Universitat Oberta de Catalunya uoc.edu

Diseño de un prototipo demostrador de un sistema rastreador de GPS con comunicación mediante telefonía celular.

Alumno: Emilio Angulo Navarro
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación
Sistemas de comunicación

Profesor Consultor: Dr. Raúl Parada Medina
Profesor responsable de la asignatura: Dr. Carlos Monzo Sánchez

6 de enero de 2019

Esta obra está sujeta a una licencia de Reconocimiento–NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño de un prototipo demostrador de un sistema rastreador de GPS con comunicación mediante telefonía celular.*
Nombre del autor:	*Emilio Angulo Navarro*
Nombre del consultor:	*Dr. Raúl Parada Medina*
Nombre del PRA:	*Dr. Carlos Monzo Sánchez*
Fecha de entrega:	*01/2020*
Titulación:	*Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación*
Área del Trabajo Final:	*11.624 - TFG - Sistemas de Comunicación*
Idioma del trabajo:	*Español*
Palabras clave:	*Rastreador GPS – Seguimiento de vehículos - Comunicación celular – Microcontrolador - GNSS*

Resumen del Trabajo:

Los rastreadores de personas, animales y objetos – y en particular de vehículos – basados en la tecnología GPS están generando un mercado importante con unas previsiones de volumen total de negocio de unos 2.530 M$ en el año 2023, y con un crecimiento anual medio de 11.9%. Cada vez existe más oferta aplicada al seguimiento de vehículos y con funcionalidades más diversas, entre las que podemos destacar la gestión de flotas y el seguimiento de vehículos para la detección de robos.

Estos dispositivos combinan distintas tecnologías de telecomunicaciones, así como diversas aplicaciones informáticas para su control, manejo y para la obtención del máximo de información referente a la trayectoria geográfica y otros eventos que el vehículo está experimentando.

En este TFG se va a estudiar, desde un punto de vista funcional y tecnológico, los distintos tipos de rastreadores existentes en el mercado, en especial aquellos que se enfocan en la detección de sustracciones de vehículos y en el establecimiento de su seguimiento, mediante radio, para su posterior recuperación. El alcance del proyecto se completa con la generación de unas especificaciones de un producto que pueda competir con la oferta existente en el mercado, con el diseño conceptual de un sistema completo de dispositivo con estas prestaciones, así como con el diseño detallado y puesta en marcha de un primer prototipo demostrador de la arquitectura y de las tecnologías seleccionadas, basado en una plataforma de desarrollo electrónico tipo Arduino o similares.

Abstract:

People, animals and objects trackers - and in particular vehicles trackers - based on GPS technology are generating an important business, with forecasts by the year 2023 of total revenue of about 2,530 M $, and with an average annual growth of 11.9% Everyday there is more offer applied to vehicle tracking and with more diverse functionalities, among which we can highlight fleet management and vehicle tracking for theft detection.

These devices combine different telecommunication technologies, as well as various computer applications for their control, handling and in order to obtain the maximum information regarding the geographical trajectory and other events that the vehicle is experiencing.

This TFG will study, from a functional and technological point of view, the different types of trackers in the market, especially those that are focused on the detection of vehicle subtractions and the establishment of their monitoring, by radio technologies, for later recovery. The scope of the project is completed with the generation of specifications of a product that can compete with the existing offer in the market, with the conceptual design of a complete device system with these benefits, as well as with the detailed design and startup of a demonstration prototype of the selected architecture and technologies, based on an Arduino or similar electronic prototyping development platform.

Agradecimientos:

- A mi director de TFG **Dr. Raúl Parada Medina** por su soporte y ayuda a lo largo de estas semanas de trabajo.

- A la **UOC** por organizar este grado y facilitar mucho la oportunidad de seguir aprendiendo sobre el maravilloso mundo de la tecnología.

- A mi familia, **Begoña**, **Javier** y **Carlos**, por la paciencia y soporte a lo largo de estos años de estudio.

- Dedicado a **mis padres**, **Emilio** y **María**, los que siempre han querido que estudie, y en especial a la memoria de **Maria Barrera**, cuya premonición se ha hecho realidad cuarenta y cinco años después de realizarla.

Indice

1. Introducción 1
1.1 Contexto y justificación del TFG 1
1.2 Objetivos y alcance del Trabajo 2
1.3 Enfoque y método seguido 4
1.4 Planificación del TFG 5
1.5 Breve sumario de productos obtenidos 6
1.6 Breve descripción de los otros capítulos de la memoria 7
2.Estado del arte y estudio de precedentes 8
2.1 Estado del arte de productos similares 8
2.1.1 Análisis del rastreador TK102 B 13
2.1.2 Análisis del rastreador Atlantis – Moto 17
2.2 Estado del arte de las tecnologías concurrentes 19
2.2.1 Tecnologías para el desarrollo de un prototipo demostrador del TMR 21
2.2.2 Tecnologías de recepción de señal GPS y de comunicación celular 25
2.2.3 Tecnologías para detección de aceleraciones e inclinación 29
2.2.4 Tecnologías de desarrollo de aplicaciones para Android 31
2.2.5 Plataformas digitales para el almacenamiento de datos IoT 31
3. Especificación básica 34
4. Diseño conceptual y arquitectura del sistema 36
5. Diseño detallado del prototipo 40
6. Pruebas funcionales sobre el prototipo 63
7. Presupuesto de ingeniería 71
8. Conclusiones 72
9. Glosario 75
10. Bibliografía 78
11. Anexos 80

1. Introducción

En este primer capítulo se presenta el marco en el que se engloba este TFG así como las justificaciones que lo sustentan. Se describe los objetivos principales y el alcance de éste, así como el enfoque y el método seguido para su realización. El capítulo se completa con una planificación temporal de las actividades principales a realizar, así como con una descripción de los productos obtenidos y con un breve resumen de los distintos capítulos que componen esta memoria.

1.1 Contexto y justificación del TFG

Al tratarse de un proyecto fin de grado sus objetivos y su justificación se encuentran centrados en motivaciones académicas, más que en posibles justificaciones económicas o industriales.

La motivación personal que mueve al autor a la realización de este proyecto está basada en tres grandes pilares:

- Aprender distintas tecnologías de vanguardia y con multitud de campos de aplicación como son:
    - sistemas globales de posicionado y navegación por satélite, o GNSS (*Global Navigation Satellite System*)
    - sistemas de comunicaciones móviles
    - internet de las cosas, IoT (*Internet of Things*)
    - almacenamiento de información en servidores remotos (*cloud*)
    - aplicaciones en dispositivos móviles
    - plataformas hardware y software de desarrollo de prototipos (como los ecosistemas desarrollados en torno a Arduino, BeagleBone Black o Raspberry Pi)

- Realizar un proyecto multidisciplinar, donde la componente de sistema e integración juegue un papel clave en la toma de decisiones tecnologías.

- Realizar un proyecto práctico, donde algo deba funcionar, aunque sea con las múltiples simplificaciones por hacerlo a nivel prototipo, para comprobar el funcionamiento tanto de los componentes como del sistema, y que pueda disponer de fases ulteriores de desarrollo de soluciones industriales.

A pesar de que el espíritu del proyecto es realizar el prototipo de un producto, que pueda llegar a tener una aplicación real industrial comercializable, no cabe duda de que la extensión y el alcance de un TFG universitario, de unas 300 horas de trabajo de un único estudiante, limita mucho las posibilidades de realizar un proyecto sumamente innovador y que pueda llegar a competir con

productos que actualmente se comercializan. Estos productos han sido desarrollados por equipos de varios ingenieros de diversas disciplinas, con extensiones de miles de horas de ingeniería y con inversiones económicas, tanto, en útiles, pruebas de funcionalidad y regulatorias, no despreciables, contra a los que un estudiante, con un tiempo limitado de unas trescientas horas, le es difícil competir.

El mero hecho de hacer un prototipo con algunas de las funcionalidades de los productos actuales ya representa un logro significativo.

1.2 Objetivos y alcance del Trabajo

El objetivo principal del presente TFG es el desarrollo conceptual y detallado, a nivel primer prototipo demostrador, de un sistema rastreador de GPS con comunicación mediante telefonía celular, orientado al seguimiento de vehículos y en particular de motocicletas. Su funcionalidad principal se orientará a la detección de robos.

El sistema estará compuesto por varios subsistemas principales:

1. terminal móvil rastreable (TMR), que será portado por el vehículo que ha de ser rastreado y protegido. Este terminal dispondrá:

    - del dispositivo de detección de señales satelitales GPS
    - de la unidad de comunicación móvil, para envío de información al *cloud* y a otros terminales móviles
    - de sistemas de detección de aceleraciones y otra sensórica para la emisión de llamadas de emergencia
    - un dispositivo para la gestión efectiva de la energía para evitar la excesiva descarga de la batería del vehículo
    - la placa de desarrollo para integrar todos los periféricos mencionados, con su software a nivel demostración
    - carcasas sencillas para su integración en el vehículo

2. Plataforma externa de almacenamiento de datos en el *cloud* (PEADC) para recoger toda la información enviada por el terminal móvil rastreable (TMR), y su posterior descarga desde dispositivos fijos o móviles.

3. Aplicación para terminales móviles de seguimiento (TMS), con sistema operativo Android, aunque adicionalmente podría considerarse iOS. La aplicación permitirá el seguimiento en tiempo real, así como la configuración remota del sistema.

4. Adicionalmente el sistema podría disponer de terminales fijos de seguimiento (TFS) instalados en ordenadores convencionales con sistema operativo, Windows, Linux o Mac OS.

El alcance detallado que se plantea para el presente trabajo es:

- Estudio del estado del arte del sector de los GPS *trackers*, analizando compañías presentes en el mercado, dispositivos principales, así como sus prestaciones.
- Estudio del estado del arte a nivel tecnológico.
- Especificación de un primer prototipo, así como del posible producto final derivado.
- Arquitectura y diseño conceptual del sistema.
- Diseño detallado de un primer prototipo demostrador, con plataformas abiertas, incluyendo:

    - Hardware:

        - Selección de la plataforma de desarrollo (Arduino, Raspberry PI, BeagleBone Black)
        - Selección de los *shields* de comunicación:
            - GPS
            - telefonía celular
            - otros sensores y actuadores
        - Montaje y arranque del conjunto.

    - Software/firmware:

        - Programación de la plataforma de desarrollo para la simulación del TMR:
            - captación de la señal de GPS así como de otra sensórica para detectar eventos
            - envío de mensajes de emergencia directamente a celular de usuarios designados (opcional)
            - envío de la información a un plataforma *cloud*, probablemente enfocada en IoT
            - gestión de la energía
            - configuraciones del dispositivo
        - Programación o establecimiento de vínculos en la plataforma IoT *cloud*.
        - Programación de una aplicación TMS para dispositivos Android (no para iOS), a nivel demostrador (probablemente con App Inventor 2):

            - para recepción de información directamente desde el dispositivo móvil de rastreo
            - petición de información a la plataforma *cloud*
            - petición de información al dispositivo móvil de rastreo
            - configuración remota del mismo

- En el alcance del proyecto no se realizará ningún prototipo del TFS (terminal fijo de seguimiento)

- Demostración funcional del sistema.
- Planificación del proyecto.
- Presupuesto de costes de ingeniería, incluyendo el primer prototipo.
- Posibles siguientes pasos.
- En caso de disponer de tiempo, se podría estudiar la transferencia de señales de audio y video entre el dispositivo móvil de rastreo y el terminal móvil del usuario designado.

En el caso de que durante el desarrollo del proyecto se generen problemáticas que pongan en riesgo el cumplimiento del alcance se prescindirá de la realización de la funcionalidad relacionada con el envío y recuperación de información de la plataforma IoT *cloud*, centrando los esfuerzos en la correcta captación de las señales de GPS y en la comunicación vía telefonía celular entre el terminal móvil rastreable (TMR) y el terminal móvil de seguimiento (TMS).

1.3 Enfoque y método seguido

El proyecto lo podemos dividir en tres partes. Una primera parte de análisis donde se va realiza un estudio del estado del arte, con dos áreas de foco claramente diferenciadas, como son:

- Estudio de productos similares actualmente en el mercado. Se realiza un análisis más o menos detallado de aquellos productos que puedan ser más representativos de las distintos segmento de mercado.
- Estudio de las tecnologías que pueden ser utilizadas para resolver el problema. Para ello el sistema se ha de descomponer en diferentes funcionalidades – bloques funcionales -, analizándose los requerimientos de cada uno de estos bloques y las posibles soluciones que se pueden establecer para resolver cada uno de ellos.

La segunda parte comprenderá la síntesis de lo aprendido.

- Así, con lo aprendido en el estudio de estado del arte de la oferta actual, se define una especificación, más o menos detallada, de las prestaciones del producto y su prototipo demostrador.
- Tras este estudio del estado del arte tecnológico se desarrolla un diseño conceptual y arquitectónico del sistema, donde se indica los distintos bloques funcionales, así como los componentes tecnológicos que se pueden utilizar en cada uno de ellos, y se establece una comparativa de sus prestaciones.

Por último, la tercera parte, consiste en el montaje y puesta a punto del prototipo demostrador donde se pretende que todas las piezas funcionen correctamente y sistema desarrolle las prestaciones básicas para demostrar viabilidad de la tecnología y del diseño propuesto.

En cuanto a la estrategia de desarrollo del prototipo se puede destacar:

- hardware: no se desarrolla con componentes diseñados específicamente para el prototipo, sino que se utiliza una plataforma de desarrollo de prototipos electrónicos, como puede ser Arduino, Raspberry Pi o Beaglebone Black, así como componentes accesorios diseñados para estas plataformas.
- Software: los distintos componentes software tienen la intención de realizar la demostración básica de las tecnologías y son diseñadas ad hoc para este propósito, sin pretender ser la base de código de un producto futuro comercial. Los lenguajes de programación a utilizar:

    - Para TMR se utiliza el lenguaje más común en función de la plataforma de desarrollo seleccionad (C si es Arduino, Phyton si es Rapsberri Pi o BeagleBone Black)
    - Para el TMS con sistema operativo Android se va a utilizar algún lenguaje o IDE sencillo como puede ser MIT App Inventor 2.

Es por tanto evidente que, en el caso del desarrollo posterior de un producto derivado de este concepto, el diseño detallado del hardware y del software debería comenzarse prácticamente de cero, reaprovechándose solo lo aprendido a nivel sistema y arquitectura, pero no el código en sí.

1.4 Planificación del TFG

La planificación básica realizada se ha estructurado a lo largo de los 5 hitos que marcan las entregas de la 5 PAC’s que componen el TFG. Las actividades representadas en el siguiente diagrama de Gantt agrupan un conjunto de actividades más simples, pero que su descripción solo haría que complicar la planificación general. Las actividades presentadas se irán subdividiendo en actividades más sencillas conforme el TFG vaya avanzando.

Como puede observarse en el diagrama de Gantt las actividades del TFG empezaron antes del periodo lectivo del cuatrimestre de otoño del 2019. En esa fase previa se estudiaron varios posibles temas, y se seleccionó un tema de acuerdo con los profesores responsables.
Del mismo modo, durante los meses de verano de 2019 se comenzó el estudio del estado de arte, tanto de productos similares como de las tecnologías que pueden ser aplicadas.
En la primera fase, se completa con la entrega del documento con los apartados necesarios para la PAC1.

Para la entrega de la PAC2 se completan los estudios de estado del arte, así como la especificación básica del producto y del prototipo que se quiere resolver.

ID	Task Name	Duration	Start	Finish	Predecessors
1	Evaluación posible temas TFG	8 days	20/06/19	01/07/19	
2	Selección tema TFG	5 days	02/07/19	08/07/19	1
3	Estado del arte productos similares	37 days	09/07/19	28/08/19	2
4	Objetivo, alcance y metodología de trabajo	18 days	29/08/19	23/09/19	3
5	Documento PAC1	3 days	24/09/19	26/09/19	3;4
6	**Entrega PAC1**	0 days	27/09/19	27/09/19	5
7	Especificación básica	4 days	29/08/19	03/09/19	3
8	Listado tecnologías a utilizar	6 days	09/07/19	16/07/19	2
9	Estado del arte de las tecnologías necesarias	61 days	17/07/19	09/10/19	2;8
10	Documento PAC2	3 days	10/10/19	14/10/19	7;9
11	**Entrega PAC2**	0 days	15/10/19	15/10/19	10
12	Selección de las tecnologías a utilizar	3 days	10/10/19	12/10/19	9
13	Diseño conceptual y arquitectónico	2 days	14/10/19	15/10/19	12
14	Diseño detallado y construcción primer prototipo	33 days	16/10/19	29/11/19	13
15	Documento PAC3	7 days	02/12/19	10/12/19	13;14
16	**Entrega PAC3**	0 days	11/12/19	11/12/19	15
17	Tests funcionales del sistema	10 days	02/12/19	13/12/19	14
18	Presupuesto	2 days	16/12/19	17/12/19	17;14
19	Conclusiones	2 days	16/12/19	17/12/19	17;14
20	Memoria - PAC4	12 days	18/12/19	02/01/20	17;18;19;14
21	**Entrega PAC4**	0 days	03/01/20	03/01/20	20
22	Preparación defensa oral - PAC5	7 days	06/01/20	14/01/20	20
23	**Defensa Oral - PAC5**	0 days	14/01/20	14/01/20	22

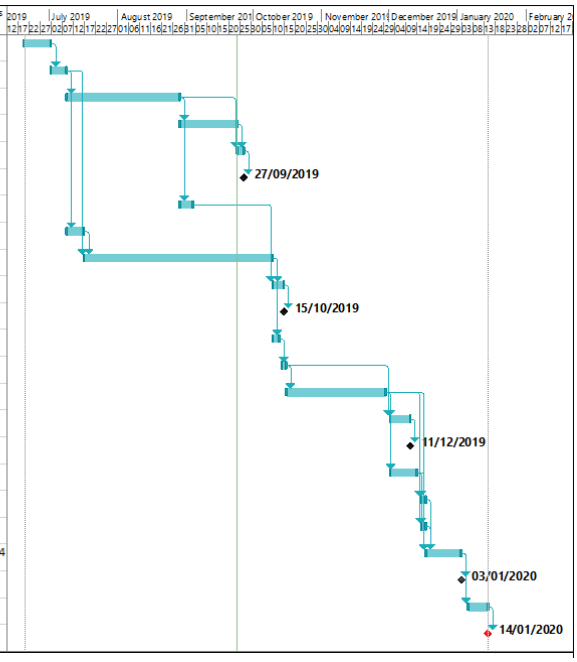

Figura 1 - Diagrama de Gantt.

Para la entrega de la PAC3, que es la más extensa se seleccionan las tecnologías finales aplicadas al prototipo, se realiza un diseño conceptual que incluye la arquitectura del sistema, así como la realización de un diseño detallado y la construcción del prototipo.

Para la PAC4 se completan las pruebas de funcionalidad básica del prototipo, y se realiza el presupuesto y las conclusiones. Además se reserva un cierto tiempo para confeccionar el documento de la memoria y los anexos del TFG.

Por último, se prepara la presentación oral, que desemboca en la PAC5, concluyendo así el TFG.

1.5 Breve sumario de productos obtenidos

Los productos obtenidos han sido:

- TMR: compuesto de una parte hardware constituida por una plataforma de desarrollo electrónico tipo Arduino, un detector inercial y un módulo de comunicaciones de telefonía celular con tecnología LTE, y una parte software desarrollada en el código de la plataforma.
- TMS: una app denominada MotoTracker versión 1.0 con las básicas, y que corre en teléfonos inteligentes con SO Android.
- PEADC: una sencilla base de datos, para manejar los datos de seguimiento de un vehículo, en una plataforma *cloud*.

1.6 Breve descripción de los otros capítulos de la memoria

Además de este primer capítulo introductorio, el presente TFG consta de seis capítulos donde se desarrollarán los distintos aspectos de éste.

En el capítulo 2 se desarrolla el “Estado del arte y estudio de precedentes”. Básicamente el capítulo contiene dos partes principales; la primera dedicada a la descripción y análisis de las prestaciones distintos dispositivos rastreadores GPS diseñados principalmente para el seguimiento de vehículos y una segunda parte dedicada al estudio de las diversas tecnologías que se utilizan o pueden utilizar en el diseño de este tipo de productos, y en particular de prototipos demostradores de la viabilidad tecnológica de este tipo de dispositivos.

En el capítulo 4 se recoge la “Especificación básica” tanto del primer prototipo como de posible productos derivados. Esta especificación básica marca la lista de requerimientos principales a cumplir por el prototipo, tanto a nivel hardware como software.

En el capítulo 5, “Diseño conceptual y arquitectura del sistema”, se describe la arquitectura del sistema, es decir:

- los módulos principales que componen el sistema (tanto a nivel hardware como software)
- las relaciones e interfaces entre los distintos módulos
- las funciones que han de cumplir cada uno de estos módulos
- las alternativas tecnológicas con las que contamos para diseñar cada uno de ellos
- la elección final de los distintos componentes de cada módulo

En el capítulo 6 se lleva a cabo el “Diseño de un primer prototipo”, como ya se ha citado basado en la utilización de alguna plataforma de desarrollo electrónico.

Seguidamente, en el capítulo 7 se describe los “Test funcionales con el prototipo” previamente construido. La idea es demostrar que el sistema funciona, y que cumple con los requerimientos básicos que se le solicitan.

Para finalizar el TFG se completa con el capítulo 8 que contiene el “Presupuesto de ingeniería” del proyecto, y el capítulo 9 con las “Conclusiones” principales que se han obtenido, así como los posibles siguientes estudios a realizar.

2. Estado del arte y estudio de precedentes

Antes de comenzar con el desarrollo arquitectónico, conceptual y detallado del sistema se va a realizar una profunda investigación del estado del arte, en dos ámbitos diferenciados:

- Estudio de los productos que se comercializan en el segmento de los sistemas rastreadores con GPS, aplicados fundamentalmente a las aplicaciones de seguimiento de vehículos como turismos, furgonetas y motocicletas particulares en aplicaciones no profesionales, como pudiese ser el seguimiento de flotas de transporte y reparto.
  Esta primera parte del estudio de precedentes será clave para construir una especificación de un hipotético producto, el cual se pretende desarrollar mediante el prototipo objeto de este proyecto.

- Estudio de las tecnologías disponibles que puede concurrir en el desarrollo de un prototipo demostrador del concepto, tanto a nivel hardware como software.
  Esta segunda parte del estudio del estado del arte será clave para el desarrollado tanto conceptual – y arquitectónico – del prototipo, así como para su posterior desarrollo detallado.

Esta investigación del estado del arte debe ser la base tanto del establecimiento de unas especificaciones detalladas del prototipo y del posible producto comercializable que se derivaría de dicho prototipo, así como del diseño arquitectónico, conceptual y detallado de éste.

2.1 Estado del arte de productos similares

Las aplicaciones y servicios relacionados con la geolocalización, normalmente basado en los sistemas GNSS (*Global Navigation Satellite System*) como GPS, han proliferado de forma evidente, pasando a ser herramientas diariamente utilizadas, por particulares y empresas, incluso desde su terminal de telefonía móvil. Las aplicaciones más comúnmente usadas conocen perfectamente la posición del usuario y le ofrecen, información y servicios relevantes como por ejemplo, servicios comerciales, opiniones de calidad del servicio recibido en el establecimiento que se ha estado, y un larguísimo etcétera.

El sector del transporte y del automóvil en particular, es uno de los grandes objetivos de estas aplicaciones. Waze, Here, Maps Me, e incluso el más generalista Google Maps, se me han incorporado al día de día de muchas personas cuando quieren conocer cómo llegar a una dirección o lugar desconocido. Estas aplicaciones informan sobre el estado de congestión de las vías, las mejores rutas, los tiempos previstos de viaje, entre otros servicios de alto.
Un segmento de productos que está creciendo de forma significativa, es el de los dispositivos de rastreo, para de vehículos, cosas en general, animales y personas.

En un artículo de Reuters consultado se afirma que los sistema de rastreo mediante GPS van a experimentar un crecimiento sostenido medio de 11.9% entre los años 2017 y 2023 hasta alcanzar una cifra de negocio de 2530 M$.

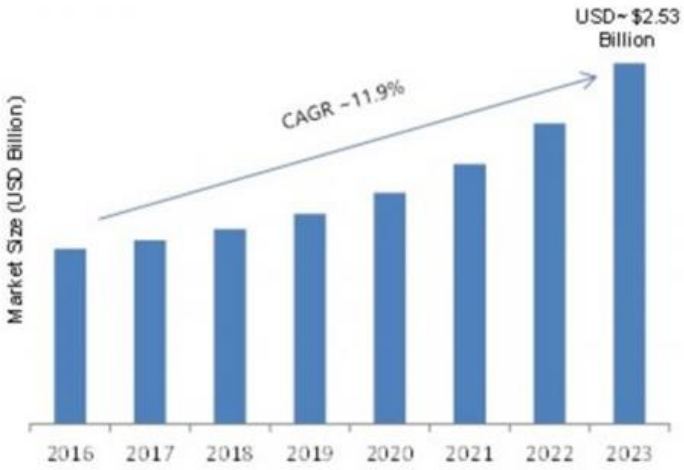

Figura 2 – Previsión de crecimeinto del sector del rastreo GPS
(fuente: www.reuters.com)

Algunas de las compañías principales que están participando de este negocio son Tomtom International Bv (Holanda), Sierra Wireless Inc. (Canadá), Atrack Technology Inc. (Taiwán), Concox Information Technology Co. Ltd. (China)
En el artículo se indica que América del Norte representa el mayor mercado de este tipo de dispositivos, debido fundamentalmente al fuerte crecimiento de los sistemas de seguridad personal. Geográficamente, el mercado global de sistemas de dispositivos de rastreo GPS se ha segmentado en cuatro regiones principales, como América del Norte, Europa, Asia Pacífico y el resto del mundo. La región de América del Norte exige la mayor parte en el mercado mundial de sistemas de dispositivos de rastreo GPS debido a la fácil adopción de tecnología avanzada y la utilización extensiva de estos dispositivos en los sistemas de respuesta de emergencia personal GPS con una creciente demanda de seguridad personal en esta región. En cualquier caso los niveles de demanda y de crecimiento del resto de regiones hace que se trate de un sector muy interesante a nivel global.

