Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

MONITOR DE CONDUCCIÓN EFICIENTE PARA VEHÍCULOS A MOTOR CON INTERFAZ OBD2

Alumno: Juan Carlos Castillo Alcántara

Tutor: Prof. D. Carlos Javier Ogayar Anguita
Dpto: Informática

Septiembre, 2018

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Carlos Javier Ogayar Anguita, tutor del Proyecto Fin de Carrera titulado: Monitor de conducción eficiente para vehículos a motor con interfaz OBD2, que presenta Juan Carlos Castillo Alcántara, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2018

El alumno: Los tutores:

OGAYAR ANGUITA CARLOS JAVIER - 26039170L

Firmado digitalmente por OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Nombre de reconocimiento (DN): c=ES, serialNumber=26039170L, sn=OGAYAR ANGUITA, givenName=CARLOS JAVIER, cn=OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Fecha: 2018.09.04 10:17:45 +02'00'

Juan Carlos Castillo Alcántara Carlos Javier Ogayar Anguita

Índice

1. INTRODUCCIÓN

1.1. Antecedentes

El transporte por medio de automóviles, camiones y autobuses es un factor clave para el desarrollo social y económico, y para la cohesión de los distintos territorios que conforman un país. Sin embargo, tiene como principal contrapartida el elevado consumo energético y los altos niveles de emisión de gases contaminantes, además de producir congestión en las redes viarias y una elevada siniestralidad.

Las personas pueden aprovechar las ventajas que ofrece el transporte por carretera y a la vez reducir sustancialmente sus impactos negativos siguiendo unos sencillos consejos referentes a la elección de vehículos más limpios, una conducción más eficiente o el uso de alternativas al vehículo turismo de baja ocupación, por ejemplo, mediante el uso del transporte público. En la mayoría de los casos, estas actuaciones conllevan un ahorro económico, una mejora del medio ambiente y un aumento de la seguridad de las personas.

En España, estas actuaciones se contemplan en el Plan de Acción de Eficiencia Energética 2017-2020, a través de distintas medidas encaminadas al logro de un transporte más eficiente y sostenible. En este plan se contemplan los siguientes campos de actuación:

- Acciones encaminadas a favorecer el cambio modal en la movilidad de personas y mercancías hacia aquellos modos menos consumidores de energía por pasajero-km o tonelada-km.
- Acciones dirigidas a mejorar la eficiencia del parque de vehículos, mediante la renovación de las flotas y la incorporación de avances tecnológicos.
- Acciones encaminadas al uso eficiente de los medios de transporte.

Sin embargo, estas actuaciones deben ser aplicadas en mayor o total medida por parte de las autoridades, fabricantes de vehículos o compañías energéticas que son las encargadas de distribuir el combustible y los tipos del mismo.

En este proyecto nos centraremos en una serie de medidas o consejos adoptados a un nivel más bajo, es decir, al del conductor del vehículo. Las claves de una conducción eficiente, promovidas por la Dirección General de Tráfico, van desde la elección de un vehículo según unos criterios de eficiencia, pasando por el proceso de arranque y puesta en marcha del vehículo, la aceleración, el frenado, hasta la utilización del cambio de marchas. Estos consejos no tienen otro objetivo que ayudar al conductor tanto de manera económica, permitiéndole a éste reducir el consumo de combustible y aumentado la vida útil de su vehículo, como incrementando la seguridad del mismo, además de garantizar la reducción de emisiones al medio ambiente.

1.2. Descripción del problema

Este Trabajo de Fin de Grado o TFG consiste en desarrollar un sistema que permita recabar información referente a los parámetros de conducción de un vehículo mientras está en marcha, como pueden ser la aceleración, la velocidad, las revoluciones o el consumo de combustible. Este proceso se realizará empleando un dispositivo de medida conocido como OBD2 (On-Board Diagnostics), muy utilizado en talleres mecánicos para revisiones de vehículos. La información recibida de este dispositivo se almacenará para después realizar un análisis del trayecto mostrando en un mapa de Google Maps el recorrido por las distintas calles o carreteras con los datos asociados de los parámetros de conducción.

De la misma forma se notificará al usuario en qué zonas o partes del recorrido el coche consume más combustible o realiza un mayor esfuerzo. Además, se presentarán la información recabada en gráficos para una mejor y completa visualización de los datos numéricos obtenidos.

1.3. Motivación y objetivos

Este TFG se enmarca dentro de la rama de Tecnologías de la Información ya que hace uso de distintas técnicas y herramientas para la recolección de información, la transmisión, el almacenamiento y la manipulación de la misma, así como el intercambio de datos entre diferentes tipos de dispositivos mecánicos.

La industria del automóvil, desde hace unos años hasta la actualidad, está incorporando a los vehículos cada vez más aplicaciones y sensores, cuyo objetivo es aumentar las prestaciones y la seguridad de sus pasajeros. El límite de esta tecnología no está aún definido, pero para que siga evolucionando y desarrollándose es totalmente necesaria la comunicación con el vehículo.

