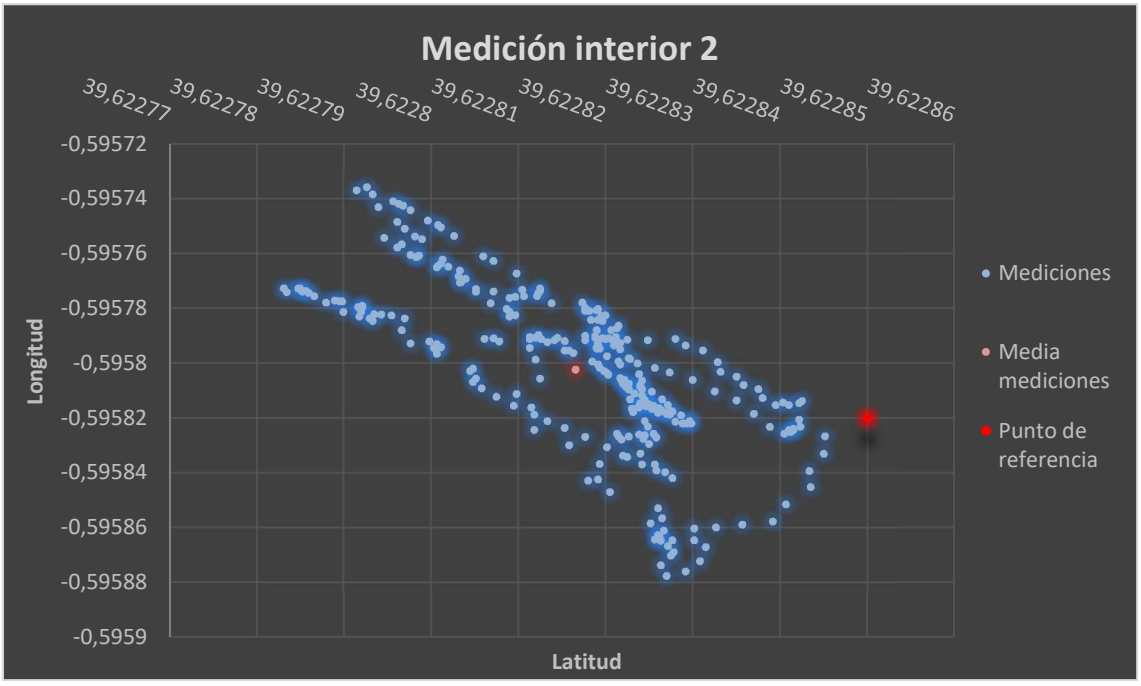


Figura 5.3 – Representación de las mediciones, el promedio de las mediciones y el punto de referencia en gráfico de dispersión. Segunda medición en interior.

La distancia entre el punto de referencia y la media de las mediciones que se distingue en el gráfico equivale a una distancia real de 4 metros que se puede observar en la figura 5.4.

Figura 5.4 – Distancia entre el punto de referencia y el valor promedio de la segunda toma de datos en interior.

Este error sigue siendo algo elevado, aunque no hay que olvidar que el receptor se encuentra en el interior de un edificio donde la dificultad de conexión con la constelación GPS es mucho mayor.

Tras concluir la toma de datos en interior se procede a realizar una segunda etapa de toma de datos en las mismas condiciones anteriores pero en una posición exterior con visión directa del cielo. Para reproducir las mismas condiciones que en la primera prueba es necesario realizar un arranque en frío del localizador y como se ha explicado anteriormente es necesario que este permanezca sin conexión o apagado más de 4 horas.

Tras esperar 4,5 horas en apagado se realiza de nuevo un arranque en frío y se realiza la segunda etapa de tomas de datos en el exterior para determinar si hay una mejora de precisión considerable.

El inicio en frío en esta posición se realizó en un tiempo de 41 segundos, un tiempo mucho menor que en la prueba de interior debido a que la antena tiene mejor visión de los satélites sin ningún tipo de obstáculo.

Tras el arranque en frío en el exterior se han obtenido los datos representados en el gráfico de la figura 5.5 en el que se puede apreciar el inicio de la medición con los datos de la zona izquierda del gráfico y como éstas coordenadas se van acercando al punto de referencia.

Aun tratándose de un arranque en frío no se aprecian puntos tan erróneos como en el primer arranque en frío en interior, esto es debido a la mayor facilidad y rapidez de conexión de dispositivo.

Figura 5.5 – Representación de las mediciones, el promedio de las mediciones y el punto de referencia en gráfico de dispersión. Primera medición en exterior.

La distancia entre el promedio de las mediciones y el punto de referencia en este caso es de tan solo 1,4 metros que como se puede apreciar en la figura 5.6 es una precisión muy buena.

Con estos resultados se aprecia la gran mejora en el posicionamiento al eliminar obstáculos que incluyen el error por bloqueo de señal.

Figura 5.6 – Distancia entre el punto de referencia y el valor promedio de la primera toma de datos en exterior.

En este caso, tras una hora de posicionamiento estático el error obtenido se ha mantenido estable con 1,3 metros entre el promedio de las mediciones y el punto de referencia.

Puesto que la principal función del sistema es la localización de vehículos, un error de 3 a 4 metros en interior no es ningún inconveniente, por lo que se puede utilizar perfectamente en interiores.

En el caso de aparcamientos subterráneos la aplicación guiará al usuario hasta la entrada, el último punto en el que el dispositivo haya podido recibir señal de los satélites.

Es a destacar el hecho de haber obtenido resultados con menos de la mitad de error en exteriores con poco más de un metro. Tras esto queda probar el sistema de localización con la aplicación y un vehículo para comprobar su efectividad.

5.2. PERIODO DE PRUEBA DEL SISTEMA

Con el fin de comprobar la fluidez del sistema y para encontrar fallos, ya sean electrónicos o de programación, se ha testado el sistema en un vehículo durante 3 días. Cada vez que se ha utilizado el vehículo se ha realizado una prueba del sistema tanto para realizar la navegación hasta el punto donde se había estacionado como para comprobar la posición mediante SMS.

En este periodo de prueba se ha encontrado y corregido problemas tanto de hardware como de software.

El principal fallo de hardware detectado ha sido una degradación del cable de alimentación del módulo GSM debido al movimiento de los elementos electrónicos fuera de la carcasa. Este fallo provocaba el constante apagado y fallo del módulo haciendo imposible la comunicación de la placa Raspberry con este. La solución ha sido retirar el tramo de cable soldado entre el interruptor y los diodos y sustituirlo por un tramo nuevo, este cable se ha duplicado para evitar este fallo en el futuro. Para mejorar esta solución se ha añadido una funda retráctil por calor para la protección de esta zona.

Durante una de las pruebas se produjo un error en el cual la aplicación había guardado como última posición un punto intermedio del trayecto y no el último punto en el que se había realizado el posicionamiento GPS. Esto se produjo porque el dispositivo UBIC perdió la señal con los GPS y por un error de código dejó de ejecutar el programa responsable del envío de las coordenadas por Wifi.

Otro factor detectado es el calentamiento del procesador de la Raspberry Pi 3, esto no ha sido un problema durante los test o el funcionamiento en lugares al aire libre o en interiores donde la temperatura del procesador ha llegado a subir como mucho a los 60ºC, una temperatura normal. Sin embargo en días más calurosos y testando el dispositivo en un vehículo aparcado a la intemperie si se han producido calentamiento excesivos debido a la dificultad para disipar el calor.

Como solución se ha pensado en incluir un ventilador de 3.3V integrado en la carcasa junto a unos disipadores de aluminio en el procesador, no obstante esta solución no solo aumenta el consumo energético del dispositivo si no que genera ruido debido al aire movido por el ventilador. La opción implantada finalmente ha sido lo que se conoce como “underclock”, esto consiste en limitar la velocidad de procesado para que nunca llegue a si velocidad máxima, de

este modo se consigue una generación de calor menor y por consiguiente un menor aumento de temperatura, además se reduce el consumo energético de la tarjeta. A esta solución también se le han añadido el disipador en la CPU mejorando la disipación pasiva por aumento de área de transmisión.