Realizando una búsqueda simple con las palabras "*GPS tracker for motorbike*" en internet aparece una cantidad importante de productos que responden a esta descripción. Para entender mejor como funcionan se han estudiado con bastante detalle el funcionamiento y las prestaciones de algunos modelos, entre los que podemos destacar los siguientes:

- TK102, con precio que oscila entre los 25 € y los 40 € (comercializado como OEM por distintas marcas)

Figura 3 – TK102B (fuente: www.amazon.es)

- RF – V8, con un precio de unos 25 $ (comercializado como OEM por distintas marcas)

Figura 4 – RF – V8 (fuente: www.amazon.es)

- Sinotrack GPS tracker 2019

Figura 5 – Sinotrack (fuente: www.amazon.es)

- Atlantis moto modelos Go y Premium, de la compañía española Atlantis, y con precios que oscilan entre los 139 € y 239 €.

Figura 6 – Modelos de Atlantos moto
(fuente: www.atlantis-technology.com/productos/)

- LEKENI TK905, con un precio de unos 55 €.

Figura 7 – LEKENI TK905
(fuente: www.amazon.es)

- Invoxia GPS, con un precio de 99 €. Tiene como particularidad que no utiliza tarjeta SIM sino una red internacional de datos como es Sigfox, que da cobertura en toda Europa y una parte del continente americano.

Figura 8 – Invoxia GPS
(fuente: www.invoxia.com/es/gps-tracker)

- XCsource AH207: con un precio de unos 25 €, cuenta con la particularidad de no contar con batería propia.

Figura 9 – XCSource AH207
(fuente: www.xcsource.com)

- Goraz: con un precio de unos 50 €, permite la desconexión del automóvil a voluntad del usuario en caso de robo.

Figure 10 – Rastreador Goraz
(fuente: www.kennelclubgrooming.com/)

Con el objetivo de no hacer este capítulo excesivamente largo, se va a comentar con más detalle un par productos que pueden ser representativos de la mayoría de las especificaciones y prestaciones de muchos de ellos.

En general podemos destacar que:

- los productos más sencillos se alimentan solo de la batería, que ha de ser extraída para ser recargada. Tienen la ventaja de no consumir la batería del vehículo, además de ser muy sencillos de instalación, y la desventaja de su autonomía y de tener que ser extraídos del vehículo.

- La gran mayoría no requieren de conexiones adicionales con el vehículo, aunque algún modelo como Sinotrack, Goraz y el Atlatis Moto que permiten actuar sobre el vehículo impidiendo su arranque a voluntad del usuario que controla el rastreador.

- La gran mayoría de los productos analizados utilizan las tecnologías de comunicación GPRS y GSM, sin necesidad de recurrir a soluciones 3G o 4G debido al bajo nivel de datos que necesitan comunicar. El rastreador de Invoxia utiliza la tecnología de comunicación para emisores de baja potencia diseñada para aplicaciones IoT desarrollada por la compañía francesa Sigfox, lo cual le permite evitar el uso de una tarjeta de telefonía móvil prepago o de contrato, en contra cuenta con algunos problemas de cobertura.

- Casi todos ofrecen alguna aplicación para poder controlar y comunicar con el dispositivo desde un teléfono móvil. Algunos usan su propia aplicación mientras que otros se conectan a plataformas compartidas.

- La mayoría ofrecen los siguientes servicios:

    - detección de la posición del vehículo bajo demanda desde el móvil (mediante una llamada, un mensaje de texto, o en los más sofisticados pulsando una tecla en una aplicación con una usabilidad mejorada). La

petición de posición puede ser única o mantenida en el tiempo con una frecuencia programable.

- Envían mensajes de emergencia si el vehículo se ha desplazado de forma inesperada, y en el caso de las motocicletas si se ha volcado de forma inesperada.

- Permiten establecer una función de barrera, de tal manera que se envíe una señal al usuario si el vehículo ha salido del perímetro de esta.

- La información es enviada vía SMS, indicando simplemente las coordenadas del vehículo, un enlace a Google Maps, y los más usable utilizan un interfaz con un programa de mapas sin necesidad de operaciones intermedias.

- Permiten conectar y desconectar alarmas.

- Permiten conocer la velocidad del vehículo, y permite emitir alarmas en el caso de que el vehículo supere el umbral seleccionado.

- Permiten conocer el estado de carga de la batería interna, y en ocasiones la del vehículo.

- Algunos permiten la desconexión remota.

- Algunos tienen un botón de SOS de tal manera que al ser pulsado realiza una llamada o SMS conteniendo las coordenadas actuales del rastreador, a un conjunto de teléfonos para solicitar ayuda.

- Muchos de ellos envían las coordenadas a plataformas en el *cloud* que pueden ser accedidas posteriormente desde internet por parte del usuario.

- Los más sofisticados empiezan a añadir aplicaciones "sociales" donde los usuarios pueden compartir rutas, información, y establecer una cierta red social.

En las siguientes páginas se van a describir dos modelos, el TK102B que puede ser considerado como un rastreador representativo de los de bajo coste, y las soluciones de Atlantis Moto, que podríamos considerarlo de gama alta. El resto de los productos disponen de prestaciones englobables en este rango.

2.1.1 Análisis del rastreador TK102 B

Se trata de un dispositivo de pequeñas dimensiones (como una caja de goma de mascar), que se alimenta por batería recargable mediante conector USB, y no requiere de ninguna instalación en el vehículo que se quiera rastrear. El interfaz se realiza a través de un teléfono móvil externo.

No requiere el pago de ninguna cuota, excepto los gastos de mantenimiento de la línea de SIM que depende de la compañía seleccionada por el usuario.
Dispone de llamada de emergencia en caso de accidente o necesidad de asistencia, donde las coordenadas son enviadas a un teléfono móvil designado.

Figura 11 - Cittatrend TK102 .
(Fuente: www.comfortsurf.com/product/gps_tracker_TK102B)

Sus características principales son:

- Localización de alta precisión de todo tipo de automóviles, personas, animales y otros de dispositivos móviles.
- Masa : 50 gr
- Dimensiones: 64 x 46 x 17 mm (ver figura 12)
- Material de la carcasa: plástico PVC
- Chip de GPS: SIRF III con posibilidad de lectura de hasta 20 satélites
- Sensibilidad de la señal de GPS: -159 dBm
- Precisión del GPS aproximada: 5 m
- Tiempo de establecimiento de la señal de GPS (*time to first fix*): entre 1 y 45 s
- Trabaja con 4 bandas: 850/900/1800/1900MHz. Red: GSM/GPRS
- GSM/GPRS: SIMCOM sim340 -159 dBm
- GPRS puede enviar señales continuamente mientras que GSM solo cada 20 s
- Rango de condiciones operativas:
    - Temperatura de funcionamiento: -20 a 55 grados
    - Rango de humedades: 5% a 95% sin condensación
    - Altitud máxima: 18000 m
    - Velocidad máxima: 515 m/s
    - Aceleración máxima: 4g
- Batería de iones de litio cargable e intercambiable 3.7 V – 800 mAh
- Autonomía en espera: 80 horas
- Voltaje de entrada del cargador desde el vehículo: 12 a 24 V DC
- Voltaje de entrada del cargado a la red: 110 a 230 V AC
- El paquete incluye:
    - 1 x localizador TK102B

    - 1 x cargador
    - 2 x batería
    - 1 x cable
    - 1 x adaptador europeo
    - 1 x tela pequeña de limpieza
- Precio en torno a unos 35€
- Está visto disponible con varios nombres comerciales lo cual indica que se trata de un OEM. En la figura se muestra el producto con la marca Cittatrend.

Figura 12 - Dimensiones externas (fuente: www.comfortsurf.com/product/gps_tracker_TK102B)

Es un dispositivo sencillo e intuitivo. En la siguiente figura se ve sus puntos de interés:

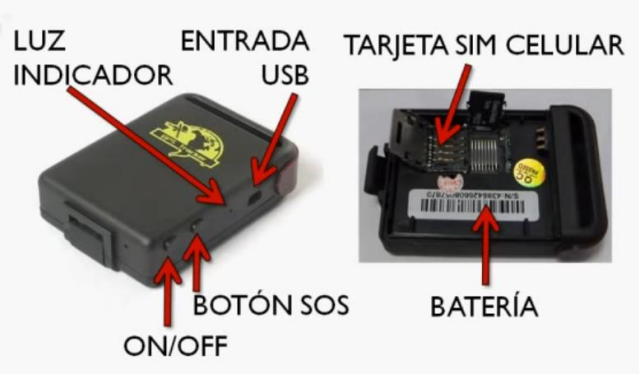

Figura 13 - Puntos de interes del TK 102 B (fuente: www.localizadorgpstracker.com.mx)

Como puede verse cuenta con:

- un interruptor de encendido y apagado (ON/OFF)
- un LED para indicar su encendido
- una entrada micro USB para el cargador de la batería
- un botón de llamada de emergencia (SOS)
- un *slot* para la inserción de la tarjeta SIM
- la ubicación para la batería de litio

Para interactuar el usuario lo hará mediante llamadas desde un teléfono móvil al número de la SIM que se ha montado en el dispositivo. El sistema contestará con un mensaje SMS con las coordenadas donde se encuentra el dispositivo, o

bien con un enlace que puede ser abierto con Google Maps (u otro sistema de mapas digitales) donde se mostrará de forma gráfica la ubicación.

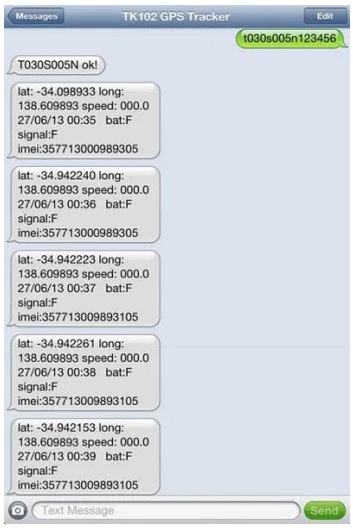

Figura 14 - Comunicación mediante mensajes SMS
(fuente: www.solidgoldsecurity.com.au/tk102-gps-tracker-command-instructions/)

Figura 15 - Comunicación mediante enlace a un sistema de mapas como Google Maps
(fuente: www.solidgoldsecurity.com.au/tk102-gps-tracker-command-instructions/)

Además, cuenta con unas funcionalidades adicionales que han de ser destacadas:

- función *Auto-Tracking* donde se puede establecer un periodo fijo de envío automático de coordenadas.
- Autorización para poder enviar la información hasta a 5 teléfonos celulares.
- Función SOS: al presionar un botón el dispositivo envía las coordenadas actuales a un teléfono celular designado.
- Función *Geo-Fence* (o cerca geográfica): el dispositivo envía una alerta en el caso de que se haya salido de un área geográfica predeterminada por el usuario.
- Límite de velocidad: el dispositivo envía una alerta si se sobrepasa una determinada velocidad previamente definida.
- Alerta de movimiento: el dispositivo envía una alerta si se ha movido unos 200 m.
- Alerta de batería baja.

-El dispositivo se apaga envía una señal de alerta antes de su total desconexión.

2.1.2 Análisis del rastreador Atlantis - Moto

A diferencia de lo que ocurre con el rastreador descrito en el apartado anterior Atlantis – Moto no publica información tecnológica detallada de sus dispositivos. Pero sí se ha podido encontrar mucha información de sus prestaciones, así como de su aplicación, que podemos considerarla de las mejores del mercado. Por tanto, este apartado se centrará más en el análisis de las prestaciones de un rastreador de altas prestaciones, así como en el funcionamiento de algunas de las funcionalidades principales de su app, y que han de servir como base para el desarrollo de un prototipo básico.
Las características principales que Atlantis – Moto publica de sus dispositivos son:

- Aviso por corte/sabotaje/descarga batería principal de la moto.
- Aviso por activación de alguna entrada digital.
- Posibilidad de configurar una velocidad máxima, y lanzar un aviso por exceso de velocidad.
- Alarma de desplazamiento no autorizado: el localizador detecta movimiento sin llave de contacto.
- Alerta de *roaming*.
- Función parking (zona dinámica): generación automática de una zona de seguridad cuando quite la llave de contacto. La alarma GPS M1 detecta salida de esta zona sin llave de contacto.
- Alerta de sabotaje/desconexión de antena GPS.
- Se pueden configurar hasta a 5 números destinatarios de los SMS de aviso/alerta.

Para entrar en la app se accede mediante el uso de un nombre de usuario y una contraseña.
La pantalla principal nos indica la ubicación de la moto en un mapa de Google Maps, el nombre y la foto del icono del vehículo se puede personalizar.
Dispone de un menú de configuración muy completo, donde el usuario puede elegir las distintas opciones de protección ofrecidas.

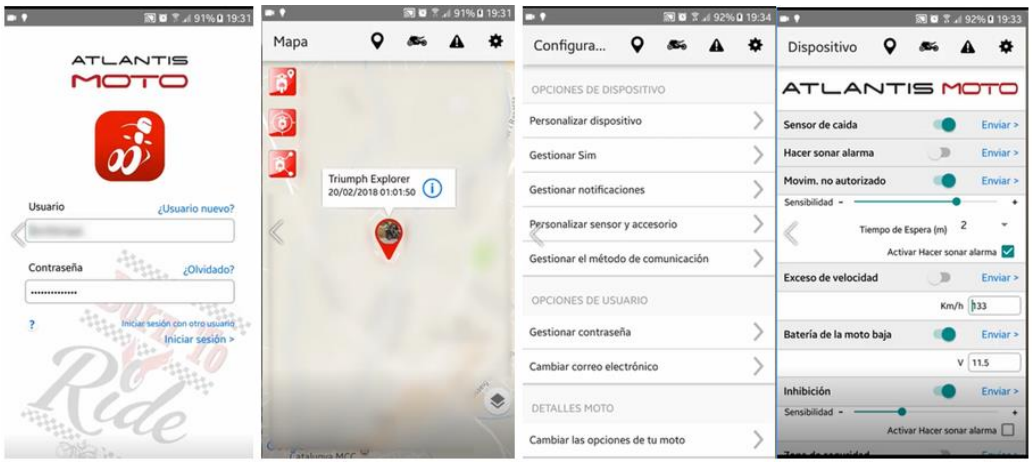

Figura 16 – Pantallas de acceso, principal y de configuración de Atlantis-Moto (fuente: Xavi Fábregas).

Dispone de opciones para gestionar la SIM que se esté utilizando en el dispositivo rastreador. Del mismo modo, permite configurar la vía de comunicación que seguirán las distintas informaciones. Las opciones son vía SMS o datos, debido a los distintos costes en función del contrato con la compañía suministradora de la SIM. Permite configurar a quien se le va a enviar las alertas en caso de que el piloto pueda estar accidentado.
Atlantis - Moto está introduciendo el concepto de red social en su aplicación, para compartir ubicaciones, rutas, etc., ofrece pantallas de configuración para determinar con quién se comparte la información.

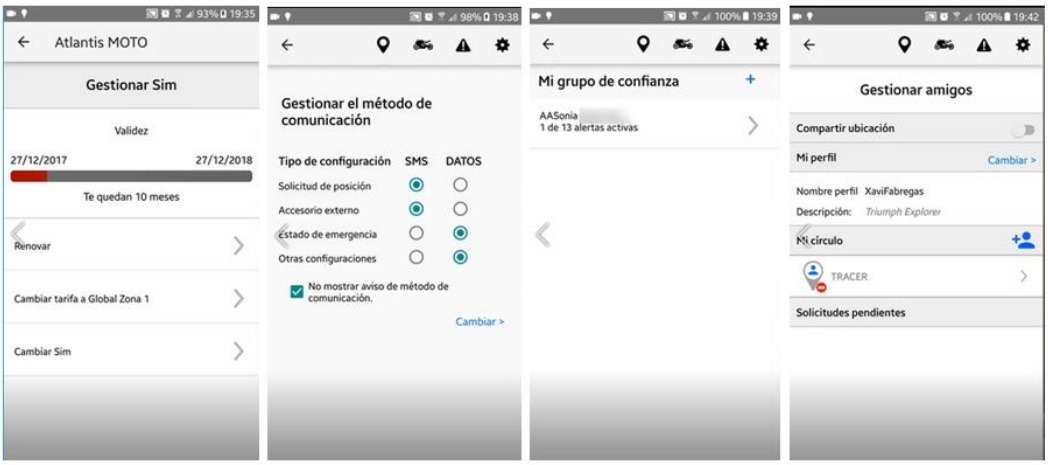

Figura 17 – Pantallas de gestión de comunicaciones, grupos de confianza y amigos (fuente: Xavi Fábregas)

Puede establecerse una zona de seguridad alrededor de la moto, de tal manera que si la moto sale de ella se genera un alerta.
La aplicación dispone de una pantalla de eventos donde se indica todas las incidencias acaecidas que pueden ser consultadas, clickando encima de ellas.

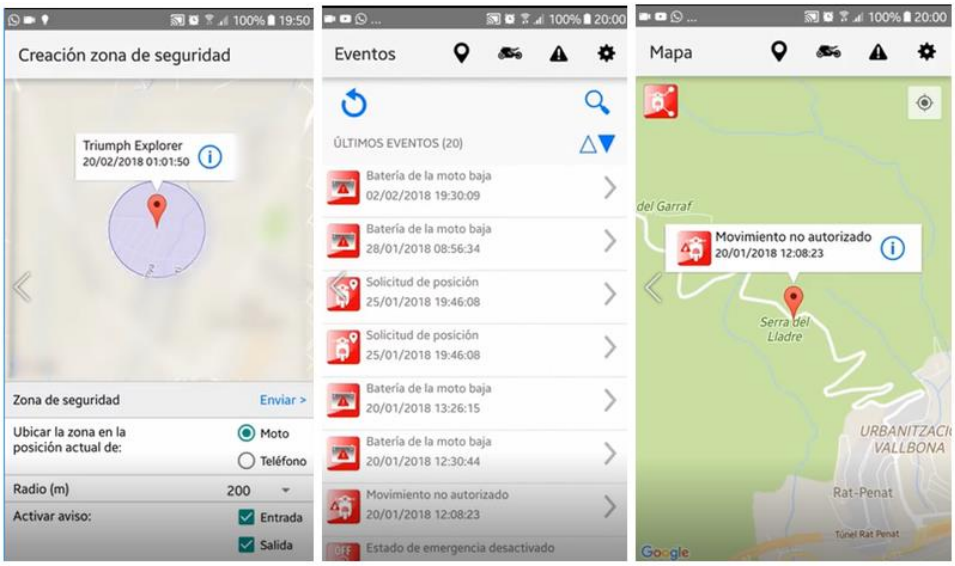

Figura 18 – Pantallas de establecimiento de zona de seguridad, listado de eventos y alarma por movimiento no autorizado (fuente: Xavi Fábregas)

Es muy interesante destacar que la app de Atlantis – Moto dispone de sistema de detección de inhibiciones, generando alertas correspondientes, así como de sabotaje de batería. Estos son mecanismos muy utilizados por los ladrones a la hora de sustraer una motocicleta, y por tanto estas características son clave para una buena detección del robo.
Este tipo de funcionalidades no formarán parte del alcance de este TFG.

2.2 Estado del arte de las tecnologías concurrentes

Después de haber analizado varios productos (sistemas) de rastreo y seguimiento de vehículos se puede plantear una arquitectura genérica que permita identificar los distintos bloques o subsistemas funcionales que componen el sistema. Esta arquitectura genérica es una representación conceptual general que contiene los distintos módulos (bloques funcionales principales o subsistemas) así como las comunicaciones e interfaces entre ellos, que permiten ofrecer la funcionalidad completa del sistema.
Es probable que ninguno de los productos presentados en el apartado anterior responda de una forma precisa a dicha arquitectura genérica, pero ésta sí sirve para explicar el funcionamiento aproximado de la mayoría de ellos.

Esta arquitectura genérica permite listar las distintas tecnologías que se han de analizar y estudiar para posteriormente poder tomar decisiones pertinentes para el obtener un diseño optimizado del sistema. El diseño conceptual y detallado del prototipo habrá de centrarse en la optimización de los objetivos que se plantean para el sistema, más que en la optimización, por separado, de los diversos bloques que lo componen.

Arquitectura genérica de los sistemas de rastreo y seguimiento de vehículos

En la siguiente figura se muestra una posible síntesis de la arquitectura de estos sistemas de comunicación.

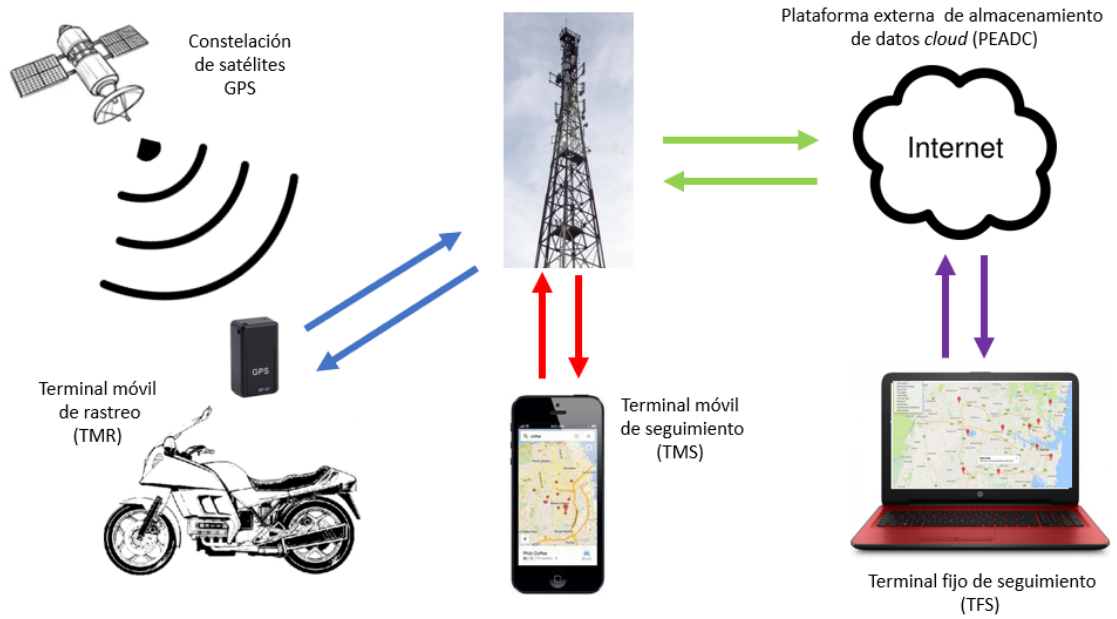

Figura 19 - Arquitectura general de sistema rastreadores.

Los subsistemas principales que la componen son:

1. Terminal móvil de rastreo (TMR), que es componente que va montado en el vehículo que ha de ser rastreado y protegido. Podríamos decir que es el elemento central de la arquitectura, y el que más determina los distintos casos de uso del producto como sistema. Cuenta con elementos tanto hardware como software para desarrollar todas sus funcionalidades, entre las que podemos destacar:

    - dispositivo de captación y tratamiento de señales satelitales GPS, cuenta con varios componentes hardware y software. A nivel hardware contiene:

        - las antenas satélites para la recepción de las ondas del sistema GNSS – en nuestro caso GPS –
        - la electrónica de tratamiento de las mismas, para computar los parámetros de salida, como *fix* (que indica si la comunicación con los satélites es la correcta para para un resultado de posicionado fiable), latitud, longitud, altura, velocidad, etc.
        - A nivel software contiene el protocolo que maneja tanto la recepción de los comandos como la entrega de los resultados al central de control del TMR.

    - Unidad de comunicación móvil, cuya función principal es la comunicación vía radio con los elementos externos como el TMS y el PEADC para envío de información tanto de alarmas, como la información de sensores, de las variables de estado y, obviamente, de las coordenadas del vehículo. A nivel hardware está compuesto tanto por la antena (o antenas) de telefonía celular, como su electrónica de control, así como el software correspondiente para su control. Ha de incorporar la SIM de la compañía que se elija para enviar la información.

    - Sensores de detección de aceleraciones y de vuelco, los cuales han de servir para disparar la emisión de llamadas de emergencia, y de posible situación de movimiento o caídas no deseados.

    - Dispositivo para la gestión efectiva de la energía para evitar la excesiva descarga de la batería del vehículo y el disparo de emergencia en caso de baja batería, tanto del vehículo como de las baterías del TMR.

    - Sistemas de desconexión eléctrica de algunos de los componentes del vehículo para facilitar su inmovilización en caso de posible robo.

- Placa electrónica central para integrar todos los dispositivos mencionados, y en donde, obviamente, se ejecutará el software de gestión.

2. Plataforma *cloud* (PEADC) para recoger toda la información enviada por el TMR, y que posteriormente puede ser consultada tanto por el TMS y TFS.

3. El TMS, que en general va a ser un terminal móvil, con sistema operativo Android o iOS. En el presente TFG solo se pretende desarrollar el software para el sistema operativo Android.

2.2.1 Tecnologías para el desarrollo de un prototipo demostrador del TMR

El TMR va a ser principalmente un sistema electrónico digital, aunque con mucha componente analógica que ha de manejar las distintas señales citadas en el punto anterior.