Lo que más me llamó la atención de este trabajo fue la posibilidad de no solo tomar datos del vehículo para que un conductor pueda conocer dónde y cómo conduce mejor o peor, sino también cómo puede adaptar o mejorar su conducción para un ahorro económico y mejora del medio ambiente. También, me interesó la posibilidad de que el sistema que se va a desarrollar en este TFG pueda servir como base para el desarrollo de un sistema mucho más avanzado que pueda ser utilizado por empresas con una flota de vehículos para la comparación de rutas en busca de la óptima o la comparación de la conducción de los diferentes trabajadores sobre una misma ruta. Además, el sistema puede utilizarse para comprobar sobre un mismo trayecto o ruta cómo se desempeñan los vehículos diesel, de gasolina, híbridos y eléctricos en términos de consumo, cómo afecta la utilización de distintos tipos de cajas de cambio, o la influencia de ayudas como el control de crucero o la conducción autónoma.

Por otro lado, me gustó la libertad que, en cierta manera, me ofreció este trabajo a la hora de desarrollar el sistema, pudiendo optar por varias opciones y herramientas y elegir la que más me gustara o en la que mejor me desenvolviera.

Los objetivos de este proyecto son los siguientes:

- Desarrollar un software para monitorizar los parámetros más importantes de un vehículo.
- Monitorizar parámetros que afecten al consumo y mantenimiento del vehículo, tales como, la velocidad, las revoluciones, la temperatura o el consumo de combustible.
- Hacer uso de un dispositivo OBD2 para la recogida de la información y un dispositivo móvil Android.
- El software debe ofrecer valoraciones cualitativas y cuantitativas sobre la eficiencia y el estilo de conducción.

- Se deberán ofrecer resultados instantáneos y globales que ayuden a la conducción ecológica/económica.
- Vincular los datos recogidos con información sobre la ubicación espacial (vía GPS) y temporal para referencias futuras.
- Identificar, mediante los datos recogidos, tramos de carretera problemáticos.
- Recopilar datos sobre recorridos y rutas con la geolocalización para referencias futuras.
- Investigar acerca de posibles librerías disponibles para Android para la comunicación con el dispositivo de OBD.

1.4. Soluciones disponibles

En el mercado existen aplicaciones, mayoritariamente para teléfonos móviles, que permiten la comunicación con el vehículo mediante un dispositivo OBD2 y monitorización en tiempo real de la información recogida.

Por supuesto, todas las soluciones ya existentes requieren que el usuario posea un dispositivo OBD2 para conectarlo al vehículo.

- **Torque:** es una aplicación para teléfonos Android desarrollada por Ian Hawkins. Torque es capaz de monitorizar en tiempo real muchos componentes del vehículo como son la aceleración, la velocidad, la posición de los pedales, las temperaturas del motor y del refrigerante, el consumo, etc. Además, permite comprobar y confirmar códigos de error que se pueden mostrar en el cuadro de mandos o comprobar el estado del sistema eléctrico del vehículo. También es capaz de generar gráficos con los datos obtenidos en diferentes momentos de tiempo.

Ilustración 1 – Logotipo de la aplicación Torque

Ilustración 2 – Captura de la interfaz de Torque

- **DashCommand:** es otro ejemplo de aplicación muy similar a Torque. Esta aplicación desarrollada por Palmer Perfomance Engineering permite la monitorización en tiempo real de muchos parámetros del vehículo, control de las luces de error o del consumo del vehículo.

Ilustración 3 – Logotipo de la aplicación DashCommand

Ilustración 4 – Capturas de la interfaz de DashCommand

1.5. Estructura del documento

Tras los apartados introductorios definidos más arriba, el documento continúa con un segundo apartado dedicado a la gestión y planificación del proyecto en cuento a costes y tiempos estimados. Además, se explicará la metodología de desarrollo empleada en el proyecto.

En el tercer apartado se expondrán las distintas herramientas, tanto hardware como software, utilizadas para el desarrollo del proyecto, así como algunas de sus principales características.

El cuarto apartado estará dedicado a las distintas fases o etapas en las que se ha dividido el desarrollo, desde el diseño, la implementación, las diversas pruebas realizadas, hasta la obtención de los resultados finales. Se explicarán los objetivos a cumplir en cada fase, los problemas encontrados, las soluciones empleadas, así como las herramientas utilizadas.

Un quinto apartado nos servirá para exponer los resultados obtenidos tras finalizar el proyecto, las conclusiones y se tratarán posibles mejoras y trabajos futuros.

En el último apartado de la memoria se expondrá los anexos con los manuales de las aplicaciones desarrolladas y la bibliografía básica utilizada para realizar el trabajo.

2. GESTIÓN Y PLANIFICACIÓN

2.1. Metodología utilizada

Para este proyecto se ha utilizado una metodología de desarrollo ágil conocida como SCRUM [1]. A diferencia de otras metodologías como en cascada o con ejecución secuencial, SCRUM se caracteriza por la división de la fase de desarrollo en hitos o iteraciones

En SCRUM se realizan entregas parciales y regulares del producto final cada cierto período de tiempo y es especialmente útil para proyectos que necesitan

obtener resultados lo más pronto posible, donde los requisitos pueden variar a lo largo del desarrollo y donde la flexibilidad y la productividad son factores imprescindibles.