En cuanto a la aplicación Android se ha detectado un fallo que cierra la aplicación si el usuario intenta abrir el mapa con la aplicación recién instalada sin haber conectado un dispositivo UBIC primero. Esto es debido a que el sistema intenta abrir un archivo que no existe y asignarle a un marcador Google Maps una posición que no tiene. La solución es dejar la aplicación unos segundos para que pueda sincronizar alguna posición del dispositivo UBIC y de esta forma ya se puede abrir el mapa o utilizar la opción de localización mediante SMS.

Por todo lo demás tanto la aplicación Android como el dispositivo GPS han funcionado a la perfección. Una vez encendido el dispositivo tarda entre 30 segundos a 1 minuto en empezar a enviar los datos de la posición, la aplicación permite el uso de otras aplicaciones mientras funciona en segundo plano comunicándose con el dispositivo. La función de localización mediante SMS no ha fallado ninguna vez durante la prueba, recibiendo siempre la posición correcta del dispositivo.

CAPÍTULO 6. CONCLUSIONES

Después de la explicación del funcionamiento del sistema GPS, su aplicación en el sistema con un receptor, el ensamblaje, conexión y programación del dispositivo de posicionamiento GPS UBIC y el diseño y desarrollo de la aplicación móvil para Android UBICAPP, se presentan las siguientes conclusiones del trabajo de fin de grado.

En cuanto al dispositivo GPS UBIC:

- Se ha diseñado un dispositivo capaz de realizar la geolocalización y gestionar esta información de la forma que se ha considerado más adecuada para garantizar su correcto funcionamiento.
- De los diversos canales de comunicación con los que gestionar el envío de datos hasta la aplicación se ha optado por el uso de una red Wifi propia con un alcance mejorado respecto a otras opciones.
- Se ha logrado la correcta integración de todos los elementos electrónicos en un solo producto en el que se ha cuidado tanto la funcionalidad como la estética.
- Con la inclusión de la batería se ha conseguido una autonomía elevada del dispositivo resultado útil para otros usos a parte de la localización de vehículos, como pueda ser el control de objetos personales como mochilas o maletas.
- Relacionado con el punto anterior, cabe destacar la mejora añadida posteriormente de la funcionalidad de comunicación a distancia que permite al usuario saber dónde se encuentra el dispositivo y como llegar hasta él incluso estando fuera del rango de conexión de este.

Respecto a la aplicación Android UBICAPP:

- Tras horas de investigación y programación se ha conseguido desarrollar una aplicación de estética cuidada y con un funcionamiento correcto testado en varios teléfonos móviles.
- Se ha tenido como principal directriz durante el desarrollo de esta mantener una sencillez y facilidad elevada para minimizar de esta forma la curva de aprendizaje del usuario y conseguir que el usuario utilice la aplicación a menudo, evitando tiempos largos en su uso que resulten tediosos.
- Después de las primeras pruebas y los numerosos fallos encontrados y resueltos, incluso con la aplicación funcionando correctamente, esta ha sufrido constantes cambios para mejorar la experiencia de usuario, optimizar procesos y depurar el código al máximo.
- Se ha reparado en la gran cantidad de información obsoleta existente referente a la programación de aplicaciones y la dificultad de encontrar información actualizada referente a las últimas versiones de Android en cuando a desarrollo se refiere.

Referente al sistema completo formado por el dispositivo UBIC y la aplicación UBICAPP:

- Se ha logrado desarrollar un producto 100% funcional que le ofrece un servicio de localización de vehículos u objetos al usuario y que es de fácil utilización y comprensión.
- Tras la comparativa de otros localizadores se ha conseguido realizar un sistema de comunicación entre dispositivo y aplicación basado en Wifi que ofrece mayor rango de comunicación que la mayoría de conexión Bluetooth de los localizadores encontrados en el mercado. También cabe destacar la gran cantidad de trabajo que ha supuesto el desarrollo de este sistema de comunicación hasta conseguir su funcionamiento.
- Se han obtenido resultados satisfactorios del producto. Cumple todas las características iniciales para las que se había proyectado y además tiene gran potencial para ampliar su funcionalidad con más prestaciones a implementar sobre la base ya creada.

CAPÍTULO 7. TRABAJO FUTURO

El producto desarrollado en este trabajo podría considerarse un prototipo sobre el cual se pueden realizar varias mejoras y ampliaciones con el tiempo y recursos pertinentes como las que se describen a continuación.

7.1. REDUCCIÓN DE TAMAÑO DEL DISPOSITIVO

Una de las mejoras consiste en modificar el hardware para conseguir un dispositivo más compacto, portable y menos pesado.

Si bien en este dispositivo se ha utilizado una Raspberry Pi 3 como núcleo, por una parte esto ha sido debido a la facilidad de esta placa para realizar las conexiones y pruebas con los periféricos a través de los cables para prototipado de rápida conexión, por otra parte se ha utilizado esta placa porque ya se disponía de ella para su uso inmediato sin necesidad de adquirirla en una tienda.

Sin embargo, en febrero de 2017 la fundación Raspberry Pi lanzó al mercado una nueva tarjeta llamada Raspberry Pi Zero W. Esta tarjeta posee todas las características de la Raspberry Pi 3 con algo menos de potencia y un tamaño mucho más pequeño.

Figura 7.1 – Raspberry Pi Zero W. Imagen de www.raspberrypi.org/products/raspberry-pi-zero-w/

En comparación con la Raspberry Pi 3 esta tarjeta tiene menor potencia de procesamiento así como menor velocidad de reloj, también dispone de la mitad de memoria RAM.

Estos datos a simple vista pueden parecer jugar en su contra, pero nada más lejos de la realidad, el hecho de sufrir esta reducción de prestaciones debida a su reducción de tamaño y acompañada por una notable reducción de precio, también le supone una reducción en el consumo de energía, lo cual es algo a destacar cuando en los dispositivos portátiles gran parte del peso es debido a las baterías, este menor consumo también se traduce en una menor generación de calor evitando sobrecalentamientos.

Todo el software y hardware desarrollado para la Raspberry Pi 3 es completamente compatible con la Raspberry Pi Zero W con la ventaja de la reducción de tamaño correspondiente, de 85 x 54 mm de la versión utilizada a 65 x 30 mm de la versión Zero. Además este cambio viene junto a una posible reducción de la batería debido al descenso del consumo, pudiendo reducir la capacidad de esta sin afectar a la autonomía del dispositivo.

Ambos cambios causarían una reducción sustancial en las dimensiones y peso del dispositivo al ser ambos los componentes más grandes que se encuentran en su interior actualmente.

Tampoco se descarta el paso de microprocesador a microcontrolador si en un futuro se dispone de alguna marca con las dimensiones bastante acotadas y las conectividades necesarias. Para esto sería necesario analizar si estos microcontroladores serían capaces de desempeñar todas las funciones que realiza actualmente el microprocesador y sopesar si el ahorro de consumo energético es lo bastante importante como para adaptar todo el software.

7.2. IMPLEMENTACIÓN DE UN MODO ALARMA

Una funcionalidad interesante que se puede añadir por software es la opción de poder activar un modo alarma ya sea desde la propia aplicación Android o desde un botón situado en el propio dispositivo.

Esta función se ha pensado para activarla una vez el usuario deje el localizador en una posición fija y se aleje de él, por ejemplo un vehículo. Con este sistema el usuario recibirá en su teléfono móvil una alerta si la posición del localizador cambia.

Esto se puede implementar con un programa que se ejecute al activar el modo alarma en el dispositivo, el funcionamiento de este sería comparar las coordenadas iniciales guardadas al activar el modo alarma con cada actualización de coordenadas. Como se ha comprobado en el apartado 5.1 con las pruebas de precisión, estando en un punto fijo se han obtenido dispersiones de varios metros por lo que se le debe dar un margen para que el dispositivo no de falsas alarmas.