Para el desarrollo de un prototipo demostrador, en el que se pretende realizar una pequeña inversión parece más que razonable una plataforma de desarrollo electrónico tipo Arduino. Desde su aparición comercial, hace ya unos años, Arduino ha permitido el desarrollo de múltiples proyectos electrónicos digitales de distinta índoles, en los ámbitos de los estudiantes, *hobbistas* y también profesionales. En la actualidad Arduino cuenta con una amplia oferta de placas, así como un gran conjunto de accesorios compatibles - denominados *shields* -, pero también han aparecido serios competidores con distintos niveles de prestaciones, como pueden ser Raspberry Pi y BeagleBone Black.

En las siguientes líneas se describe las principales características de cada uno ellos, para justificar la posterior elección del dispositivo utilizado en el prototipo.

Arduino:

Se trata de una placa de desarrollo que integra un microcontrolador, y que puede adquirirse a muy bajo coste, en torno a los 35 €. Cuenta con las siguientes características principales:

- Alimentación con CC entre 5 y 12 V (en función del modelo), bajos consumos en general.
- Cuenta con un número importante de entradas y salidas tanto digitales como analógicas.
- Cuenta con temporizadores internos.
- Interfaz serie I2C y USB.
- Dispone de una gran base de *shields* compatibles, GPS, telefonía celular, Bluetooth, Wifi, acelerómetros, sensores de todo tipo, actuadores diversos, y un larguísimo etcétera.
- No requiere de sistema operativo.

- El lenguaje de programación está basado en C y se compila y monta externamente, antes de ser descargado en la memoria de programa del microcontrolador.
- Existe una gran variedad de modelos enfocados a distintas aplicaciones.
- Cuenta con un enorme conjunto de librerías así como muchos programas disponibles, y una gran base de desarrolladores de código abierto entusiastas dispuestos a ayudar al colectivo.

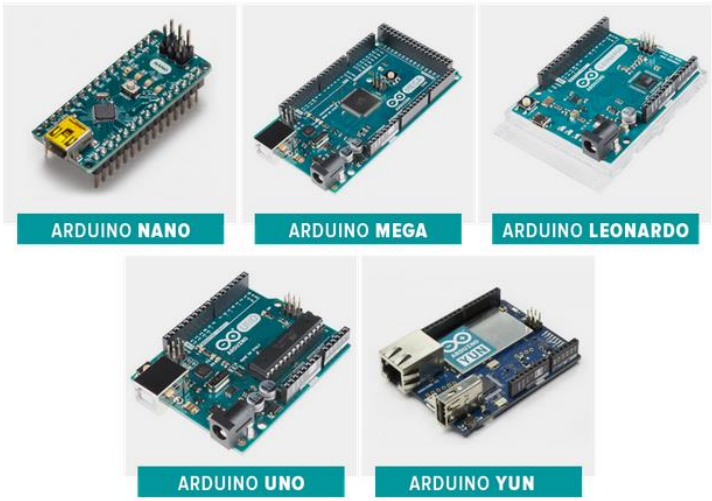

Figura 20 - Algunos miembros de la familia Arduino (fuente: www.arduino.cl)

Entre sus inconvenientes podemos citar:

- Cuenta con una memoria muy pequeña (unos 2 KB de RAM y 32 KB de flash) lo cual puede causar problemas en algunos proyectos de una cierta complejidad.
- Este problema se ve acentuado con el uso de *shields* que requieren de librerías que pueden llagar a pesar bastante.

Raspberry Pi:

Este dispositivo es un microcomputador integrado en una única placa de dimensiones muy reducidas, y a un precio realmente reducido, ya que ronda los 50 €. Utiliza un sistema operativo Linux con un interfaz similar a Ubuntu, utilizando ventas, y multiproceso.
Entre sus ventajas se puede destacar:

- Puede ejecutar aplicaciones visuales como cualquier PC, en múltiples campos, correo electrónico, administración del PC, navegadores de internet, programas de audio, video, programas de matemáticas, editores de texto, hojas de cálculo, y un larguísimo etcétera.
- Dispone de conectores para pantalla como HDMI, además de USBs para conectar teclado, ratón, *pen drive*, etcétera, así como conexión ethernet.

- Entrada para tarjetas SD.
- Su memoria y capacidad de cálculo (cuenta con un microprocesador interno y no con un microcontrolador) es, por tanto, muy superior a la de Arduino.
- Dispone de entradas y salidas digitales (GPIO), a nivel señales individuales.
- Cuantas con salidas con PWM.
- Se puede programar en varios lenguajes de programación, y en particular en Phyton.
- Funcionando como PC de sobremesa tiene unas prestaciones muy destacables.

Figura 21 – Raspberry Pi
(fuente: www.wikipedia.es)

Entre sus inconvenientes se puede indicar:

- No dispone de lectura de señales analógicas, se ha de realizar mediante un conversor A/D.
- A pesar de que cuenta con muchos accesorios no hay tantos como en la familia Arduino.
- Existe un número importante de librerías y programas en general disponibles para resolver muchos problemas, pero la comunidad no está tan extendida ni es tan dinámica como la de Arduino.
- Sus GPIOs no pueden manejar tanta corriente como Arduino, requiriendo por tanto, etapas de amplificación externa.
- Consume mucho más que Arduino, lo cual limita su uso en algunas aplicaciones (en especial aquellas que son móviles y alimentadas por batería)

BeagleBone Black:

Se podría decir que es el competidor directo de Raspberry Pi, ya que es un microcomputador, con un precio similar, en torno a unos 50 €. Sus características técnicas son muy similares a Raspberry Pi, en lo que se refiere a interfaces de comunicaciones, sistema operativo, y aplicaciones que puede ejecutar. Pero cuenta con algunas características destacadas sobre éste:

- Cuenta con el doble número de pines de entrada y salida que Raspberry Pi.
- Dispone de pines para hacer lectura analógica (como Arduino).

Entre las desventajas:

- La experiencia como PC de sobremesa es peor que Raspberry Pi.
- Existe mucha menos literatura y mucho menos software disponible que para su competidor -aunque mucho de este software es casi portable entre ambas plataformas - y muchísimo menos que para Arduino.

Figura 22 – BeagleBone Black
(fuente: www.beagleboard.org)

A partir de la información analizada referente a las distintas opciones, se ha decidido seleccionar para el desarrollo del prototipo la plataforma Arduino. Los motivos fundamentales han sido:

- Existen distintos *shields* para las necesidades del prototipo:
  - Recepción de señal GPS
  - Comunicación con tecnología celular
  - Gran variabilidad de sensores de aceleración, de ángulo con respecto a la gravedad, de voltaje, etc.
  - Actuadores de mayor potencia, tanto electrónicos, como relés.
- Posibilidad de interactuar con plataformas IoT
- Bajo consumo.
- Gran cantidad de información disponible.

El mayor riesgo que se corre con esta elección es la limitación de memoria, problema que debe ser manejado con cuidado durante el desarrollo del proyecto.

Obviamente el TMR ha de incluir, además de la placa central que gestiona el conjunto de información, módulos para:

- Recepción de la señal de GPS
- Envió de la información vía radio, tanto al TMS como al PEADC.
- Sensores de detección de movimiento y vuelco.

Estás tecnologías serán investigadas en los siguientes apartados.

Inicialmente no se van a buscar componentes que puedan interactuar con el vehículo para desconectarlo en caso de evidencia de robo. Básicamente se ha

decidido dejarlo fuera del alcance del proyecto, aunque su complejidad incremental no es excesiva.

2.2.2 Tecnologías de recepción de señal GPS y de comunicación celular

En el mercado existen muchas placas que pueden realizar esta función, y que pueden trabajar de forma fácil con plataformas de desarrollo como Arduino, que pobremente será el diseño escogido para realizar el prototipo.

Se analizado varios modelos pero hay tres que queremos destacar:

- Adafruit Ultimate GPS Breakout v3: se trata de un módulo que proporciona señal de GPS a distintas plataformas de desarrollo entre ellas a Arduino. El dispositivo está construido alrededor del chip MTK3339, que puede llegar a monitorizar hasta 22 satélites con una alta sensibilidad que permite trabajar con señales de -165 dBm. Dispone de una antena externa, y puede actualizar su localización hasta 10 veces por segundo. La alimenta se realiza con voltajes de 3.3 a 5 V y con bajos consumos de 20 mA durante la navegación. Dispone de LEDs de señalización de estatus, y tiene un precio significativamente para un prototipo que oscila entre los 35 y los 40 €.

Figura 23 – Adafruit Ultimate GPS (fuente: www.adafruit.com)

- Arduino SIM 808 GPS/GPRS/GSM para Arduino. Visto que las prestaciones de GPS de Adafruit pueden satisfacer perfectamente las necesidades del prototipo, se ha realizado una búsqueda de componentes que puedan realizar la comunicación celular para el envío de las alarmas, información a transmitir, así como la recepción de los comandos necesarios. Inicialmente, debido al bajo volumen de datos que se han de intercambiar, y que no se requiere una bajísima latencia, casi cualquier sistema de comunicación puede solucionar las necesidades del proyecto.
  Un módulo muy popular encontrado es el 808, del que existe mucha información y muchos ejemplos resueltos. Además cuenta con la particularidad de que incluye un sistema de recepción de señales de GPS, lo cual podría permitir el uso de una *shield* para ambas funciones (GPS y comunicación celular) simplificando el dispositivo. Además todo esto a un precio muy asequible alrededor de 40 €. Buscando información se ha encontrado que existe una nueva versión la 908, que se pasa a describir seguidamente.

Figura 24 – SIM 808 (fuente: www.dfrobot.com)

- SIM 908 GPS/GPRS/GSM para Arduino. Se trata de una placa con cuádruple banda que funciona en las frecuencias EGSM 900MHz, DCS 1800MHz y GSM850 MHz, y PCS 1900MHz. Dispone de una antena GSM flexible de 85 mm (conector u.FL). Se controla mediante comandos AT GSM07.07, 07.05 y comandos AT mejorados SIMCOM.
  Además contiene un dispositivo de captura y tratamiento de señales GPS, incluyendo una antena SMD de alta ganancia para GPS.
  Su precio es significativamente superior a la versión anterior, rondando los 80 €.

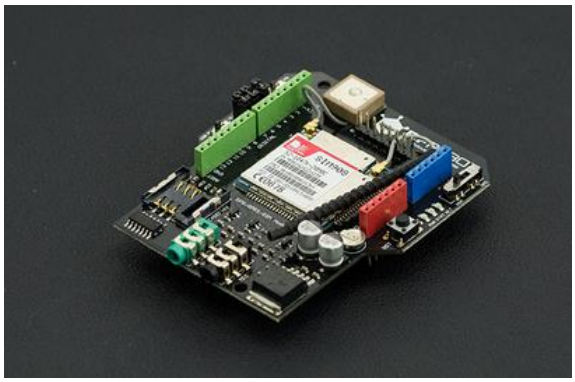

Figura 25 – SIM 908 (fuente: www.dfrobot.com)

Un listado más detallado de sus características es el siguiente:

- Alimentación: 6 a 12v, con unos consumos de 100 mA a 7 V, trabajando en modo GSM.
- Cumple con los estándares:
  - Class 4 (2 W @ 850/900 MHz).
  - Class 1 (1 W @ 1800/1900MHz).
- Quad-Band 850/900/18001900MHz.
- GPRS multi-slot clase 10, GPRS estación móvil clase B y compatible con GSM phase 2/2+.
- Control vía comandos AT (GSM07.07 ,07.05 y los comandos avanzados SIMCOM).
- Puede realizar conexiones punto a punto MO y MT.
- Posibilidad de enviar SMS en modo broadcast.
- Dispone de los modos de texto y PDU.
- Soporte para la tarjeta SIM incluido.

- Conectores para micrófono y altavoz.
- Navegación por satélite GPS.
- Antenas de alta ganancia embebidas SMD para GPS y GSM.
- Receptor de 42 canales, GP.S L1 C/A *code*, High-performance STE *engine*.
- Dispone de una muy buena sensibilidad con señales de -160 dBm para realizar seguimiento y de -143 dBm para el establecimiento de la recepción.
- Tiempo de establecimiento de señal (*Time-To-First-Fix*) en torno a los 30 s (*cold start*) y en torno a 1 s (*hot start*).
- Precisión en la posición horizontal < 2.5m CEP.
- Consumos del GPS (GSM modulo desconectado): 77mA durante la adquisición de la señal y 76 mA durante el seguimiento.
- Programable desde Arduino vía USB.
- Indicadores de estatus LED.
- Vías conductoras de la placa recubiertas de oro.
- Tamaño: 81x70mm.

Botlectics SIM7000E para Arduino y otras plataformas. A pesar de que está desarrollada por otra empresa, esta placa puede considerarse una evolución de las anteriores, y está mucho menos extendido, y hay mucha menos información disponible. Se trata de una placa de comunicaciones que incorpora las tecnologías CAT-M1 y NB-IoT, pero manteniendo los protocolos 2G (GSM/GPRS) y 2.5G (EDGE), incluyendo también un sensor de señal GPS y GLONASS.

Figura 26 – SIM7000 (fuente: www.botletics.com)

Cuenta con un muy bajo consumo y está pensado para ser utilizado con baterías o pequeñas placas solares. Incluye una antena plana adhesiva. Existen distintas versiones para las distintas regiones, en el caso de este proyecto se ha escogido la versión E para Europa. En la figura anterior se muestra la *shield* SIM7000 montada en una Arduino UNO
Un listado de especificaciones detalladas es:

- Compatible con Arduino Uno, Arduino Mega, and Arduino Leonardo o prácticamente cualquier 1.8-5V microcontrolador como ESP8266, ESP32 y Raspberry Pi.
- Soporte de las tecnologías LTE CAT-M1 y NB-IoT además de 2G (GSM/GPRS) y 2.5G (EDGE). Con un canal de subida de 375 Kbps y uno de bajada de 300 Kbps.
- Bajos consumos:
    - Cuando está en modo apagado: 7uA
    - En modo de bajo consumo: 9uA
    - Modo *Sleep*: 1mA
    - Cuando el GNSS está activo: 30mA
    - Con comunicación LTE activada (bandas LTE B12): 167mA @ 23dbm, 109mA @ 10dbm
- Conexión para batería LiPo de 3.7 V.
- La placa cuenta con una línea de carga de la batería.
- Soporta el envío y recepción de mensajes de texto SMS.
- Soporta los protocolos TCP/IP, incluyendo MQTT, PPP, POP3 para envío de correo electrónico y también los protocolos FTP, HTTP y SSL.
- Sistema multi-GNSS con soporte de GPS, GLONASS, Galileo, entre otros, con precisiones de unos 2.5 m.
- Incluye un sensor de temperatura MCP9808 de alta precisión de unos 0.25° y con un canal de comunicación por el interfaz I2C.
- Conector Micro USB para la utilización de comandos AT.
- LEDs para indicación del estatus de la placa.
- Conectores u.FL para las antenas tanto para los GNSS y para LTE.

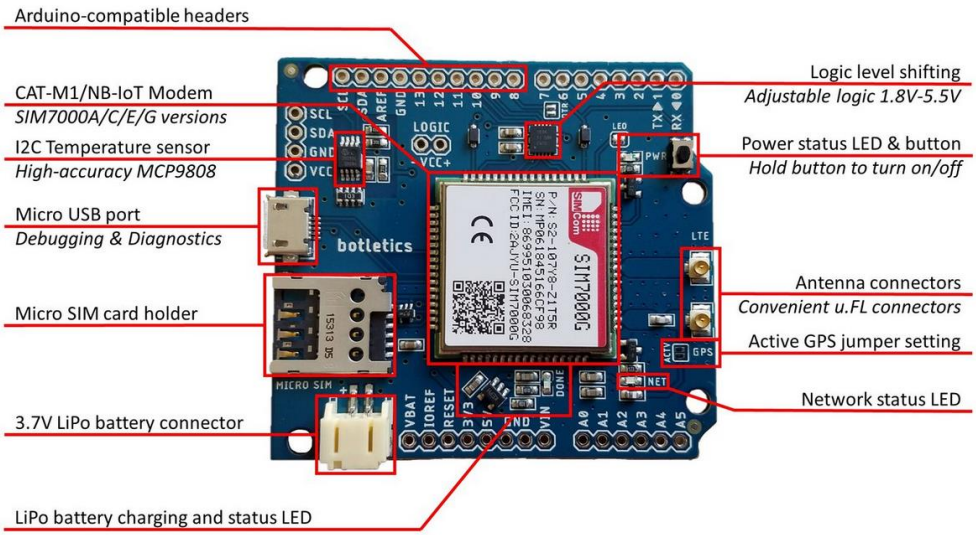

Figura 27 – SIM7000 (fuente: www.botletics.com)

Inicialmente esta última *shield* es la escogida para llevar adelante el proyecto, ya que debería satisfacer todas las especificaciones requeridas, además de soportar las tecnologías LTE CAT-M1 y NB-IoT. Ambas tecnologías de bajo consumo - *Low Power Wide Area* (LPWA) - desarrolladas para aplicaciones IOT. Ambas cuentan con un ancho de banda bajo que conectan a internet

dispositivos que necesitan transmitir pequeñas cantidades de datos, a bajo coste y con una alta duración de la batería.

2.2.3 Tecnologías para detección de aceleraciones e inclinación

En el sistema se requiere detección de movimiento y vuelco de la motocicleta. Para ello se debe disponer de sensores de este tipo. Investigando se ha encontrado que dentro del ecosistema Arduino existen acelerómetros, que permite detectar la aceleración en cualquiera de los tres ejes, giróscopos que detectan las velocidades de giro en las tres direcciones e IMUs – unidad de medición inercial – que dan señales que permiten conocer el estado de movimiento y la inclinación en la que se encuentra, básicamente haciendo uso tanto de acelerómetros como de giróscopos, y en ocasiones magnetómetros e incluso barómetros para conocer la altura con respecto al nivel del mar.

Se ha realizado una pequeña investigación de la oferta disponible, así como de sus prestaciones y se ha encontrado los siguientes dispositivos:

- MPU 6050: varios fabricante producen placas que utilizan este chip IMU con 6 grados de libertad, de reducido coste – en torno a 3 € - compatible con Arduino. Cuenta con giróscopos de 3 ejes y acelerómetros de 3 ejes, permitiendo tener acceso a las 6 señales de forma independiente.

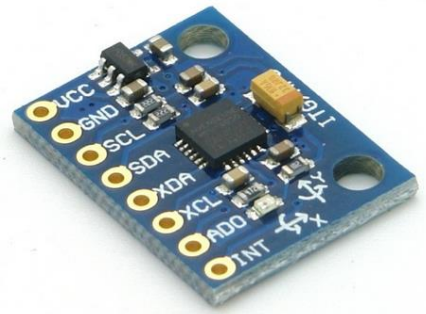

Figura 28 – GY-521 MPU 6050 (fuente: www.hotmcu.com)

Presenta las siguientes características.

- Alimentación: 3.3 a 5 V.
- Comunicación I2C.
- Rangos de detección aceleraciones: ±2 g, ±4 g, ±8 g, ±16 g.
- Rangos de detección de velocidades angulares: ± 250, 500, 1000, 2000 °/s.
- Masa: en torno a los 3 gr.
- Paso entre pines: 2.54 mm.

- Investigando se han encontrado otras opciones como el sensor de inclinación C7230, pero que no dispone de acelerómetro ni giróscopos.

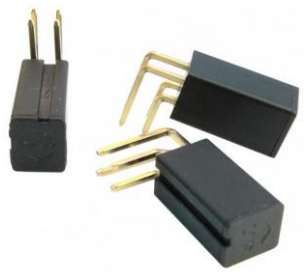

Figura 29 – Sensor inclinación C-7230
(fuente: www.fadisel.com)

MMA 7361: se trata de un acelerómetro con medida de inclinación, de reducido coste, en torno a los 7 €, y con las siguientes características:

- Alimentación: 2.2 V - 3.6 V o 4 V – 7 V, en función del pin al que se conecte.
- Rangos de detección seleccionables: ±1.5 g, ±6 g.
- Sensibilidad: 800mV/g @ 1.5 g.
- Tiempo de arranque: 0.5 ms.
- 3 salidas analógicas.
- Masa: 2.5 gr.
- Paso entre pines: 2.54 mm.

Figura 30 – Acelerómetro MMA 7361
(fuente: www.addicore.com)

ADXL 345: se trata de un acelerómetro digital de 3 dimensiones, de bajo coste con las siguientes características:

- Rangos de aceleración seleccionables: ±2 g, ±4 g, ±8 g, ±16 g
- Resolución de 10 bits a 13 bits cuando se incrementa el rango de detección.
- Paso entre pines: 2.54 mm .

Figura 31 – ADXL345
(fuente: www.analog.com)

Tres de las opciones investigadas pueden satisfacer las necesidades del proyecto, pero, en este la elección inicial es el IMU MPU6050. Durante el diseño detallado se determinará si el funcionamiento es el correcto, y en caso contrario se modificará la selección.

2.2.4 Tecnologías de desarrollo de aplicaciones para Android

Para el prototipo del terminal móvil de seguimiento (TMS) se va a emplear un teléfono móvil con sistema operativo Android y con una aplicación muy específicamente desarrollada para demostrar la tecnología.

Son infinitas las opciones existentes para el desarrollo de aplicaciones en el entorno Android, entre las que podemos listar, los IDEs NetBeans, Eclipse e Intellij IDEA para Java, Basic 4 Android, Mono para Android, LiveCode, InDesign CS6, Appcelerator, Titanium, Adobe Air, Ruboto, DropSource, Andromo, Thunkable, Mobincube…
En este caso se ha decidido que la app del proyecto será desarrollada con MIT App Inventor 2, básicamente porqué el autor del TFG cuenta con algo de experiencia con este IDE. App Inventor 2 es un producto en línea desarrollado por MIT - *Massachusetts Institute of Technology* - y cuenta con las siguientes características:

- Se trata de software libre, sin necesidad de adquirir licencias.
- Es multiplataforma: requiere sólo de un navegador y la máquina virtual de Java instalada, con "*java web start*".
- Es una herramienta muy usable y visual, y no requiere la escritura de código propiamente dicha, aunque tener experiencia en programación ayuda a utilizar la herramienta con rapidez, calidad y sencillez.
- Se puede emular de forma rápida y eficiente el funcionamiento de la App y transferirla al dispositivo de forma casi instantánea, con lo que la convergencia de la funcionalidad se hace de forma rápida.
- Existe mucha información disponible de cómo resolver problemas.
- Entre los inconvenientes principales está que solo sirve para Android, que en esta caso no es un problema, aunque sí podría serlo su portabilidad a iOS en un futuro.
- No genera código Java que podría ser utilizado como base para fases de industrialización posteriores.

Tal vez podría considerarse que App Inventor es una herramienta poco profesional, y tal vez sea así, pero es más que suficiente para la propuesta de hacer un prototipo demostrador del rastreador. Obviamente, en caso de realizarse una segunda fase de industrialización de éste, se requeriría de programadores profesionales utilizando herramientas más sofisticadas.

2.2.5 Plataformas digitales para el almacenamiento de datos IoT

Para completar el funcionamiento del sistema se ha de enviar los datos a una plataforma IoT, a este componente lo hemos denominado plataforma externa de almacenamiento de datos en el *cloud* – PEADC – Obviamente la alternativa que nos parece más apropiada es utilizar una plataforma IoT que sea gratis,

para poder almacenar la escasa información que durante el desarrollo del proyecto se va a generar.
Las opciones aquí con varias, podemos agruparla en dos grupos, aquellas enfocadas a aplicaciones empresariales, en general bastante complejas de utilizar (vía SDK en algunas ocasiones), y que asumimos que tienen unos costes de subscripción elevados:

- Amazon AWS IoT Core
- Amazon AWS 1-Click
- Jasper de Cisco Systems
- Microsoft Azure IoT
- Watson IoT de IBM
- Cloud IoT Core de Google

Y aquellas que están pensadas para desarrolladores, muchas de ellas con bajos costes de subscripción, en incluso opciones gratis:

- ThingSpeak: es una plataforma orientada a añadir, ver y analizar flujos de datos de una manera sencilla y bastante visual. Permite análisis en tiempo real, y tiene un interfaz especial para realizar el análisis con Matlab. La información subida se organiza por canales. Se puede utilizar de forma gratuita, al menos cuando los volúmenes de datos manejados son moderados.
- Emoncms.org: se trata de una plataforma de código abierto que forma parte del proyecto OpenEnergyMonitor.org. Está enfocada en la gestión de la energía. No se ha encontrado información clara de los costes de acceso.
- ThingWorx: se trata de una plataforma muy popular en la actualidad para aplicaciones IoT. Permite a las empresas desarrollar soluciones innovadoras como realidad aumentada de forma rápida y sencilla. A pesar de que es de pago se puede tener acceso gratuito durante 120 días.
- Kaa: también de código abierto, dispone de un gran conjunto de herramientas IoT que pueden ser fácilmente interconectadas. No se ha encontrado que existan disponibles versiones gratuitas.
- Thinger.io: también código abierto, entre sus características se puede destacar la calidad del interfaz altamente interactivo. Es gratuita para quien disponga de hasta 2 dispositivos. Los planes básicos de pago son bastante económicos.
- Ubidots: cuanta también con interfaz sencillo e intuitivo, dispone de una versión educativa totalmente gratuita.
- Firebase: se trata de la plataforma ubicada en la nube e integrada con Google Cloud Platform, que usa un conjunto de herramientas para la creación y sincronización de proyectos. Originalmente no se creó enfocada en aplicaciones de IoT, siendo, más bien, una plataforma de nube universal. Sin embargo, funciona bien con IoT, es muy sencilla de utilizar, y conecta muy bien con cualquier hardware que pueda comunicarse con su API REST, incluidas las plataformas de desarrollo como Arduino.