En cada hito o entrega definida se debe proporcionar un resultado lo más completo posible para pasar a una nueva iteración. El proceso de aplicación de esta metodología parte de un conjunto de requisitos y objetivos a los que el cliente (Product Owner) les da una determinada prioridad y quedan repartidos en iteraciones y entregas. En cada fase de desarrollo se incluyen procesos de planificación, análisis, diseño, implementación, revisión y documentación.

Roles que pueden extraerse de SCRUM aplicados a este proyecto:

- El **equipo de desarrollo**: es el equipo que tiene la responsabilidad de entregar el producto. Este rol es adoptado por la persona que presenta el TFG.
- **Stakeholders**: son las personas a las cuales el proyecto producirá beneficio. En este caso representan a las personas que hacen uso de la aplicación generada para mejorar su conducción hacia una conducción eficiente.
- **Product Owner**: es un tipo de cliente que se encarga de que el equipo de desarrollo trabaje de manera adecuada y también de priorizar requisitos u objetivos. Es un rol que puede ser adoptado por el tutor del TFG.
- **ScrumMaster** o **facilitador**: se encarga de que el equipo de desarrollo conozca los principios teóricos y prácticos de SCRUM con el objetivo de que lo utilicen en los procesos de toma de decisiones. Es otro rol que puede ser adoptado nuevamente por el tutor del TFG.

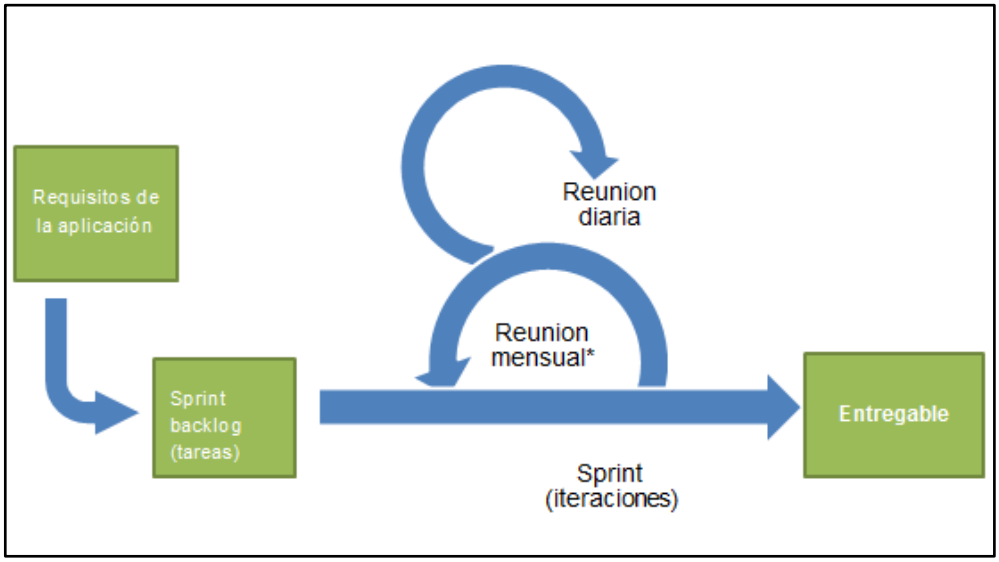


Ilustración 5 – Esquema del proceso SCRUM

2.2. Planificación del proyecto

En este TFG el sprint, es decir, el período de tiempo entre cada entrega, se ha definido de manera mensual. Cada mes se han presentado los avances logrados y los resultados obtenidos al tutor del trabajo.

Los objetivos y requisitos de este TFG definidos en el capítulo 1, establecieron las siguientes entregas o iteraciones:

- Mes 1: investigación del sistema OBD2, funcionamiento de la comunicación, estudio acerca de la conducción eficiente y su aplicación al proyecto en desarrollo, posibles herramientas o librerías software disponibles.
- Mes 2: desarrollo e implementación de una aplicación móvil que estableciera comunicación con el dispositivo OBD2 mediante Bluetooth y realizara consultas sobre los parámetros del vehículo.
- Mes 3: geolocalización del vehículo y almacenamiento de los datos recabados del mismo.
- Mes 4: procesamiento de los datos recogidos del vehículo mediante el desarrollo de una aplicación que muestre información cuantitativa y cualitativa de los mismos.

- Mes 5: aplicación de los principios de conducción eficiente al proyecto.
- Mes 6: pruebas y obtención de los resultados finales.

El proyecto se inició el día 24 de enero de 2018, tras una primera reunión con el tutor se dio por comenzada la primera iteración del proyecto. A lo largo del desarrollo del proyecto se encontraron diversos problemas que derivaron en retardos en las entregas. El principal problema fue la imposibilidad de poder probar las aplicaciones desarrolladas en mi vehículo. Debido a la antigüedad de éste, la comunicación OBD2 no funcionaba correctamente y no se podían obtener resultados, por lo que se necesitó la ayuda de vehículos de terceros para probar el funcionamiento del sistema. Este problema derivó en atrasos en las iteraciones donde era necesario establecer comunicación con el dispositivo OBD2 para realizar pruebas de funcionamiento. La iteración número 2 fue la que mayormente se vio afectada por el problema, por lo que se decidió a lo largo de la misma ampliarla un mes más.

