Resumen

Desde que se ha conseguido reducir el tamaño de los computadores, junto con el precio que cuesta producirlos, se ha incrementado un aumento de la presencia de sistemas informáticos en nuestro día a día. Así ha surgido la computación ubicua, la cual ha demostrado ser ya una realidad y busca hacernos la vida más sencilla. Uno de los mayores ejemplo de la madurez de esta tecnología es la ciudad de New Songdo (Corea del Sur), una metrópoli la cual tiene desplegada sistemas de información en los aparcamientos, cines, viviendas... los cuales interactúan con las personas a través de tarjetas.

Uno de los áreas aún en investigación de los sistemas ubicuos es el desarrollo y despliegue de sistemas inteligentes en la carretera (ITS). Éstos persiguen el incrementar la seguridad de todos los agentes de la carretera y ofrecer una mayor rapidez a la hora de responder a eventos de emergencia. Este tipo de tecnologías han demostrado ser lo suficientemente robustas para utilizarse como apoyo para lidiar con la gestión de tráfico, mejora de la seguridad vial, aportar información al conductor... Actualmente están apareciendo vehículos que además de tener un propio sistema de auto-diagnostico, son capaces de interactuar con su entorno.

Descriptores

redes vehiculares, agentes vulnerables, ciudades inteligentes.

Índice general

Índice de figuras

Capítulo 4

Capítulo 6

Capítulo A

Índice de tablas

Capítulo 4

Capítulo 7

Índice de algoritmos

Capítulo 4

Capítulo A

1. INTRODUCCIÓN

Según la Directiva General de la Comisión Europea para el Transporte y la Movilidad, en 2014, algo más de 25.700 accidentes de tráfico fueron informados en la Unión Europea. Aunque el número de accidentes está decrecimiento sustancialmente, el informe sobre Transporte ha anunciado un objetivo estratégico para la seguridad en las carreteras europeas para el período de 2011 a 2020: reducir el número de muertes en carretera a la mitad. Si las estadísticas son analizadas, en el período de 2010 a 2012, el número de ciclistas muertos en siniestros ha aumentado un 6 %; siendo el único agente de la carretera cuyos resultados vayan a peor. Esto se explica, al menos parcialmente, por un aumento de la presencia ciclista en la carretera. Se podría decir que el ciclismo es un medio de transporte donde los Usuarios Vulnerables de la Carretera (VRU) tienen un mayor contacto con el tráfico de mayor afluencia y velocidad. Cuando están involucrados en un accidente, son los que sufren las consecuencias más graves derivadas de una colisión con otro agente de la carretera; ya que están completamente expuestos a otros vehículos.

Basados en estos estudios, los accidentes en la que están involucrados VRUs ocurren frecuentemente en vías diseñadas para viandantes y ciclistas; por ejemplo en pasos de peatones y caminos para ciclistas cercanos a infraestructuras comunes de tráfico, las carreteras. Por lo tanto, la pregunta es: ¿Cómo se pueden reducir los accidentes de VRUs, y cómo minimizar la gravedad de un siniestro y sus consecuencias? Se pueden tomar varias soluciones: mejorar el diseño y trazado de las vías de comunicación, mejorar la iluminación, instalar más infraestructuras de protección, promocionar equipamiento de seguridad y enseñar cómo utilizarlo...

Sin embargo, hay otras soluciones viables aparte del re-diseño de las infraestructuras existentes, o soluciones pasivas como el uso del casco de seguridad. Una opción que está ganando fuerza en los países desarrollados es el desarrollo de soluciones para la movilidad en el entorno de ciudades inteligentes. Aunque el término de ciudad inteligente pueda parecer confuso, se podría decir que se considera *inteligente* cuando se ha aplicado tecnologías de la información y comunicación para mejorar la calidad de vida en áreas como la seguridad, gasto energético, reducción de costes, y gobierno y transporte, permitiendo una participación efectiva y activa por parte de los ciudadanos.

En el dominio de las Ciudades Inteligentes, las soluciones para transporte diseñadas tratan de hacer un uso más seguro, sostenible y eficiente de la carretera a través de un mejor entendimiento del estado de tráfico, la posición de los los vehículos y usuarios, y el registro de eventos que suceden durante el transporte. Estas soluciones combinan la capacidad y beneficios de los sensores, dispositivos, infraestructura física y arquitecturas de comunicación combinada con sistemas de información en la nube, y la capacidad de analizar grandes volúmenes de datos.

En este contexto, los Sistemas Inteligentes de Información (ITS) emergen como una respuesta tecnológica para una mejor monitorización y caracterización del tráfico. Estos sistemas permiten al mismo tiempo mejorar el uso y eficiencia de la carretera, así como la seguridad de los usuarios, particularmente aquellos definidos como vulnerables; ciclistas, peatones o motoristas. Los ITS

actuales requieren el uso de cámaras de tráfico, paneles informativos, o sensores de inducción que obtengan datos para ser posteriormente mandados y procesados en la central de gestión de tráfico. A diferencia de estas soluciones que requieren el uso de sensores, actualmente surgen sistemas conocidos como Información Vehicular Flotante (FCD), que se encargan de reunir información de los sistemas de posicionamiento global (GPS) obtenidos de terminales móviles y el uso de páginas web colaborativas como Waze, que permite a los conductores obtener y proveer información sobre la carretera sin necesidad de ningún sensor en la carretera. Este tipo de soluciones basadas en FCD tienen a favor la monitorización del estado de los usuarios de manera ubicua, pero su fiabilidad depende del número de vehículos y usuarios informando sobre los eventos y aportando datos.

En el dominio de los ITS, los ITS Cooperativos (C-ITS) son sistemas que permiten la conexión directa entre vehículos (comunicaciones V2V) ó entre vehículos e infraestructuras (comunicaciones V2I) para intercambiar de información con el objetivo de mejorar la seguridad vial y la gestión del tráfico. Estos enlaces son posibles gracias a las unidades de abordo (OBU), dispositivos C-ITS dedicados que habilitan interfaces de comunicación, y dispositivos localizados en infraestructuras llamados RSU.

En un escenario C-ITS, hay generalmente cuatro agentes a considerar: dos entidades móviles (OBUs y peatones), y dos entidades estacionarias (la RSU y el sistema central). Estas entidades son capaces de ejecutar cuatro tipos diferentes de aplicaciones: seguridad activa en la carretera, tráfico eficaz cooperativo, servicios locales cooperativos, y servicios globales en Internet. Sobre cada tipo, hay diferentes definiciones de casos de uso y aplicaciones, donde cada agente puede ser considerado un sensor que genera información. Dependiendo de la aplicación y las restricciones de tiempo, el intercambio de información entre las entidades se puede clasificar como:

Mensajes de alerta: se definen como notificaciones descentralizadas y pueden ser enviados desde cada vehículo o RSU.

Mensajes de latido o *"beacons"*: son usados por los OBU para notificar su posición, velocidad e identidad a las RSU que forman parte del FCD. Además, estos mensajes también son usados para conocer la situación actual del tráfico. Por ello, el Acceso Inalámbrico en Entornos Vehiculares (WAVE) define los Mensajes de Aviso Cooperativo (CAMs), que son transmitidos periódicamente a todos los vehículos en área de alcance.

Mensajes sobre infotainment: notificaciones no relacionadas con la seguridad, sino que son usados para aportar mayor información y confort al conductor; datos turísticos, acceso a Internet, asistencia en navegación, etc.

En el campo de los servicios C-ITS, una gran variedad de aplicaciones y casos de uso se centran en incrementar la seguridad del usuario. Teniendo en cuenta requisitos estratégicos, económicos y de organización, características del sistema así como requisitos legales y de estandarización, el Comité del Instituto Técnico Europeo de Estándares en la Telecomunicación ha definido un conjunto básico de aplicaciones para usar como referencia en ITS para desarrolladores. Entre ellos, los Avisos a Usuarios Vulnerables en Carretera tratan de proveer notificaciones a los vehículos sobre la presencia de usuarios vulnerables, por ejemplo ciclistas, y en caso de existir situaciones de peligro también se avisa a los VRU sobre la presencia de un vehículo cercano.

Siguiendo los requisitos presentados por el ETSI, este proyecto presenta un sistema que emplea a los vehículos y ciclistas como sensores móviles que aportan información sobre su posición, velocidad y rumbo con el objetivo de detectar la proximidad entre estas dos entidades y avisarles en el caso de detectar peligro. Esta solución tiene un sistema centralizado que despliega comunicación inalámbrica vehicular, conectividad móvil y computación en la nube, y gestiona la información obtenida por los usuarios (vehículos y ciclistas). El sistema ha sido desplegado y verificado en un dominio real, y se han realizado pruebas de rendimiento en diferentes escenarios para comprobar el correcto funcionamiento de las comunicaciones en los peores escenarios.

Actualmente, existe un proyecto llamado Detección de agentes Inteligentes Cooperativos para la mejora de la eficiencia en el tráfico (ICSI), con similares aplicaciones que se encuentra bajo desarrollo usando una proximación descentralizada con la finalidad de mejorar el rendimiento y la seguridad.

1.1. ANTECEDENTES

La importancia de las tecnologías C-ITS para la administración pública y la Comisión Europea está reflejada en la directiva 2010/40/EU, donde la UE reconoce la capacidad de los C-ITS para mejorar los sistemas de gestión de tráfico actuales y de dirigir los procesos de implantación y despliegue de estos sistemas en las carreteras europeas. Tras decenas de investigaciones y proyectos de desarrollo tales como Sistemas Cooperativos Vehículo-Infraestructura (CVIS), Sistemas Cooperativos para la seguridad en carretera *vehículos inteligentes en ciudades inteligentes* (SAFESPOT), El transporte flexible y adaptable del mañana (TEAM), el despliegue masivo de sistemas C-ITS se está acercando. Un ejemplo es el Memorandum del Entendimiento (MOU) firmado por la industria automovilística y organizaciones constructoras con el objetivo de comenzar a desplegar soluciones basadas en C-ITS en 2015. Las administraciones públicas también están trabajando en la misma dirección, resaltando el tratado que han pactado Alemania, Austria y Holanda para desplegar sistemas C-ITS en las vías que comunican estos tres países.

Se puede destacar de la misma manera el anuncio realizado en Febrero del 2014 por la Administración Nacional de Seguridad del Tráfico en Autopistas (NHTSA), que pertenece al Departamento de Transporte de los Estados Unidos (USDOT), sus intenciones para dar los pasos necesarios para el despliegue de sistemas V2V cooperativos en los próximos años, exactamente a partir de 2017, para vehículos comerciales.