La investigación realizada para seleccionar una plataforma no ha sido suficientemente profunda, ya que no se cuenta con la experiencia y los conocimientos de que requerimientos se van a necesitar, ni que prestaciones nos va a permitir cada una de las plataformas indicadas. La elección ha sido utilizar Firebase, ya que parece una de las más extendidas en pequeños proyectos, y en particular aquellos que se realizan con dispositivos tipo Arduino. Otro punto muy importante es que App Inventor 2 cuenta con objetos desarrollados para trabajar con esta plataforma, de forma sencilla y cómoda. Además, en las pruebas realizadas, los tiempos de latencia en la actualización de datos han sido realmente muy cortos y más que suficientes para la aplicación que estamos desarrollando.

3. Especificación básica

Después de haber completado el estudio del estado del arte de los productos que concurren en este segmento, se está en disposición de formular una especificación básica del prototipo que se va a desarrollar. Por tratarse de una especificación básica de un primer prototipo va a centrarse más en la prestaciones cualitativas del mismo, y no tanto en parámetros cualitativos que evalúen cada uno de los aspectos.

- El interfaz con el usuario ha de ser gráfico, sin la necesidad de que éste tenga que cortar y pegar coordenadas para ser visualizadas en un mapa.
- El sistema ha de informar de la localización actual del vehículo, y de un histórico de las localizaciones inmediatamente anteriores.
- El TMR instalado en el vehículo ha de dejar su información en una plataforma *cloud* (PEADC) para que el TMS pueda acceder en cualquier momento. Es decir si el TMR se encuentra sin cobertura de telefonía celular, el TMS podrá subir al histórico más reciente.
- Precisión: inferior a unos 15 m (como los sistemas de GPS tradicionales)
- Los parámetros principales han de poder ser configurados desde el terminal móvil (TMS), y variados en tiempo real si fuese necesario. Algunos de estos parámetros principales son:

    - Detalles de identificación del vehículo (marca, modelo y matrícula)
    - Números de SIM tanto del TMS como del TMR.
    - Periodo de muestreo.
    - Parámetros de visualización por pantalla.

- El TMS ha de incluir su propia posición en el mapa, para facilitar la búsqueda del TMR.
- El TMR ha de enviar información directa al TMS (vía SMS) en situaciones de vibraciones o desplazamientos inesperados, y por supuesto en la situación de vuelco.
- Del mismo modo el TMR ha de avisar a terceras personas, vía SMS, de una situación de vuelco prolongada.
- El TMR ha de enviar información de varios parámetros, con el periodo de muestreo establecido. Estos parámetros son:

    - Latitud en la que se encuentra el vehículo.
    - Longitud en la que se encuentra el vehículo.
    - Fecha y hora de la localización.
    - Está el vehículo encendido.
    - Estado dinámico, en el sentido de si el vehículo no está estático.
    - Estado de inclinación del vehículo, para detectar vuelcos y caídas.
    - Velocidad del vehículo.

- El TMR ha de poder conectarse a la batería del vehículo, pero contar con su propia batería para no agotar a la anterior.
- El TMR ha de tener un tamaño reducido ara poder ser instalado en una motocicleta sin ser claramente detectado.

- Los tiempos de latencia de las comunicaciones entre TMR y TMS han de ser bajos, por debajo de los 2 o 3 segundos, aunque estas comunicaciones se establezcan a través del PEADC.

Durante el desarrollo de estas especificaciones se planteó la posibilidad de realizar un documento de requerimientos del software, tanto del TMR (el que corre en Arduino) como del TMS (el que corre en el teléfono inteligente o Tablet), con el formato de la normativa establecida por el IEEE 830. Esta alternativa hubiera sido la más completa, menos ambigua y más profesional para realizar una buena especificación, pero a sencillez de esos dos programas (recordemos que se trata de un primer prototipo demostrador) así como la escasez de tiempo disponible, desaconsejaban seguir ese camino.

4. Diseño conceptual y arquitectura del sistema

Después de haber completado el estudio del funcionamiento de algunos de los rastreadores existentes en el mercado, se realizó lo que se podría denominar una síntesis de una arquitectura genérica “de máximos”, que puede responder a la gran mayoría de funcionalidad vistas en los distintos productos analizados. Esa arquitectura se plasmó en la figura 19.

Las especificaciones básicas listadas en el capítulo 3, resumen también, las líneas generales de las prestaciones de los productos analizados.

Es decir se cuenta con una propuesta de arquitectura que puede satisfacer – si se implementa correctamente – los requerimientos establecidos en este TFG.

Por otra parte, las investigaciones realizadas sobre las tecnologías disponibles han permitido seleccionar una serie de componentes que se va a utilizar para implementar a nivel conceptual y detallado la arquitectura propuesta.

En la siguiente figura se muestra una adaptación de la figura 19, con los componentes seleccionados.

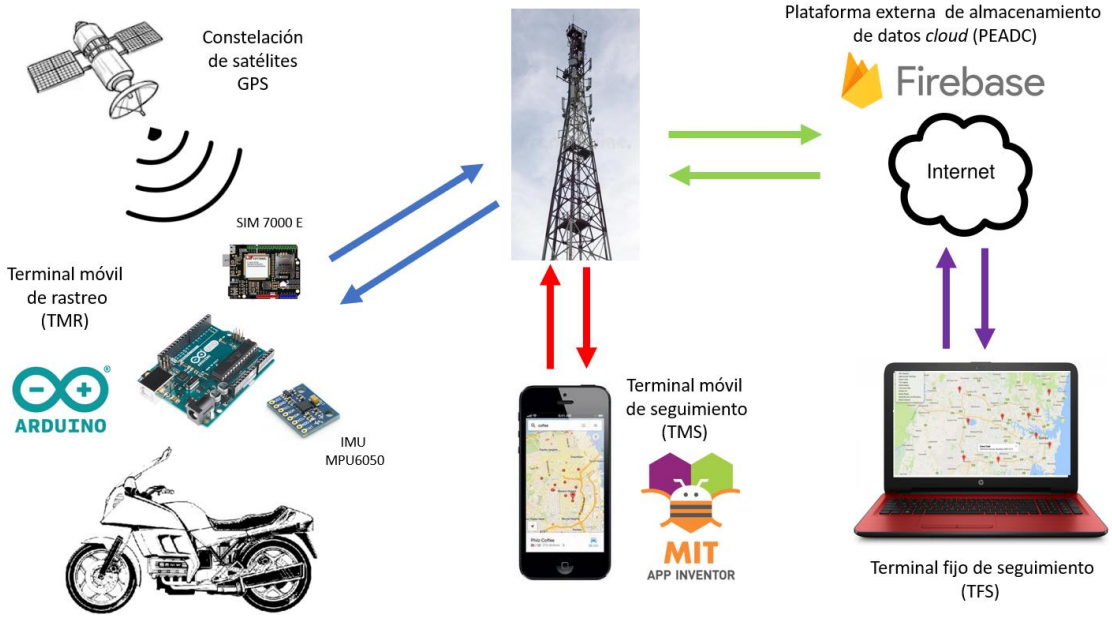

Figura 32 – Arquitectura básica del sistema y selección de tecnologías para completar el prototipo de rastreador.

A lo largo del desarrollo detallado y de la fase experimental han aparecido problemas que han requerido de cambios en algunos de los componentes seleccionados, pero que no han modificado sustancialmente el diseño conceptual del sistema, ni, por supuesto, su arquitectura básica.

A continuación se va a realizar una explicación de cómo funcionan a alto nivel los distintos componentes – integrados en esta arquitectura – dejando para el siguiente capítulo los comentarios acerca de los detalles.

El siguiente diagrama de bloques ayuda a entender los flujos de información y los protocolos utilizados en cada una de ellas.

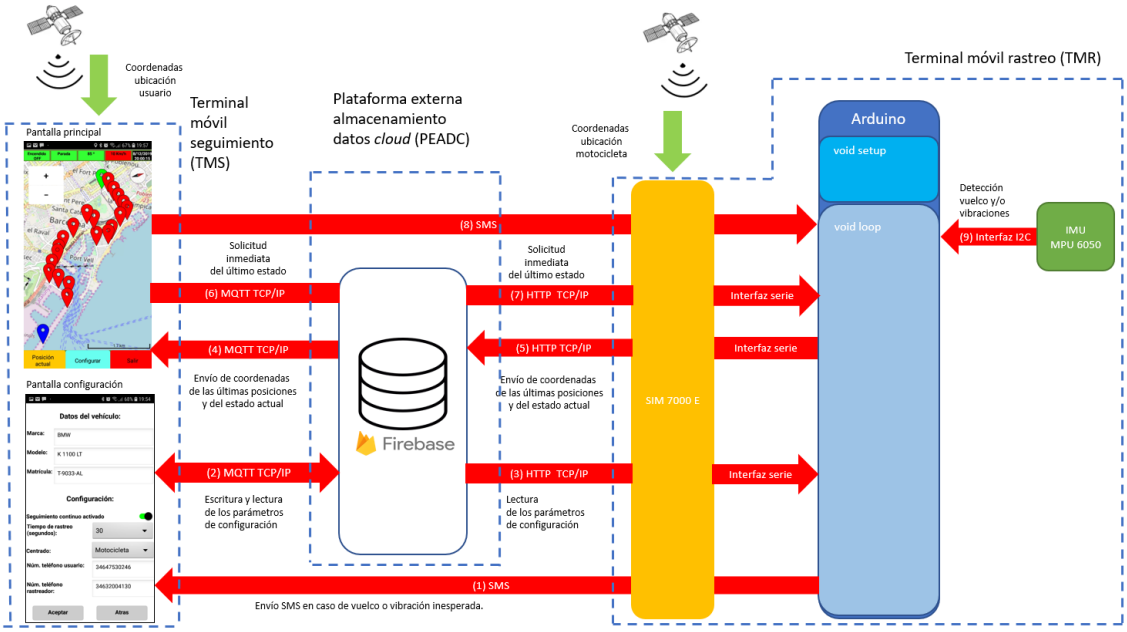

Figura 33 – Flujos de datos entre los distintos bloques que componen la arquitectura.

Terminal móvil de rastreo (TMR):

Está integrado por 5 componentes principales:

- Arduino.
- SIM7000E.
- Antena para LTE y GPS.
- Batería de 3.7 V Li Po.
- IMU MPU6050

Su funcionamiento es el siguiente. Arduino está trabajando dentro de un bucle infinito donde espera alguno de los siguientes eventos:

-detección de movimiento de vibración y/o vuelco desde el IMU (flecha roja indicada con el número 9 en la figura superior). En el caso de que este evento se dé, Arduino ordenará a la SIM 7000E de un SMS indicando al usuario a través del TMS que el evento se ha producido, y que la motocicleta puede estar siendo robada, o cuando menos manipulada, sin el consentimiento de su sueño (flecha roja con el número 1). Del mismo modo Arduino actualizará todos los parámetros que tenga disponibles en la base de datos de Firebase (flecha roja número 5), haciendo que esta información esté accesible también desde el TMS (flecha roja número 4). Lo cierto es que el envío del SMS no estrictamente necesario, ya que la aplicación que corre en el TMS podría alertar al usuario al detectar un cambio en la base de datos. Se ha preferido hacer con SMS para asegurar que se capta mejor la atención del usuario, aunque no tenga la aplicación activa.

- Detección de la recepción de una señal a través de la SIM 7000E en la se le indique un cambio de configuración de alguno de sus parámetros - número de teléfono del TMS, periodo de enviar señales, etcétera – (flecha roja número 2 hacia la izquierda) Esta información será recibida cuando se produzca un cambio en la base de datos (no SQL) contenida en Firebase.
- Detección de una señal de petición de estatus, por parte del usuario que maneja el TMS. De nuevo esta solicitud puede realizarse a través de SMS (flecha roja número 8) o de un cambio en un registro de Firebase, donde se solicite la información (flechas rojas 6 y 7).
- Detección de un cambio de parámetros (flecha roja número 3)

Terminal móvil de seguimiento (TMS):

En este caso el dispositivo es un teléfono inteligente con sistema operativo Android. En el corre una aplicación que se ha denominado MotoTracker, en su versión 1.0 y que cuenta con dos funcionalidades instanciadas en otras tres pantallas principales.

Se ha desarrollado:

- una pantalla de inicio muy sencilla con el logo, el nombre y la versión de la aplicación. Se ha considerado la opción de añadir un acceso con nombre de usuario y palabra clave, pero se ha descartado, ya que no añadía una funcionalidad muy relevante.

- Una pantalla principal donde se reciba la mayoría de la información, y permita:
    - conocer el estado actual de la motocicletas,
    - Su posición actual y el histórico de sus posiciones sobre una representación de un mapa.
    - La posición relativa del usuario.
    - Botones con las acciones principales, como solicitar posición, ir a la pantalla de configuración o salir de la aplicación.
- Un pantalla de configuración donde el usuario pueda fijar los parámetros para controlar correctamente la funcionalidad de la aplicación.

El diseño, la funcionalidad y los aspectos de programación de las tres pantallas y de la aplicación en general, se detallan en el siguiente capítulo.

Plataforma externa de almacenamiento de datos – *cloud* (PEADC):

La PEADC ha de almacenar los datos relevantes, tanto lo que se refiere a parámetros de configuración, como los propios datos del seguimiento de la motocicleta.

Como se ha comentado la plataforma que se está utilizando es Firebase de Google. Se trata de una plataforma muy accesible y que ofrece la posibilidad de almacenamiento gratis.
La base de datos generada es muy sencilla, y puede ser accedida y editada desde cualquier navegador.
Cuenta con la ventaja principal de App Inventor 2 (que es el IDE de desarrollo utilizado para el desarrollo de la aplicación de la TMS) cuanta con objetos que comentan automáticamente y de forma sencilla con esta plataforma *cloud*. En el siguiente capítulo se muestran los objetos y métodos utilizados.

Por su parte, la comunicación con Arduino debe hacerse mediante el protocolo HTTP.

Para el terminal fijo de seguimiento - TFS - no se especifica ninguna tecnología, porqué, como se ha comentado anteriormente no forma parte del alcance del proyecto. En cualquier caso su funcionamiento no divergiría mucho del de TMS.

5. Diseño detallado del prototipo

En este capítulo se va a describir algunos de los detalles de implementación de la solución, tanto del hardware como del software del terminal móvil de rastreo como de la aplicación del terminal móvil de seguimiento.

Terminal móvil de rastreo (TMR) – hardware:

En el capítulo anterior se ha indicado cuales son los componentes hardware utilizados en el diseño del TMR, Ahora se va a describir las conexiones entre los distintos componentes. En la siguiente figura se muestra una fotografía del montaje de los componentes del TMR:

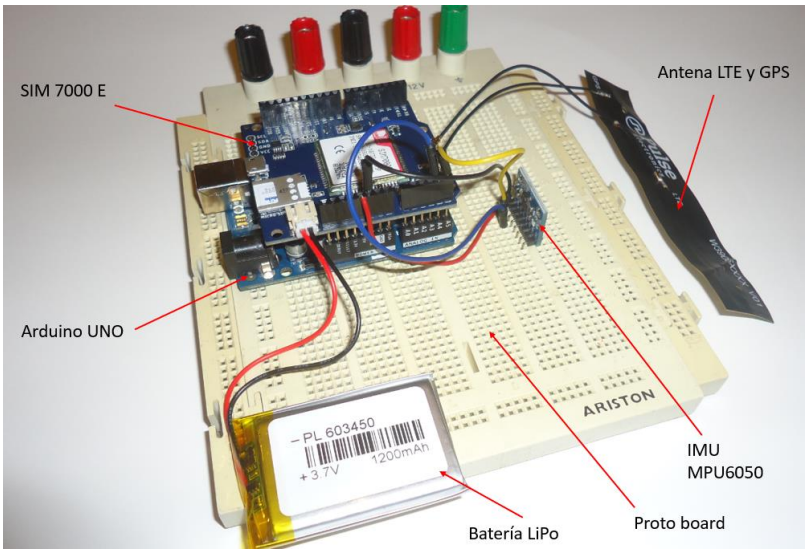

Figura 34 – Montaje hardware del TMR.

La figura siguiente ilustra las conexiones entre Arduino UNO y el IMU MPU6050. Como puede verse consiste solo en cuatro conexiones:

- De los 5 V de Arduino a Vcc del MPU6050.
- Conexión entre ambos GND's.
- De la pata A5 de Arduino al SCL del MPU6050.
- De la pata A4 de Arduino al SDA del MPU6050.

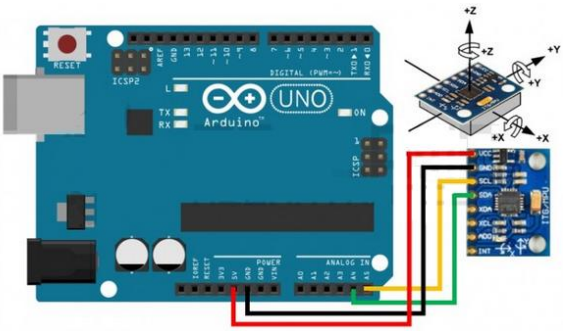

Figura 35 – Esquema de conexión entre Arduino y el IMU MPU6050 (fuente: www.naylampmechatronics.com)

Obviamente las dos primeras se utilizan para la alimentación del IMU. Las dos segundas configuran la conexión I2C entre Arduino y el IMU.
I2C es un bus de comunicaciones en serie síncrono. El acrónimo de su nombre significa *Inter-Integrated Circuit*. La primera versión - 1.0 - data del año 1992 y la versión 2.1 del año 2000, y fue desarrollada por Philips. La velocidad de transmisión es moderada, alrededor de 100 kbit/s en el modo estándar, aunque también permite velocidades superiores de hasta 3.4 Mbps. El uso de este bus de comunicaciones está muy extendido en la industria, utilizándose para comunicar microcontroladores y sus periféricos en sistemas integrados (*Embedded Systems*) Una parte importante de su éxito se debe a su sencillez ya que I2C sólo utiliza dos líneas para transmitir la información: una para los datos (SDA) y otra para la señal de reloj (SCL). También es necesaria una tercera línea, pero esta sólo es la referencia de voltaje (GND). En la comunicación de define un dispositivo como el maestro (en este caso Arduino) que es el que genera la señal de reloj (de ahí la naturaleza síncrona de este bus), pero la comunicación, que puede ser bidireccional, se establece por la conexión de datos (SDA). I2C puede utilizarse también entre un maestro y varios esclavos que son accedidos mediante direccionamiento.
En este caso utilizamos Arduino como maestro y MPU6050 como esclavo, con dirección 0x68.

Por su parte la SIM7000E va montada en forma de *shield*, es decir todos los pines de dicha placa conectan entran en los pines hembra de Arduino. A pesar del montaje en forma de *shield*, que hace el montaje mucho más compacto y fiable, son pocas las conexiones que realmente están funcionando.

- De los 5 V de Arduino a Vcc del SIM7000E
- Conexión entre ambos GND's.
- El pin digital 6 de Arduino se utiliza para indicar a SIM7000E la señal PWRKEY.
- El pin digital 7 de Arduino se utiliza para indicar a SIM7000E la señal de RST (*reset*).
- Los pines digitales 10 y 11 de Arduino se utilizan para la comunicación serie con SIM7000E (se denominan TX y RX).

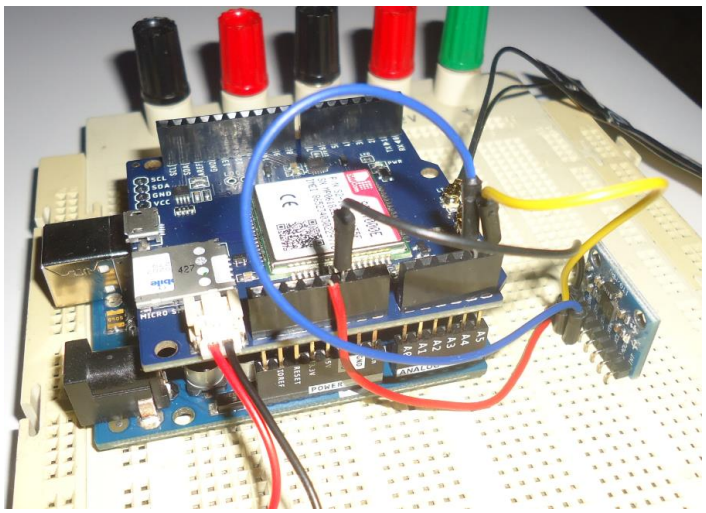

Figura 36 – Conexión del IMU MPU6050 a través de SIM7000E montado en forma de *shield*.

A pesar de que exista conexión entre la alimentación de Arduino y de la placa SIM7000E, ésta última requiere de la conexión a la batería para su funcionamiento correcto (parece que el ruido de la alimentación proveniente de Arduino, o de otras posibles fuentes de alimentación externas, no permite un buen funcionamiento).

Así, disponiendo la placa SIM7000E encima de Arduino, las conexiones con MPU6050, se realizan en los conectores hembra de aquel, como se muestra en la fotografía de la figura anterior.

Terminal móvil de rastreo (TMR) – software:

Los programas de control del TMS se han desarrollado con el IDE de Arduino. Los programas para Arduino se suelen denominar *sketches*, tienen una extensión .ino así como un conjunto de instrucciones y una sintaxis muy parecida a los lenguajes C y C++.
En este presente TFG se han desarrollado dos programas distintos para Arduino:

- MotoTracker_cloud_rev0_1.ino: en este programa responde a la arquitectura de concepto mostrada en la figura 33, es decir realiza una parte importante de la comunicación a través de la base de datos de Firebase, lo cual permite la integración con aplicación desarrolla para Android en el TMS (y que se describe en detalla más abajo) De alguna manera se puede decir que este programa permitiría al futuro producto acercarse mucho a las prestaciones de los rastreadores de gama alta.
  Como se comentará en el capítulo 6 - pruebas funcionales sobre el prototipo -, este programa no ha funcionado por una incompatibilidad del chip SIM7000 – incluido en la *shield* de comunicaciones escogida – y el set de comandos AT necesarios para la comunicación con el protocolo HTTP que utiliza Firebase. En cualquier caso el programa se va a describir, ya que con un cambio de *shield* de comunicaciones, reemplazando la Botlectics SIM7000E por la Botlectics 7500E, el problema estaría resuelto. Este cambio de placa no se ha podido realizar dentro de este TFG por falta de tiempo para recibir la placa y hacer de nuevo las pruebas.

- MotoTracker_solo_SMS_rev0_2.ino: este programa realiza las comunicaciones unidireccionales desde el TMR al TMS únicamente vía SMS y por tanto no requiere de la aplicación desarrollada para el TMS. Esta solución iguala casi en su totalidad las prestaciones de los rastreadores de gama baja (presentados en el estado del arte), pero claramente no cumple con los objetivos fijados en las especificaciones básicas.

En los siguientes párrafos se va a describir la funcionalidad de ambos programas, aunque solo el segundo programa ha podido desarrollado en su totalidad y ha podido refinarse hasta alcanzar el funcionamiento deseado.

Como es sabido, los programas de Arduino tienen una estructura fija que se puede describir de la siguiente manera:

- un programa principal que se auto arranca al cargarse en el chip. Sería algo así como el programa main() de C. Es en esta parte principal donde se incluyen las librerías necesarias para la compilación, el montaje y la posterior ejecución del programa. Es en esta zona donde se declaran las variables globales, que pueden ser accedidas desde cualquier punto del programa. También en esta zona se declaran las definiciones (#DEFINE) que se utilizan durante la compilación y que facilitan la lectura del programa. Del mismo modo es en esta zona del programa donde suelen ser definidas las funciones que el programador necesite incorporar a su programa para estructurarlo y simplificarlo de una forma apropiada.
- Una función void setup(). En ella se define e inicializa aquellas asignaciones que se realizan para el comienzo del programa, como asignaciones de distintas funcionalidades a los pines de Arduino - entrada, salida, analógico, digital, etcétera – Por esta función no se vuelve a pasar excepto que sea llamada por otra parte del programa, por tanto se ha de asegurar que incluye todas las inicializaciones necesarias.
- Función void loop(). Se podría decir que es la parte más importante, ya que consiste en un bucle infinito que Arduino va ejecutando constantemente, excepto que alguna instrucción interna de esta función loop() decida pararlo. Es en esta función donde suele estar la algorítmica principal del *sketch*.

Los dos programas desarrollados tienen la parte principal y la función setup() muy parecidos, estando las diferencias fundamentales en la función loop()

Las librerías que se han incluido en el programa han sido las siguientes:

```
// Librerias SIM7000
#include "Adafruit_FONA.h"

//Librerias del MPU6050
#include "I2Cdev.h" // Librería del I2C
#include "MPU6050.h" //Libreria para el control del IMU
#include "Wire.h"

//Librería generales
#include <SoftwareSerial.h>
```

Para la SIM7000 se ha utilizado la librería Adafruit_FONA que incorpora una cantidad importante de métodos para los objetos creados con esta clase. Entre los métodos utilizados en los programas podemos destacar (no se indica los parámetros de entrada):

.begin(): genera en objeto por donde se establece la comunicación.
.type(): en este proyecto la respuesta es 7000E.
.getIMEI(): obtiene en IMIE de la tarjeta SIM empleada.
.setFunctionality(): arranca la funcionalidad del objeto de comunicación.
.setNetworkSettings(): en este proyecto se ha utilizado una SIM de la compañía Lycamobile. En este método se le ha de decir a la SIM7000E cual es el APN, el nombre de usuario y la clave de acceso a este proveedor. En este caso:

APN = data.lycamobile.es
User = lmes
Password = plus

Figura 37 – Tarjeta SIM Lycamobile.

Obviamente se podría haer utilizado otra compañía, s eha escogida esta porque tiene una modalidad prepago bastante económica que incluye un número ilimitado de mensajes SMS.