Una solución sería implementar el algoritmo con el que el programa calcule la distancia entre las dos coordenadas, si esta distancia es mayor a 10 metros durante 3 mediciones consecutivas el dispositivo UBIC manda un SMS al usuario con la información correspondiente de posición e incluso puede ser interesante el envío de la velocidad a la que se está desplazando.

Debido a que en la gran mayoría de los casos el dispositivo va a estar en la misma posición no es necesario realizar esta comprobación constantemente o con un periodo de tiempo muy corto como pueda ser 0,5 o 1 segundos, aumentando el tiempo entre comprobaciones mientras estas estén dentro del rango permitido se conseguirá una menor carga de procesamiento y un ahorro de energía. Un periodo válido podría ser entre 1 y 10 minutos, dando la opción al usuario de elegir dependiendo de cuánto tiempo vaya a estar el dispositivo en funcionamiento.

7.3. GUARDAR TRAYECTO

Ya sea como complemento al modo alarma descrito anteriormente o como función independiente, se ha estudiado la posibilidad de que el dispositivo guarde las posiciones durante un trayecto para que el usuario pueda ver la ruta por la que se ha movido el dispositivo.

En resumen esta función sirve para saber los desplazamientos que ha realizado un vehículo o persona en un mapa o para guardar la ruta de un desplazamiento o viaje con el fin de repetirla o usarla como referencia.

Esta función combinada con el modo alarma puede dar información precisa al usuario de donde se encuentra el dispositivo, que trayecto ha realizado y en qué dirección avanza.

Para comprobar hasta qué punto se pueden conseguir buenos resultados con la mejora de esta aplicación se ha realizado una prueba con mediciones reales.

Se ha realizado un trayecto en un vehículo mientras el dispositivo graba las posiciones, de esta forma se evaluara la precisión y capacidad del dispositivo de realizar este trabajo y si la velocidad a la que se desplaza puede llegar a causar problemas en el posicionamiento.

Se ha realizado una ruta que incluye calles estrechas de población, tramos de carretera y autovía, durante todo el trayecto se han ido registrando la posición y velocidad del vehículo.

En total se han registrado 1220 posiciones representadas en el gráfico de la figura 7.2.

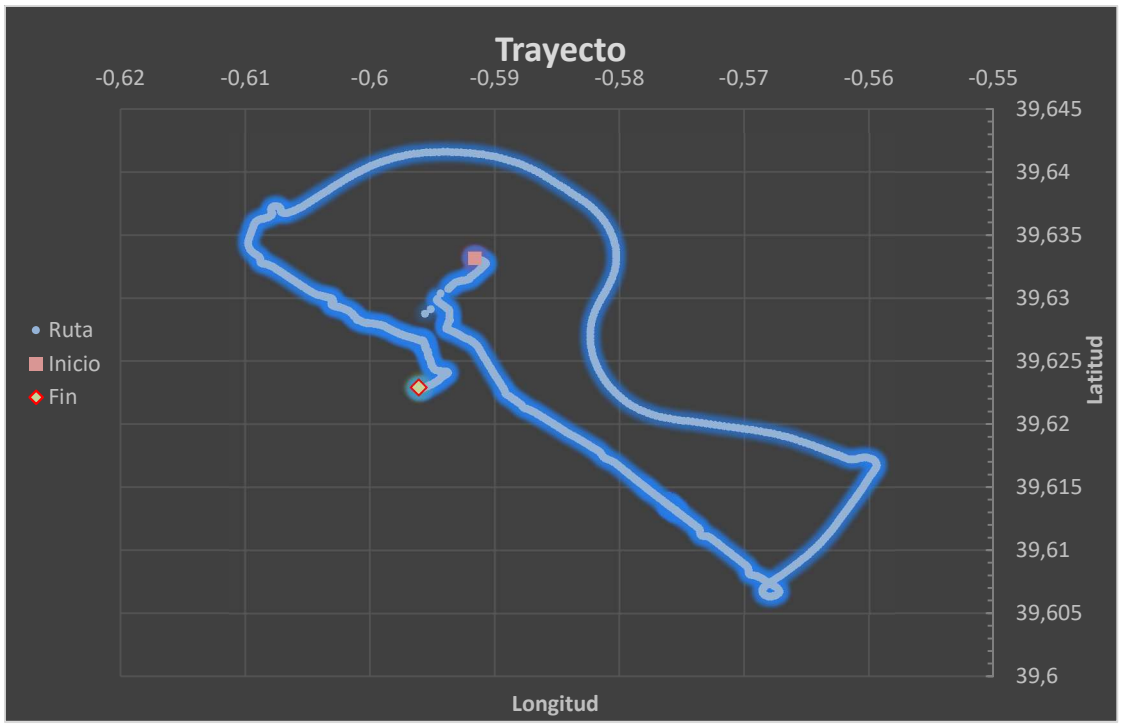


Figura 7.2 – Gráfico con representación de las posiciones obtenidas durante un trayecto

Aun siendo un gráfico de puntos discretos se aprecia una línea continua debido a la gran cantidad de datos y la escala del mapa. Superponiendo este trayecto en un mapa con la herramienta de Google Maps Engine se puede apreciar mucho mejor la ruta seguida por el vehículo en el que se encontraba el dispositivo localizador.

Al realizar la superposición de las coordenadas con el mapa se ha comprobado que la ruta obtenida en el gráfico de Excel encaja a la perfección con las carreteras del mapa proporcionado por Google.

En la figura 7.3 se pierde información sobre el trazado seguido debido a la escala de la imagen, sin embargo en las imágenes en detalle se puede apreciar como las posiciones obtenidas del GPS corresponde con la calzada a la perfección.

Figura 7.3 – Representación de ruta sobre mapa con Google Maps Engine**.**

Se pueden apreciar diferencias entre varios tramos en cuanto a densidad de datos, esto es debido a las variaciones de velocidad del vehículo ya que el tiempo de medición del GPS es fijo. En los trayectos con los puntos más separados el vehículo se encuentra circulando a una mayor velocidad (carretera, autovía), en los tramos con una densidad de puntos mayor el vehículo está parado o circulando a una velocidad baja (semáforos, población). Si bien esto puede servir como referencia, en el programa con el que se realizó la toma de datos también se incluyó un registro de la velocidad en cada punto, de esta forma se puede saber la velocidad a la que circula el vehículo en cada instante.