2. DEFINICIÓN DE OBJETIVOS

El principal objetivo de este trabajo es el desarrollo de aplicaciones para hacer posible la comunicación entre vehículos y ciclistas, a través de redes vehiculares y móviles. Se desea que estos dos agentes de la carretera posean información para poder conocer la posición de los agentes a su alrededor y así poder evitar accidentes. Las aplicaciones desarrolladas serán desplegadas por un lado en los sistemas informáticos de los vehículos y la infraestructura en la carretera, en los móviles inteligentes de los ciclistas y en la nube, donde un servidor central permitirá la comunicación entre diferentes plataformas. Se busca la creación de una plataforma lo más abierta posible a diferentes tecnologías, así como el uso de software libre para su desarrollo.

Los resultados que permitan verificar el sistema se harán a través de las pruebas unitarias que se han creado para probar el sistema, las pruebas que se han realizado en la calle y a las conclusiones que se han llegado tras analizar las mediciones obtenidas. Se desea verificar la calidad de las comunicaciones entre los diferentes agentes de la carretera, así como la precisión de los algoritmos desarrollados para prever los accidentes y obtener información a través de los datos recogidos por los mensajes vehiculares. Para saber más sobre cómo se han verificado los resultados del proyecto ir al capítulo 5.

Las aplicaciones que serán desarrolladas deben poseer una serie de características:

- Universalidad: el sistema a desarrollar debe ser flexible a diferentes tecnologías. Actualmente existen diversos fabricantes que aún no cumplen los estándares que se están estableciendo, por lo que se debe permitir que la adaptación de los diferentes sistemas sea lo más sencillo posible.
- Interfaz gráfica: la interacción del usuario con las aplicaciones debe ser sencilla, que no requiera de ningún conocimiento técnico para realizar sus funciones.
- Localización de agentes en carretera: los vehículos y ciclistas intercambian información sobre sus posiciones a través de mensajes CAM.
- Predicción de accidentes: los usuarios deben poder percibir las situaciones de peligro. En el caso de los vehículos a motor se reproduce una alarma sonora, y en el de los ciclistas se enciende un LED en el casco que llevan puestos, además de mostrarse la alerta en el Smart-phone; en caso de tenerlo a la vista.

3. METODOLOGÍA

4. DESARROLLO

Tras obtener una descripción completa del proyecto, se han realizado varias reuniones en las cuales se han obtenido las especificaciones que se desea en el sistema a desarrollar. Durante la fase inicial de obtención de requisitos, se han recogido y priorizado de la siguiente forma los requisitos funcionales:

1. Empleo de comunicaciones IEEE 802.11p y 3G entre dispositivos móviles y módulos de NEC.
2. Debe ser una comunicación bidireccional.
3. Baja latencia en el envío de mensajes; menor a 100 ms.
4. Incremento de la seguridad para los VRU.
5. Comunicación de ciclistas individuales y en grupo.
6. Se debe conocer la posición de los ciclistas en los vehículos a motor, y recibir avisos.

Se han clasificado las siguientes especificaciones como requisitos no funcionales, ya que no son vitales para el despliegue principal de la plataforma, pero que pueden ser integrados tras el despliegue inicial:

- Aplicación con monitor de actividad para ciclistas.
- Posibilidad de crear logs durante las pruebas.
- Simpleza al crear un grupo de ciclistas, que no requiera conocimientos técnicos.
- Posibilidad de mandar mensajes desde el servidor central.

En el diagrama de casos de uso mostrado en 4.1 se pueden observar las tareas que se han pedido integrar en el proyecto. Los ciclistas envían sus posiciones al iniciar la salida, pueden crear un grupo o no, monitorizar su actividad y emparejar sus dispositivos con el casco BLE. Los conductores al iniciar su aplicación son notificados de los eventos de alerta que se pueden dar en la carretera, mientras en segundo plano notifican su posición. La plataforma que es implementada en la nube recibe las notificaciones y cuando detecte alguna posibilidad de peligro, notifica a los ciclistas y conductores. Además, desde la aplicación de escritorio en la nube se pueden enviar mensajes de gestión de carretera a la unidad Unidad de carretera (RSU).

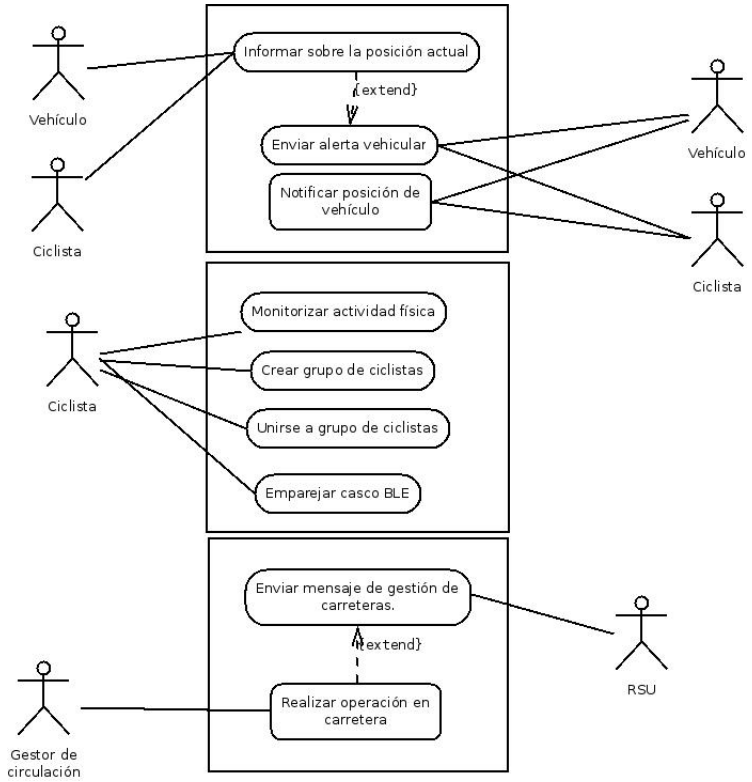

Figura 4.1: Casos de uso

Una vez analizados los requisitos de la plataforma, se procede a diseñar la solución del sistema. Una primera decisión que se debe tomar es si el sistema debe ser distribuido o centralizado. El primero tiene la ventaja de proveer un mejor rendimiento, ya que tan solo debe encargarse de un área, pero puede perder información ya que no tiene un mapa completo del área total. Por el contrario, un sistema centralizado requiere de una mayor infraestructura cuanto más área provea de cobertura, pero tiene en todo momento una visión completa de los escenarios. Al ser una solución experimental que va a ser desplegada en un pequeño escenario y los requisitos de seguridad son prioritarios, se ha elegido un sistema centralizado.

Se deberá crear diferentes aplicaciones: un servidor central que permita la comunicación entre diferentes tecnologías, una app móvil para ciclistas y conductores, y aplicaciones para ser desplegadas en unidades Unidad de a bordo (OBU) y RSU.

4.1. ARQUITECTURA DEL SISTEMA

A grandes rasgos, existe en el medio del sistema una aplicación en la nube denominada *Nube de Conductores* que se encarga de hacer llegar los mensajes procedentes de los ciclistas a los vehículos a motor, y los mensajes enviados por los vehículos a motor a los ciclistas. Además, filtra los mensajes que han sido mal formados, monitoriza las posiciones de todos los vehículos en la carretera y es capáz de predecir cuándo se puede dar la posibilidad de que haya un choque entre dos vehículos; en cuyo caso avisa a los conductores de esta posibilidad.

Los vehículos a motor mandan continuamente beacons a través de un OBU anunciando su posición, velocidad y la dirección a la que se dirigen. No se comunican directamente con la *Nube de Conductores*, sino que los mensajes enviados son escuchados por una unidad desplegada en carretera llamada RSU. Esta última recoge los mensajes que escucha y los reenvía a la nube por conectividad 3G.

Por otro lado, los ciclistas envían información a la *Nube de Conductores* a través de 3G o 4G; dependiendo de la disponibilidad. Éstos también pueden agruparse empleando *Wi-Fi 802.11*, mediante la creación de un *HUB* de dispositivos móviles en el cual se envían notificaciones sobre los eventos que aparezcan.

En la figura 4.2 se puede observar de qué elementos está compuesto el sistema y cómo se comunican entre ellos. Como puede apreciarse, hay diferentes tecnologías de comunicación y desarrollo en cada una de las plataformas, por lo que uno de los requisitos es que la solución desarrollada sea flexible a los cambios de tecnología.

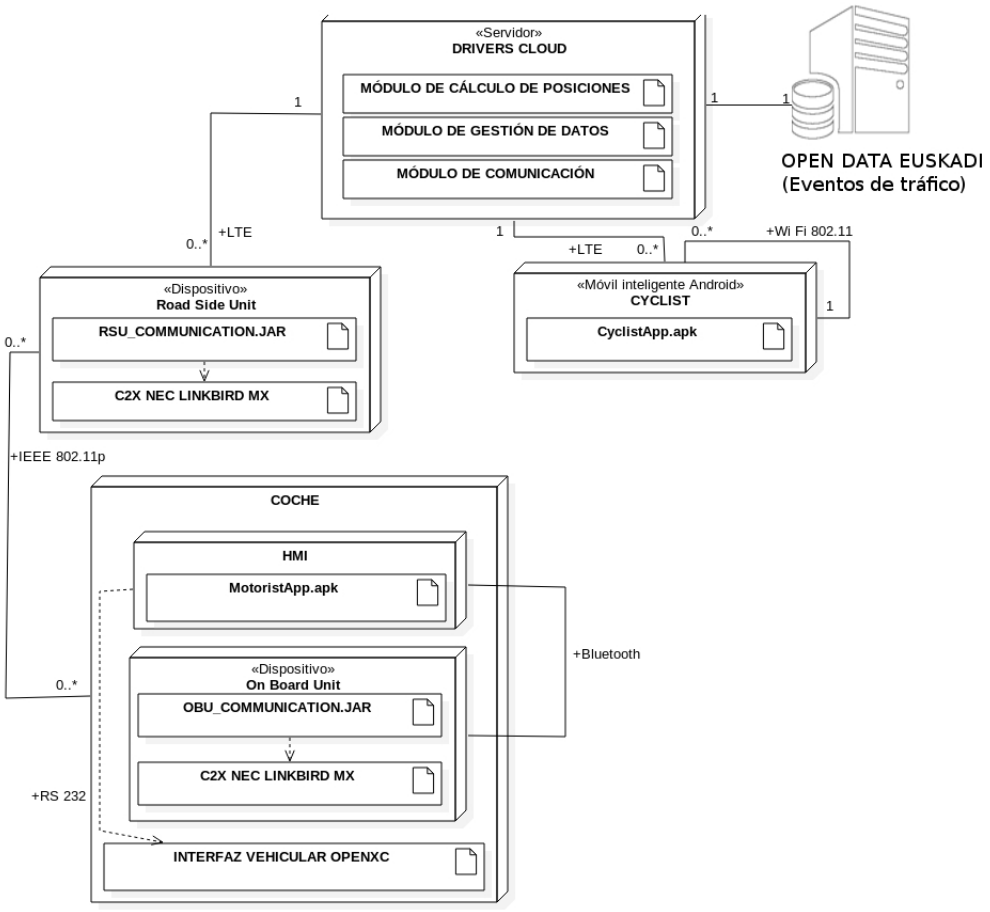

Figura 4.2: Arquitectura del sistema

4.1.1. Comunicación entre plataformas

La conexión entre la parte de los vehículos a motor y de los ciclistas hacia la nube se establece a través de tecnología móvil Long Term Evolution (LTE) o Third Generation (3G), dependiendo de

la disponibilidad, aunque la manera de comunicarse con la *Nube de Conductores* es diferente. La Nube de Conductores actúa como intermediario entre las aplicaciones desarrolladas en el lado de los motoristas y el de los ciclistas.