.setPreferredLTEMode(): este método activa por defecto la comunicación LTE CAT1 M a través del objeto de comunicación. Durante la fase de experimental no sé utilizó desde el principio este método, lo cual dio unos 2 días de problemas en las comunicaciones. A partir de utilizar este método el sistema pasó a funcionar establemente.
.enableGPS(): activa y desactiva la conexión GPS.
.getGPS(): solicita a la placa que envíe la información captada. La SIM7000E envía un número importante de parámetros, como si hay no *fix*, latitud, longitud, velocidad, *heating*, altitud, etcétera.
.getAdVoltage(): solicita el voltaje de entrada del chip.
.getBattVoltage(): solicita el voltaje de la entrada de la batería LiPo.
.getBattPercent(): solicita el porcentaje de carga de la batería (es muy aproximado).
.getSIMCCID: solicita el CCIS de la SIM.
.getRSSI(): solicita en nivel de cobertura, se puede traducir a dBm.
.getNetworkStatus(): solicita el estatus de conexión a la red de datos.
.getTime(): solicita el tiempo de la red.
.getNetworkInfo(): solicita información de la red de datos.
.sendSMS(): envía un mensaje SMS, los dos parámetros que utiliza son el número del receptor y el texto del mensaje.

Para trabajar con el IMU PSU6050 se han incluido las tres librerías indicadas arriba:

- I2Cdev.h y Wire.h: para la comunicación I2C con el IMU.
- MPU6050.h: para llamar a métodos específicos de este sensor.

Los métodos utilizados en los programas de estas librerías han sido:

MPU6050 mpu(mpuAddress): la dirección utilizada es Ox68, este método crea el objeto mpu que es el que va a utilizar para extraer la información del IMU MPU6050.

mpu.initialize(): método que inicializa el objeto de comunicación con MPU6050.
mpu.getAcceleration(&ax, &ay, &az): solicita al objeto mpu las tres aceleraciones, en cada uno de los ejes, y la pone en tres variables tipo float.
mpu.getRotation(&gx, &gy, &gz):solicita al objeto mpu las tres velocidades de giro, y la pone en tres variables tipo float.
Wire.begin(): inicializa la comunicación a través del I2C.

Por último, como librería de uso general hemos utilizado la Software.Serial para establecer comunicaciones serie tanto con el monitor para visualizar las trazas, así como también para la comunicación serie con la SIM7000.

En la función setup() general se realiza una inicilaización general tanto del IMU como de la SIM7000E con muchas de los métodos descritos arriba. En el anexo 1 de este documento se muestra los detalles del segundo programa, que utiliza SMS como método de comunicación, pero que es común con la del primero.

Descripción de bucle loop() del programa MotoTracker_cloud_rev0_1.ino:

En el bucle iterativo se están monitorizando básicamente tres eventos, que son los que van a desencadenar las distintas acciones:

- Llegada de un SMS procedente del TMS del usuario donde se indique que ha habido un cambio en los parámetros principales en la base de datos en Firebase. En este caso el sistema lee el parámetros y empieza a utilizarlo (por ejemplo el periodo de monitoreo)
- Llegada de un SMS procedente del usuario donde se solicite el estado del vehículo. En este caso el bucle pide a los sensores IMU y GPS los fatos, y los empaqueta los sube a la base de datos Firebase (esta es la funcionalidad que no se ha conseguido) La actualización de la base de datos hará que la aplicación del TMS se actualice y refresque automáticamente, sin ninguna acción adicional por parte del usuario.
- El estado de los sensores del IMU. Si detecta una vibración o una inclinación por encima de unos valores prefijados el programa enviará un SMS de advertencia al TMS del usuario (mensaje SMS normal al gestor de SMS's) y además actualizará la base de datos en Firebase (lo que no funciona) para que el usuario pueda acceder a la información.

Las señales que llegan IMU entregan los tres valores de aceleración y las tres valores de velocidad angular, en torno a los tres ejes. Para que esta información sea tratada se han de realizar algunas conversiones, como son:

- Pasarlas a unidades más inteligibles desde un punto de vista físico, en este caso las aceleraciones de han pasado m/s^2 y las

velocidades angulares a º/s. Los factores de conversión utilizados en el programa han sido:

```
const float accScale = 2.0 * 9.81 / 32768.0; //Conversión aceleraciones al SI
const float gyroScale = 250.0 / 32768.0;     //Conversión velocidades angulares a º/s.
```

- Restarles los valores de offset que muestra el sensor en reposo, podríamos decir que es una especie de calibración o ajuste del sensor.

```
//Valores de calibración para corregir el offset de los sensores.
const float cal_acel_x= (10.5-9.81)/accScale;
const float cal_acel_y= -0.11/accScale;
const float cal_acel_z= -0.53/accScale;
const float cal_vel_ang_x = -3.50/gyroScale;
const float cal_vel_ang_y = 0.25/gyroScale;
const float cal_vel_ang_z = -0.8/gyroScale;
```

- Por último se han realizados los cálculos para determinar el ángulo de inclinación con respecto a la vertical, así como el valore de velocidad angular total.

```
//Calcular los angulos de inclinacion y la velocidad angular total.
ang_x = atan((ax-cal_acel_x) / sqrt(pow((ay-cal_acel_y), 2) + pow((az-cal_acel_z), 2)))*(180.0 /
3.14);
gtotal=sqrt(pow((gx-cal_vel_ang_x),2)+pow((gy-cal_vel_ang_y),2)+pow((gz-
cal_vel_ang_z),2))*gyroScale;
```

- Finalmente estos valores se comparan los valores umbrales para decidir si la motocicleta está siendo manipulada y si se ha volcado:

```
//Limite ángular a partir del cual se envía SMS al usuario
const float ang_limite = 45;
//Limite velocidad angular a partir del cual se envía SMS al usuario
const float vel_ang_limite= 3;
```

Descripción del bucle loop() del programa MotoTracker_solo_SMS_rev0_2.ino:

Este programa ha renunciado a la escritura de datos en la base de datos *cloud* en Firebase, y solo atiende al tercer evento citado en el punto anterior. Es decir solo atiende a las variables del IMU para enviar mensajes. Este tercer bucle de vigilancia es común en los dos programas.
Pero en este caso, como no existe comunicación con la base de datos, el TMR envía las coordenadas geográficas del vehículo al usuario, vía un mensaje SMS con el enlace automático a Open Street Map, lo que permite que el usuario pueda ver la posición del vehículo sobre un mapa, pero sin verlo en la aplicación específica.

Para poder enviar el enlace el programa genera una cadena de caracteres que incluye en el enlace (URL de OPS) donde se incluyen las coordenadas tanto del punto central del mapa, como del marcador que indica la posición del vehículo.

Para realizar esto se ha utilizado el siguiente fragmento de código:

```
if (geo){ // Si el mensaje está geolocalizado se ha de llamar a GPS y completar la estructura del enlace.
    //Declaración variables necesarias.
   float latitud;
   float longitud;
```

```
float vel_kph, cabecera, altitud;
char url[150];
char aux[20];
strcpy(url,"");

char latstring [12];
char longstring [12];


if (fona.getGPS(&latitud, &longitud, &vel_kph, &cabecera, &altitud)) { //Llamada a la función que lee las
coordenadas
  Serial.println(F("--------------------"));
  Serial.print(F("Latitud: ")); Serial.println(latitud,10);
  Serial.print(F("Longitud: ")); Serial.println(longitud,10);
  //Serial.print(F("Velocidad: ")); Serial.println(vel_kph);
  //Serial.print(F("Cabecera: ")); Serial.println(cabecera);
  //Serial.print(F("Altitud: ")); Serial.println(altitud);

  delay (2000);
//Generación de la URL

    strcpy(url,"https://www.openstreetmap.org/?mlat=");
    dtostrf(latitud,3 ,10, latstring);
    strcat(url, latstring);
    strcpy(aux, "&mlon=");
    strcat(url,aux);
    dtostrf(longitud,3 ,10, longstring);
    strcat(url, longstring);
    strcpy(aux,"#map=17/");
    strcat(url,aux);
    dtostrf(latitud,3 ,10, latstring);
    strcat(url,latstring);
    strcpy(aux,"/");
    strcat(url,aux);
    dtostrf(longitud,3 ,10, longstring);
    strcat(url,longstring);
    strcpy(aux,"&layers=N");
    strcat(url,aux);

 } else{
strcpy (url, "Sin cobertura de GPS\n");
 }
strcpy(envio, mensaje);
strcat(envio,url);

 } else{
strcpy(envio, mensaje);
  }
```

Además para evitar la sobrecarga de recepción de SMS (cosa que ni ocurre en el programa que usa Firebase) se hace que los SMS solo puedan ser enviados con una frecuencia fijada (con un mínimo periodo configurable de 15 segundos) Si las perturbaciones cesan los mensajes SMS también cesaran. Si no los mensajes continúan hasta que el dispositivo TMR es reinicializado.

En el siguiente capítulo se muestran las pruebas hechas, así como figuras con las pantallas que se reciben en el TMS del usuario.

Terminal móvil de seguimiento (TMS) – hardware:

El terminal móvil de seguimiento que se está utilizando es un teléfono inteligente con sistema operativo Android, de la marca Samsung Galaxy A3, pero podría haberse utilizado cualquier otro teléfono compatible con el sistema operativo de Google. Es decir en este apartado nos vamos a centrar solo en el software.

Terminal móvil de seguimiento (TMS) – software:

Para el funcionamiento del TMS se ha desarrollado en este TFG la de la aplicación MotoTracker. Como se ha indicado existen tres pantallas, la de inicio, la principal y la de configuración.
El diseño detallado ha sido bastante básico, y no especialmente complicado utilizando la IDE de App Inventor 2, en cualquier caso se pasa a describir con detalle las tres pantallas así como se ha realizado su programación.

App Inventor 2 ofrece dos vistas para la programación visual del código:

- la vista denominada “Diseñador”, donde se configura el aspecto de las distintas pantallas, y se carga los componentes que se van a utilizar en esa pantalla,
- la vista denominada “Bloques”, donde realmente se construye el algoritmo a partir de bloques funcionales generales, así como los de los componentes que han sido cargados en la correspondiente pantalla Diseñador.

Por tratarse de un lenguaje que podríamos denominar gráfico, no es fácil de comentar en modo texto – como es esta memoria – y, por tanto, vamos a incluir figuras con los bloques más representativos de cada una de las tres pantallas que configurar la aplicación MotoTracker.

Pantalla de inicio:

Como ya se ha comentado en la descripción del concepto, se trata de una pantalla muy sencilla, que aparece al invocar la aplicación y que desaparece después de 3 segundos danto paso a la pantalla principal, y casi no cuenta con funcionalidades reseñables.
En la siguiente figura se muestra la pantalla de inicio de la aplicación MotoTracker.

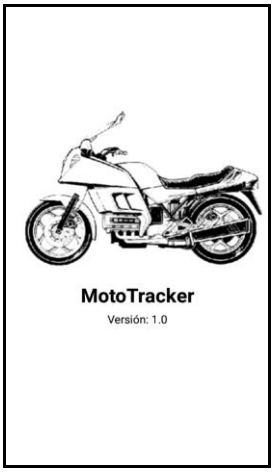

Figura 38 – Pantalla de inicio.

La vista “Diseñador” de esta pantalla es muy sencilla, y solo incluye una disposición vertical y una horizontal, algunas etiquetas, un componente reloj que realiza la cuenta atrás de 3 segundos para dar paso a la siguiente pantalla

y un fichero con la imagen de portada, que además sirve de icono de la aplicación. En la siguiente figura se muestra una porción de dicha vista en el IDE de App Inventor 2.

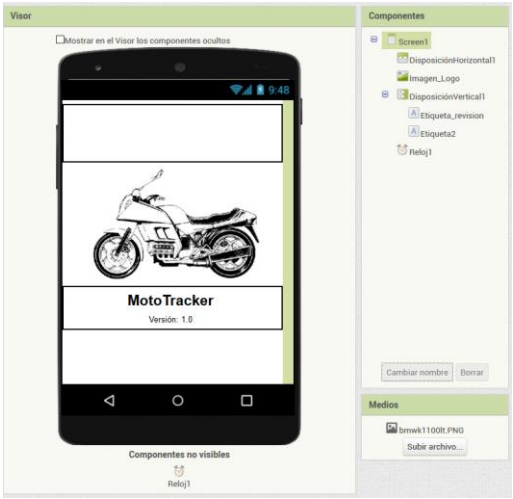

Figura 39 – Vista "Diseñador" de la pantalla de inicio.

Obviamente la vista de "Bloques" de esta pantalla es también extremadamente sencilla, ya que solo contiene la funcionalidad de la cuenta atrás del componente reloj, y la gestión del evento "ir atrás" que en esta pantalla equivaldría a cerrar la aplicación.

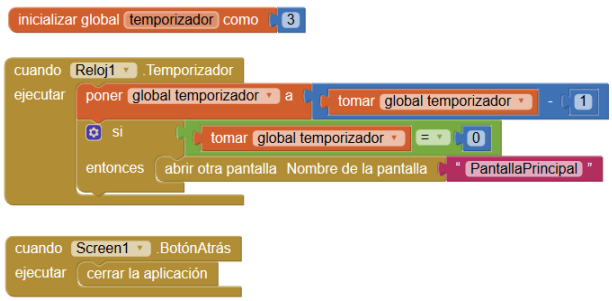

Figura 40 - Vista "Bloques" de la pantalla de inicio.

Pantalla principal:

Es en esta pantalla principal donde el usuario puede acceder a un parte de la información, y desde donde puede desencadenar distintas acciones.
En la parte superior aparece la barra de estado con información detallada del estado actual de la motocicleta. Por actual se entiende la última vez que el TMR se conectó al *cloud*.

Figura 41 - Pantalla principal de la aplicación.

Está información no llega al TMS directamente del TMR sino que no hace de la base de datos de Firebase. La información disponible consiste en (de izquierda a derecha)

- Si la motocicleta se encuentra con el contacto puesto (ON) o quitado (OFF)
- Si la motocicleta está parada, en el sentido de que no está sufriendo vibraciones y movimientos debidos a la marcha o a posibles manipulaciones.
- El ángulo de inclinación en el que se encuentra para intentar detectar posibles vuelcos o caídas.
- La velocidad aproximada a la que se está moviendo.
- Fecha y hora de la última actualización.

Si el usuario puede navegar por el mapa, girarlo, hacer zoom con los dedos o mediante el uso de los botones.
Además de la información del estado el usuario puede ver en formato de mapa la última ubicación de su vehículo, así como un histórico de las últimas 20 ubicaciones. La ubicación última se muestra con un globo de color azul, mientras que el histórico de posiciones se muestra con globos de color rojo. Además sobre el mismo mapa el usuario puede comprobar su propia ubicación (indicada con un globo de color verde)
Pulsando encima de los globos, puede obtener información de la fecha y la hora en la que el vehículo se encontraba en esa posición.

En la parte inferior de esa misma pantalla principal existen tres botones:

- para solicitar al TMR una actualización inmediata del estado del vehículo
- acceso a la pantalla de configuración
- salir de la aplicación

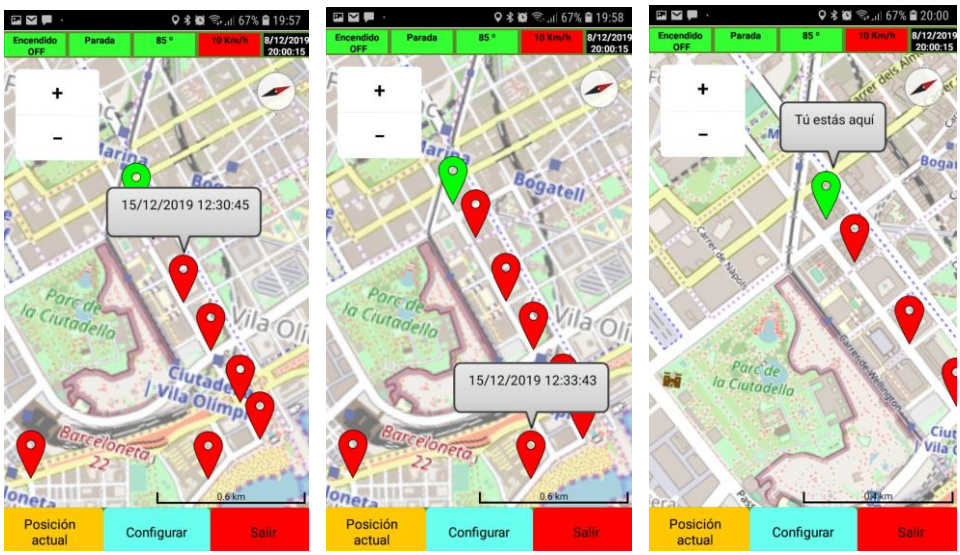

Figura 42 – Pantalla principal, con dos localizaciones en el tiempo del vehículo, y la posición del usuario.

Desde la pantalla principal se puede salir de la aplicación pulsando en el botón Salir.

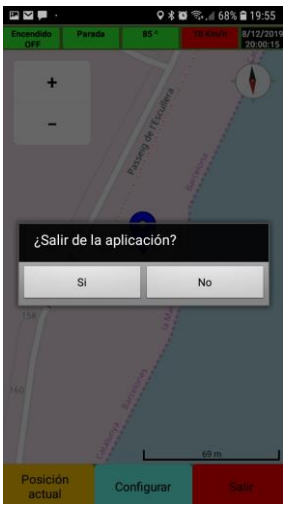

Figura 43 – Salir de la aplicación desde la pantalla principal.

En la siguiente figura se muestra la vista “Diseñador” de la pantalla principal, que se ha utilizado para el desarrollo de su interfaz de usuario. Como ya se ha descrito, cuenta con una barra de último estado en su parte superior, un mapa con las localizaciones en su parte central y una fila de botones con las tres acciones:

- para solicitar al TMR una actualización inmediata del estado del vehículo
- acceso a la pantalla de configuración
- salir de la aplicación

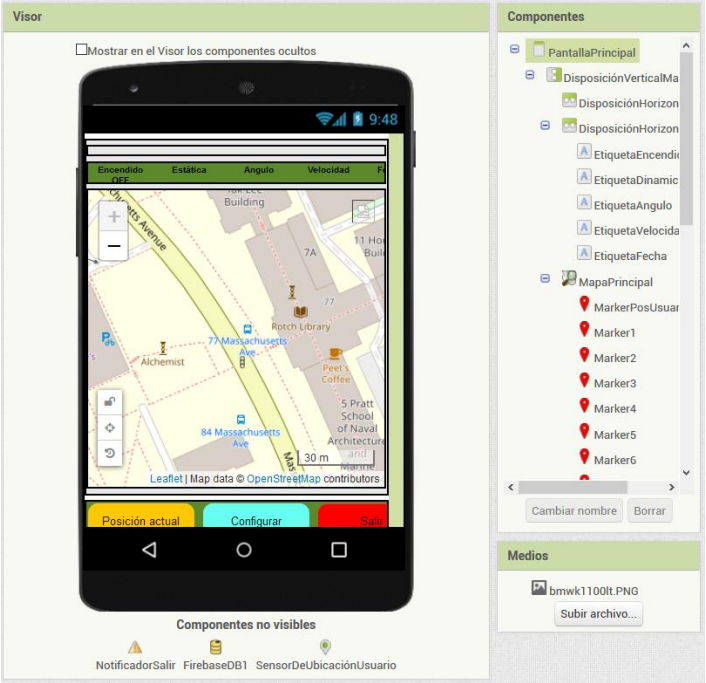

Figura 44 – Vista "Diseñador" de la pantalla principal.

En esta pantalla se han incluido un número importante de componentes (que aparecen en el lado derecho de la vista, aunque no todos los componentes son visibles), como son, disposiciones verticales y horizontales, etiquetas, tres botones, un componente mapa, 22 marcadores de mapa (para las posiciones del usuario, el histórico del vehículo, y la última posición del mismo), un notificador, un sensor de ubicación (que accede al GPS del teléfono inteligente y que sirve para detectar la posición del usuario), y por último un componente que da acceso a la base de datos que se encuentra almacenada en Firebase.

La programación de esta pantalla cuenta con varios bloques funcionales, de los cuales destacaremos los principales.

En la figura 45 se muestra el bloque de inicialización de la pantalla principal, en este bloque se genera una lista (podríamos decir que es como un vector o un *array* de otros lenguajes) los 20 marcadores que indican las 20 posiciones históricas almacenadas en Firebase. Además en la parte de abajo del bloque se solicita a Firebase (a través del objeto FirebaseDB1, que es el que apunta a la base de datos que se está utilizando) los datos de esas 20 posiciones, además de algunos parámetros más, como son el tipo de centrado (en el vehículo o en el usuario), la posición actual (referido a la última), la hora de actualización, la velocidad del vehículo, el estado dinámico o estático del vehículo así como su ángulo de inclinación.
Este bloque de inicialización solicita las posiciones, mediante una llamada al método .ObtenerValor del objeto FirebaseDB1) pero esto sólo solicita a la base de datos que se envíe el objeto, y se ha de diseñar un nuevo bloque para la lectura y clasificación en las variables correctas de la información recibida.

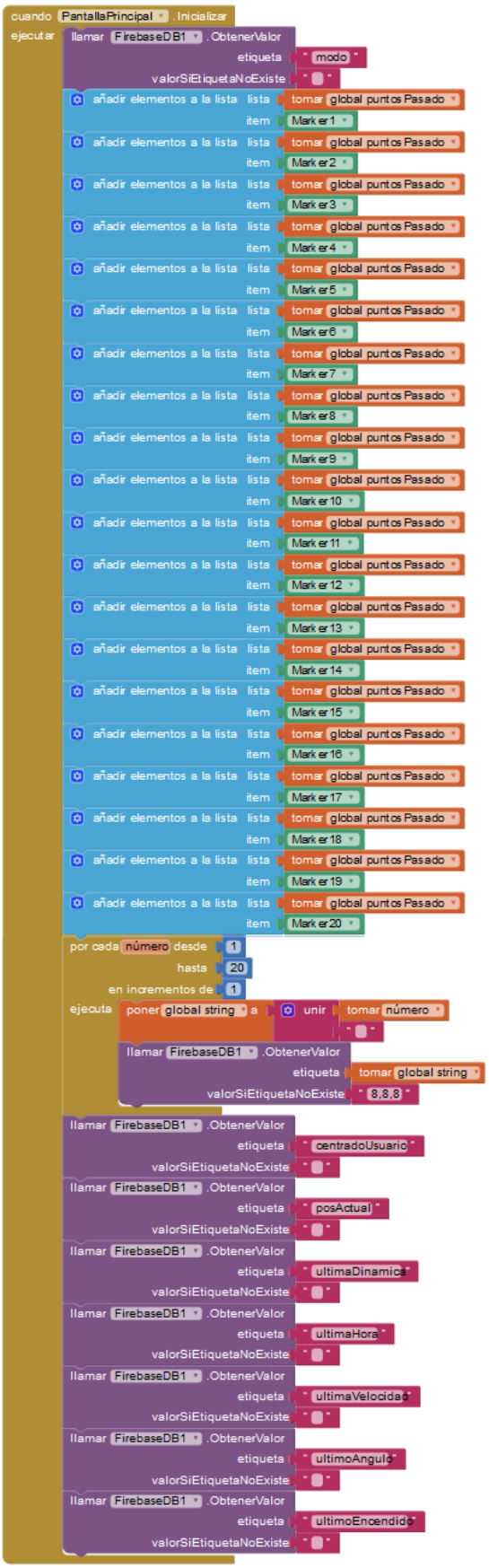

Figura 45 – Bloque de inicialización y solicitud de Información a la base de datos, de la pantalla principal.

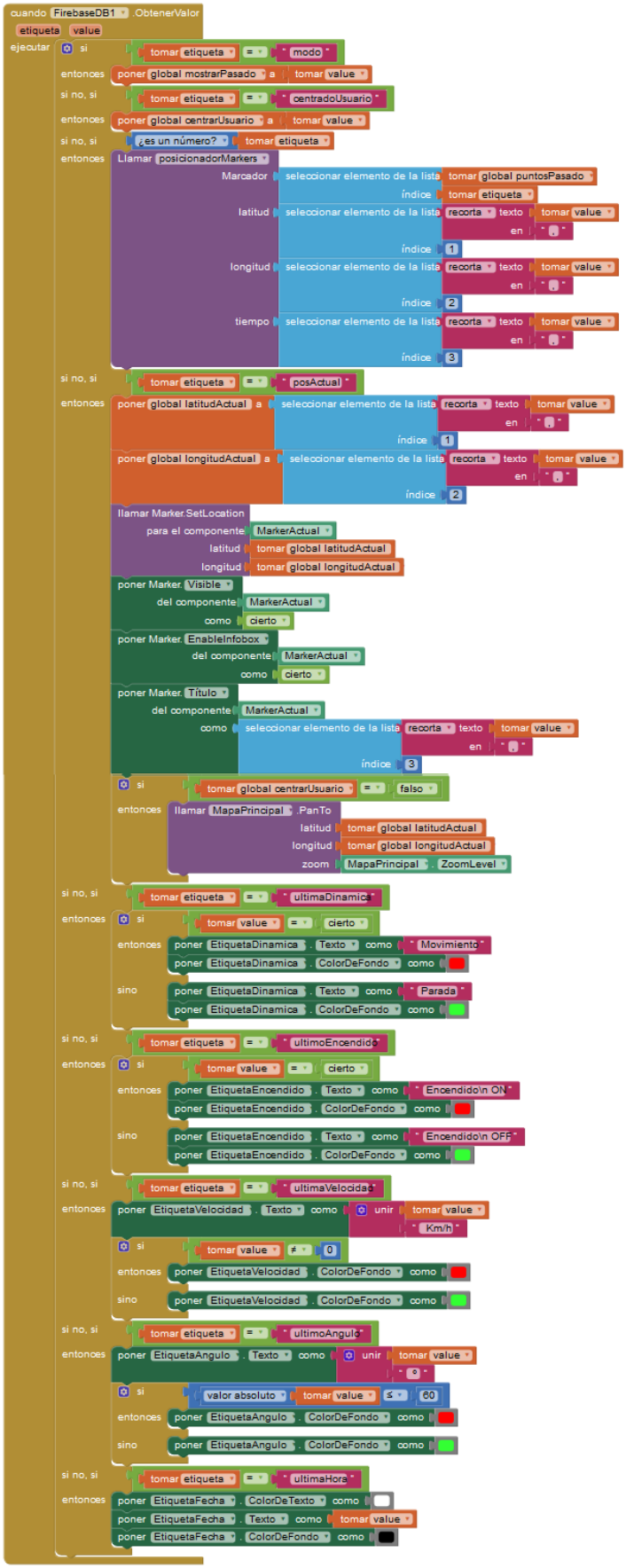

Figura 46 – Bloque de lectura y asignación de la pantalla principal.