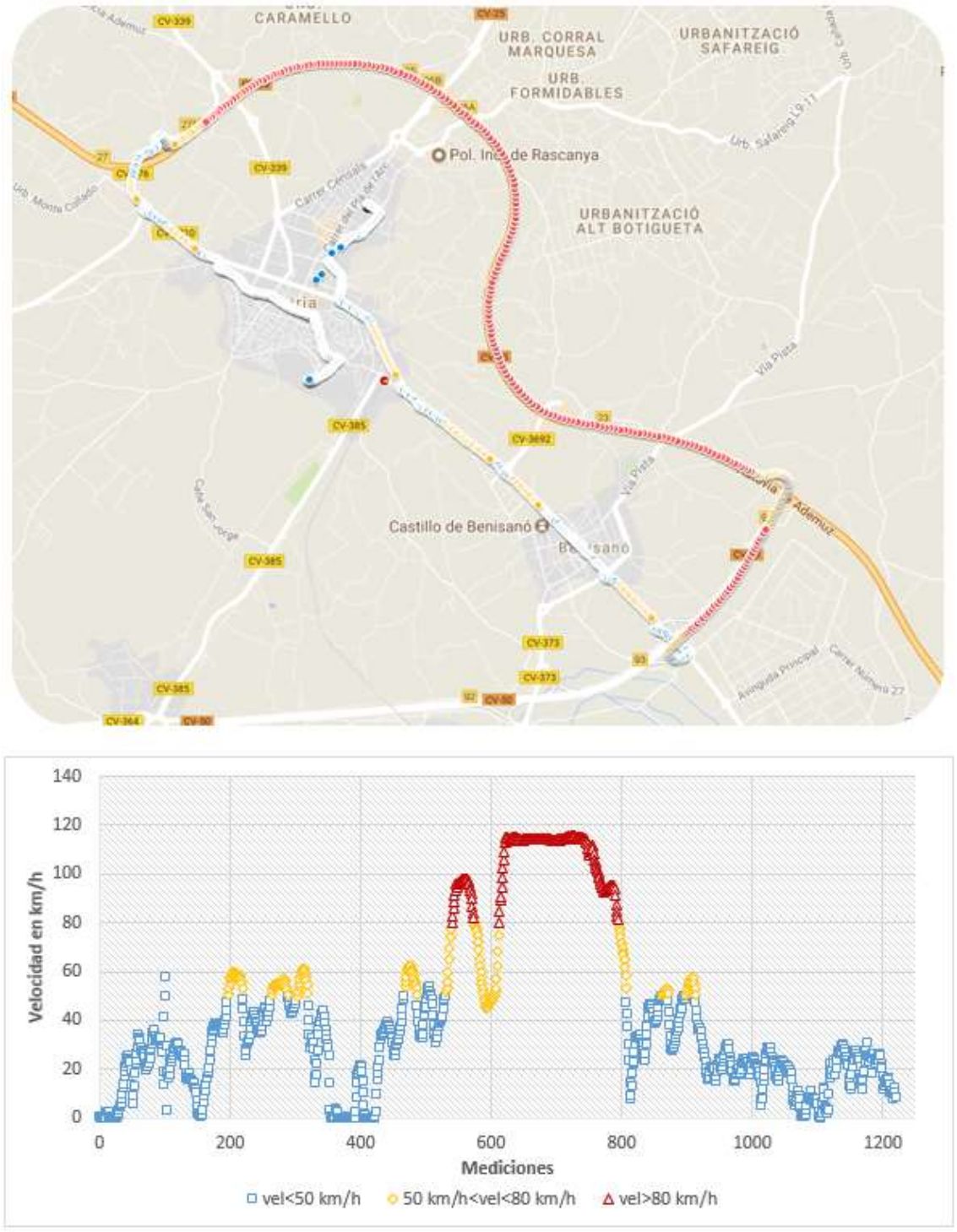


Figura 7.4 – Gráfico del perfil de velocidades registrado durante el trayecto y representación en el mapa según rango de velocidad.

En la parte inferior de la figura 7.4 se encuentra un gráfico en el que se muestra el perfil de velocidades registradas durante el trayecto con el GPS, la parte superior se trata de un mapa en el que está representado el trayecto en el que se puede apreciar tramos de 3 colores diferentes equivalentes a los tramos de velocidad del gráfico.

La implementación de esta función se podría realizar de múltiples maneras, una opción es el control total del usuario quien tiene que conectar y desconectar el modo seguimiento de trayecto de forma manual cada vez, otro caso podría ser mediante un modo alarma como el descrito anteriormente, esta alarma puede activar de forma automática el modo seguimiento de trayectoria para registrar las posiciones en las que se encuentra el dispositivo durante este periodo, también existe la posibilidad que el modo se active y se desactive de manera automática si el dispositivo se está moviendo o está parado con la información de la velocidad obtenida por el GPS. Todas estas opciones se pueden implementar de forma separada o en conjunto de manera que se complementen.

Después de la realización de la prueba se puede afirmar que el dispositivo sería completamente capaz de realizar el seguimiento de la trayectoria en un periodo de tiempo acotado, el único problema sería el envío de estos archivos a larga distancia debido a la gran cantidad de datos que pueden llegar a contener. Una opción podría ser limitar el periodo de tiempo máximo de seguimiento de ruta hasta un tamaño de archivo el cual pueda enviarse por mensaje multimedia o MMS, sin embargo esto genera limitaciones al usuario, otra solución es dotar al módulo SIM800L de una tarjeta con conexión a internet para el envío de estos archivos.

BIBLIOGRAFÍA

Nanda, U., & Kumar Pattnaik, S. (2016). Universal Asynchronous Receiver and Transmitter. *3rd International Conference on Advanced Computing and Communication Systems*, (pág. 5). Bhubaneswar, India.

Blewitt, G. (1997). Basics of the GPS Technique: Observation Equations. En G. Blewitt, *Geodetic Applications of GPS.* Swedish Land Survey.

Departamento de Electrónica, Univesidad de Málaga. (s.f.). *El.uma.es.* Obtenido de http://www.el.uma.es/marin/Practica4_UART.pdf

Google. (s.f.). *Android Developers*. Obtenido de https://developer.android.com/develop/index.html

Mehaffey, J. (12 de Julio de 1999). Obtenido de http://www.elgps.com/documentos/barras/barras.htm

Molino, L. d. (s.f.). *Sallesat.* Obtenido de http://www.sallesat.org/pdfs/ParametrosKeplerianos.pdf

Ortega, V., & Moya, M. (4 de Abril de 2013). *catedrais defe etsit upm*. Obtenido de http://catedraisdefe.etsit.upm.es/wiki/index.php/Los_or%C3%ADgenes_del_Sistema_de_Posicionamiento_Global_(GPS)

Raspberry Pi Foundation. (s.f.). *RaspberryPi.org*. Obtenido de https://www.raspberrypi.org/documentation/

The National Coordination Office for Space-Based Positioning, N. a. (6 de Junio de 2017). *GPS.org*. Obtenido de http://www.gps.gov/systems/gps/

Trimble Navigation Ltd. (s.f.). *TopoServic*. Obtenido de http://toposervic.com/gps-diferencial/

u-blox. (05 de Diciembre de 2011). *u-blox.com.* Obtenido de https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf

PRESUPUESTO

1. INTRODUCCIÓN

En presente presupuesto se recogen los elementos que han sido necesarios para la realización de este proyecto, tanto la mano de obra como el software y material utilizado.

La mano de obra ha sido realizada por una sola persona la cual ha realizado los trabajos de:

- Diseño del hardware del dispositivo GPS (40h)
- Montaje del dispositivo GPS (60h)
- Diseño de la carcasa del dispositivo GPS (8h)
- Programación del dispositivo GPS (80h)
- Diseño de la aplicación Android (32h)
- Programación de la aplicación Android (80h)

Estos puntos se pueden resumir en 2 puntos que engloban los anteriores:

- Diseño, montaje y programación del dispositivo GPS (188h)
- Desarrollo de la aplicación Android (112h)

En cuanto a materiales y software utilizados a tener en cuenta en este presupuesto se incluyen los elementos electrónicos montados en el dispositivo GPS, materiales para la construcción de la carcasa y programas y sistemas operativos utilizados.

2. MANO DE OBRA

Siguiendo los dos puntos presentados en el punto anterior como mano obra realizada en el proyecto, se muestran en la siguiente tabla los precios correspondientes a cada trabajo:

Tabla 1. Precios mano de obra

Nº	Unidades	Descripción	Cantidad	Precio unitario (€)	Precio Total(€)
1	h	Diseño y montaje del dispositivo GPS	170	30	5100
2	h	Desarrollo de la aplicación Android	130	30	3900
				Total	9000

3. MATERIALES

En la tabla 2 se incluyen todos los materiales necesarios utilizados para la realización del proyecto incluyendo las herramientas informáticas utilizadas así como el coste que han supuesto cada uno de ellos.