A continuación, se muestra un diagrama de Gantt del proyecto donde puede verse gráficamente el desarrollo de cada una de las iteraciones y tareas y el solapamiento entre ellas.

Iteraciones y tareas	Inicio	Fin
Proyecto	24/01/2018	31/08/2018
Iteración 1	24/01/2018	24/02/2018
Definición de requisitos	24/01/2018	04/02/2018
Estudo del sistema OBD2	04/02/2018	24/02/2018
Estudio sobre conducción eficiente	04/02/2018	14/02/2018
Conducción eficiente aplicada al proyecto	14/02/2018	24/02/2018
Estudio de herramientas disponibles	14/02/2018	24/02/2018
Iteración 2	24/02/2018	24/04/2018
Diseño de la aplicación	24/02/2018	04/03/2018
Conectividad Bluetooth	04/03/2018	24/03/2018
Comunicación OBD2	14/03/2018	24/04/2018
Pruebas	04/03/2018	24/04/2018
Iteración 3	24/04/2018	24/05/2018
Corrección de las respuestas erróneas	24/04/2018	04/05/2018
Geolocalización	04/05/2018	14/05/2018
Almacenamiento de datos	14/05/2018	24/05/2018
Pruebas	24/04/2018	24/05/2018
Iteración 4	24/05/2018	24/07/2018
Retardos en la comunicación	24/05/2018	14/06/2018
Diseño de la aplicación	14/06/2018	24/06/2018
Creación de gráficas con los datos	24/06/2018	14/07/2018
Mostrar mapa en la aplicación	14/07/2018	24/07/2018
Trazar el trayecto en la aplicación	04/07/2018	24/07/2018
Pruebas	24/05/2018	24/07/2018
Iteración 5	24/07/2018	24/08/2018
Cambios y mejoras en la interaz	24/07/2018	04/08/2018
Rutas en función del esfuerzo/consumo	04/08/2018	24/08/2018
Iteración 6	24/08/2018	31/08/2018
Obtención resultados finales	24/08/2018	31/08/2018

Ilustración 6 – Calendario de la planificación

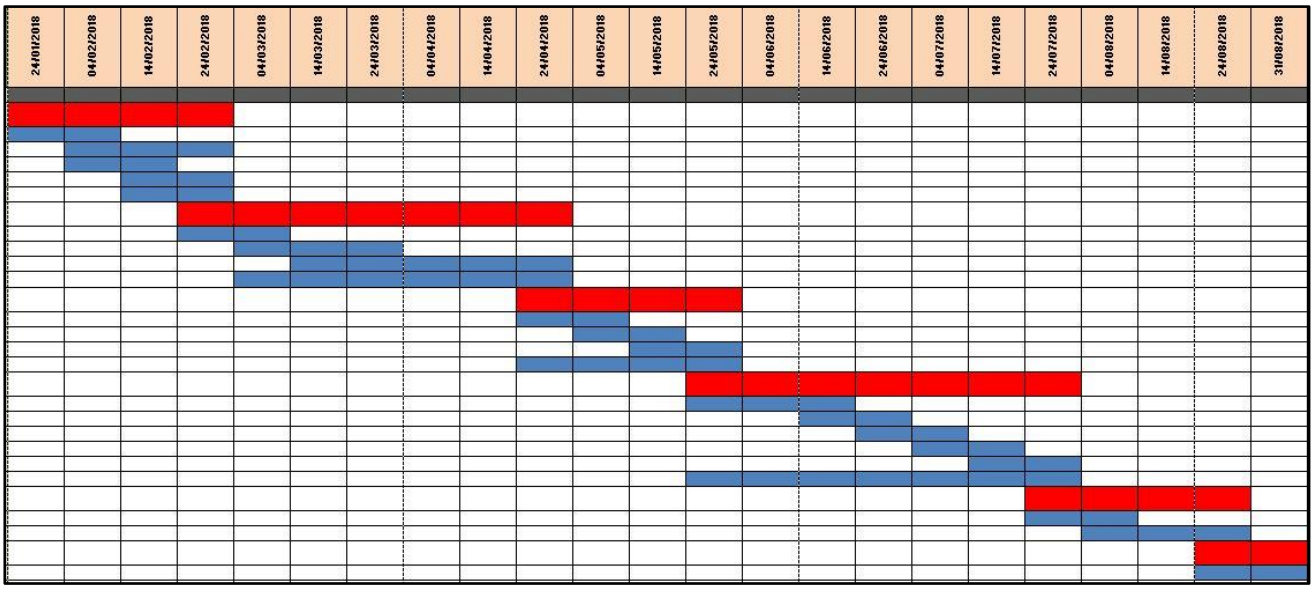


Ilustración 7 – Diagrama de Gantt

A continuación se presenta un diagrama tipo Burndown muy característico y empleado en la metodología SCRUM. El diagrama representa la cantidad de trabajo restante hasta la finalización del proyecto pasando por las distintas iteraciones, es decir, representa cuánto trabajo falta por realizar para finalizar el proyecto en cada instante de tiempo. El eje de coordenadas Y del diagrama representa la cantidad de trabajo restante en porcentaje, mientras que el eje X representa el transcurso del tiempo.