Mensajes a ciclistas

Gracias al actual predominio de Smart-Phones en la vida de todos los habitantes, el despliegue de aplicaciones móviles vehiculares es bastante sencillo. Se han convertido en dispositivos potentes y versátiles, los cuales permiten utilizarlos para una gran variedad de utilidades. Gracias que tienen GPS integrado podemos obtener una posición bastante aproximada de los usuarios, dependiendo de la calidad del dispositivo se obtendrá una localización más precisa. También poseen conexión móvil con una gran variedad de conexiones como Bluetooth, USB, Wi-Fi, LTE, GSM, UMTS y NFC.

Actualmente el predomino del mercado se encuentra en el Sistema Operativo móvil Android. Este sistema, actualmente en desarrollo por Alphabet, esta orientado principalmente a dispositivos móviles y embebidos. Esta basado en Linux, y es un proyecto que tiene devoción por los estándares abiertos existentes; prueba de ello es la pertenencia a la alianza comercial Open Handset Alliance, la cual se dedica a desarrollar estándares abiertos para su uso en dispositivos móviles.

Android posee un completo entorno de desarrollo, el cual incluye un depurador de código, biblioteca, un simulador de teléfono, documentación, ejemplos de código y tutoriales. Para el desarrollo en esta plataforma se puede optar por dos opciones, la instalación del IDE de desarrollo Android Studio, ó descargar el SDK de Android e integrarlo con el IDE que el programador desee. Los lenguajes de programación con los que es posible desarrollar son Java y C/C++, aunque este segundo solo se recomienda su uso para el desarrollo de librerías que requieran de un gran rendimiento. La aplicación resultante es un paquete apk que es ejecutado en un Sandbox dentro de Android.

Se ha desarrollado una aplicación *Android* desde la cual se manda a la nube actualizaciones sobre la posición del usuario o el grupo que el usuario haya creado. Éste recibe notificaciones sobre las posiciones de los vehículos próximos, y otros diferentes eventos que pueden darse en la carretera; por ejemplo, un accidente de tráfico. Se ha contemplado la posibilidad de salidas en grupo de ciclistas, para ello se ha habilitado una modalidad específica mediante la cual se crea un grupo que se comunica entre sus miembros a través de una red privada *Wi-Fi 802.11*. Los diferentes miembros se mantienen actualizados sobre los diferentes eventos a través de un nodo denominado líder, el cual es el enlace a la nube tanto para reportar la posición del grupo de ciclistas como para recibir mensajes de la nube y retransmitir éstos al resto de miembros.

Para comunicarse con los dispositivos Android se requiere un sistema de comunicación por el cual aunque los ciclistas no tengan en un momento determinado cobertura, los mensajes no se pierdan. Se ha elegido la plataforma Google Cloud Messaging (GCM), la cual se encarga de gestionar que los mensajes lleguen al destino aunque éste se encuentre temporalmente inaccesible mediante tecnología *Push* [A.1].

El mensaje debe respetar el formato que la API de GCM indica y puede observarse en el algoritmo 4.1, donde *ID_ANDROID* es el identificador del dispositivo Android al que se le va a enviar el mensaje, y *DATOS* un objeto *JSON* con la información se que desea enviar. Para crear un iden-

tificador único, se puede emplear el que crea Android cuando se introduce la cuenta de correo personal en el móvil. El identificador de Android está formada por una cadena hexadecimal de 64 bit, la cual es poco probable que se repita. En el caso de que se quisiese reducir aún más la probabilidad de repetición, se puede mezclar el identificador de Android con el que la compañía de telefonía emplea para identificar nuestro dispositivo; aunque esto último aumenta el tamaño de los mensajes.

```
1 { "registration_ids": [ "ID_ANDROID" ], data: { /*DATOS*/ }}
```

Algoritmo 4.1: Envío de mensajes mediante GCM

Mensajes a vehículos a motor

Poseen en el vehículo un dispositivo *OBU* que permite comunicarse con la infraestructura en la carretera a través de una red *IEEE 802.11p*. A traves de las *RSU* dispuestas en la carretera, las cuales actúan de intermediario, se envían y reciben los mensajes de la nube. Por tanto puede decirse, que las *RSU* actúan de *gateway* de comunicaciones entre los vehículos y las aplicaciones desplegadas en la *Nube de Conductores*. La *OBU* al recibir el mensaje lo muestra en un Interfaz Humano-Máquina (*HMI*) que posee el vehículo. La información del vehículo puede ser recogida a través de un interfaz *OpenXC* ó/y la *OBU*, dependiendo de la tecnología que se emplee para ello.

Los mensajes enviados a los vehículos a motor siguen el formato mostrado en la sección 4.1.2. La *Nube de Conductores* envía mensajes HTTP/1.1 a través del método POST un mensaje con contenido JSON a la *RSU*. Ésta se encarga de comunicarlo al vehículo a través de la red *IEEE 802.11p*.

4.1.2. Formato de los mensajes

Para poder realizar la conexión desde diferentes plataformas y entornos de desarrollo, se ha optado por buscar el diseño más abierto y flexible posible. Los datos son almacenados y transmitidos en formato plano con la codificación de caracteres *UTF-8*, para que puedan ser manipulados desde cualquier plataforma. Estos mensajes están construidos en formato JavaScript Object Notation (*JSON*) para facilitar su análisis. A continuación, se muestra un ejemplo de la forma que tienen los mensajes recibidos de vehículos a motor:

```
1 { "type": "motorist_position", "id": "a3553743", "timestamp": "12343242344",
2 "latitude": "43.270880", "longitude": "-2.937973", "altitude": "20",
3 "heading": "53", "speed": "5" }
```

Algoritmo 4.2: Formato de mensajes

En las siguientes secciones se explica en detalle el formato de los mensajes que son enviados y recibidos a través de la *Nube de Conductores*.

Mensaje de posición de vehículo a motor

Indican la información geográfica de un vehículo. Los mensajes entrantes en la *Nube de Conductores* tienen que tener todos los campos indicados, mientras que los mensajes salientes se usarán los campos que sean necesarios.

Tabla 4.1: Formato de mensaje Vehículo a Motor

Tipo	*Uso*	*Descripción*
type	String	Identificador del tipo de mensaje. Su valor es *motorist_position*.
id	String	Identificador del vehículo. Se emplea el ID del router Linkbird-MX
timestamp	Integer	Marca de fecha y hora a la que se envía el mensaje.
latitude	Double	Latitúd en la que se encuentra el vehículo.
longitude	Double	Longitúd en la que se encuentra el vehículo.
altitude	Integer	Altitúd en la que se encuentra el vehículo.
heading	Float	Dirección que mantiene el vehículo respecto al Norte magnético.
speed	Float	Velocidad a la que circula el vehículo.

Mensaje de posición de ciclista

Indican la información geográfica de uno o más ciclistas. Los mensajes entrantes en la *Nube de Conductores* tienen que tener todos los campos indicados, mientras que los mensajes salientes se usarán los campos que sean necesarios

Tabla 4.2: Formato de mensaje Ciclista

Tipo	*Uso*	*Descripción*
type	String	Identificador del tipo de mensaje. Su valor es *cyclist_position*.
id	String	Identificador del vehículo. Se emplea el identificador de Android.
timestamp	Integer	Marca de fecha y hora a la que se envía el mensaje.
latitude	Double	Latitúd en la que se encuentra el vehículo.
longitude	Double	Longitúd en la que se encuentra el vehículo.
altitude	Integer	Altitúd en la que se encuentra el vehículo.
heading	Float	Dirección que mantiene el vehículo respecto al Norte magnético.
speed	Float	Velocidad a la que circula el vehículo.
components	Integer	Número de ciclistas sobre los que se informa. Permite la creación de grupos de ciclistas.

Mensaje de alerta

Cuando la *Nube de Conductores* detecta que un ciclista y un vehículo a motor tienen una gran probabilidad de encontrarse, se envía este tipo de mensaje para comunicar la distancia entre los vehículos y su posición relativa.

Tabla 4.3: Formato de mensaje Ciclista

Tipo	*Uso*	*Descripción*
type	String	Identificador del tipo de mensaje. Su valor es *alert*.
distance	String	Distancia a la que se encuentra un vehículo.
relative_angle	Integer	Ángulo relativo al que se encuentra el vehículo.

4.2. NUBE DE CONDUCTORES

El núcleo del sistema es una aplicación desplegada en la nube, la cual se ha denominado *Nube de Conductores*, donde se concentran en una base de datos la información relativa a ciclistas y vehículos a motor. Un servicio de aplicación web embebido llamado *Jetty* se encarga de recibir y atender los mensajes HTTP/1.1 que son enviados desde la parte de vehículos a motor y ciclistas. Dos *Handler* independientes se encargan de filtrar los mensajes que no han sido correctamente construidos, es decir, tienen un formato inválido, e insertar y actualizar los datos de la base de datos.

Para el despliegue de la aplicación se utilizado una máquina virtual *Ubuntu Server 14.04 LTS* que cuenta con 2048 MiB de memoria RAM y 2 núcleos para procesamiento. También se ha reservado un dominio público para que las peticiones puedan ser enviadas al servidor. Gracias a la herramienta *ANT* se puede cambiar fácilmente la plataforma donde se distribuya la aplicación, además esta configurada para poder ser ejecutada directamente con el comando *run*.