Tabla 2. Precios materiales

Utilización	Nº	Unidades	Descripción del material	Cantidad	Precio Unitario (€)	Precio Total(€)
Dispositivo GPS	1	Ud	Raspberry Pi 3 Modelo B	1	36.5	36.5
	2	Ud	Módulo GPS NEO-6M GY-GY-GPS6MV2	1	8.64	8.64
	3	Ud	Batería Poweradd Slim 2 5000 mAh	1	7.99	7.99
	4	Ud	Módulo GSM GPRS SIM800L cuatribanda	1	14.5	14.5
	5	Ud	Tarjeta de memória 32GB micro SD Samsung Evo clase 10	1	9.99	9.99
	6	Ud	Cables para prototipado	10	0.064	0.64
	7	Ud	Adaptador serial TTL a USB	1	7.08	7.08
	8	Ud	SO Raspbian strech with desktop Kernel version: 4.9	1	0	0
	9	Ud	Carcasa impresa 3D	1	6	6
Aplicación Android	10	h	Android Studio 2.3.1	80	0	0
					Total	91.34

4. PRECIOS UNITARIOS

Tabla 3. Precios unitarios

Nº Actividad	Unidades	Descripción unidad	Medición	Precio (€)	Importe (€)
1	DISEÑO, MONTAJE Y PROGRAMACIÓN DEL DISPOSITIVO GPS				
1.1	u	DISEÑO Y ENSAMBLAJE DEL HARWARE	1	3338.34	3338.34
1.2	u	DESARROLLO DEL SOFTWARE DEL DISPOSITIVO GPS	1	2400	2400
			TOTAL UNIDAD		5738.34
2	DESARROLLO DE LA APLICACIÓN ANDROID				
2.1	u	DESARROLLO DE LA APLICACIÓN ANDROID	1	3360	3360
			TOTAL UNIDAD		3360

5. PRECIOS DESCOMPUESTOS

Tabla 4. Precios descompuestos

Nº Actividad	Código	Unidades	Descripción	Rendimiento	Precio Unidad (€)	Importe (€)
1			DISEÑO, MONTAJE Y PROGRAMACIÓN DEL DISPOSITIVO GPS			
1.1		u	DISEÑO Y ENSAMBLAJE DEL HARWARE. Estudio previo y elección de los componentes del dispositivo, ensamblaje y conexión de los mismos.			
	001	h	Diseño y elección de componentes	40	30	1200
	002	u	Raspberry Pi 3 Modelo B	1	36.5	36.5
	003	u	Módulo GPS NEO-6M GY-GY-GPS6MV2	1	8.64	8.64
	004	u	Batería Poweradd Slim 2 5000 mAh	1	7.99	7.99
	005	u	Módulo GSM GPRS SIM800L cuatribanda	1	14.5	14.5
	006	u	Tarjeta de memória 32GB micro SD Samsung Evo clase 10	1	9.99	9.99
	007	u	Cables para prototipado	10	0.064	0.64
	008	u	Adaptador serial TTL a USB	1	7.08	7.08
	009	u	SO Raspbian strech with desktop Kernel version: 4.9	1	0	0
	010	h	Montaje y conexión del hardware	60	30	1800
	011	h	Diseño carcasa 3D	8	30	240
	012	u	Impresión carcasa 3D	1	13	13
					TOTAL UNIDAD	3338.34
1.2		u	DESARROLLO DEL SOFTWARE DEL DISPOSITIVO GPS. Preparación del sistema operativo, configuración de comunicación con los periféricos y programación.			
	001	h	Instalación y configuración sistema operativo en la Raspberry Pi	8	30	240
	002	h	Configuración de la comunicación con periféricos	22	30	660
	003	h	Programación Python	50	30	1500
					TOTAL UNIDAD	2400
2			DESARROLLO DE LA APLICACIÓN ANDROID			
2.1		u	DESARROLLO DE LA APLICACIÓN ANDROID. Estudio previo de opciones de programación, preparación del entorno de trabajo, diseño funcional y estetico y programación			
	001	h	Diseño funcional y estetico	32	30	960
	002	u	Software para desarrollo Android Studio	1	0	0
	003	h	Instalación y configuración del entorno de trabajo	4	30	120
	004	h	Programación aplicación	76	30	2280
					TOTAL UNIDAD	3360

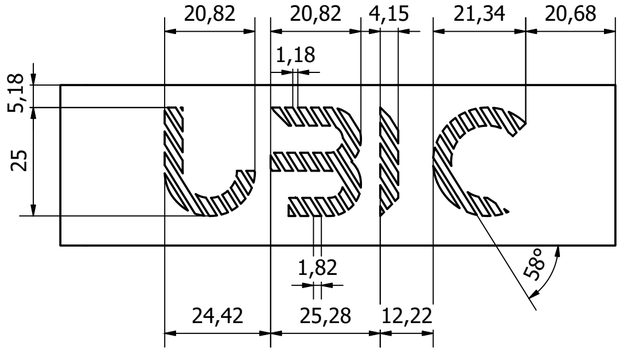
6. RESUMEN DEL PRESUPUESTO

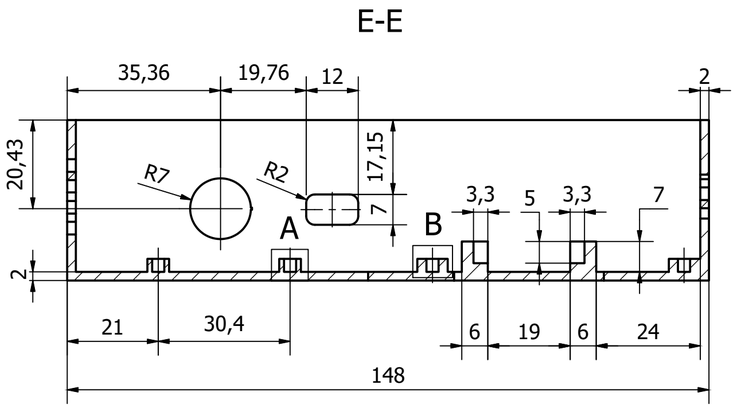
Tabla 5. Resumen del presupuesto

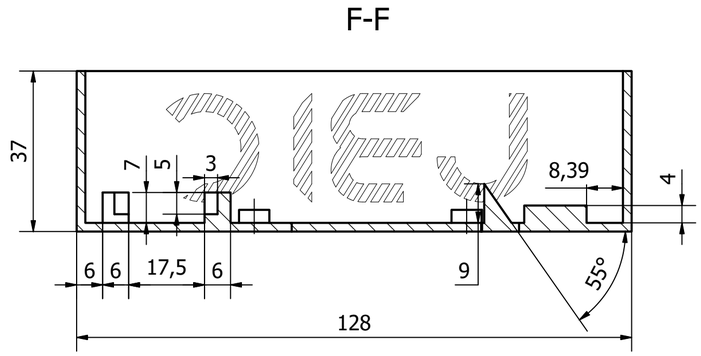
Nº Actividad	Descripción	Importe (€)
1	Diseño, montaje y programación del dispositivo GPS	5738.34
2	Desarrollo de la aplicación Android	3360
	Presupuesto de ejecución material	9098.34
	Gastos generales (13%)	1182.78
	Beneficio industrial (6%)	545.90
	Presupuesto de ejecución por contrata	10827.02
	IVA (21%)	2273.68
	Presupuesto base licitación	13100.70

El importe del presente presupuesto base de licitación asciende a: **TRECE MIL CIEN EUROS CON SETENTA CÉNTIMOS.**