El diagrama Burndown asociado al desarrollo de este proyecto es el siguiente:

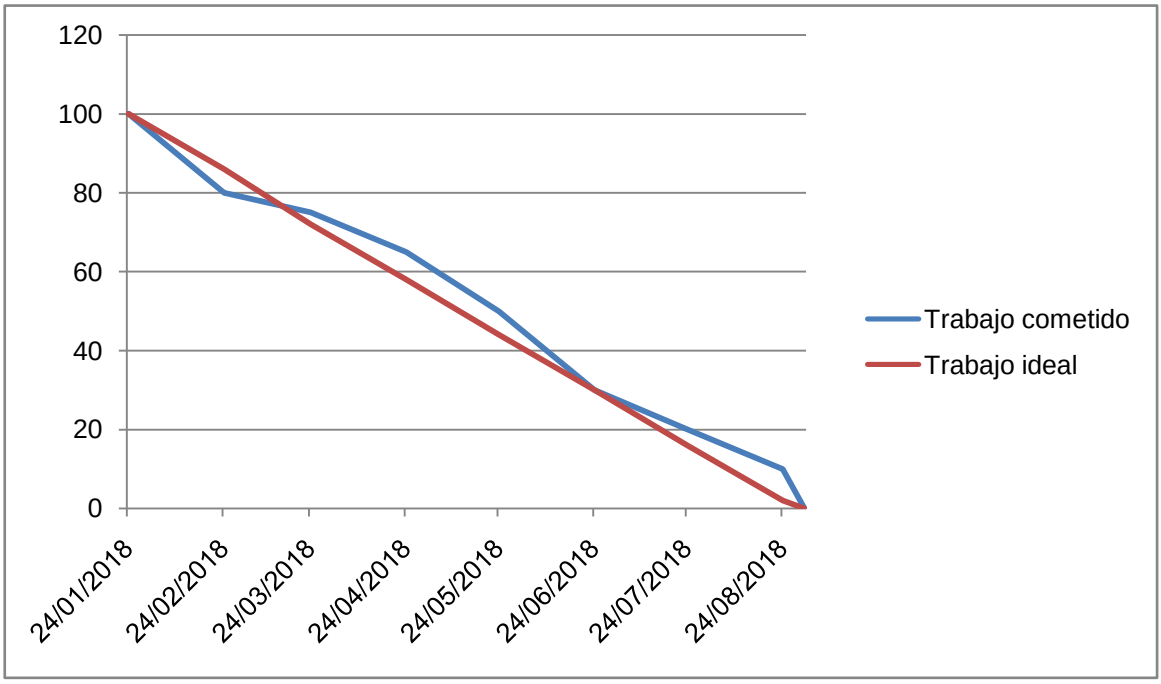


Ilustración 8 – Diagrama de Burndown

2.3. Análisis de costes

Para el desarrollo de este proyecto serán necesarios los siguientes recursos (especificados con mayor detalle en el apartado 3.1):

- Un ordenador para la implementación de la aplicación móvil. Los requisitos del ordenador a utilizar no tienen por qué ser excesivamente altos. En este caso se ha hecho uso de un ordenador portátil con precio en 2014 de 490€.
- Un smartphone Android para probar la aplicación en desarrollo con versión Ice Cream Sandwich 4.0.3 – 4.0.4 o superior. En este caso se ha

hecho uso del Smartphone BQ Aquaris U Plus con precio de 190€ en 2017.

- Un dispositivo de diagnóstico OBD2 para conectar al coche y establecer comunicación con la aplicación a desarrollar. En este caso se ha hecho uso del dispositivo ELM327 Mini con un coste de 8,99€ en la página web de Amazon.
- El vehículo empleado para la realización de las pruebas de la aplicación y la conexión del dispositivo OBD2 fue un Nissan Yuke con matrícula de 2011 con un precio de compra cercano a los 18.000€.

Al estar en propiedad de algunos recursos de antemano calcularemos la amortización para el período total del proyecto, 217 días. La Ley del Impuesto sobre Sociedades publicada en el Boletín Oficial del Estado estipula que el período de amortización para el hardware es de 8 años, en total 2.920 días. Mientras que el período de amortización para los elementos de transporte externo como los vehículos son de 14 años, en total 5.110 días.

Herramienta	Coste total	Período de amortización	Amortizado/día	Amortizado en total (217 días)
Ordenador portátil	490 €	2920 días	0,18 €	39,06 €
Smartphone	190 €	2920 días	0,07 €	15,19€
Dispositivo OBD2	8,99 €	-	-	8,99 €
Vehículo	18.000 €	5110 días	3,53 €	766,01€
Costes indirectos (agua, luz, combustible, etc.)	165,85€	-	-	165,85 €
Total				995,10 €

Tabla 1 – Tabla de costes de las herramientas hardware y el vehículo

En cuanto al coste de las herramientas software utilizadas en este proyecto, se detalla en la siguiente tabla:

Herramienta	Coste
Windows 10 Home (incluído en el ordenador portátil)	0 €
Android Studio	0 €
Visual Paradigm (versión estudiante)	0 €
Balsamiq Mockups (prueba gratuita 30 días)	0 €

Total	0 €

Tabla 2 – Tabla de costes de las herramientas software

Por lo tanto, el coste total del desarrollo del proyecto es de 995,10€ según las herramientas y materiales utilizados en este caso. El coste total podría variar en función de los elementos explicados anteriormente, tales como el vehículo, el dispositivo móvil utilizado, etc.