4.2.1. Procesos

A través del API de *Jetty* la aplicación crea un servidor con dos manejadores de mensajes, uno para ciclistas y otro para vehículos a motor. A través de ellos la *Nube de Conductores* recibe datos de ciclistas y vehículos a motor, almacenándolos en una base de datos interna sin necesidad de utilizar un DBMS, ya que no hace falta que los datos sean persistentes más tiempo de lo que los vehículos estén emitiendo su posición. Cada manejador posee un ThreadPool con el que crea un gestor para cada mensaje recibido, este esta limitado a un número de hilos para evitar que la aplicación se colapse.

Un registro se considera antiguo cuando no ha sido refrescado en un período de un minuto. Para evitar que emplee información obsoleta, se ejecuta una rutina que tan solo mantiene en memoria los registros que periódicamente están siendo actualizados; esto se realiza gracias al campo de *timestamp*.

Paralelamente, otro algoritmo compara las posiciones de los vehículos. Cuando se detecta que los vehículos a motor y los ciclistas están próximos - en un rango menor a 200 metros - se manda a ambos vehículos una alerta avisándoles de su proximidad *[Algoritmo 4.3]*.

```
for (Motorist m : lMotorist) {
  for (Cyclist c : lCyclist) {
    if (isCollisionDanger(m, c)) {
      sendWarningToMotorist(c);
      sendCyclistPositionToMotorist(c);
    }
  }
}
```

Algoritmo 4.3: Cálculo de la proximidad de los vehículos

4.3. APLICACIÓN DE CICLISTAS

Con el objetivo de incrementar la seguridad de los ciclistas en las carreteras, se ha desarrollado una aplicación móvil. Ésta permite propagar información sobre el tránsito de vehículos en la carretera y de esta forma, el ciclista puede colocarse en una mejor posición a la hora de ser adelantado por otro vehículo, ó puede saber qué se va a encontrar en una zona de visibilidad reducida antes de aproximarse.

Se ha elegido la plataforma Android debido al predominio de este sistema en el mercado actual, de esta forma se puede maximizar la recepción. Para este desarrollo se ha usado la API 23 de Android con retro-compatibilidad hasta la API 15. Si en un futuro se desea ampliar la plataforma a un mayor mercado, la solución tendría que pasar por adaptar el código escrito en Android a C# y utilizar la plataforma Xamarin para generar una aplicación para cada plataforma.

Esta solución requiere de la *Nube de Conductores* para funcionar, ya que la información de los ciclistas es enviada a la misma y, de la misma forma, se puede recibir información sobre otros vehículos en la carretera, además de las alertas que manda la nube en caso de detectar una aproximación a un vehículo.

4.3.1. Modos de funcionamiento

Existen dos modalidades de funcionamiento diferentes, en se muestra la misma información al ciclista aunque su modo de proceder variará:

1. Modo individual: el usuario manda mensajes con su posición a través de *HTTP/1.1* a *Driver's Cloud*. Los mensajes provenientes de la nube son mandados al dispositivo mediante el servicio *GCM* de *Google*.
2. Modo grupal: uno de los terminales de los integrantes del pelotón actuará como HUB, y se encargará de gestionar todos los mensajes que lleguen desde la nube; se denomina *líder* del grupo. Este líder retransmitirá los mensajes a los demás miembros del grupo; denominados *seguidores*. Los mensajes que llegan al líder utilizan el mismo método que el modo de funcionamiento individual, pero al reenviar los mensajes que envían datagramas *UDP* dentro del *Hub*. El establecimiento de la comunicación se realiza de la siguiente forma:
   a) El dispositivo que actúa como líder crea el *Hub* automáticamente al entrar en la opción *líder* de la aplicación.
   b) Los seguidores entran en el modo seguidorde la aplicación, y seleccionan el grupo al que desean ingresar. El dispositivo enviará una petición al líder.

c) El líder al recibir una petición, la acepta o rechaza. Dependiendo si su dispositivo está sincronizando dispositivos o no.
d) Si el líder ha aceptado la petición el seguidor queda a la espera hasta que el líder dé comienzo a la salida.
e) En cuanto comience la salida el líder mandará mensajes a través de broadcastcada vez que reciba notificaciones de la nube.

En las figuras 4.3 y 4.4 se muestra la interfaz gráfica con la que se encuentra el usuario. Nótese en la interfaz del seguidor que puede buscar un grupo de dos maneras: (1) buscando el grupo de manera manual a través de una lista, ó (2) dejando que la aplicación auto-detecte una red y trate de unirse a ella.

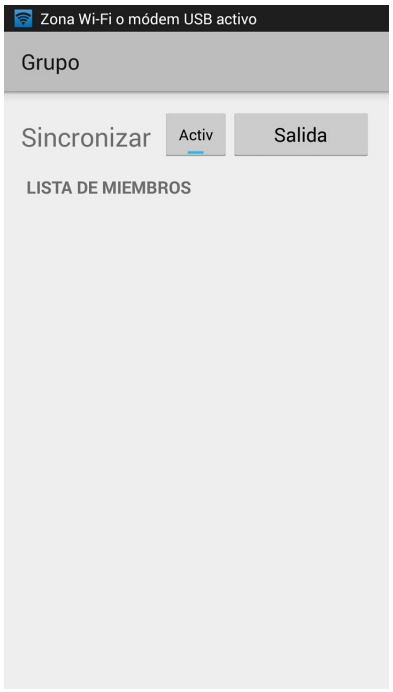

Figura 4.3: *Hub* del líder

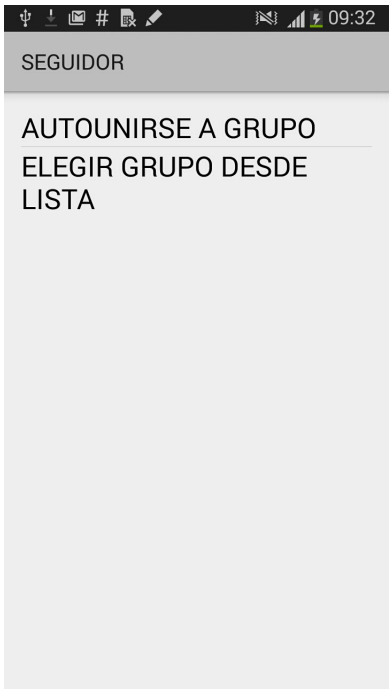

Figura 4.4: Ingreso al grupo del seguidor

Para mantener un registro de la ruta que se esta realizando, un controlador mantiene toda la información sobre la salida que se esta realizando. En la figura 4.6 se observa la estructura de este controlador, el funcionamiento es como siguiente:

AJourney y AGroupJourney interfáz gráfica que se muestra al usuario (figura 4.5).

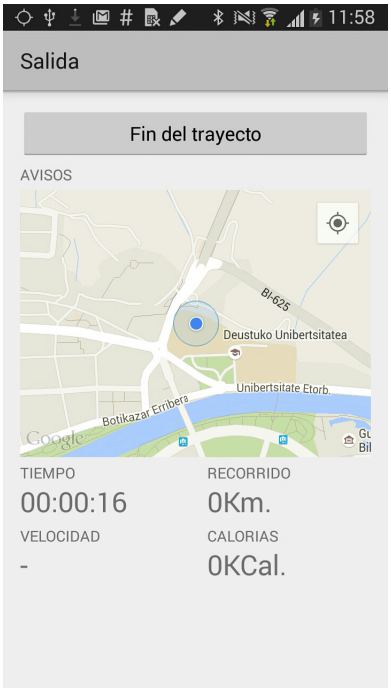

Figura 4.5: UI de la salida

UIUpdateListener escuchador de los eventos que se generan en cuanto un mensaje es recibido. Actualizará la interfaz gráfica para mostrar la información al usuario.

GPSController encargada de activar el *GPS* y subscribirse a las actualizaciones de posición. Se ha configurado para refrescar la posición cada dos segundos, cuando esto sucede se envía una notificación a la nube con los datos recogidos.

JourneyController gestor de la salida. Controla los datos relacionados con la salida: tiempo, distancia recorrida, ruta y calorías quemadas. Permite ser pausada y reanudada.

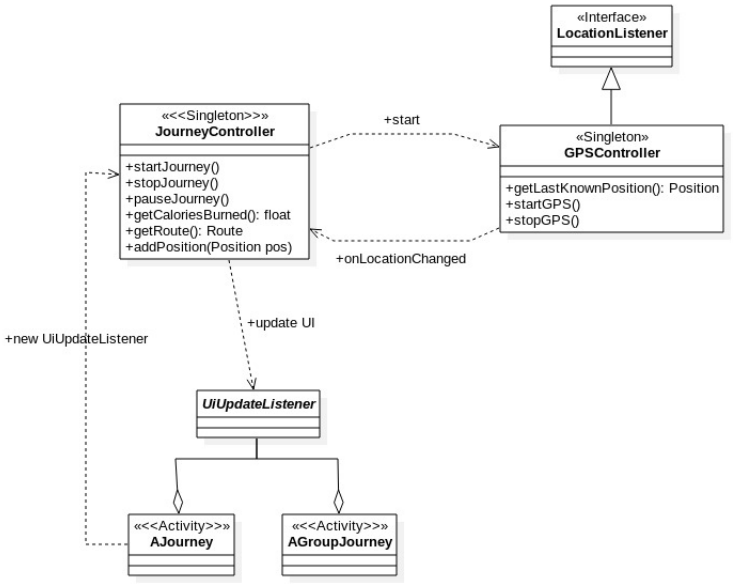

Figura 4.6: Controlador de la salida

4.3.2. Comunicación con la nube

Cuando la posición del ciclista es actualizada, se formatean los datos en un objeto JSON y se envían a la nube por medio de un mensaje *HTTP/1.1 POST*. El dominio del servidor es fijo, por lo que siempre se tendrá localizado la dirección de destino [Algoritmo 4.4]. Cuando el mensaje es recibido por la nube, la aplicación comprobará si el ciclista tiene algún peligro cerca. Si se detecta un vehículo cercano, la aplicación desplegada en la nube contestará con un mensaje de alerta con la información del vehículo detectado y la distancia que les separa.

```
1  HttpClient httpClient;
2  HttpPost httpPost;
3  String data;
4
5  data ="{\"id\":\"" + cyclist.getIdentifier() + "\"," +
6    "\"type\"": + "\"cyclist_position\"," +
7    "\"latitude\":\"" + cyclist.getPosition().getLatitud() +  "\"," +
8    "\"longitude\":\"" + cyclist.getPosition().getLongitud() + "\"," +
9    "\"altitude\":\"" + cyclist.getPosition().getAltura() + "\"," +
10   "\"heading\":\"" + cyclist.getPosition().getRumbo() + "\"," +
11   "\"speed\":\"" + cyclist.getSpeed() + "\"," +
12   "\"components\":\"" + cyclist.getPersonas() + "\"," +
13   "\"timestamp\":\"" + new Timestamp(new Date().getTime()) + "\"}";
14 httpClient = new DefaultHttpClient();
15 httpPost = new HttpPost("http://cloud.mobility.deustotech.eu/cyclist");
16 httpPost.setEntity(new StringEntity(data));
17 httpClient.execute(httpPost);
```

Algoritmo 4.4: Envío de peticiones desde la aplicación de ciclistas a la Nube de Ciclistas

Para la recepción de mensajes, la aplicación tiene que pedir un ”token” de registro único del

servidor *GCM* (*Google Cloud Messaging*)[1]. Para ello el dispositivo tiene que tener instalado los servicios *Google Play*. Una vez obtiene el ”token”, cuando se envíe un mensaje al servidor se incluirá este identificador dentro del contenido para que la aplicación en el servidor pueda saber a qué dispositivo debe responder. Tras haber realizado la autentificación con los servicios de Google, un escuchador espera a nuevas notificaciones y los procesa una vez han llegado [4.7].