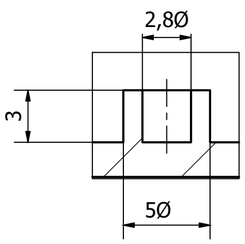
PLANOS



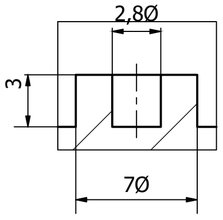


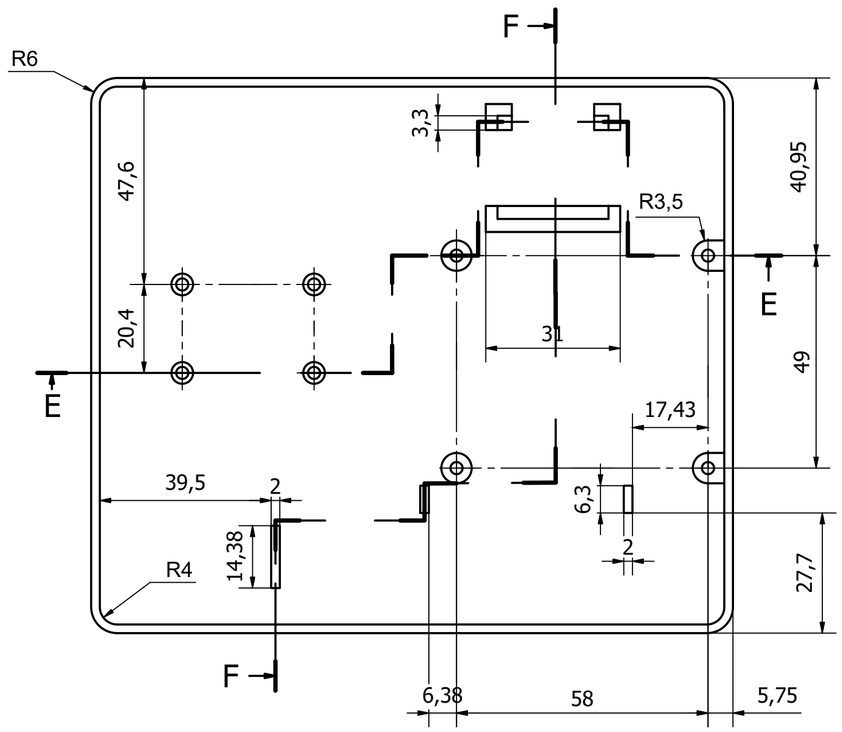


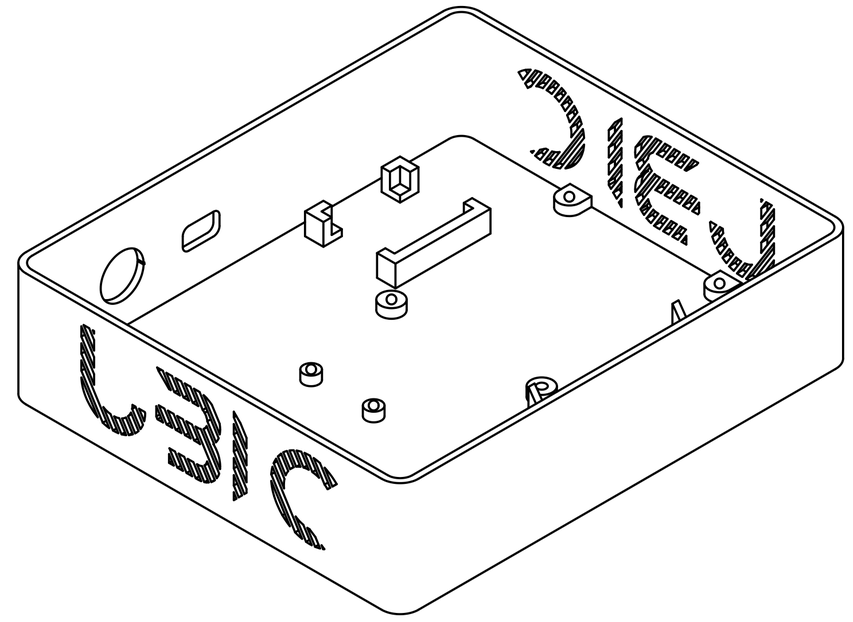
A (4 : 1)



B (4 : 1)







TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Héctor Sancho Chilet
Autor proyecto

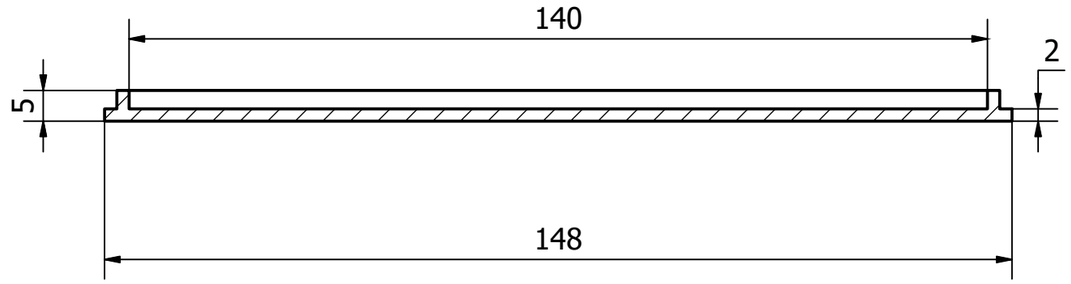
Proyecto:
DESARROLLO DE UN SISTEMA DE LOCALIZACIÓN Y APLICACIÓN MÓVIL PARA VEHÍCULOS EN APARCAMIENTOS

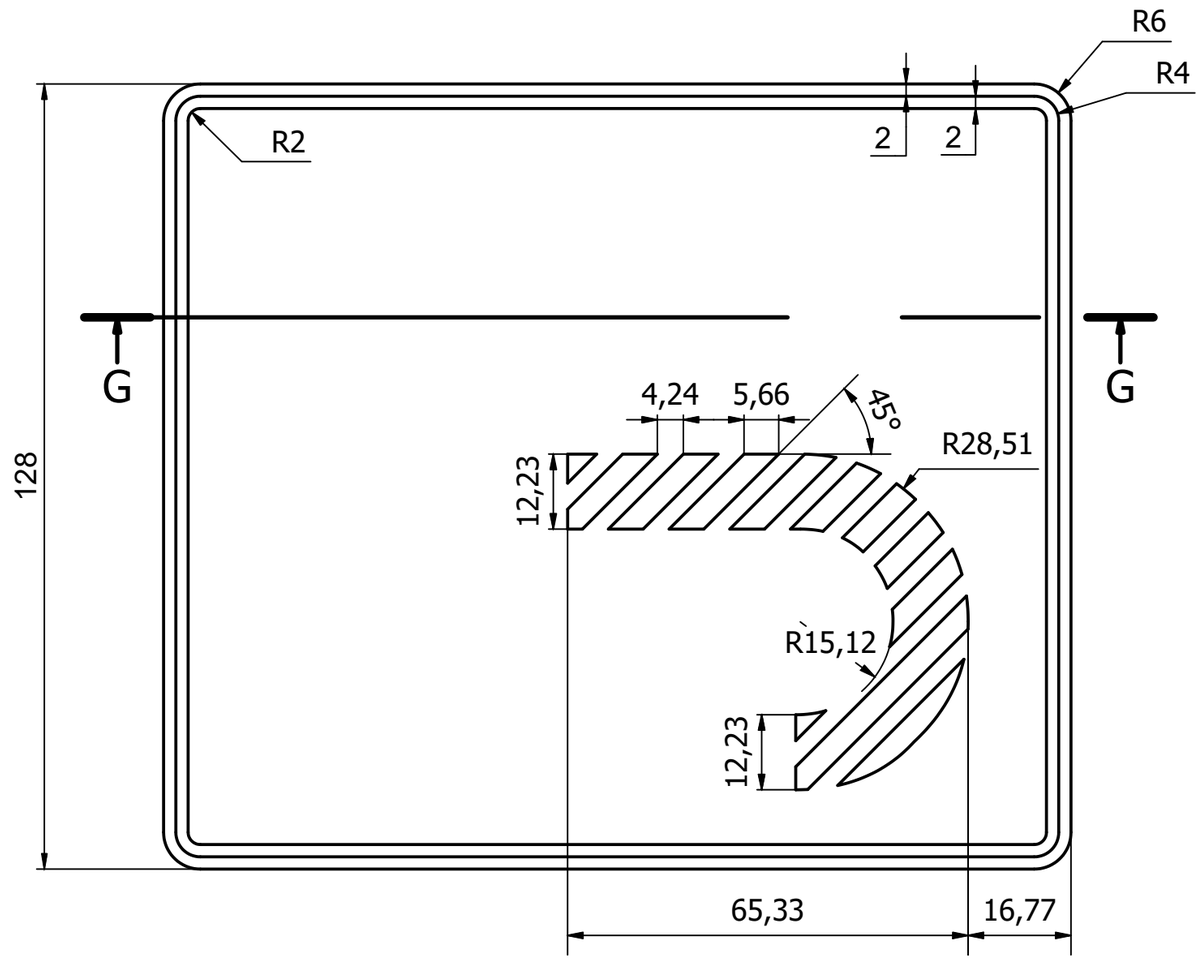
Fecha: Agosto 2017

Escala: 1/1

Plano: Base de la carcasa del dispositivo UBIC

Nº Plano: 1





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto:
DESARROLLO DE UN SISTEMA DE LOCALIZACIÓN Y APLICACIÓN MÓVIL PARA VEHÍCULOS EN APARCAMIENTOS