Este segundo bloque (que se muestra en la figura 46) consiste en unas estructuras del tipo Si-Entonces-Sino anidadas que van detectando las etiquetas de la información recibida desde Firebase y la va tratando de forma correcta. Así la información del estado de la motocicleta se utiliza para escribir las etiquetas de la barra superior, y a seleccionar el fondo de dichas etiquetas, en función de su contenido. Así una etiqueta con fondo verde indica que la situación del vehículo es normal, o es la esperable, mientras que un fondo rojo indica que el estado es de alta sospecha de que pueda estar siendo manipulado de forma incorrecta, o también en caso de accidente.

Para la lectura de la información de los marcadores de posición se ha desarrollado una función denominada posicionadorMarker, que se muestra en la figura 47, y que simplemente lee la información, desde Firebase, de las coordenadas y la leyenda de las posiciones históricas del vehículo y las asigna a las variables de los marcadores. Además dará, o no, visibilidad a estos marcadores en función del usuario ha solicitado ver el histórico o sólo la última posición.

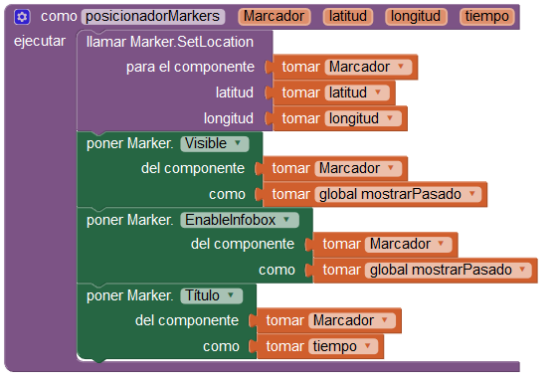

Figura 47 – Bloque de la función posicionadorMarkers.

Como se ha indicado, los bloques comentados son ejecutados cuando la pantalla principal se inicializa, pero esta no es el único evento en que se requiere que la información de esta pantalla se actualice, de hecho, cada vez que el TMR modifique una posición en la base de datos remota la información de la pantalla debería actualizarse con esta última información.

Para conseguir que esto ocurra se ha realizado un nuevo bloque que se activa cada vez que existe un cambio en la base de datos (determinada por el objeto FirebaseDB1), el evento que detecta este cambio es Cuando FirebaseDB1.ObtenerValor, en ese instante se vuelve a solicitar a FirebaseDB1 que se obtengan todos los valores, que cuando llegan a la aplicación son gestionado, por el segundo bloque (figura 46) del mismo modo que cuando la pantalla se inicializa. Este bloque de actualización se muestra en la figura 48.

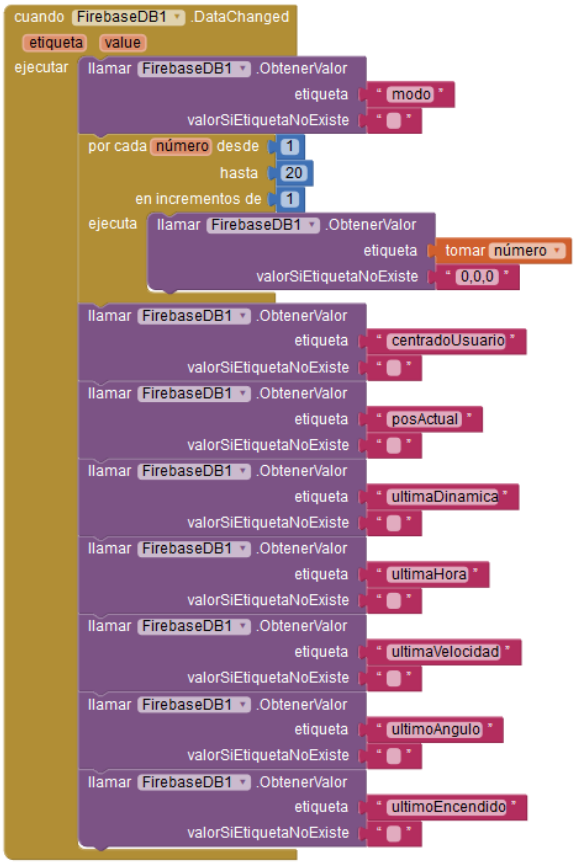

Figura 48 – Bloque de actualización automática.

Aunque existe algunas funcionalidades más en esta pantalla principal vamos a comentar solo 2 bloques más.

En la figura 49 se muestra el bloque que genera la posición del usuario, utiliza el objeto del sensor de ubicación incluido en la pantalla, y en concreto el método que detecta un cambio de ubicación de dicho sensor, que se ha de recordar que no es más que el GPS del teléfono inteligente.

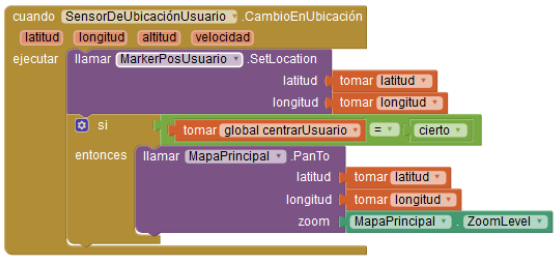

Figura 49 – Bloque que actualiza el marcador de la posición actual del usuario.

Por último, en la figura 50, se muestra el bloque que cierra la aplicación en el caso de que el usuario lo solicite.

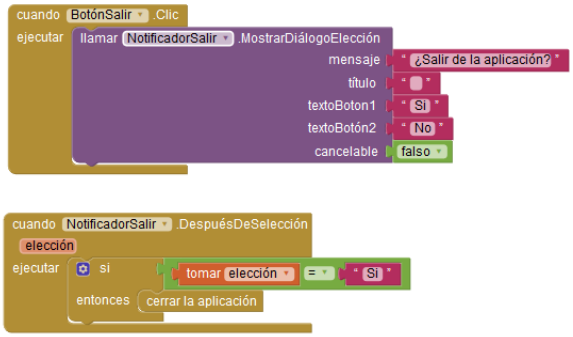

Figura 50 – Bloque de cierre de la aplicación.

Pantalla de configuración:

En esta pantalla, como su nombre indica, el usuario puede comprobar los parámetros de configuración del sistema y modificarlos en el caso que así lo considere. En la siguiente figura se muestra se muestra el aspecto de esta pantalla de configuración.

Figura 51 – Pantalla de configuración.

Los parámetros que pueden ser configurados están almacenados en la base de datos de Firebase y zona accesibles tanto por la aplicación como por Arduino. Los parámetros son los siguientes:

- marca, modelo y matrícula de la motocicleta. Estos parámetros no tienen una funcionalidad en sí, sirven solo para identificar la motocicleta.
- Seguimiento continuo: en el caso de que esté activado, en el mapa se muestran la trayectoria seguida por el vehículo, si está desactivado solo se muestra la posición última registrada del mismo.
- Periodo de rastreo que indica cada cuanto tiempo el TMR envía la posición y el estado de la motocicleta al *cloud.* Las opciones que puede adoptar este parámetros pueden ser 15, 30, 60 y 300 segundos.

- Centrado. El mapa puede ubicar en el centro de la pantalla al usuario o la motocicleta que se está rastreando.
- Teléfono del usuario. Arduino lo lee de Firebase y lo utiliza para realizar comunicaciones directas SMS con el usuario, en caso de urgencia.
- Número de la SIM del rastreador, en este caso el la app la que utiliza este dato para comunicar directamente con el rastreador en el caso de que lo requiera.

Cuando el usuario pulsa el botón de Aceptar la aplicación salva los nuevos parámetros en Firebase y quedan accesibles de nuevo para la propia aplicación o para el Arduino que configura el TMR.

Desde esta pantalla, obviamente, el usuario puede regresar a la pantalla principal.

En la siguiente figura se muestra tres situaciones en las que se está modificando distintos parámetros dentro de la pantalla de configuración.

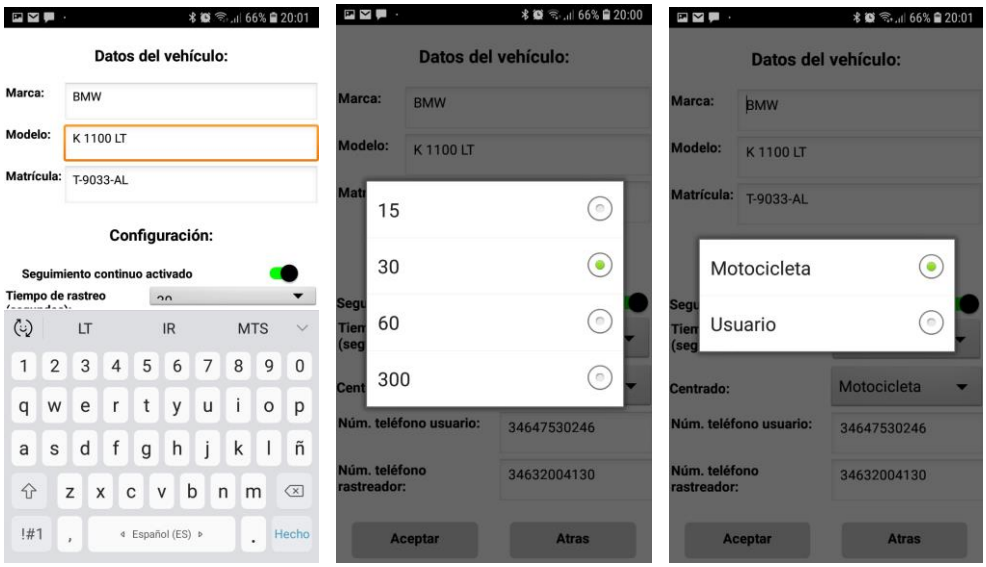

Figura 52 – Pantalla de configuración, en distintas situaciones.

Como ya se ha mencionado esta aplicación ha sido desarrollada con App Inventor 2 y utiliza los mapas de Open Street Maps (OMS) que es un proyecto colaborativo para crear mapas de acceso libres y editables por los miembros de la comunidad. También se podría haber utilizado Google Maps, pero se quería explorar otro tipo de producto que le diese a la aplicación un aspecto no tan habitual como son los grafismos de los mapas de Google.

El diseño de esta pantalla de configuración ha sido mucho más sencillo que el de la anterior. Su vista “Diseñador” se muestra en la siguiente figura.

Figura 53 – Vista "Diseñador" de la pantalla de configuración.

Contiene componentes disposiciones horizontales y verticales, varias etiquetas, dos botones (uno para validar los datos introducidos y otro para regresar a la pantalla principal, marcado con la leyenda "Atrás"), componentes campo de texto (para introducir los datos por teclado, un notificador, un interruptor y de nuevo un componente para el acceso a la base de datos Firebase.

Entre los bloques funcionales podemos destacar los siguientes.

Cuando la pantalla se inicializa se solicita remotamente a los servidores de Firebase los valores de configuración, ya mencionados, marca, modelo, matrícula, periodo de lectura, los número de teléfono y SIM, y los modos de seguimiento y centrado. La programación gráfica de este bloque de solicitud de lectura desde el *cloud* se muestra en la siguiente figura.

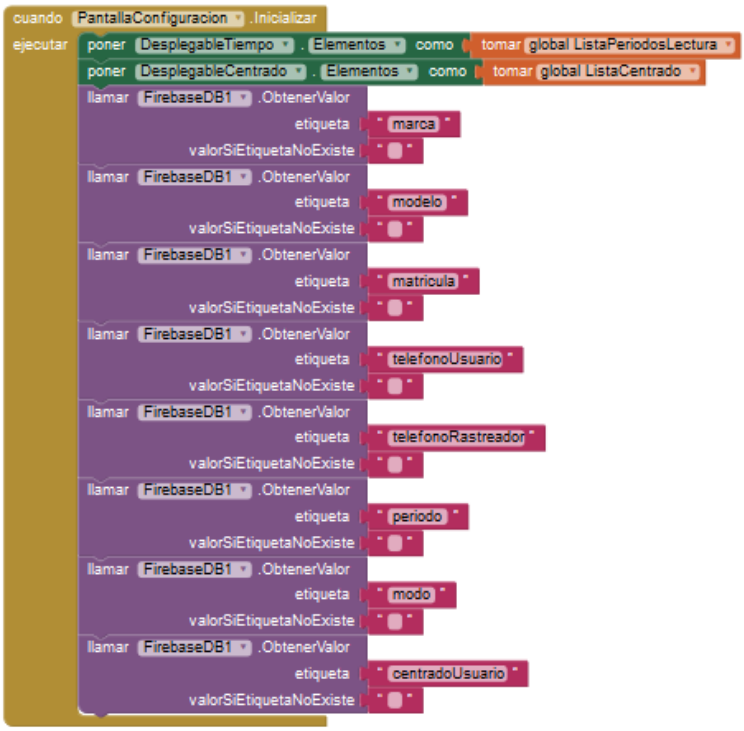

Figura 54 – Bloque de solicitud de lectura desde Firebase, de la pantalla de configuración.

Como puede verse la petición se realiza mediante el proceso .ObtenerValor del objeto FirebaseDB1, que apunta a la base que se ha generado en Firebase para este proyecto. En cada solicitud se incluye la etiqueta que direcciona el valor en la base de datos. Además se pone un valor por defecto en el caso de que la etiqueta no se encuentre en la base de datos.
Cuando los valores se van recibiendo en la aplicación han de ser correctamente asignados a los objetos componente que los reciben. Para la realización de esta asignación se ha generado una anidación de estructuras alternativas Si-Entonces-Sino (algo similar a lo realizado en la pantalla principal, pero más sencilla ya que en esta pantalla se leen menos valores). Obviamente el filtrado se realiza mediante la correcta selección del campo etiqueta que se recibe desde Firebase a través del objeto FirebaseDB. Los componentes receptores son los valores de los campos de texto, y el valor por defecto de los desplegables, y también de los interruptores. Este bloque puede verse en la figura 55.

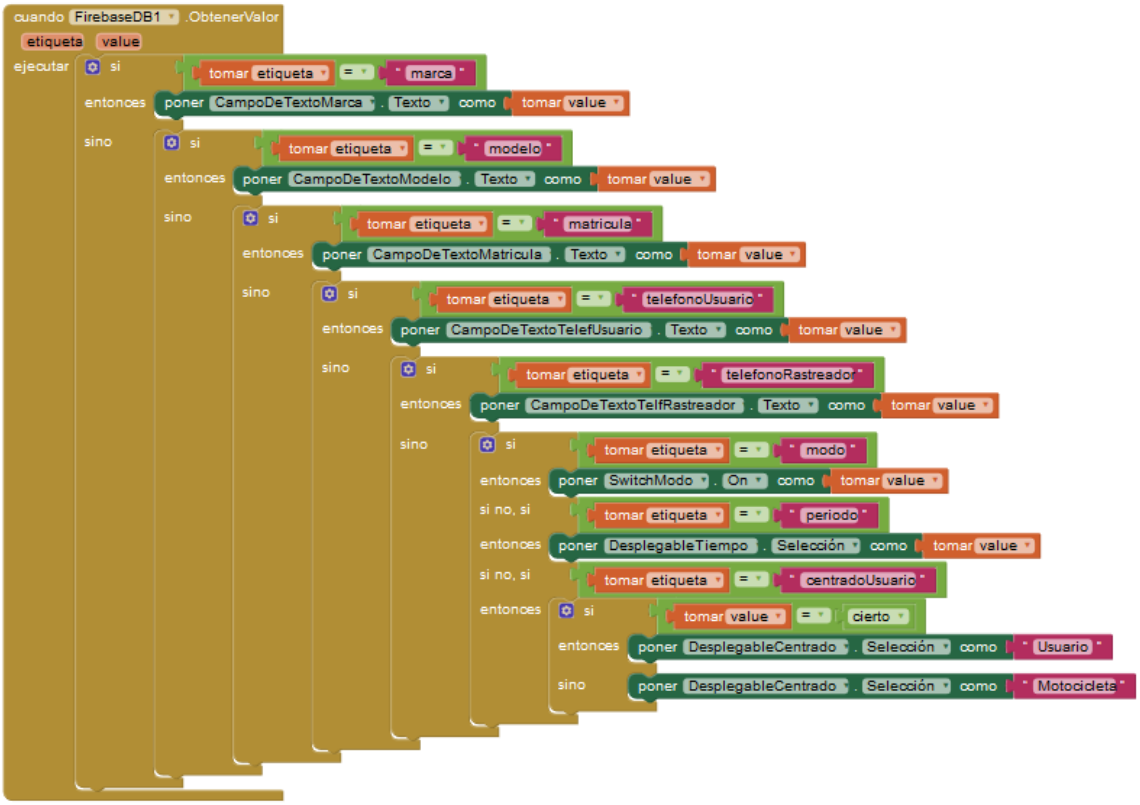

Figura 55 – Bloque de lectura desde Firebase y asignación, de la pantalla de configuración.

Se puede destacar un tercer bloque, que se activa cuando se pulsa el botón aceptar de esta pantalla. En este caso los valores son salvados en la misma base de datos con la que se está trabajando, representada con el objeto FirebaseDB1. Cada valor se guarda en el *cloud* asignándosele la etiqueta correspondiente. Este bloque puede verse en la figura 56.

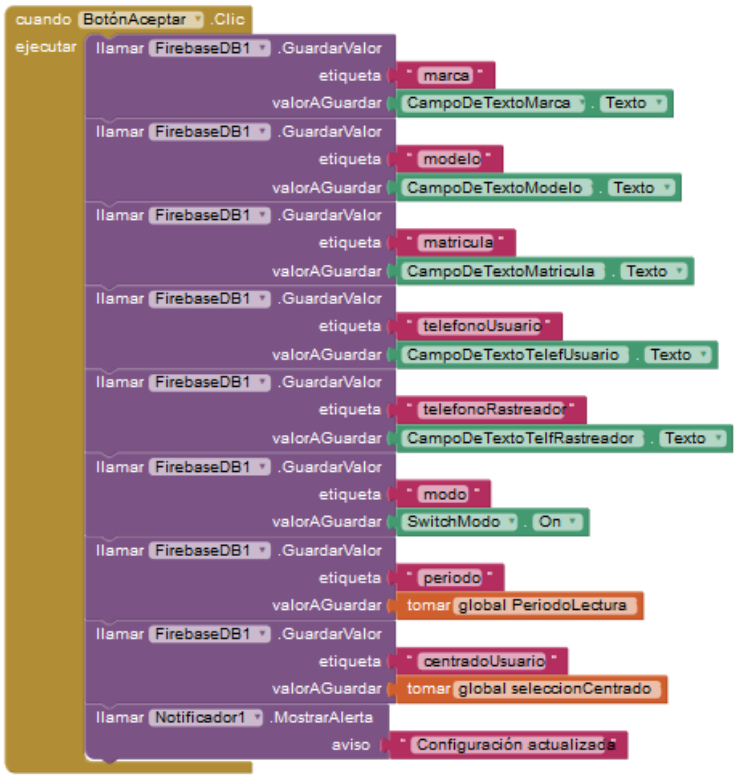

Figura 56 – Bloque de escritura en Firebase, de la pantalla de configuración.

Existe algún pequeño bloque adicional, para el regreso a la pantalla principal, y para la inicialización de alguna de las variables, pero no se explicarán por su sencillez.

Plataforma externa de almacenamiento de datos – *cloud* (PEADC):

La base de datos utilizada la hemos denominado MotoTracker y hemos utilizado un tipo de base de datos denominada de tiempo real. Dentro del IDE de Firabase se han editado las reglas y se han configurado para que permita tanto la lectura como la escritura en la base de agentes externos (en esta caso Arduino y la aplicación anteriormente comentada)

Figura 57 – Selección y configuración de la base de datos.

La base de datos del prototipo demostrador es muy sencilla y almacena una cantidad pequeña de datos. Cada datos utiliza una etiqueta identificativa que es el método utilizado para su direccionamiento.

Los datos pueden ser extraídos en formato JSON.
La estructura de la base de datos, que no es más que una lista muy moderada de parámetros puede visualizarse en la siguiente figura:

https://moto-tracker-51ddf.firebaseio.com/

moto-tracker-51ddf
- 1: "\"41.3916972,2.1892417,15/12/2019 12:30:06\
- 2: "\"41.389735,2.191508,15/12/2019 12:30:45\
- 3: "\"41.388274,2.193407,15/12/2019 12:31:10\
- 4: "\"41.386722,2.195483,15/12/2019 12:31:56\
- 5: "\"41.385598,2.196874,15/12/2019 12:33:00\
- 6: "\"41.383674,2.194944,15/12/2019 12:33:43\
- 7: "\"41.380248,2.192830,15/12/2019 12:35:05\
- 8: "\"41.378920,2.191977,15/12/2019 12:37:15\
- 9: "\"41.378082,2.188206,15/12/2019 12:38:38\
- 10: "\"41.382014,2.186511,15/12/2019 12:40:10\
- 11: "\"41.383065,2.183955,15/12/2019 12:40:38\
- 12: "\"41.379165,2.180192,15/12/2019 12:41:29\
- 13: "\"41.375713,2.177459,15/12/2019 12:42:43\
- 14: "\"41.373375,2.176468,15/12/2019 12:43:50\
- 15: "\"41.372161,2.176326,15/12/2019 12:44:21\
- 16: "\"41.369448,2.175053,15/12/2019 12:44:47\
- 17: "\"41.367026,2.174712,15/12/2019 12:45:25\
- 18: "\"41.366139,2.177934,15/12/2019 12:46:00\
- 19: "\"41.364770,2.181696,15/12/2019 12:46:25\
- 20: "\"41.361687,2.181860,15/12/2019 12:46:56\
- centradoUsuario: "false"
- marca: "\"BMW\""
- matricula: "\"T-9033-AL\"
- modelo: "\"K 1100 LT\"'
- modo: "true"
- periodo: "\"30\""
- posActual: "\"41.351866,2.175771, ahora\'
- telefonoRastreador: "\"34632004130\"
- telefonoUsuario: "\"34647530246\"
- ultimaDinamica: false
- ultimaHora: "\"8/12/2019 20:00:15\"
- ultimaVelocidad: 10
- ultimoAngulo: 85
- ultimoEncendido: false

Figura 58 – Estructura y contenido de la base de datos utilizada.

La conexión con esta plataforma ha sido muy sencilla (uno de los motivos importantes por los que se eligió) solo se ha tenido que tomar dos decisiones tres decisiones:

6. Pruebas funcionales sobre el prototipo

En cualquier proyecto de ingeniería – por sencillo que pueda parecer – la parte experimental es clave para comprobar el funcionamiento de todo el diseño realizado, así como para descubrir aquellos errores cometidos tanto en la fase de diseño conceptual, como en la fase de diseño detallado. Cualquier proceso de desarrollo de un producto requiere de fases de desarrollo teórico que se vayan alternando con fases más prácticas y experimentales, y que tras una serie de iteraciones - que pueden incluir cambios de diseño significativos – consigan que el producto cumpla plenamente con estas especificaciones.

En el caso de este proyecto - debido al tiempo total disponible – solo se ha podido hacer una única e inicial iteración de diseño, sin posibilidad de realizar una segunda para introducir algunos cambios significativos con las modificaciones necesarias para resolver los problemas aprendidos en esta primera iteración.

En este TFG la fase experimental ha sido muy compleja pero muy útil y esclarecedora, y ha forzado a la introducción de algunos pequeños cambios y ajustes. El cambio más significativo que se ha podido introducir a tiempo ha sido la sustitución de Arduino UNO por Arduino Mega 2560. La razón del cambio no ha sido otra que la escasa memoria del primero hacía que el sistema empezase a fallar en cuanto los programas crecían su tamaño.

Arduino UNO cuenta con un microcontrolador que funciona a 16 KHz, una memoria *flash* de 32 KB, una memoria SRAM 2 KB y EEPROM de 1 KB, mientras que Arduino Mega2560 cuenta con un chip que funciona también a 16 KHz, una memoria *flash* de 256 KB, una memoria SRAM 8 KB y EEPROM de 4 KB. Además cuenta con más pines de entrada y salida, aunque no eran necesarios para este proyecto. En la siguiente figura se muestra el montaje actualizado con Arduino Mega2560.