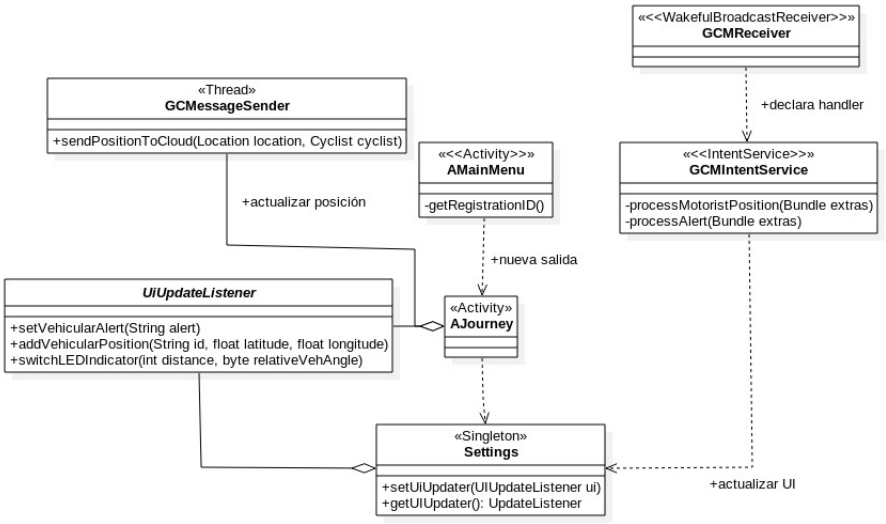

Figura 4.7: Estructura de la comunicación GCM

4.3.3. Comunicación del grupo

Entre los seguidores y el líder se enviarán notificaciones sobre el estado actual de cada nodo (Tabla **??**) a través de del modo *Hub* que tienen los dispositivos; la explicación completa puede encontrarse en la subsección

Tabla 4.4: Tipo de mensajes en grupo

MENSAJE	*Descripción*	Campos extra
REGISTER	Petición de ingreso de un seguidor al *Hub*.	*nombre*
ACCEPT	Respuesta de aceptación de ingreso de un seguidor al *Hub*.	-
KICK	El administrador echa del *Hub*a un seguidor.	-
START	Notificación de comienzo de la salida.	-
STOP	Fin de una salida.	-
PAUSE	Notificación de pausa de la salida.	-
RESUME	Notificación de reanudado de la salida.	-
ALERT	Alerta por vehículo cercano.	Tabla 4.3
MOTORIST_POSITION	Posición de un vehículo.	Tabla 4.1

Los mensajes son enviados a través del protocolo de transporte *UDP* para que la recepción de mensajes sea lo más rápido posible. Al ser un canal poco fiable se ha implementado una capa

[1]Servicio de mensajería ofrecido por *Google* para enviar y recibir mensajes desde diferentes plataformas.

para garantizar la recepción del mensaje: cuando un mensaje es enviado, se almacena en un HashTable utilizando como clave la IP del destinatario y un número identificativo del mensaje[2]. El receptor mandará un ACK al emisor cuando un mensaje le llegue, y este último eliminará del HashMap el registro previamente almacenado. En caso de que un mensaje no llegue, un *timeout* provocará que el emisor vuelva a enviar el mismo mensaje al receptor; el *timeout* se incrementará al doble cada re-envío. En caso de que un mensaje no sea recibido al quinto intento, se dejará de intentarlo (figura 4.8).

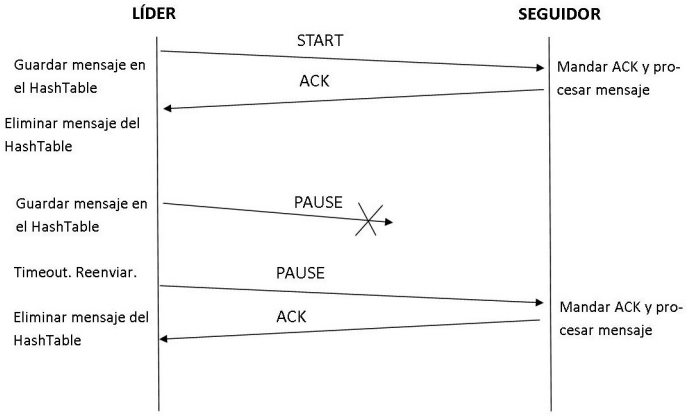

Figura 4.8: Comunicación líder-seguidor

4.3.4. Casco BLE

El ciclista no puede estar pendiente de los avisos de su Smartphone continuamente, ya que esto puede poner en riesgo su seguridad. Para que el ciclista pueda mantener la vista en la vía y tenga la posibilidad de saber si hay algún vehículo que pueda ponerle en riesgo, se ha integrado una *mota Texas CC2540* en el casco del ciclista conectado a varios LEDs que según un código de colores le informan al ciclista sobre eventos que sean peligrosos. Si por ejemplo hay un vehículo acercándose por el lado izquierdo, un LED amarillo o rojo se encenderá, dependiendo si la distancia es menor de 50 ó 20 metros respectivamente.

Este dispositivo utiliza el estándar BLE para comunicarse con la aplicación móvil mediante cortos mensajes de 8 bytes, los cuales contienen un código hexadecimal que representa la combinación de LEDs que deben encenderse.

Programación de la mota La mota contiene un pequeño programa escrito en lenguaje C que se encuentra flasheado en su ROM (Read Only Memory). Este programa configura el micro-controlador para actuar como servidor (denominado *Central*), a la espera de ser emparejado y recibir mensajes. Hasta que se sincroniza con un dispositivo, cada 50 mili segundos propaga una señal para que los dispositivos puedan emparejarse. Cuando un dispositivo se conecta, el micro controlador espera a recibir un pequeño mensaje con el código de la señal que le especificará qué LEDs debe encender. Cuando llega el mensaje, si el código recibido es el correcto enciende el LED correspondiente. Para el desarrollo de este programa se ha

[2]El HashTable solo permite una clave, por lo que se ha creado una clase que calcula un Hash en base a las dos claves con que deseamos identificar el datagrama.

trabajado sobre una plantilla, que provee Texas Instrument junto con el dispositivo CC2540, al que se ha añadido el servicio necesario para encender el LED al recibir una señal. En el algoritmo **??** se explica cómo se ha implementado un nuevo servicio para gestionar los mensajes entrantes.

Programación de la app La aplicación de ciclistas actúa como cliente, por lo que el usuario debe primero emparejarse con el casco para que pueda comenzar a comunicarse con la mota. El proceso de conexión y envío de mensajes consiste:

1. Buscar los servicios Bluetooth disponibles.
2. Conectar al dispositivo en cuestión.
3. Descubrir los servicios que ofrece el dispositivo. Esto devolverá varios UUIDs con los servicios que tiene disponibles la mota. Una vez se sabe cuál es el servicio que controla la recepción de mensajes, hay que obtener una referencia.
4. Descubrir las características que contiene el servicio. Empleando la referencia del servicio, se pueden obtener uno o varios UUID que representan variables en las que se puede escribir un valor. Aquí es donde se depositará el código de combinación de LEDs que se desea encender [A.2].
5. Escribir sobre la característica que gestiona los LEDs [A.3].

```
1  public void conectarBLE(Device dispositivo) {
2    // El primer argumento indica que la propia clase gestionará los eventos,
3    // el segundo argumento que se autoconectará al dispositivo, y el tercer
4    // argumento a qué dispositivo va a conectarse.
5    bluetoothGatt = device.connectGatt(this, false, dispositivo);
6  }
7
8  public void mandarMensajeBLE(byte msg) {
9    // obtener el servicio que contiene la característica que se va a modificar
10   servicio = bluetoothGatt.getService(UUID\_SERVICIO);
11
12   // obtener la característica (local)
13   caracteristica.getCharacteristic(msg);
14
15   // modificar el valor de la característica (local)
16   caracteristica.setValor(msg);
17
18   // aplicar cambios en el dispositivo remoto
19   bluetoothGatt.writeCharacteristic(caracteristica);
20 }
```

Algoritmo 4.5: Envío de mensajes LED desde la aplicación de ciclistas

Tabla 4.5: Tabla de la verdad de señales LED

SEÑAL	*MNEMÓNICO*	DESCRIPCIÓN
0x00	NONE	Sin peligro. Apagar los LED
0x01	AL_RIGHT	Vehículo a menos de 20 metros por la derecha. Encender luz roja derecha.
0x02	AL_LEFT	Vehículo a menos de 20 metros por la izquierda. Encender luz roja izquierda.
0x03	AL_BACK	Vehículo a menos de 20 metros por detrás. Ambas luces rojas encendidas.
0x04	AL_FRONT	Vehículo a menos de 20 metros por delante. Ambas luces rojas encendidas parpade
0x11	W_RIGHT	Vehículo a menos de 50 metros por la derecha. Encender luz amarilla derecha.
0x12	W_LEFT	Vehículo a menos de 50 metros por la izquierda. Endernder luz amarilla izquierda.
0x13	W_BACK	Vehículo a menos de 50 metros por detrás. Ambas luces amarillas encendidas.
0x14	W_FRONT	Vehículo a menos de 50 metros por delante. Ambas luces amarillas parpadeando e

4.4. COMUNICACIÓN VEHICULAR

Los RSU y OBU son los encargados de proveer las posiciones de los vehículos a motor de la *Nube de Conductores*. El OBU se encuentra integrado en el vehículo, y envía a través de *IEEE 802.11p* las posiciones de los vehículos a la RSU. La RSU recoge los datos enviados por la OBU y los retransmite a la nube, al igual que recibe datos de la nube y los retransmite al OBU.