Plano:
Tapa de la carcasa del dispositivo UBIC

Autor:
Héctor Sancho Chilet

Fecha:
Agosto 2017

Escala:
1:1

Nº Plano:
2

ANEXO 1 – CÓDIGO PROGRAMA UBIC.PY

```
import gps
import socket
import sys
import time
import RPi.GPIO as GPIO
import os

GPIO.setmode(GPIO.BCM)
UDP_IP = "192.168.42.10"
UDP_IP2 = "192.168.42.11"
UDP_IP3 = "192.168.42.12"
UDP_IP4 = "192.168.42.13"
UDP_IP5 = "192.168.42.14"
UDP_PORT = 5005

posicion = "0"
session = gps.gps("localhost", "2947")
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
    try:

        latitud = 0
        report = session.next()

        if report['class'] == 'TPV':
            if hasattr(report, 'lat'):
               latitud = report.lat
               posicion = str(report.lat)+'|'+str(report.lon)+'|'
               print posicion
               fichero = open ('coordenadas.txt','w')
               fichero.write(posicion)
               fichero.close()
            if latitud != 0:
               sock.connect(("192.168.42.1", 80))
               sock.sendto(posicion, (UDP_IP, UDP_PORT))
               sock.sendto(posicion, (UDP_IP2, UDP_PORT))
               sock.sendto(posicion, (UDP_IP3, UDP_PORT))
               sock.sendto(posicion, (UDP_IP4, UDP_PORT))
               sock.sendto(posicion, (UDP_IP5, UDP_PORT))
               print 'enviado:   ' + posicion

    except KeyError:
                pass
    except KeyboardInterrupt:
                quit()
    except StopIteration:
                session = None
                print "GPSD has terminated"
```

ANEXO 2 – CÓDIGO PROGRAMA UBICSMS.PY

```
import serial
import time
from curses import ascii
sSerie = serial.Serial('/dev/ttyS0',9600)
time.sleep(30)
estado = 0
sSerie.write('AT\r\n')
while True:
    time.sleep(0.5)
    try:
        llamando1 = sSerie.readline()
        llamando2 = llamando1.split()
        for l in llamando2:
            if l == "RING":
                print l
                estado = 1
                sSerie.write('ATH\r\n')
            if estado == 1:
                if l[:5] == "+CLIP":
                    cadena =l
                    numero = llamando2[1].split(",")[0]
                    print 'Te ha llamado: ' + numero
                    estado = 0
                    fichero = open ('coordenadas.txt')
                    coordenadas = fichero.read()
                    fichero.close()
                    try:
                        sSerie.write("AT\r\n")
                        time.sleep(1)
                        print sSerie.readline()
                        print sSerie.readline()
                        sSerie.write("AT+CMGF=1\r\n")
                        time.sleep(1)
                        print sSerie.readline()
                        print sSerie.readline()
                        sSerie.write('AT+CMGS=' + numero + '\r\n')
                        time.sleep(2)
                        sSerie.write(coordenadas + ascii.ctrl('z'))
                        time.sleep(4)
                        print sSerie.readline()
                        print sSerie.readline()
                        print sSerie.readline()

                    except ValueError:
                        print "Error"


    except KeyboardInterrupt:
                quit()
```

ANEXO 3 – CÓDIGO APLICACIÓN ANDROID

3.1. ANDROIDMANIFEST.XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.hector.tfgapp">


    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission
android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>


    <application
        android:allowBackup="true"
        android:icon="@mipmap/logo5"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!--
             The API key for Google Maps-based APIs is defined as a
string resource.
             (See the file "res/values/google_maps_api.xml").
             Note that the API key is linked to the encryption key
used to sign the APK.
             You need a different API key for each encryption key,
including the release key that is used to
             sign the APK for publishing.
             You can define the keys for the debug and release targets
in src/debug/ and src/release/.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
```

```
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps" />

        <service android:name=".MiIntentService" />


        <receiver
android:name="com.example.hector.tfgapp.MessageReceiver"
android:enabled="true">
            <intent-filter android:priority="2147483647">
                <action
android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

3.2. MAINACTIVITY

3.2.1. MainActivity.java

```
kage com.example.hector.tfgapp;

import android.Manifest;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.net.wifi.WifiManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.AppCompatImageButton;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    boolean infovisible = false;
    public Intent serviceIntent;
    private WifiManager wifiManager;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        AppCompatImageButton navegar = (AppCompatImageButton)
findViewById(R.id.navegar);
        final AppCompatImageButton info = (AppCompatImageButton)
findViewById(R.id.info);
        AppCompatImageButton sms = (AppCompatImageButton)
findViewById(R.id.sms);
```

```
        final TextView infotext = (TextView)
findViewById(R.id.infotext);
        Log.d("TAG", "intent");
        int permissionCheck = ContextCompat.checkSelfPermission(
                MainActivity.this, Manifest.permission.CALL_PHONE);
        if (permissionCheck != PackageManager.PERMISSION_GRANTED) {
            Log.i("Mensaje", "No se tiene permiso para realizar
llamadas telefónicas.");
            ActivityCompat.requestPermissions(MainActivity.this, new
String[]{Manifest.permission.CALL_PHONE}, 225);
        } else {
            Log.i("Mensaje", "Se tiene permiso!");
        }
        int permissionCheck2 = ContextCompat.checkSelfPermission(
                MainActivity.this, Manifest.permission.RECEIVE_SMS);
        if (permissionCheck2 != PackageManager.PERMISSION_GRANTED) {
            Log.i("Mensaje", "No se tiene permiso para realizar
llamadas telefónicas.");
            ActivityCompat.requestPermissions(MainActivity.this, new
String[]{Manifest.permission.RECEIVE_SMS}, 225);
        } else {
            Log.i("Mensaje", "Se tiene permiso!");
        }

        serviceIntent = new Intent(MainActivity.this,
MiIntentService.class);


        startService(serviceIntent);

        navegar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                wifiManager = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
                wifiManager.setWifiEnabled(false);
                Intent intent = new Intent(v.getContext(),
MapsActivity.class);
                startActivity(intent);

            }
        });
        sms.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Intent i = new
Intent(android.content.Intent.ACTION_CALL);
                i.setData(Uri.parse("tel:617200383"));
                int permissionCheck =
ContextCompat.checkSelfPermission(
                        MainActivity.this,
Manifest.permission.CALL_PHONE);
                if (permissionCheck !=
PackageManager.PERMISSION_GRANTED) {
                    Log.i("Mensaje", "No se tiene permiso para
realizar llamadas telefónicas.");

ActivityCompat.requestPermissions(MainActivity.this, new
String[]{Manifest.permission.CALL_PHONE}, 225);
                } else {
```

```
                    Log.i("Mensaje", "Se tiene permiso!");
                }
                startActivity(i);
            }
        });


        info.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(infovisible){
                    infotext.setVisibility(View.GONE);
                    infovisible=false;
                }else{
                    infotext.setVisibility(View.VISIBLE);
                    infovisible=true;
                }



            }
        });

    }
}
```

3.2.2. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@mipmap/fondo4"
    tools:context="com.example.hector.tfgapp.MainActivity">

    <android.support.v7.widget.AppCompatImageView
        android:layout_width="368dp"
        android:layout_height="296dp"
        android:src="@mipmap/logo7"
        android:scaleType="center"
        android:foregroundGravity="center"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="10dp"
        app:layout_constraintLeft_toLeftOf="parent" />
    <android.support.v7.widget.AppCompatImageButton
        android:layout_width="127dp"
        android:layout_height="61dp"
        android:src="@mipmap/map1"
        android:scaleType="centerCrop"
        android:background="@color/transparente"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="180dp"
        android:id="@+id/navegar"
        android:layout_marginStart="119dp" />
    <android.support.v7.widget.AppCompatImageButton
        android:layout_width="227dp"
        android:layout_height="161dp"
        android:src="@mipmap/botonsms"
```