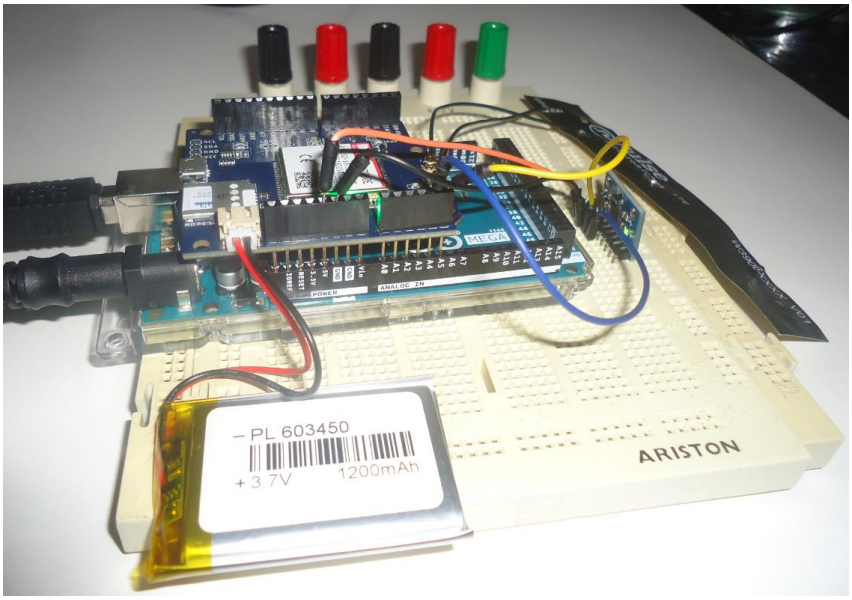

Figura 59 – Sistema con Arduino Mega 2560 instalado.

Además de el cambio de placa, se ha debido mover los pines de SDA y SCL del bus I2C a los pines 20 y 21 que son los que utiliza Arduino Mega2560 para esta función. El resto de conexiones así como el software ha sido totalmente compatible, excepto que se ha cambiado la compilación indicando que era para esta nueva placa.

A parte de este cambio en el hardware, la fase experimental ha servido:

- para comprobar el buen funcionamiento de la aplicación MotoTracker que funciona en el TMS, y en especial su buena integración con la plataforma *cloud* Firebase, con un nivel muy bajo de fallos.
- Para comprobar que el sistema operativo Android, en sus últimas revisiones, no permite que las aplicaciones tengan acceso directo a los SMS, sino que sea la aplicación general de SMS la que controle su recepción. Esto ha impedido que los SMS – procedentes de Arduino, y que pretendían advertir al usuario de eventos inesperados en el estado de su vehículo - se integren completamente con la aplicación MotoTracker, y solo puedan dar advertencia a través de la aplicación general de administración de mensajes de texto.
- Para comprobar el buen funcionamiento de la integración entre el IMU MPU6050 con Arduino. Ha funcionado perfectamente desde el primer momento, sin generar comportamientos erráticos de ningún tipo. Tal solo se hubo de proceder a incluir unas líneas de código en Arduino para compensar los errores de offset del sensor. Se ha conseguido de forma fácil una medida del ángulo de inclinación de la moto (para detección de posibles caídas , así como una detección de movimientos.
- Para comprobar el buen funcionamiento de la SIM7000 en las funcionalidades de:

    - la conexión con la red LTE CAT1 M, de bajo consumo y que encaja muy bien con aplicaciones del tipo IoT, como podríamos considerar la geolocalización de un vehículo.
    - El envío y recepción de mensajes SMS a través de la citada red.
    - La conexión entre la SIM7000 y algunas plataformas *cloud*, como por ejemplo dweet.io.
    - Pero por desgracia, también ha servido para detectar un error de integración entre la SIM7000 y la plataforma *cloud* Firebase.

Dado que este último punto ha sido el que más ha condicionado la obtención de los objetivos propuestos, se va a comentar en primer lugar.

Problemas de comunicación entre SIM7000 y Firebase:

La selección de la plataforma *cloud* se basó fundamentalmente en dos ventajas que esta plataforma mostraba con respecto a otras opciones:

-La facilidad de integración con App Inventor 2.
-La facilidad de integración con placas de comunicación para Arduino.

La primera ventaja se ha demostrado ampliamente cierta. El IDE de App Inventor 2 dispone de objetos Firebase, y posibilita un desarrollo sencillo, ágil, y con unos resultados muy fiables. En uno de los siguientes apartados se muestra alguna evidencia de esto último.

En la segunda ventaja claramente se han cometido errores en la interpretación. Es cierto que existe mucha documentación de cómo integrar un dispositivo Arduino con la plataforma Firebase. Existen varias librerías, pero en particular ArduinoFirebase que cuenta con un gran número de métodos para realizar operaciones sobre las bases de datos residentes en Firebase.
Cuando estas librerías se incorporaron al sketch que se ejecuta en Arduino UNO o Mega2560 (ambos se comportan igual), el propio IDE avisaba de que esas librerías no se podían utilizar con Arduino sino con otra placa de desarrollo denominada ESP8266, que tiene características similares a Arduino, y que incorpora electrónica para la comunicación WiFi basadas en el estándar 802.11. Muchos de los métodos hubieran funcionado bien – al menos desde un punto de vista lógico – pero están pensados para comunicación de objetos – tipo IoT- en una LAN WiFi, pero no en la red celular LTE o GSM/GPRS con la que nuestro proyecto trabaja.

Una vez detectado este primer error se realizó una búsqueda de librerías similares para trabajar con la SIM7000, pero no se encontró ninguna solución disponible. En este punto se intentó otra estrategia como es utilizar el método .post de la librería Adafruit_FONA (que es la base de comunicaciones celulares utilizada en este TFG) para comunicar mediante el protocolo HTTP con la base de datos. Las primeras pruebas se realizaron con la base de datos dweet.io, y el resultado fue el correcto, solo que ya se detectaron algunas limitaciones importantes, como son que la librería solo soporta los comandos HTTP GET y POST, y no PATCH que probablemente sería el más apropiado para la funcionalidad que se le pretendía dar al TMR.

Cuando se intentó comunicar Arduino a través de la SIM7000 con Firebase se presentaron muchos problemas, para intentar entenderlos mejor se descargaron varias API REST, como Insomnia y AdvancedRestClient 14.0.4. REST es una acrónimo de las palabras inglesas *Representational State Transfer*, se trata de un estilo de arquitectura de comunicación para diseñar aplicaciones en red. La ventaja fundamenta de REST es que ofrece una alternativa sencilla para tecnologías complejas como CORBA, RPC o SOAP, ya que utiliza HTTP, siendo el motivo fundamental que este protocolo permite compartir información entre un cliente (portátil, teléfono móvil, tableta, etc.) y un servidor. En el caso de este TFG se realizó una comunicación con Insomnia y se consiguió comunicar con la base de datos establecida en Firebase. El código HTTP utilizado se muestra en la siguiente figura:

```
1  POST /modelo.json HTTP/1.1
2  Content-Type: application/json
3  Host: conexionbasica-47951.firebaseio.com
4  Content-Length: 21
5
6  {
7    "amigo":"manolo"
8  }
```

Figura 60 – Codigo HTTP generado por Insomnia cuando comunica con Firebase.

Como se puede ver en la transacción se intercambia varios parámetros, como son:

- el comando en sí, en este caso la instrucción POST a un objeto JSON de la base de datos.
- El tipo de contenido: Content-Type, que en ese caso es un application/json
- El Host, que indica la URL de la base de datos (en este caso es una base de prueba en Firebase)
- La longitud del contenido.
- El contenido del apunte en la base en formato JSON.

Cuando se intentó realizar el envío del mismo formato se comprobó, que Adafriut_FONA cuenta con métodos que permiten el mismo formato indicado, pero solo están disponibles cuando el chip es SIM7500, muestras que en proyecto se había adquirido una placa con un SIM7000. Parece que las razones son que SIM7500 permite utilizar un conjunto de comandos AT (también denominados comandos Hayes) más amplio que el que utiliza la SIM7000, y entre esos comandos AT adicionales, se encuentran los que permiten realizar la comunicación con el formato HTTP correcto. Preguntado el fabricante de la placa que integra el chip SIM7000 de SimCom en la placa *shield* adquirida para el desarrollo de este TFG, confirmó la imposibilidad de utilizar las funciones de Adafruit_FONA para conseguir comunicar con Firebase.

Esta problemática se descubrió el día 30 de diciembre, casi ya sin tiempo para poder finalizar el TFG. En este contexto se plantearon varias posibles soluciones:

a) Adquirir una nueva *shield* que utilizase el chip SIM7500. Esta era probablemente la mejor solución, pero la placa no podría testearse hasta mediados de enero (debido al plazo de entrega de la nueva *shield*) y además no queda claro si puede soportar el comando PATCH de HTTP.

b) Generar código propio basado en el set de comandos AT que soporta SIM7000. Esta solución se descartó por no disponer de la experiencia con los comandos AT (ni el tiempo para adquirirla). Además no se tenía

la certeza de que con ese set de comandos se pudiese conseguir el objetivo.

c) Cambiar la plataforma *cloud* a otra más sencilla, como por ejemplo dweet.io. Esta solución parece ser factible, pero hacía que todo el código de la aplicación del TMS tuviera que realizarse de nuevo, sin quedar tiempo material para ello.

d) Cambiar Arduino por una ESP8266. Se descartó porque la comunicación se quiere establecer a través de las líneas celulares, y no por conexión WiFi. La comunicación WiFi no es la correcta para un rastreador que se puede encontrar en cualquier punto.

e) Simplificar y reducir la funcionalidad del prototipo a la utilización de comandos SMS. Esta solución, a pesar de que es mucho menos elegante, y claramente no cumple con las especificaciones del proyecto, iguala las prestaciones de algunos de los productos – los de gama baja – del mercado pero puede ser desarrollado con el poco tiempo restante hasta la entrega del TFG. Para ello se desarrolló el programa para Arduino Mega2560, MotoTracker_solo_SMS_V0_2.ino, que ha sido comentando en el capítulo 5, y que está listado en el Anexo de este mismo documento.

Los mensaje SMS al usuario por el citado programa se envían mediante una conexión LTE CAT1 M de bajo consumo.

Funcionamiento del IMU MPU 6050 integrado con la SIM7000 para el envío de mensajes de advertencia:

Esta pequeña placa se encuentra perfectamente integrada en el ecosistema Arduino. Tanto su hardware como el software (librerías) han funcionado perfectamente desde el primer momento.
Como ya se ha comentado el IMU MPU6050 es el encargado de dos funciones:

- de informar el estado dinámico del vehículo, detección de movimientos y vibraciones, así como en la detección del ángulo de inclinación de la motocicleta.
- Disparar el envío de SMS cuando se detecten vibraciones y movimientos y/o ángulos de inclinación por encima del umbral para considerarlos caída.

Para testear su funcionalidad las pruebas han sido muy sencillas. Se ha movido el *protoboard* donde se encuentra montado Arduino, la SIM7000E y el IMU, de distintas formas para comprobar la funcionalidad.

- Prueba de vibraciones: se ha movido el sensor, pero sin llegar a inclinarlo más 45º con respecto a la vertical, como se muestra en la siguiente figura. En este caso el sistema ha enviado un SMS indicando que el evento había acaecido, pero como en ese momento aún no tenía cobertura de GPS el SMS lo indica

claramente. Cuando se cuenta con cobertura de GPS el TMR envía el SMS acompañado de un enlace a Open Street Map con las coordenadas de donde se encuentra el vehículo. En la siguiente figura se muestra la pantalla del TMS con los tres mensajes recibidos:

- Sistema arrancado.
- Detección de vibraciones, sin cobertura.
- Detección de vibraciones, con cobertura de GPS y por tanto con un enlace.

Si el usuario pulsa en el enlace, inmediatamente se abre una nueva ventana donde puede visualizar en Open Street Map las coordenadas geográficas desde donde ha sido enviado el mensaje – indicadas con un marcador azul –
En este TFG se ha utilizado OPS pero podría haberse realizado con Google Maps, u otras plataformas de mapas abiertas.

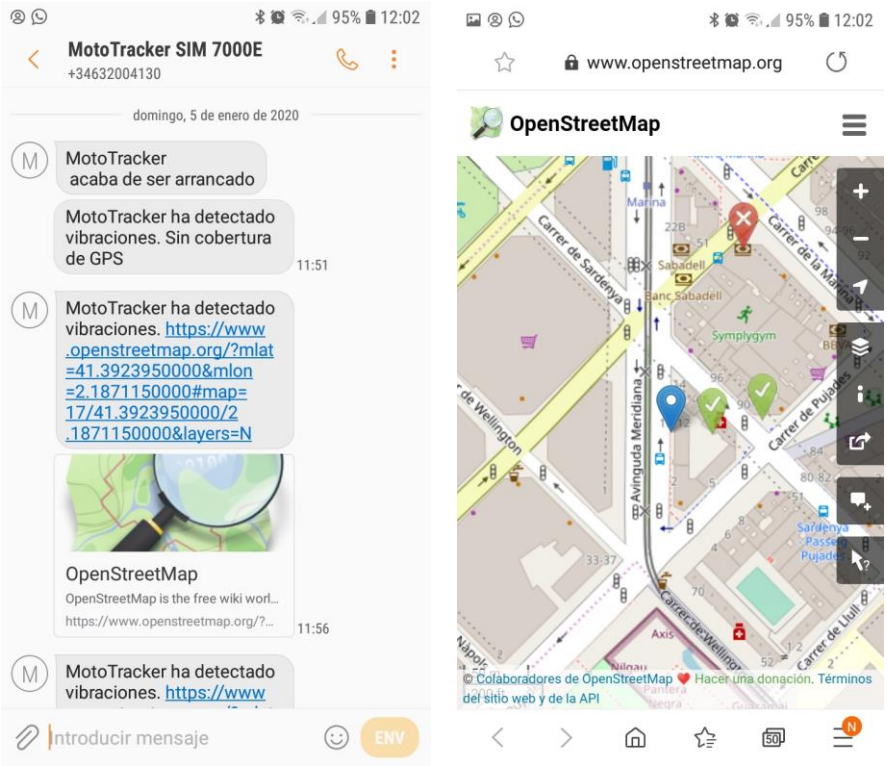

Figura 61 - Pantalla del TMS con los SMS recibidos, y la pantalla despues de pulsar sobre el enlace.

- Prueba de inclinación y vuelco: en este caso la placa se ha movido hasta que el ángulo si ha sobrepasado los 45º, y por tanto el sistema avisa indicando las dos posibles alteraciones vibraciones y vuelco. Esta situación podría responder a una posible vuelco de la moto, por intento de hurto, o por accidente. El sistema ha funcionado según lo esperado, en general siempre que se produce un vuelco el primer aviso es por vuelco y vibraciones.

- Por último se ha mantenido el sistema inmóvil (sin vibraciones ni movimiento alguno) pero con una inclinación por encima del umbral citado. En este caso el mensaje recibido indica solo la posibilidad de vuelco.

A partir del primer envío de SMS las notificaciones se reproducen con un periodo que coincide con el periodo de refresco indicado por el usuario. Los mensajes cesarían cuando las alteraciones desaparecen durante 5 periodos consecutivos -esta funcionalidad no se ha implementado en el programa MotoTracker_solo_SMS_V0_2.ino aunque sería muy sencillo incorporarlo. El permanente envió de SMS permitiría el seguimiento del vehículo en caso de robo, o de accidente.
En la siguiente figura se muestra la recepción sucesivas de mensajes según el periodo especificado (en este caso 30 segundos, aunque el valor es programable). Si la alteración -vibraciones o vuelco – desaparecen y no son detectadas por el IMU los mensajes SMS dejan de enviarse.

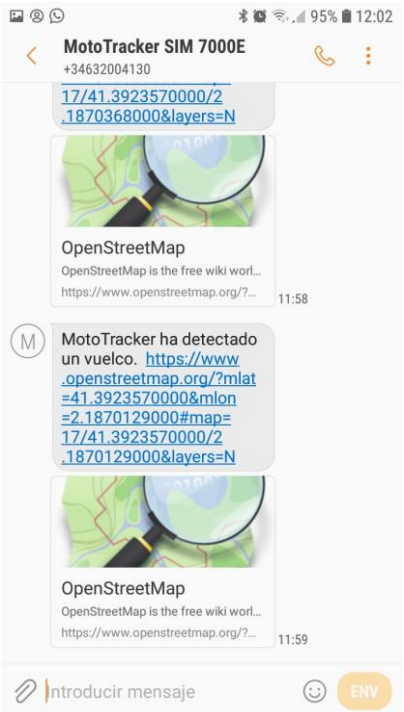

Figura 62 – Llegada de mesnajes cada 30 segundos, sino desaparece la alteración.

Como ya se ha mencionado, la funcionalidad de que toda esta información sea subida al *cloud*, para poder ser monitorizada más cómodamente desde la app, no se ha podido probar por la incompatibilidad de la SIM700E con Firebase. Por tanto, las pruebas realizadas con la aplicación

Funcionamiento de la aplicación MotoTracker:

Como ya se ha mencionado, la funcionalidad de que toda esta información sea subida al *cloud*, para poder ser monitorizada más cómodamente desde la app, no se ha podido probar por la incompatibilidad de la SIM700E con Firebase.

Por tanto, las pruebas realizadas con la aplicación no han sido distintas a las realizadas durante el desarrollo de la misma.

En el capítulo anterior ya se ha mostrado una parte importante del funcionamiento de las pantallas.

Para poder chequear la funcionalidad se han realizado varios test.

Se han recorrido varios kilómetros con la aplicación abierta, y con el parámetros de centrado en el usuario. La aplicación ha funcionado perfectamente, pero se ha descubierto una posible mejora, y es que la aplicación no le indica al sistema operativo Android que no entre en modo ahorro de energía, lo cual origina que el sistema operativo haga que la pantalla se pague, y para recuperarla se requiera introducir la palabra clave de acceso del usuario. Esta funcionalidad es una mejora obligada para una segunda versión de la aplicación.

Por otra parte se han realizado pruebas de actualización de las posiciones en Firebase (introducidas a mano, y no con Arduino y la shield SIM7000 como se pretendía) y el funcionamiento ha sido el esperado, sin ningún modo de fallo detectado.

Es decir podríamos decir que la aplicación MotoTracker ha alcanzado todos los objetivos marcados inicialmente, excepto la recepción directa de mensajes SMS, que como ya se ha indicado está bloqueada por el sistema operativo Android de Google.

7. Presupuesto de ingeniería

En este capítulo se presenta una sencilla valoración de los costes incurridos para la realización del TFG. No presenta una estimación de los posibles beneficios que podría llegar a generar el proyecto, ya que se reduce a una primera fase de prototipaje, lejana de una posterior industrialización.
El presupuesto se ha dividido en dos bloques:

- materiales del proyecto: que engloba los mínimos gastos realizados en la adquisición de componentes para la puesta en marcha del prototipo.
- Recursos de ingeniería, donde se ha realizado un cómputo de las horas empleadas en las distintas actividades, así como una tasación en función de la naturaleza de la actividad. Los costes horarios están basados en estimaciones de coste de mercado, que empresas de ingeniería cargan a sus clientes.

Con todo ello, el coste aproximado del proyecto sería de unos 14.000 euros.

Presupuesto materiales del proyecto:

#	Concepto	Cantidad	Precio unitario	Precio
1	Arduino UNO	1	25,0 €	25,0 €
2	Botletics SIM 7000 E	1	95,0 €	95,0 €
3	IMU MPU-6050	6	2,5 €	15,0 €
4	Bateria Li-PO 3.V 1200 mAh	1	14,0 €	14,0 €
5	Protoboard	1	10,0 €	10,0 €
6	Cables	50	0,0 €	0,5 €
7	Fuente alimentación 12 V Arduino	1	10,7 €	10,7 €
8	Cable USB Arduino	1	5,5 €	5,5 €
9	Tarjeta prepago SIM LycaMobile - registro	1	2,5 €	2,5 €
10	Tarjeta prepago SIM LycaMobile - cuota mensual	2	10,0 €	20,0 €
	Total - materiales del proyecto			**198 €**

Presupuesto recursos de ingeniería:

#	Concepto	Cantidad	Precio unitario (€/h)	Precio
1	Análisis productos rastreadores (estado del arte)	50	25	1.250 €
2	Análisis del mercado (estado del arte)	4	25	100 €
3	Experimentación con productos reales	2	25	50 €
4	Investigación y selección de componentes tecnológicos (estado del arte)	100	50	5.000 €
5	Análisis de plataformas cloud gratuitas (estado del arte)	5	50	250 €
6	Desarrollo de la arquitectura y el diseño conceptual	25	50	1.250 €
7	Compra de los componentes	2	15	30 €
8	Montaje e integración	10	50	500 €
9	Desarrollo software de Arduino	45	50	2.250 €
10	Desarrollo de la app TMS	20	50	1.000 €
11	Experimentación/testeo prototipo	50	35	1.750 €
12	Confección memoria	16	25	400 €
	Total horas	**329**		
	Total - recursos de ingeniería			**13.830 €**

Total ejecución proyecto	**14.028 €**

8. Conclusiones

Las conclusiones principales de este TFG han sido, en lo que hace referencia al producto desarrollado:

- El mercado de los rastreadores GPS es un sector en crecimiento, y con una oferta muy variada, tanto en prestaciones, como en precio, como en formato. La oferta es muy superior a lo que se podía esperar en un primer momento. Existe una importante oferta de productos de pocas prestaciones, a un precio muy reducido, y una gama especializada en aplicaciones concreta, que ofrecen una app propia, y que, en consecuencia, tienen un precio más alto.

- La gran mayoría utilizan las redes de telefonía celular, pero empieza a aparecer productos que utilizan redes especializadas en IoT como puede ser LPWAN (como el de la empresa Sigfox).

- En general cuentan con arquitecturas similares, donde las comunicaciones suelen darse entre los terminales de rastreo y seguimiento, pero muchos de ellos, además, vuelcan datos en algún servidor *cloud*.

En cuanto a las tecnologías elegidas para hacer el prototipo:

- El ecosistema Arduino es muy apropiado para el desarrollo de un prototipo conceptual demostrador, ya que además del microcontrolador, existe oferta de *shields* para las comunicaciones celulares LTE, y también dispositivos para la detección de aceleraciones y giros. Aunque no ha formado parte del alcance de este TFG se podría haber añadido de forma fácil algún dispositivo actuador, que desconectase algunos de los servicios básicos del vehículo o por el contrario, disparase alguna alarma.
- En cualquier caso, Arduino en ocasiones parece poco profesional, debido a sus limitadas prestaciones. De empezar este TFG de nuevo, se exploraría con más profundidad la plataforma de desarrollo Rapsberry Pi.

- Se ha podido comprobar que hay varios proyectos que han utilizado tecnologías parecidas para hacer rastreadores GPS, tanto basados en Arduino, como en otras plataformas de desarrollo.

En cuanto a la metodología empleada y los retos principales encontrados durante el desarrollo del TFG:

- Una de las complejidades del TFG ha consistido en realizar una buena arquitectura y diseño conceptual, ya asegurar que se pueden cumplir todas las prestaciones incluidas en las especificaciones del prototipo.

- La segunda gran complejidad es hacer que todo funcione según lo previsto. La parte práctica es compleja, debido en parte a que la documentación existente suele ser muy informal, y en multitud de ocasiones se ha de recurrir a las fuentes para conseguir resolver los problemas.

- La tecnología cada vez es más compleja, y para cualquier nuevo desarrollo se requiere de mucho conocimiento detallado, el cual no está siempre accesible. Desarrollar un prototipo en tan poco tiempo y fuera de un entorno profesional y colaborativo ha sido complejo y estresante.

- La parte práctica ha sido clave para aprender cosas que inicialmente se pensaba debían funcionar, pero que en realidad no lo hacía. “El diablo está en los detalles”.

- La planificación se ha intentado seguir a rajatabla, pero han parecido retrasos fundamentalmente en aquellas actividades prácticas, donde algo se tenía que hacer funcionar. En ocasiones los fallos descubiertos en la puesta en marca de alguna funcionalidad a requerido de la compra de nuevos componentes con el retraso que eso ha originado. Otra fuente de retraso han sido la carga profesional del autor, con viajes, y picos de trabajo que han dificultado la realización de algunas actividades, de nuevo en especial las actividades practicas con los prototipos, ya que requieren algo más que un PC.

- La metodología seguida ha sido la propuesta en un principio y ha funcionado bien en líneas generales.

- A pesar de que el TFG no tiene una fuerte componente innovadora, el mero hecho de realizar una implementación práctica le aporta una dosis muy importante de complejidad, en especial en un periodo de tiempo con corto como son los tres meses disponible para su realización.

En cuanto a los resultados obtenidos:

- Los resultados que se han alcanzado han sido satisfactorios, pero no se ha podido completar los objetivos marcados. Con lo aprendido hasta ahora se necesitaría unas 3 o 4 semanas adicional para volver a reintentar el arranque del prototipo integrado con la base de datos Firebase, esta vez utilizando el chip SimCom SIM7500.

- Creo que estaría bien que el TFG, que son 12 créditos, pudiese desarrollarse en una modalidad de 2 cuatrimestre (6 créditos y 6 créditos) para poder tener un poco más de tiempo para poder madura el proyecto, y completarlo. En especial si tiene parte experimental como éste.

En cuanto a los posibles siguientes pasos:

- Obviamente el siguiente paso sería volver a realizar las pruebas o bien con el nuevo chip mencionado, o bien utilizando una base de datos, que pueda ser accedida con el SIM7000, y las librerías existentes.

- Además, este TFG podría ser ampliado, mediante la realización de una especificación de producto más innovadora que incluya elementos diferenciales con respecto a la competencia, que refuercen la propuesta de valor del producto.

- Además debería hacerse una diseño de una electrónica (y su correspondiente software/firmware) ad-hoc para este producto, basada en un microprocesador o microcontrolador, que permitiera:

    - Nuevas funcionalidades.
    - Menor coste.
    - Mayor integración y menor volumen.
    - Menores consumos.

9. Glosario

2G: segunda generación de telefonía móvil.