4.4.1. Unidad en carretera

Formato por un router que se comunica a través de *IEEE 802.11p*, y un computador conectado tanto al router como a una red LTE. Actúan como puente entre las OBU instaladas en los vehículos y la nube. El RSU escucha y escribe a través de dos canales:

1. En el canal LTE recibe los mensajes HTTP/1.1 que provienen de la Nube de Conductores, así como envía las posiciones de los vehículos a la nube. Un Servicio web posibilita la gestión de estos mensajes; el formato es el mismo que el explicado en la sección 4.1.2.
2. Escucha el canal IEEE 802.11p los mensajes que son enviados por los vehículos, y los redirige a la nube a través de mensajes HTTP/1.1.

Funcionamiento

1. Se comprueba que el formato del mensaje es correcto. Si no lo es, se descarta.
2. Lectura del campo TYPEdel mensaje. Dependiendo de su contenido se aplica un proceso diferente: el mensaje puede ser retransmitido a través de Broadcast, se puede mostrar una notificación en un panel informativo en carretera...

Una propuesta para añadir funciones adicionales consiste en conectar el RSU a elementos informáticos que pueden existir en la carretera, por ejemplo los paneles informativos, y al enviar un mensaje desde la Nube de Conductores a una RSU en concreto se muestra la información deseada.

4.4.2. Unidad en el vehículo

Dentro del ecosistema vehicular existen varias unidades con diferentes papeles: el OBU, el HMI (Human-Machine interface) y la interfaz OpenXC (o tecnología equivalente). Para darse la comunicación con todas las plataformas se depende de la existencia de RSU desplegadas en la carretera.

OBU

Formado por un router que se comunica a través de IEEE 802.11p, un dispositivo GPS conectado al router, y un computador conectado al router a través de un conector RJ-45. A traves de esta red se reciben mensajes de diferentes RSU que se encuentran desplegadas en la carretera, al igual que se envían mensajes informando sobre la posición del vehículo a través de Broadcast. Su funcionamiento consiste en:

- Obtener la posición del vehículo realizando peticiones al router.
- Mandar periódicamente las posiciones a través de broadcast.
- Escuchar los mensajes que mandan las RSU.
- Ofrecer y proveer al usuario la información de los mensajes a través del HMI.

Información al usuario

Para proveer información al conductor se emplea el HMI, el cual consiste en un ordenador de a bordo que contiene diferentes apps. adicionalmente, se puede conectar al OBDII (On-Board Diagnostic)[3] a través de un puerto RS 232 para obtener información del vehículo; como por ejemplo, el estado de los neumáticos.

Se ha desarrollado una aplicación conectada por Bluetooth al OBU con la que se muestra al vehículo un mapa con las posiciones de los ciclistas cercanos [Imagen 4.9]. Cuando un vehículo se acerca a un ciclista o grupo de ciclistas una notificación salta para informar al conductor. Las posiciones de los ciclistas son obtenidas a través de la nube, y almacenadas en el vehículo durante un período de 30 segundos; según llegan las nuevas posiciones desde la nube, se van actualizando. Pasado ese tiempo, los registros que no hayan sido refrescados son eliminados, ya que esto significaría que los nodos han dejado de transmitir.

[3] Se trata del sistema de diagnóstico del vehículo. Provee información sobre el estado de los diferentes subsistemas del vehículo.

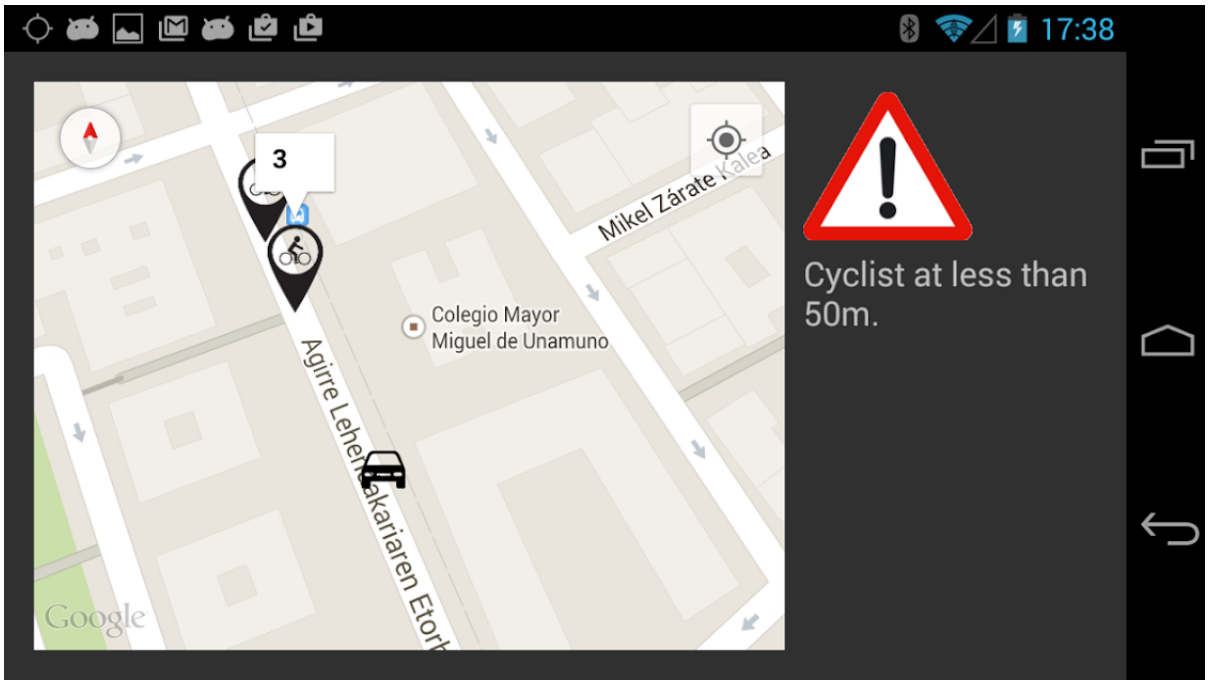

Figura 4.9: UI de la aplicación instalada en el HMI

Comunicación HMI-OBU

Para crear el servidor Bluetooth que envíe los mensajes recibidos por el OBU a la aplicación instalada en el HMI, se ha utilizado la librería Bluecove. En el 4.6 se abre un punto de acceso para clientes Bluetooth, y se envían los mensajes que se han recibido en el OBU y deben ser mostrados al usuario.

```
BluetoothServer server;
StreamConnection cnn;
BufferedWriter writer;
final String UUID = "btspp://localhost:432814212fd123e;name=obu";
StreamConnectionNotifier notifier;

// 1. Para que el HMI pueda conectarse al OBU la conexión primero se debe hacer
// visible bajo un UUID determinado.
LocalDevice.getLocalDevice().setDiscoverable(DiscoveryAgent.GIAC);
cnn = notifier.acceptAndOpen();

// 2. Se espera a que un cliente se conecte al servicio Bluetooth
writer = new BufferedWriter(new OutputStreamWriter(cnn.openDataOutputStream()));

// 3. Una vez conectado, se envían mensajes al cliente en cuanto son recibidos, mediante
// una cola de mensajes
[...]

// 4. Se cierra el socket al finalizar la conexión
writer.close();
cnn.close();
```

Algoritmo 4.6: Creación de un punto de acceso Bluetooth en el OBU

5. RESULTADOS Y PRUEBAS

6. PLANIFICACIÓN: FASES Y CALENDARIO

En el siguiente apartado se describe cuál ha sido la planificación del proyecto, y cómo se ha organizado. Esta planificación se ha desarrollado para poder mantener un ritmo contínuo de desarrollo de tipo ágil, y conseguir un aprovechamiento máximo del proyecto. Durante el desarrollo de este proyecto se ha realizado paralelamente pequeños proyectos, por lo que era necesario que se pudiesen observar los resultados del desarrollo de la forma más inmediata posible. Para ello, como se ha explicado en el capítulo 3, se realizan pequeños prototipos de una funcionalidad que más adelante son integradas en el proyecto.

La estrategia seguida durante el proyecto puede observarse en la imagen 6.1. Inicialmente se reunen los requisitos necesarios y se realiza un diseño de las funcionalidades a incluir, se producen varias iteraciones hasta que todos los requisitos son satisfechos. Una vez finalizado el diseño se pasa al desarrollo, el cual incluye depuración y pruebas unitarias. Cuando el desarrollo ha finalizado correctamente, se pueden añadir nuevos requisitos o pasar a pruebas en calle si han sido programadas con anterioridad; algunas pruebas requieren de adaptar parte del programa al caracter de las pruebas. Una vez finalizadas las pruebas, si se han descubierto fallos en el software, es reparado y si se decide que es requerido ampliar la plataforma se vuelve a iniciar el ciclo.

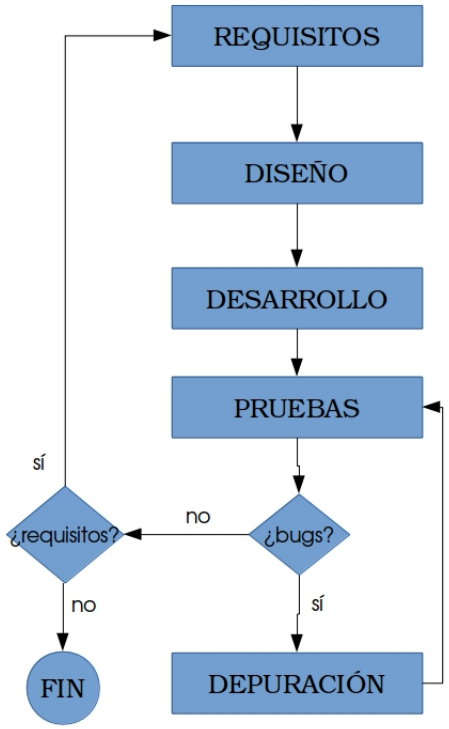

Figura 6.1: Estrategia del desarrollo

En las imágenes 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 y 6.8 se presentan varios diagramas Gantt en los que se puede observar las diferentes tareas y fases por las que ha pasado el proyecto, presentando las correspondientes fechas de inicio y fin que se han estimado durante la ejecución del proyecto. Al tratarse de un proyecto incremental mediante generación de prototipos, se han realizado varias iteraciones durante el desarrollo.

Las fases del proyecto se puede dividir en los siguientes bloques:

- Recolección de requisitos del sistema y estudio de la tecnología necesaria.
- Desarrollo de la Nube de conductores.
- Desarrollo de la aplicación para ciclistas.
- Desarrollo de la RSU, OBU y HMI.
- Verificación del sistema en la calle.