```
        android:scaleType="centerInside"
        android:background="@color/transparente"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="16dp"
        android:id="@+id/sms"
        android:layout_marginStart="73dp"
         />
    <android.support.v7.widget.AppCompatImageButton
        android:id="@+id/info"
        android:layout_width="47dp"
        android:layout_height="46dp"
        android:src="@mipmap/info"
        android:background="@color/transparente"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="16dp" />

    <TextView
        android:id="@+id/infotext"
        android:gravity="center"
        android:layout_width="344dp"
        android:layout_height="481dp"
        android:background="@color/background"
        android:textSize="24dp"
        android:text="@string/infointro"
        android:visibility="gone"
        android:textColor="@color/blanco"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="10dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
         />


</android.support.constraint.ConstraintLayout>
```

3.3. MIINTENTSERVICE.JAVA

```
package com.example.hector.tfgapp;

import android.app.IntentService;
import android.content.Context;
import android.content.Intent;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.support.annotation.Nullable;
import android.util.Log;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
```

```
import java.util.List;

/**
 * Created by Hector on 25/05/2017.
 */

public class MiIntentService extends IntentService {

    public boolean actualizando = true;
    private WifiManager wifiManager;
    private WifiInfo wifiInfo;
    private WifiConfiguration wifiConfig = new WifiConfiguration();
    private String connectedID, ultimascoordenadas;
    private String ssid = "UBIC";
    private String myssid = "\"UBIC\"";
    private String key = "sistemaUBIC";
    private String[] coordenadas;
    public File file;
    public MiIntentService() {
        super("MiIntentService");
    }

    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        wifiManager = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        while(actualizando) {
            if (!wifiManager.isWifiEnabled()) {
                wifiManager.setWifiEnabled(true);
                esperar(5);
            }
            wifiInfo = wifiManager.getConnectionInfo();
            connectedID = wifiInfo.getSSID();
            if (!comprobar()) {
                connect();
                Log.d("Service", "conectando");
                esperar(10);
                wifiInfo = wifiManager.getConnectionInfo();
                connectedID = wifiInfo.getSSID();
                Log.d("Service", "conectado a:  " + connectedID);
            }
            while (comprobar()) {

                Log.d("Service", "Recibiendo coordenadas");

                try {
                    DatagramSocket clientsocket= new
DatagramSocket(5005);
                    byte[] receivedata = new byte[30];
                        while (comprobar()) {
                            try {
                                DatagramPacket recv_packet = new
DatagramPacket(receivedata, receivedata.length);
                                Log.d("Service", "S: Receiving...");
                                clientsocket.receive(recv_packet);
                                String receivedstring = new
String(recv_packet.getData());
                                Log.d("Service", " Received String: "
+ receivedstring);
                                InetAddress ipaddress =
recv_packet.getAddress();
```

```
                                int port = recv_packet.getPort();
                                Log.d("Service", "IPAddress : " +
ipaddress.toString());
                                Log.d("Service", "Port : " +
Integer.toString(port));
                                coordenadas =
receivedstring.split("\\|", 3);
                                ultimascoordenadas =
coordenadas[0]+"|"+coordenadas[1];
                                Log.d("Service", coordenadas[0] + "---
--" + coordenadas[1]);
                                file = new File(getFilesDir(),
"ultimaposicion.txt");

Log.d("Service",getFilesDir().toString());
                                try{
                                    Log.d("Service","creando
fichero");
                                    FileOutputStream out = new
FileOutputStream(file);

out.write(ultimascoordenadas.getBytes());
                                    Log.d("Service","fichero escrito"
+ ultimascoordenadas);
                                    out.close();
                                }catch (IOException e){
                                }
                            }catch (Exception e){
                                Log.d("service","fallo recibiendo");
                            }
                            if(!wifiManager.isWifiEnabled()){
                                actualizando=false;
                                break;
                            }
                            esperar(5);
                            wifiInfo =
wifiManager.getConnectionInfo();
                            connectedID = wifiInfo.getSSID();
                        }
                    } catch (SocketException e) {
                        Log.e("UDP", "Socket Error", e);
                    } catch (IOException e) {
                        Log.e("UDP", "IO Error", e);
                    }
                }
                esperar(5*60);
            }
        }
        public boolean comprobar(){
            if(connectedID.equals(myssid)){
                return true;
            }else {
                return false;
            }
        }

        public void esperar(int segundos) {

            try {
                Thread.sleep(segundos*1000);
            } catch (Exception e){
```

```
            Log.d("Service","esperar ha fallado");
        }
    }
    public void connect(){
        wifiConfig.SSID=String.format("\"%s\"",ssid);
        wifiConfig.preSharedKey = String.format("\"%s\"", key);
        wifiManager.addNetwork(wifiConfig);
        List<WifiConfiguration> list =
wifiManager.getConfiguredNetworks();
        for( WifiConfiguration i : list){
            if(i.SSID != null && i.SSID.equals(myssid)){
                wifiManager.disconnect();
                wifiManager.enableNetwork(i.networkId,true);
                wifiManager.reconnect();
                break;
            }
        }
    }


}
```

3.4. MAPSACTIVITY

3.4.1. MapsActivity.java

```
package com.example.hector.tfgapp;

import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {
    private GoogleMap mMap;
    private double lat,lon;
    private String lectura;
    private String[] coordenadas;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        lat=0;
        lon=0;
        try {
            int linea = 0;
            InputStreamReader is = new
InputStreamReader(openFileInput("ultimaposicion.txt"));
            BufferedReader fich = new BufferedReader(is);
```

```
            lectura = fich.readLine();
            is.close(); // Cerrar el fichero
        } catch (IOException ex){
            ex.printStackTrace();
        }
        coordenadas = lectura.split("\\|", 3);
        lat = Double.parseDouble(coordenadas[0]);
        lon = Double.parseDouble(coordenadas[1]);
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        LatLng ubic = new LatLng(lat, lon);
        mMap.addMarker(new
MarkerOptions().position(ubic).title("Posición de UBIC"));
        CameraUpdate cameraUpdate =
CameraUpdateFactory.newLatLngZoom(ubic,17 );
        mMap.animateCamera(cameraUpdate);

    }
}
```

3.4.2. Activity_maps.xml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.hector.tfgapp.MapsActivity" />
```

3.5. MESSAGERECEIVER.JAVA

```
package com.example.hector.tfgapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * Created by Hector on 25/07/2017.
 */

public class MessageReceiver extends BroadcastReceiver{
    private static final String SMS_RECEIVED =
"android.provider.Telephony.SMS_RECEIVED";
    private int num;
    private String UBIC="+34617200383";
```

```
    public File file;

    @Override
    public void onReceive(Context context, Intent intent) {

        if (intent.getAction().equals(SMS_RECEIVED)) {
            Bundle bundle = intent.getExtras();


            if (bundle != null) {
                Object[] sms = (Object[]) bundle.get("pdus");
                for (int i = 0; i < sms.length; i++) {
                    SmsMessage mensajes =
SmsMessage.createFromPdu((byte[]) (sms[i]));
                    String numero =
mensajes.getDisplayOriginatingAddress();
                    String coordenadas =
mensajes.getMessageBody().toString();
                    if (numero.equals(UBIC)) {
                        file = new
File("/data/user/0/com.example.hector.tfgapp/files",
"ultimaposicion.txt");
                        try{
                            Log.d("Service","creando fichero");
                            FileOutputStream out = new
FileOutputStream(file);
                            out.write(coordenadas.getBytes());
                            Log.d("Service","fichero escrito" +
coordenadas);
                            out.close();

                        }catch (IOException e){

                        }
                        Toast.makeText(context, "UBIC encontrado,
sígueme...", Toast.LENGTH_LONG).show();
                        Intent intent2 = new
Intent(context.getApplicationContext(), MapsActivity.class);

intent2.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK);
                        context.startActivity(intent2);


                        //Guardar en fichero cuando funcione...
                    }

                }

            }
        }
    }
}
```