3. HERRAMIENTAS HARDWARE, SOFTWARE Y TECNOLOGÍAS UTILIZADAS

3.1. Herramientas hardware

3.1.1. Dispositivo OBD2 ELM327 Mini

El ELM327 es un dispositivo OBD2 fabricado por la empresa canadiense ELM Electronics que produce circuitos integrados y microchips desde 1998. En 2005, desarrollaron el ELM327, dispositivo muy famoso y muy utilizado hasta la fecha por su bajo coste y gran fiabilidad.

El dispositivo hace uso de la conexión Bluetooth para la comunicación con el mismo. Implementa la mayor parte de los protocolos de comunicación OBD y soporta gran cantidad de comandos para las consultas al vehículo.

Ilustración 9 – Dispositivo ELM327

3.1.2. Ordenador portátil Toshiba Satellite L50D-P-18F

Es el ordenador que se ha empleado para la implementación del proyecto, algunas de sus características son:

Componente	Características
Procesador	AMD A8-6410 APU 2.00Ghz
Memoria RAM	8 GB DDR3L 1600 MHz
Almacenamiento	Disco duro 1TB Serial ATA Velocidad de rotación: 5.400 rpm
Tarjeta gráfica	AMD Radeon R5
Sistema operativo	Windows 10
Conexiones	3 USB (2x USB 3.0 1x USB 2.0) Ranura para tarjetas SD Salida HDMI Salida VGA 1x Entrada/Salida de audio 1x Micrófono integrado RJ-45
Comunicación	Internet 10/100 Mbps Wireless LAN 802.11
Batería	Litio 4 celdas de 3.000mAh
Dimensiones	380 x 259.9 x 23.5 mm
Peso	2.26 Kg
Precio (2014)	490€

Tabla 3 – Características del ordenador portátil

Ilustración 10 – Toshiba Satellite L50D-P-18F

3.1.3. Smartphone BQ Aquaris U Plus

Es el dispositivo móvil utilizado para instalar las aplicaciones desarrolladas y establecer la comunicación Bluetooth con el dispositivo OBD2. Algunas de sus características:

Componente	Características
Procesador	Qualcomm Snapdragon 430 – 8 núcleos
Memoria RAM	2GB
Almacenamiento	8GB Espacio para tarjeta SD de hasta 256GB
GPU	Adreno 505
Versión Android	Oreo 8.0.0
Cámara	Trasera 16 megapíxeles Frontal 5 megapíxeles
Conectividad	LTE, Bluetooth 4.2, Wi-Fi 802.11b/g/n
Otros	Sensor de huellas dactilares
Batería	3.080mAh
Dimensiones	144 x 70,5 x 7,8 mm
Peso	142 gramos
Precio (2017)	190€

Tabla 4 – Características del smartphone

Ilustración 11 – BQ Aquaris U Plus

3.2. Balsamiq Mockups

Balsamiq Mockups es un programa de escritorio disponible para los sistemas operativos Windows y macOS que permite al usuario crear de manera simple pero

muy detallada, diseños para interfaces de usuario de programas de escritorio, páginas web o aplicaciones para móviles. Dispone de una versión de prueba totalmente gratuita de 30 días con toda la funcionalidad incluida.

El funcionamiento del programa es sencillo e intuitivo, basándose en el sistema "Drag and Drop". Los elementos que queramos añadir a nuestro diseño deben ser localizados en la barra de tareas y arrastrados hasta la zona de trabajo, conocida como mockup.

Cada elemento posee multitud de opciones de personalización como el tamaño, la posición, la orientación, el color, etc.

El diseño final puede guardarse en nuestro ordenador en distintos formatos de imagen. Si vamos a editar el diseño a posteriori, podemos guardarlo en formato bmpr para poder abrirlo más tarde con Balsamiq y seguir editando.