- Estudio de los resultados y generar documentación.

Los hitos del proyecto son las distintas pruebas que se han hecho de funcionalidades diferentes del sistema. Durante estas pruebas se analiza el rendimiento que se obtiene de la comunicación del sistema, se proponen y añaden mejoras que serán incluidas en siguientes pruebas.

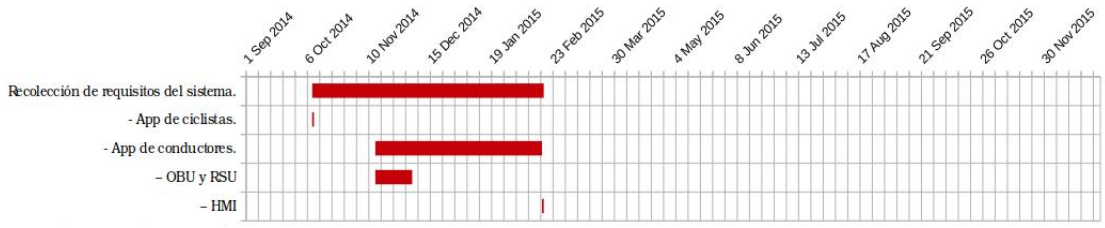

Figura 6.2: Diagrama de Gantt - Especificaciones

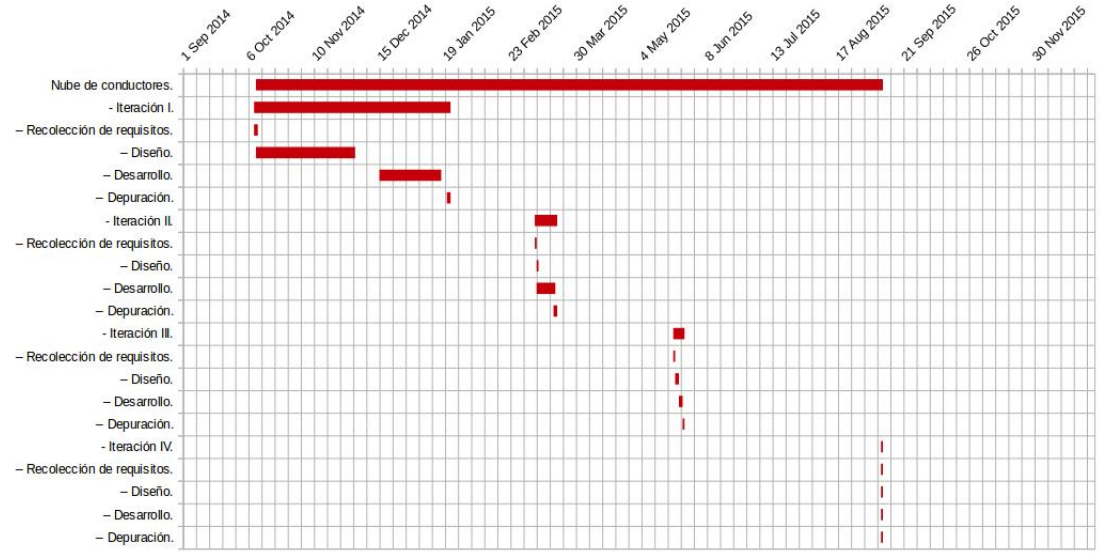

Figura 6.3: Diagrama de Gantt - Nube de conductores

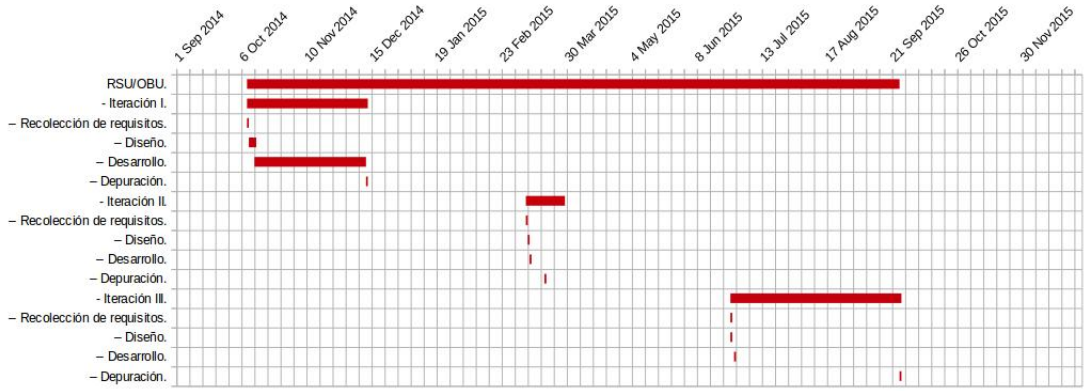

Figura 6.4: Diagrama de Gantt - Especificaciones

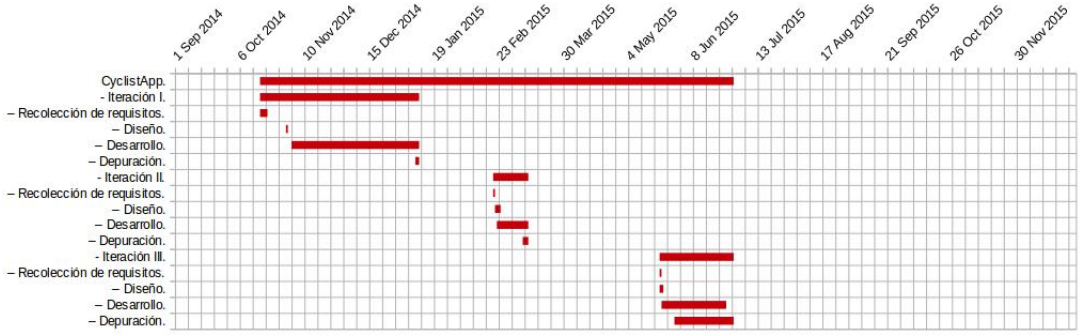

Figura 6.5: Diagrama de Gantt - Aplicacion de ciclistas

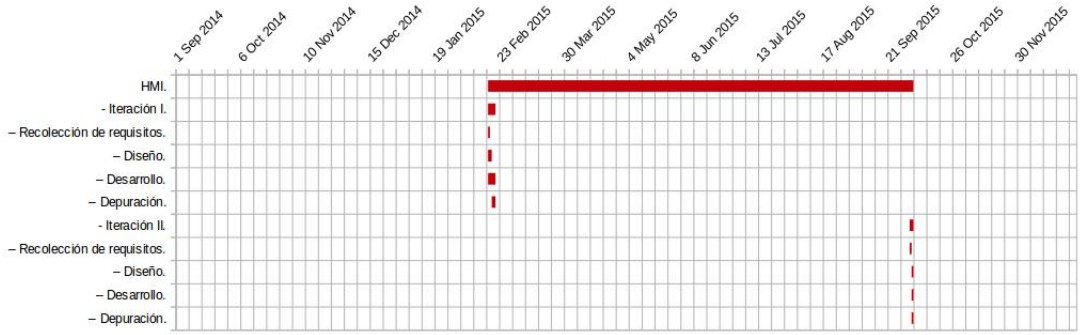

Figura 6.6: Diagrama de Gantt - HMI

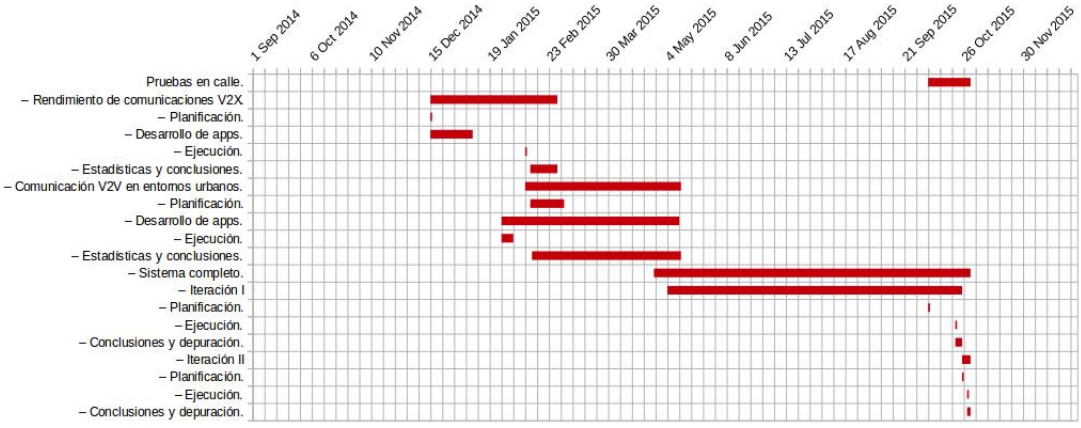

Figura 6.7: Diagrama de Gantt - Pruebas en la calle

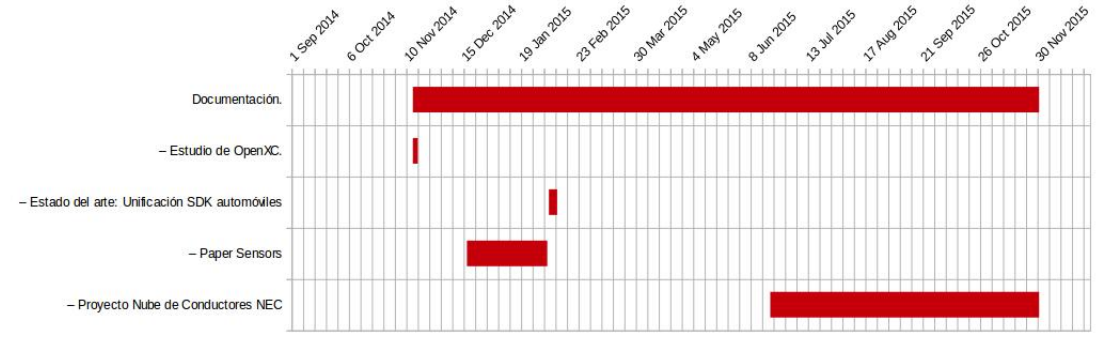

Figura 6.8: Diagrama de Gantt - Documentación

7. PRESUPUESTOS

Se ha realizado una recopilación de todos los gastos que se han dado en el proyecto. Estos costes tienen que ver con la adquisición de equipo, contratación de diferentes servicios y software para ser posible el estudio, desarrollo, verificación y documentación del sistema.