A/D: convertidor analógico a digital.
Android: sistema operativo de Google para dispositivos móviles.
API: *Application Programming Interface*, interfaz de programación de aplicaciones, es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
App: aplicación o programa informático.
App Inventor 2: IDE de desarrollo de aplicaciones móviles para el sistema operativo Android desarrollado por el Instituto de Tecnología de Massachusetts (MIT - *Massachusetts Institute of Technology*)
AT: referido a los comandos AT (de atención) y también conocidos como comandos Hayes, es un lenguaje desarrollado por la compañía Hayes Communications que prácticamente se convirtió en estándar abierto de comandos para configurar y parametrizar módems.
CC: corriente continua.
Cloud: servidor en la nube.

Firmaware: programa informático inalterable que controla al más bajo nivel los circuitos electrónicos de un dispositivo de cualquier tipo.
Fix: se refiere al estado de un GPS cuando se encuentra correctamente conectado con los satélites de geolocalización.
Flash: se refiere a la memoria *flash*, que es una clase de chip que se emplea para el almacenamiento y el traslado de datos. Se podría definir como una EEPROM cuya programación y su borrado se concreta en bloques de gran tamaño.

Geo Fence: geovalla o geocerca es un perímetro virtual de un área geográfica del mundo real.
GND: *Ground*, normalmente referido al pin de masa o 0 V.
GNSS: *Global Navigation Satellite System*, sistema global de navegación por satélite.
GPIO: *General Purpose Input/Output,* entrada/salida de propósito general, hace referencia a un pin genérico en un chip, cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede programar por el usuario en tiempo de ejecución.
GPRS: *General Packet Radio Service*, servicio general de paquetes vía radio.
GPS: *Global Positioning System*, sistema de posicionamiento global.
GPS *tracker*: rastreador GPS.
GSM: *Groupe Spécial Mobile*, *Global System for Mobile communications* sistema global para las comunicaciones móviles.

HDMI: *High-Definition Multimedia Interface*, interfaz multimedia de alta definición, se trata de una norma de video, cifrado sin compresión apoyada por la industria para que sea el sustituto de las anteriores normas de video análogo como son el conector RCA y el euroconector.

HTTP: *Hypertext Transfer Protocol*, protocolo de transferencia de hipertexto. Es el protocolo de comunicación original que permite las transferencias de información en la *World Wide Web*.

I2C: *Inter-Integrated Circuit*, circuito inter-integrado, bus de comunicación serie síncrono entre distintos componentes electrónicos.
IDE: *Integrated Development Environment*, entorno de desarrollo integrado, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
IEEE: *Institute of Electrical and Electronics Engineers*, Instituto de Ingeniería Eléctrica y Electrónica es una asociación mundial de ingenieros dedicada a la normalización y el desarrollo en áreas técnicas.
iOS: sistema operativo de Apple para dispositivos móviles.
IoT: *Internet of Things*, internet de las cosas.
IMU: *Inertial Measurement Unit*, unidad de medición inercial.

JSON: *JavaScript Object Notation*, o notación de objetos para JavaScript, se trata de un formato que permite la clasificación e intercambio de datos, principalmente utilizado en bases de datos.

LED: *Light-Emitting Diode*, diodo emisor de luz.
LiPo: Litio Polímero, se refiere a un tipo muy popular de baterías.
LPWAN: *Low Power Wide Area Network,* red de área ancha y baja potencia.
LTE: *Long Term Evolution*, estándar de la 3GPP para comunicaciones inalámbricas de transmisión de datos de alta velocidad para teléfonos móviles y terminales de datos para el acceso por radio en 4G.

MQTT: *Message Queing Telemetry Transport*, transporte de telemetría de mensajes en cola y se trata de un protocolo de comunicación entre máquinas muy utilizado en IoT.

NB-IoT: *Narrowband Internet of Things*, red de banda estrecha del Internet de Cosas, es una red de baja potencia para habilitar servicios para dispositivos móviles.

OEM: *Original Equipment Manufacturer,* fabricante de equipo original.

PC: *Personal Computer*, ordenador personal.
PEADC: Plataforma Externa de Almacenamiento de Datos *Cloud*.
POP3: *Post Office Protocol*, protocolo de oficina de correo que se utiliza en clientes locales de correo para obtener los mensajes de correo electrónico almacenados en un servidor remoto.
PPP: *Point-to-Point Protocol*, protocolo punto a punto.
PWM: *Pulse-Width Modulation,* modulación por ancho de pulsos de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

REST: *Representational State Transfer*, o transferencia de estado representacional.
Roaming: itinerancia, se refiere al servicio que permite utilizar un teléfono móvil en otros países distinto al originario del contrato de telefonía.

SCL: *Serial Clock*, pin del reloj del I2C.
SD: *Secure Digital*, formato de tarjeta de memoria *flash*.
SDA: *Serial Data*, pin de datos del bus I2C.
SDK: *Software Development Kit*, kit de desarrollo de software es generalmente un conjunto de herramientas de desarrollo de software que permite a un desarrollador de software crear una aplicación informática para un sistema concreto.
Shield: placa que se apila sobre Arduino de forma que permite ampliar sus capacidades con distintas funcionalidades. En general, estas *shields* se alimentan a través del Arduino mediante los pines de 5V y GND.
SIM: *Subscriber Identity Module*, hace referencia las tarjetas con este nombre, módulo de identificación de abonado.
SMD: *Surface-Mount-Device,* referido a las antenas que están cosntruidas en un plano, en un circuito flexible o bien en una placa.
SMS: *Short Message Service*, servicio de mensajes cortos o servicio de mensajes simples.
SQL: *Structured Query Language*, lenguaje de consulta estructurada, es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.
SSL: *Secure Sockets Layer*, capa de puertos seguros es un protocolo criptográfico, que proporciona comunicaciones seguras por una red, normalmente internet.

TCP/IP: *Transmission Control Protocol/Internet Protocol,* protocolo de control de transmisión/ protocolo de internet, designan comúnmente al conjunto de protocolos de Internet del modelo OSI.
TFG: Trabajo Fin de Grado.
TMR: Terminal Móvil de Rastreo.
TMS: Terminal Móvil de Seguimiento.

USB: *Universal Serial Bus*, Bus Universal en Serie.

10. Bibliografía

Estudios del segmento:

https://www.reuters.com/brandfeatures/venture-capital/article?id=120812
Consultado: agosto de 2019

https://www.marketwatch.com/press-release/gps-tracker-2019-market-segmentationapplicationtechnology-market-analysis-research-report-to-2023-2019-04-23
Consultado: junio de 2019

Rastreadores GPS para vehículos y, en especial motocicletas: de productos del segmento:

https://www.carbibles.com/motorcycle-gps-tracker/
https://trackimo.com/how-gps-tracker-works/
https://motor10.top/los-mejores-localizadores-gps-para-motos/
https://viajes10.top/mejores-localizadores-gps/
https://www.equipar.es/el-mejor-localizador-gps-para-moto-barato/
https://www.adslzone.net/listas/gadgets/localizador-gps-coche-moto
https://www.invoxia.com/es/gps-tracker
https://wemob.es/localizador-gps.html
https://www.localizadorsherlog.es/
https://www.atlantis-technology.com/
https://www.trackerway.com/es/supported/
Consultados: septiembre de 2019

Diseño de la aplicación con App Inventor 2:

https://es.wikipedia.org/wiki/OpenStreetMap
Consultado: septiembre de 2019
https://www.youtube.com/watch?v=I_nIeu3OcZA&t=3805s
https://www.youtube.com/watch?v=bCiguJ8bH_o&t=44s
https://www.youtube.com/watch?v=xWB04QOs5ME
https://www.youtube.com/watch?v=L6hbMv45gFY&list=PL_jqLzL9gwRFaIo11koCXzVave0fLTt_L
https://www.youtube.com/watch?v=D4ZIV8we9jY
Consultados: octubre de 2019

Plataformas *cloud*:

https://repository.ucatolica.edu.co/bitstream/10983/15029/1/TrabajoGrado%20Harold%20y%20Kilian.pdf
http://openaccess.uoc.edu/webapps/o2/bitstream/10609/42812/6/rloureiroTFC0615memoria.pdf
https://www.youtube.com/watch?v=0Bhrwc8VfGw
https://www.youtube.com/watch?v=8aVp6kSeu5Q

http://190.131.241.186/bitstream/handle/10823/1215/Documento_Trabajo_Grado.pdf?sequence=1&isAllowed=y
Consultados: septiembre de 2019.

Componentes electrónicos:

MPU6050:
https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Aceler%C3%B3metro-y-Giroscopio.html
https://www.luisllamas.es/como-usar-un-acelerometro-arduino/
https://www.youtube.com/watch?v=awjXzXnRuwo
https://www.youtube.com/watch?v=xEGLYBQCBqg
Consultados: diciembre-2019

SIM7000:
https://www.botletics.com/
Consultado: noviembre de 2019

SIM800 y SIM900:
https://www.youtube.com/watch?v=oslIQBlfFHc
https://www.youtube.com/watch?v=AOeOzeRBpkA
https://www.youtube.com/watch?v=UMcWh_RyMYo
Consultados: julio de 2019

Proyectos similares:

http://www.toptechboy.com/
https://www.cooking-hacks.com/projects/arduino-realtime-gps-gprs-vehicle-tracking
https://create.arduino.cc/projecthub/botletics/real-time-2g-3g-lte-arduino-gps-tracker-iot-dashboard-01d471
Consultados: julio de 2019

API REST:
https://ayudawp.com/wordpress-rest-api-que-es-como-funciona/
Consultado: diciembre de 2019

Programación en C:

Libro: Acera García, Miguel Angel: C/C++, Edición revisada y actualizada. Editorial Anaya, Madrid – 2010.

Libro: Schildt, Herbert: Turbo C/C++ Manual de referencia, Edción 3.1. Editorial Osborne McGraw-Hill. Madrid – 1994.

Internet:

Libro: Kurose, James y Ross, Keith: Computer Networking: A Top-Down Approach, Global Edition – 7ª Edición. Pearson, New Yprk – 2016.

11. Anexos

El programa MotoTracker_cloud_rev0_1.ino no es incluido en estos anexos ya que no ha podido ser verificado en su totalidad debido a los problemas de compatibilidad del set de comandos AT (Hayes) con el chip SimCom SIM7000E.

Si se incluye el programa MotoTracker_solo_SMS_rev0_2.ino que ha funcionado correctamente. Se ha representado con una fuente muy pequeña para que pueda ser recortado y movido a Arduino, más que para ser leído en este documento.

Para el funcionamiento correcto de este programa se ha de utilizar Arduino Mega2560, ya que la memoria necesaria genera problemas severos de funcionamiento en Arduino UNO.

```
/*
 *  Programa: MotoTracker_solo_SMS
 *  Revisión: 0.2
 *
 *
 *  Autor: Emilio Angulo
 *  Fecha: 1 de Diciembre 2019
 *
 *  Notas:
 *
 *  Funcionalidad:
 *
 *      -Integra la primera versión de la comunicación a traves de SMS,
 *       a partir de las señales de aceleración recibidas desde el IMU, pero envía los SMS geolocalizados.
 *     - No incluye nada de Firebase, debido a los problemas de comunicación con la SIM7000
 *
 *  Esquema de conexionado del IMU:
 *     GND - GND
 *     VCC - VCC
 *    Para Arduino UNO:
 *     SDA (I2C)- Pin A4
 *     SCL (I2C)- Pin A5
 *    Para Arduino Mega2560:
 *     SDA (I2C)- Pin A20
 *     SCL (I2C)- Pin A21
 *
 */

// Librerias SIM7000
#include "Adafruit_FONA.h"

//Librerias del MPU6050
#include "I2Cdev.h" // Librería del I2C
#include "MPU6050.h" //Libreria para el control del IMU
#include "Wire.h"

//Librería generales
#include <SoftwareSerial.h>
//#include <string.h">

// Definiciones
#define SIMCOM_7000
//Pines utilizados
#define FONA_PWRKEY 6
#define FONA_RST 7
#define FONA_TX 10
#define FONA_RX 11

//Incialización de variables SIM7000
char replybuffer[255]; //Buffer de respuestas
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX); //Conexión serie para hablar con SIM7000E
//SoftwareSerial *fonaSerial = &fonaSS; (No sé para que sirve, tampoco lo encuentro más abajo
Adafruit_FONA_LTE fona = Adafruit_FONA_LTE(); //Creación del objeto principal para la comunicación.
uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout = 0);
uint8_t type;
char imei[16] = {0}; // MUST use a 16 character buffer for IMEI!
char telf_usuario[21]="+34647530246";

//Inicialización variables y factores de conversión, relacionados con el IMU MPU6050
const int mpuAddress = 0x68;  // Dirección del MPU para I2C (0x68)
int ax, ay, az; //Aceleraciones leídas
int gx, gy, gz; // Velocidades angulares medidas
float ang_x;//Angulo de inclinación
float gtotal; //Velocidad angular total
const float accScale = 2.0 * 9.81 / 32768.0; //Conversión aceleraciones al SI
const float gyroScale = 250.0 / 32768.0;     //Conversión velocidades angulares a º/s.
const float ang_limite = 45; //Limite ángular a partir del cual se envía SMS al usuario
const float vel_ang_limite= 3; //Limite velocidad angular a partir del cual se envía SMS al usuario
const float cal_acel_x= (10.5-9.81)/accScale; //Valores de calibración para corregir el offset de los sensores.
const float cal_acel_y= -0.11/accScale;
const float cal_acel_z= -0.53/accScale;
const float cal_vel_ang_x = -3.50/gyroScale;
const float cal_vel_ang_y = 0.25/gyroScale;
const float cal_vel_ang_z = -0.8/gyroScale;
//Tiempos de ciclo.
unsigned int periodo =(30-5)*1000;
unsigned int tiempo1=0;
unsigned int tiempo2;
bool unavez = false;

MPU6050 mpu(mpuAddress); //Creación del objeto de control MPU6050
```

```
// Bucle SETUP
void setup() {

// Inicializa la comunicación serie del monitor
  Serial.begin(9600);//Puerto serie general de comunicación
  Serial.println("EJECUTANDO SETUP");
//Setup del SIM7000E
          // Las siguientes instrucciones resetean y arrancan SIM7000E
            pinMode(FONA_RST, OUTPUT); //El pin se inicializa de salida
            digitalWrite(FONA_RST, HIGH); //El pin se pone en estado HIGH
            pinMode(FONA_PWRKEY, OUTPUT); //El pin se inicializa de salida
          // Arranca el módulo pulsando PWRKEY a LOW por un tiempo.
          // El tiempo depende del modulo específico
            digitalWrite(FONA_PWRKEY, LOW);
            delay(100); // Para SIM7000
            digitalWrite(FONA_PWRKEY, HIGH);
          //Nota: SIM7000 baud rate se resetea despues del ciclo de arranque, y tarda unos 3 s en arrancar
            fonaSS.begin(115200); // Baud rate de SIM7000 por defecto
              Serial.println(F("Configurando a 9600 baud"));
            fonaSS.println("AT+IPR=9600"); // Set baud rate
              delay(100); // Pequeña pausa para que el comando actue
            fonaSS.begin(9600);
            if (! fona.begin(fonaSS)) {
              Serial.println(F("No puedo encontrar el objeto FONA"));
              while(1); //No arranca hasta que no encuentra el objeto
            }
            type = fona.type();
            Serial.println(F("FONA está OK"));
            Serial.print(F("Encontrado "));
            switch (type) {
              case SIM7000A:
                Serial.println(F("SIM7000A (Americano)")); break;
              case SIM7000C:
                Serial.println(F("SIM7000C (Chino)")); break;
              case SIM7000E:
                Serial.println(F("SIM7000E (Europeo)")); break;
              case SIM7000G:
                Serial.println(F("SIM7000G (Global)")); break;
              default:
                Serial.println(F("???")); break;
            }
            // Imprime el número del módulo IMEI.
            uint8_t imeiLen = fona.getIMEI(imei);
            if (imeiLen > 0) {
              Serial.print("Módulo IMEI: "); Serial.println(imei);
            }
            // Arranca "modem" a funcionalidad total.
            fona.setFunctionality(1); // AT+CFUN=1
            // Configuración de GPRS APN, username y password.
            fona.setNetworkSettings(F("data.lycamobile.es"), F("lmes"), F("plus"));
            fona.setPreferredLTEMode(1); // Uso de LTE CAT-M, no NB-IoT. Esta instrucción es clave. Sin esto no funciona.

            // Arranque de GPS
              Serial.println("Arrancando GPS");
              if (!fona.enableGPS(true))
              Serial.println(F("Fallo de arranque de GPS"));

 //Chequeos de los parámetros de la SIM7000E
      // Lectura del ADC
              uint16_t adc;
              if (! fona.getADCVoltage(&adc)) {
                Serial.println(F("Failed to read ADC"));
              } else {
                Serial.print(F("ADC = ")); Serial.print(adc); Serial.println(F(" mV"));
              }

       // Lectura del voltaje y porcentaje de la batería
              uint16_t vbat;
              if (! fona.getBattVoltage(&vbat)) {
                Serial.println(F("Failed to read Batt"));
              } else {
                Serial.print(F("VBat = ")); Serial.print(vbat); Serial.println(F(" mV"));
              }

              if ( (type != SIM7500A) && (type != SIM7500E) ) {
                if (! fona.getBattPercent(&vbat)) {
                  Serial.println(F("Failed to read Batt"));
                } else {
                  Serial.print(F("VPct = ")); Serial.print(vbat); Serial.println(F("%"));
                }
              }

      // Lectura del CCID
              fona.getSIMCCID(replybuffer);  // make sure replybuffer is at least 21 bytes!
              Serial.print(F("SIM CCID = ")); Serial.println(replybuffer);

      //Lectura del estado de cobertura
      delay(5000);

            // Lectura del RSSI
                    uint8_t n = fona.getRSSI();
                    int8_t r;
                    Serial.print(F("RSSI = ")); Serial.print(n); Serial.print(": ");
                    if (n == 0) r = -115;
                    if (n == 1) r = -111;
                    if (n == 31) r = -52;
                    if ((n >= 2) && (n <= 30)) {
                      r = map(n, 2, 30, -110, -54);
                    }
                    Serial.print(r); Serial.println(F(" dBm"));

            //Lectura del estatus de la red
                    uint8_t nn = fona.getNetworkStatus();
                    Serial.print(F("Estatus de la red "));
                    Serial.print(nn);
                    Serial.print(F(": "));
                    if (nn == 0) Serial.println(F("No registrado"));
                    if (nn == 1) Serial.println(F("Registrado (home)"));
                    if (nn == 2) Serial.println(F("No registrado (buscando)"));
                    if (nn == 3) Serial.println(F("Denegado"));
                    if (nn == 4) Serial.println(F("Desconocido"));
                    if (nn == 5) Serial.println(F("Registrado con roaming"));

             // Envio de un SMS de arranque
                    char sendto[21], message[141];
                    //flushSerial();
                    strcpy(sendto, telf_usuario);
                    envioSMSGeo (telf_usuario, "MotoTracker\n acaba de ser arrancado",false);

            //Lectura del tiempo
```

```
                char buffer[23];

                fona.getTime(buffer, 23);  // make sure replybuffer is at least 23 bytes!
                Serial.print(F("Tiempo = ")); Serial.println(buffer);

            //Tipo de conexión , banda celular, nombre del carrier, etc.
                fona.getNetworkInfo();

            // flush input
            //  flushSerial();
              //while (fona.available()) {
               // Serial.write(fona.read());
              //}

//Setup de la IMU MPU6050
            Wire.begin();
            mpu.initialize();
            Serial.println(mpu.testConnection() ? F("IMU iniciado correctamente") : F("Error al iniciar IMU"));

          Serial.println("SETUP FINALIZADO");
// FINAL DEL SETUP.
}



void loop() {
//PRINCIPIO BUCLE LOOP
//Arranca el contador de tiempo.
tiempo2=millis();


// Leer las aceleraciones y velocidades angulares

   Serial.println("EJECUTANDO LOOP");

   if (tiempo2>tiempo1+periodo or unavez==false){
   mpu.getAcceleration(&ax, &ay, &az);
   mpu.getRotation(&gx, &gy, &gz);

   //Calcular los angulos de inclinacion y la velocidad angular total.
   ang_x = atan((ax-cal_acel_x) / sqrt(pow((ay-cal_acel_y), 2) + pow((az-cal_acel_z), 2)))*(180.0 / 3.14);
   gtotal=sqrt(pow((gx-cal_vel_ang_x),2)+pow((gy-cal_vel_ang_y),2)+pow((gz-cal_vel_ang_z),2))*gyroScale;


  // Alertas SMS.
   if (ang_x<=ang_limite)
   {
    if (gtotal>=vel_ang_limite){
    Serial.println("Envio SMS inclinación y vibraciones");
    Serial.print(ang_x);Serial.println(F("º"));
    Serial.print(gtotal);Serial.println(F("º/s"));

      // Envio de un SMS
      envioSMSGeo (telf_usuario, "MotoTracker ha detectado un vuelco y vibraciones. ",true);
        tiempo1=millis();
        unavez=true;
    }else{
    Serial.println("Envio SMS inclinación");
    Serial.print(ang_x);Serial.println(F("º"));

      // Envio de un SMS
      envioSMSGeo (telf_usuario, "MotoTracker ha detectado un vuelco.  ",true);
        tiempo1=millis();
        unavez=true;
   }}else{
   if (gtotal>=vel_ang_limite){
    Serial.println("Envio SMS vibraciones");
     Serial.print(gtotal);Serial.println(F("º/s"));

      // Envio de un SMS
       envioSMSGeo (telf_usuario, "MotoTracker ha detectado vibraciones. ",true);
       tiempo1=millis();
       unavez=true;
          }
   }
   delay(1000);
 }


//FINAL DEL LOOP
//delay(1000);
}

// Funcion de envío SMS

void envioSMSGeo (char sendto[21], char mensaje[50], bool geo)
{
    Serial.println("ESTOY EN envioSMSGeo");
    flushSerial();
    char envio[150];
    strcpy(envio,"");

    if (geo){ // Si el mensaje está geolocalizado se ha de llamar a GPS y completar la estructura del enlace.
        //Declaración variables necesarias.
       float latitud;
       float longitud;
       float vel_kph, cabecera, altitud;
       char url[150];
       char aux[20];
       strcpy(url,"");

       char latstring [12];
       char longstring [12];

      if (fona.getGPS(&latitud, &longitud, &vel_kph, &cabecera, &altitud)) { //Llamada a la función que lee las coordenadas
         Serial.println(F("---------------------"));
         Serial.print(F("Latitud: ")); Serial.println(latitud,10);
         Serial.print(F("Longitud: ")); Serial.println(longitud,10);
         //Serial.print(F("Velocidad: ")); Serial.println(vel_kph);
         //Serial.print(F("Cabecera: ")); Serial.println(cabecera);
         //Serial.print(F("Altitud: ")); Serial.println(altitud);

         delay (2000);
       //Generación de la URL

            strcpy(url,"https://www.openstreetmap.org/?mlat=");
            Serial.print("Paso 1: ");Serial.println(url);
            dtostrf(latitud,3 ,10, latstring);
            strcat(url, latstring);
            Serial.print("Paso 2: ");Serial.println(url);
            strcpy(aux, "&mlon=");
```

```
            strcat(url,aux);
            Serial.print("Paso 3: ");Serial.println(url);
            dtostrf(longitud,3 ,10, longstring);
            strcat(url, longstring);
            Serial.print("Paso 4: ");Serial.println(url);
            strcpy(aux,"#map=17/");
            strcat(url,aux);
            Serial.print("Paso 5: ");Serial.println(url);
            dtostrf(latitud,3 ,10, latstring);
            strcat(url,latstring);
            Serial.print("Paso 6: ");Serial.println(url);
            strcpy(aux,"/");
            strcat(url,aux);
            Serial.print("Paso 7: ");Serial.println(url);
            dtostrf(longitud,3 ,10, longstring);
            strcat(url,longstring);
            Serial.print("Paso 8: ");Serial.println(url);
            strcpy(aux,"&layers=N");
            strcat(url,aux);
            Serial.print("Paso 9: ");Serial.println(url);

        } else{
        strcpy (url, "Sin cobertura de GPS\n");
        }
        strcpy(envio, mensaje);
        strcat(envio,url);

        } else{
        strcpy(envio, mensaje);
         }
        //Envio del SMS
         if (!fona.sendSMS(sendto, envio)) {
                    Serial.println(F("Fallo enviando SMS"));
                  } else {
                    Serial.println(F("SMS enviado"));
                  }
    //Fin de la función.
    }


void flushSerial() {
  while (Serial.available())
    Serial.read();
}

char readBlocking() {
  while (!Serial.available());
  return Serial.read();
}
uint16_t readnumber() {
  uint16_t x = 0;
  char c;
  while (! isdigit(c = readBlocking())) {
    //Serial.print(c);
  }
  Serial.print(c);
  x = c - '0';
  while (isdigit(c = readBlocking())) {
    Serial.print(c);
    x *= 10;
    x += c - '0';
  }
  return x;
}

uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout) {
  uint16_t buffidx = 0;
  boolean timeoutvalid = true;
  if (timeout == 0) timeoutvalid = false;

  while (true) {
    if (buffidx > maxbuff) {
      //Serial.println(F("SPACE"));
      break;
    }

    while (Serial.available()) {
      char c =  Serial.read();

      //Serial.print(c, HEX); Serial.print("#"); Serial.println(c);

      if (c == '\r') continue;
      if (c == 0xA) {
        if (buffidx == 0)   // the first 0x0A is ignored
          continue;

        timeout = 0;         // the second 0x0A is the end of the line
        timeoutvalid = true;
        break;
      }
      buff[buffidx] = c;
      buffidx++;
    }

    if (timeoutvalid && timeout == 0) {
      //Serial.println(F("TIMEOUT"));
      break;
    }
    delay(1);
  }
  buff[buffidx] = 0;  // null term
  return buffidx;
}
```