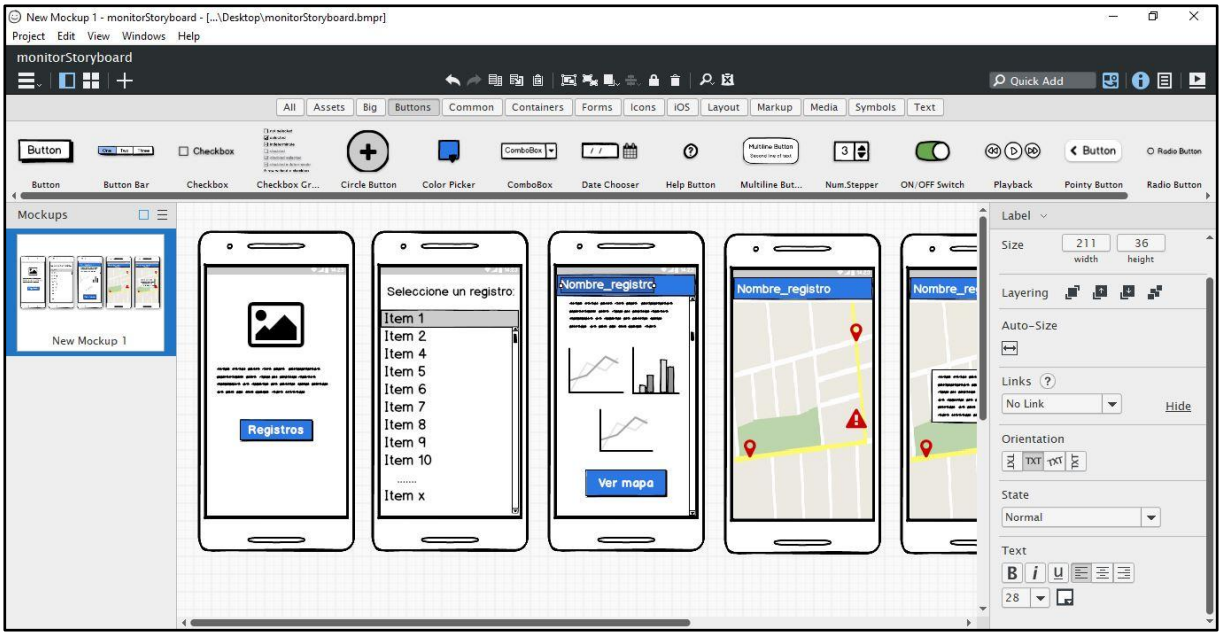


Ilustración 12 – Captura de la interfaz de Balsamiq Mockups

3.3. Visual Paradigm

Visual Paradigm es un programa de escritorio multiplataforma con una licencia gratuita y otra comercial, muy útil para la ingeniería del software. Dispone de múltiples herramientas que soportan el ciclo de vida completo del proceso de

desarrollo de software, desde la planificación, implementación o generación de código fuente, poniendo gran énfasis en la fase análisis y diseño del software.

Para ello Visual Paradigm nos ofrece una serie de diagramas de modelado UML como los diagramas de casos de uso, de clases, de actividad, de comunicación, de secuencia o de objetos.

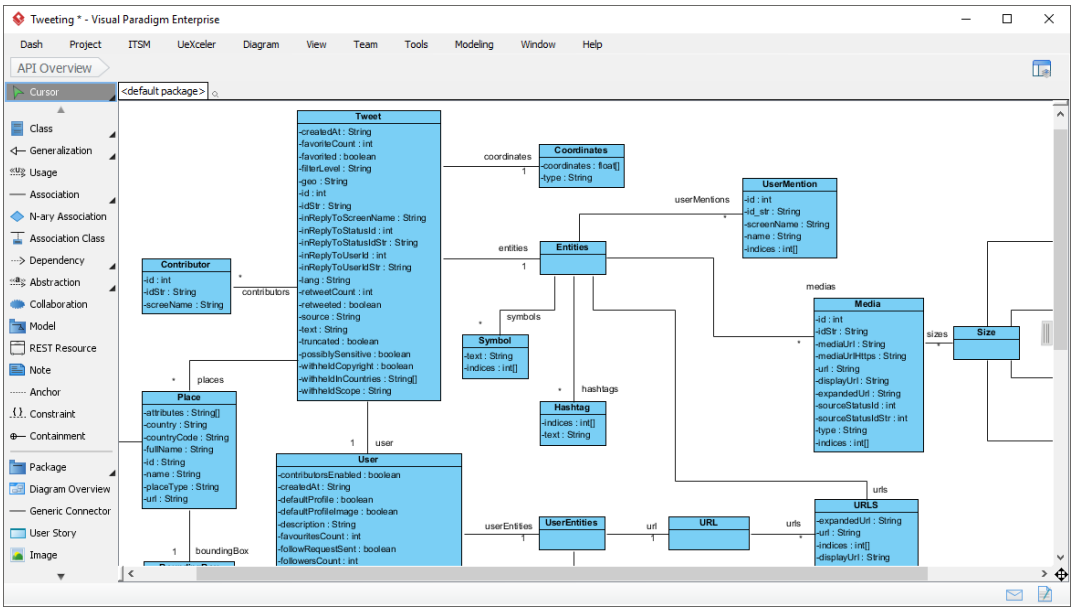


Ilustración 13 – Captura de la interfaz de Visual Paradigm

3.4. Android Studio

El entorno con el que se ha desarrollado el proyecto ha sido Android Studio en su versión 3.0. Este IDE propiedad de Google fue lanzado en 2014 y sustituyó a Eclipse como IDE oficial para el desarrollo de aplicaciones para Android.

Está basado en el software IntelliJ IDEA de JetBrains y se distribuye de forma gratuita bajo licencia Apache 2.0.