Se requiere que los gastos sean lo más bajos posibles, ya que se trata de un proyecto para la investigación y no está pensado para ser comercializado. Debido a ello se han empleado plataformas Open Source para el desarrollo y despliegue de las aplicaciones, ya que no generan ningún tipo de gasto. Al desarrollarse el proyecto bajo amparo de la Universidad de Deusto, existen gastos de instalaciones, mantenimiento, servicios y plataformas que son nulos; como por ejemplo el acceso repositorios Git.

Los beneficios de esta investigación se traducen en el impacto que tengan los artículos que puedan ser generados y publicados en diferentes revistas, diarios...

Los costes se han dividido en tres categorías diferentes: los gastos de material y equipo necesario [7.1], costes laborales del personal [7.2] y el coste de otros proyectos subcontratados [7.3]. Finalmente, se presenta un resumen de todos los gastos [7.4].

Tabla 7.1: Costes de material y equipo

Concepto	Tipo	Unidades	Coste por unidad	Coste total
NEC Linkbird MX	Hardware	2	2.960,00 EUR	5.920,00 EUR
Dell Latitude D520	Hardware	1	134,75 EUR	134,75 EUR
Acer Aspire 4810TZG	Hardware	1	596,87 EUR	596,87 EUR
Estación de trabajo	Hardware	1	799,00 EUR	799,00 EUR
Cable UTP	Hardware	2	0,00 EUR	0,00 EUR
Adaptador RS-232 Hembra a Macho	Hardware	2	1,00 EUR	2,00 EUR
Adaptador RS-232 Null	Hardware	2	1,00 EUR	2,00 EUR
HI 204-III GPS USB	Hardware	2	59,00 EUR	59,00 EUR
Fuente de alimentación Belkin	Hardware	2	50,00 EUR	100,00 EUR
Casco BLE + Texas CC2540	Hardware	1	65,00 EUR	65,00 EUR
Sistemas operativos: Ubuntu y Debian	Software	3	0,00 EUR	0,00 EUR
Sistema operativo Windows 7	Software	1	0,00 EUR	0,00 EUR
Software de desarrollo Java y C	Software	1	0,00 EUR	0,00 EUR
LaTeX	Software	1	0,00 EUR	0,00 EUR
LibreOffice	Software	1	0,00 EUR	0,00 EUR
Kit de desarrollo NEC	Software	1	0,00 EUR	0,00 EUR
Máquina virtual Ubuntu	Servicio	1	0,00 EUR	0,00 EUR
Dominio web	Servicio	1	0,00 EUR	0,00 EUR
Repositorio GitLab	Servicio	1	0,00 EUR	0,00 EUR
Google Cloud Messaging	Servicio	1	0,00 EUR	0,00 EUR
Google Maps	Servicio	1	0,00 EUR	0,00 EUR
				7,732.62 EUR

Tabla 7.2: Costes laborales

Concepto	Coste por hora	Horas	Coste total
Estudios módulos NEC.	5	20	100,00 EUR
Desarrollo de prototipos para pruebas.	5	20	100,00 EUR
Recolección de requisitos del sistema.	30	23	690,00 EUR
Formación en Android SDK.	5	20	100,00 EUR
Desarrollo de la Nube de Conductores.	5	114	570,00 EUR
Desarrollo de la OBU/RSU.	5	74	370,00 EUR
Desarrollo del HMI.	5	17	85,00 EUR
Desarrollo de CiclistasApp.	5	170	850,00 EUR
Pruebas en la calle.	30	43	1.290,00 EUR
Generar documentación.	30	55	1.650,00 EUR
			5.805,00 EUR

Tabla 7.3: Costes de subcontrataciones

Concepto	Coste total
PFG de Idoia de la Iglesia.	32.310,00 EUR
	32.310,00 EUR

Tabla 7.4: Resumen de costes

Concepto	**Coste total**
Hardware	7.737,62 EUR
Software	0,00 EUR
Servicios	0,00 EUR
Personal	5.805,00 EUR
Subcontrataciones	32.310,00 EUR
	45.852,62 EUR

8. CONCLUSIONES

A. APÉNDICE

A.1. CÓDIGO FUENTE DEL FUNCIONAMIENTO DE MENSAJES GCM

```
HttpURLConnection httpRequest;
final String KEY = AIzaSyAu2LXHXn7_rPOOUinzizQg5r5mgln4Q-Y;

try {
  // abrir conexión con el gestor GCM
  URL url = new URL("https://android.googleapis.com/gcm/send");

  httpRequest = (HttpURLConnection) url.openConnection();

  // enviar datos mediante POST
  httpRequest.setRequestMethod("POST");

  // establecer el encabezado
  httpRequest.setRequestProperty("Content-Type", "application/json");
  httpRequest.setRequestProperty("Authorization", "key=" + KEY);
  httpRequest.setDoOutput(true);

  // enviar mensaje y leer respuesta
  [...]
} catch (IOException e) {
  logger.error(e.getMessage());
}
```

Algoritmo A.1: Envío de mensajes mediante GCM

A.2. CÓDIGO FUENTE CASCO BLE

```
// Declarar el UUID del perfil ATT (Atribute Protocol)
#define PROFILE_VEHICULAR POS   5

// Declarar el UUID del servicio
#define SIMPLEPROFILE_SERV_UUID 0xFFF0

//Declarar el UUID de la característica, darle un tamaño y asignar
//permisos de escritura
#define PROFILE_VEHICULARPOS_UUID 0xFFF6
CONST uint8 simpleVehicularPositionProfileUUID[ATT_BT_UUID_SIZE] = {
^^ILO_UINT16(PROFILE_VEHICULARPOS_UUID), HI_UINT16(PROFILE_VEHICULARPOS_UUID)
};
static uint8 vehicularPositionProps = GATT_PROP_WRITE;
static uint8 vehicularPosition = 0;
```

Algoritmo A.2: Declaración del servicio LED

```
// gestionar que se ha hecho una escritura en la característica
static bStatus_t simpleProfile_writeAttrCB(uint16 connHandle, gattAttribute_t *PAttr,
    uint8 *pValue, uint8 len, uint16 offset) {
[...]
notifyApp = PROFILE_VEHICULARPOS;
[...]
}

// gestionar la petición
static void simpleProfileChangeCB(uint8 paramID) {
[...]
case PROFILE_VEHICULARPOS:
  SimpleProfile_GetParameter(PROFILE_VEHICULARPOS, &newValue);
  cambiarLED(paramID);
[...]
}

// escritura de la característica
bStatus_t SimpleProfile_SetParameter(uint8 param, uint8 len, void* value) {
[...]
  case PROFILE_VEHICULARPOS:
    if (len == SIMPLEPROFILE_CHAR5_LEN) {
      vehicularPosition = *((uint8 *) value);
    } else {
      ret = bleInvalidRange;
    }
    break;
[...]
}
```

Algoritmo A.3: Implementación del callback para el servicio LED

A.3. POSICIÓN VEHICULAR RELATIVA

La posición vehicular relativa se refiere a en qué lado de un vehículo A se encuentra un vehículo B. Es decir, si se toma como referencia el primer vehículo, en qué lado se encuentra el segundo; derecha, izquierda, delante o detrás.

El ”heading” o rumbo, es la dirección hacia la que se está dirigiendo un vehículo con respecto al Norte (0 grados). En la figura A.1, en el eje de coordenadas cartesiano. Cada vez que se realice una operación que involucre los ángulos, se emplea una función para mantenerlos en un rango de entre 0 a 360 grados.

El algoritmo A.4 mostrado se puede condensar en:

1. Calcular el ángulo existente entre los dos vehículos, sin tener en cuenta la dirección del primer vehículo.
2. Se resta el heading del vehículo de referencia al ángulo que hay entre los dos vehículos.
3. Se contrasta con una serie de casos ya conocidos, y se devuelve el ángulo relativo [Imagen A.2].

```
function calcularPosicionRelativa(heading, oLatitude, oLongitude, pLatitude, pLongitude) {
  var degrees = calculateAngleBetweenTwoPoints(oLatitude, oLongitude, pLatitude, pLongitude);
  var relativeAngle = correctDegrees(degrees - heading);

  var tmp = "";
  if (relativeAngle <= 15 || relativeAngle >= 345) {
    return "FRONT";
  } else if (relativeAngle >= 120 && relativeAngle <= 230) {
    return "BACK";
  } else if (relativeAngle < 120 && relativeAngle > 15) {
    return "LEFT";
  } else {
    return "RIGHT";
  }
}

function calcularAnguloEntreDosPuntos(ox, oy, x, y) {
  return toDegrees(Math.atan2(y - oy, x - ox));
}
```

Algoritmo A.4: Cálculo de la posición relativa vehicular.

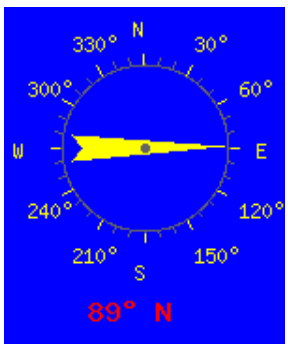

Figura A.1: Rumbo: el ángulo 0º está desplazado 90º con respecto al eje cartesiano.

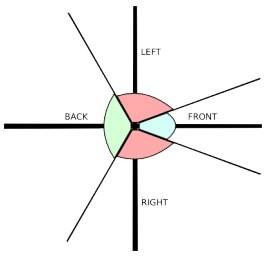

Figura A.2: Posición relativa vehicular.

A.4. POSICIÓN VEHICULAR RELATIVA

Bluetooth Low Energy. A diferencia del clásico Bluetooth, está diseñado para consumir una cantidad significativamente más pequeña de energía. Algunas diferencias durante el desarrollo respecto Bluetooth estándar a tener en cuenta en BLE son:

1. El dispositivo está continuamente durmiendo y despertándose para ahorrar batería.
2. La cantidad de información transmitida es pequeña, como máximo 216 bytes.
3. La transmisión de información se hace de manera rápida para poder poner el dispositivo a dormir tan pronto como se haya terminado de transmitir la información; latencias de hasta 2 ms por ráfaga.

A.4.1. GATT

Generic Attribute Profile. Establece cómo se va a transmitir la información sobre los perfiles y datos en la conexión BLE. GATT emplea el ATT (Attribute Protocol) como protocolo de transporte para intercambiar datos entre los dispositivos. Los datos están organizados jerárquicamente en secciones llamadas ”servicios”, los cuales tienen piezas relacionadas con ellos denomienadas ”características”.

A.4.2. UUID

Universal Unique Identifier. Es un identificador de 128 bits que está garantizado que es único. Los UUID nos permiten identificar los servicios y características, además de poder operar con ellos.