Ilustración 14 – Logo de Android Studio

Algunas de las características más importantes que Android Studio posee:

- Un sistema de compilación basado en Gradle.
- Un emulador rápido con múltiples funciones.
- Entorno unificado para realizar desarrollos para todos los dispositivos Android.
- Instant Run para aplicar cambios mientras una app se ejecuta sin la necesidad de compilar un nuevo APK.
- Integración de plantillas de código y GitHub para ayudar a compilar funciones comunes de las apps e importar ejemplos de código.
- El SDK (Software Development Kit) de Android es un conjunto de herramientas de desarrollo que posee Android Studio, compuesto por un depurador, un simulador de teléfono, documentación, ejemplos de código y tutoriales.

Cualquier proyecto de Android está dividido en cuatro partes bien diferenciadas:

- El archivo **AndroidManifest.xml**: es un archivo donde especificaremos las principales opciones de configuración de la aplicación que vamos a generar, tales como los permisos que requerirá del teléfono, el icono o el nombre que poseerá al instalarla en dispositivo móvil.

- El directorio con los archivos de código fuente en formato java donde el desarrollador implementará las funcionalidades de a aplicación.

- El directorio **res**: incluye archivos en formato XML con los que definiremos el diseño de las distintas pantallas de las aplicaciones. Como pueden ser la colocación de los distintos elementos en pantalla, las distancias entre ellos y los bordes de la pantalla, la configuración de los atributos de cada elemento, colores, estilos, fuentes, etc.

- El **build.gradle**: archivo donde se definirán dependencias de la aplicación o se enlazarán repositorios remotos necesarios para incluir librerías de software a nuestro proyecto.

3.4.1. Java

Java es el lenguaje de programación utilizado por Android Studio, es un lenguaje de propósito general, concurrente y orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.

Java es uno de los lenguajes de programación más populares en uso. Fue originalmente desarrollado por James Gosling y publicado en 1996. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos.

Es un lenguaje que se caracteriza por su fácil aprendizaje gracias a su sintaxis sencilla y a la gran comunidad de desarrolladores que posee, que facilita la obtención de documentación y la resolución de prácticamente cualquier duda que le pueda surgir a la persona que se inicie en su aprendizaje.

4. DESARROLLO DEL PROYECTO

4.1. Reunión inicial

En la primera reunión con el tutor del Trabajo de Fin de Grado se concretaron los pasos iniciales para comenzar el proyecto, el tutor comentó la posibilidad de buscar librerías de software para el sistema OBD2 y me proporcionó un dispositivo OBD2 para probar en mis vehículos el funcionamiento del sistema. El dispositivo en concreto fue un Konnwei KW903 con conexión Wifi. El dispositivo además, proporcionaba un CD con la APK de la aplicación Torque versión Pro.

Ilustración 15 – Konnwei KW903

En las pruebas realizadas con el dispositivo y la aplicación Torque no conseguí establecer comunicación con ninguno de mis vehículos, un Volvo S40 del año 2000 y un Opel Frontera del año 1997. A través de la aplicación, Torque era capaz de establecer comunicación Wifi con el dispositivo OBD2, pero éste no era capaz de establecer comunicación con la ECU, el ordenador de a bordo de los vehículos, por lo tanto la aplicación no recibía los valores de los parámetros del vehículo consultados.

Tras comentar lo sucedido con el tutor del TFG se decidió cambiar de dispositivo OBD2 a uno con conexión Bluetooth para ver las posibilidades que podían obtenerse con el nuevo. Se cambió el Konnwei KW903 por un ELM327 Mini. Tras probar el nuevo dispositivo se volvió a encontrar el mismo problema de comunicación con los vehículos. Así que finalmente se decidió continuar con el proyecto haciendo uso del dispositivo ELM327 en vehículos de terceros para realizar las pruebas pertinentes durante el transcurso del proyecto.

4.2. Iteración 1 - Definición de requisitos, estudio del sistema OBD2 y conducción eficiente.

4.2.1. Definición de requisitos

Los requisitos de un software son la descripción completa del sistema que se va a desarrollar y de las interacciones que los usuarios tendrán con él. Además

incluyen restricciones en el diseño, normas y estándares de calidad. Distinguimos dos tipos de requisitos.

Los requisitos funcionales describen las interacciones con el software y los casos de uso. Los requisitos funcionales para el sistema que vamos a diseñar son:

- Capturar parámetros de conducción de un vehículo cuando esté en marcha.
- Almacenar los valores recogidos para un análisis posterior.
- Poder acceder a todos los registros de trayectos guardados por la aplicación.
- Establecer comunicación con el ordenador de a bordo del vehículo a través de un dispositivo OBD2.
- Generar información cualitativa y cuantitativa con los datos recabados del trayecto del vehículo.
- Trazar en un mapa el trayecto realizado para mostrar el recorrido por las distintas calles y carreteras.

Los requisitos no funcionales o complementarios imponen restricciones y normas al diseño y la implementación. Los requisitos no funcionales con los que cuenta nuestro sistema son:

- Implementar la parte del sistema dedicada a la captura de datos lo más simple y sencilla posible para evitar que el usuario tenga que interactuar de manera innecesaria con la aplicación mientras conduce.
- La única acción que el usuario tendrá que realizar una vez haya establecido conexión con el dispositivo OBD será la de finalizar la comunicación con el mismo para evitar distracciones en la conducción que pudieran poner en peligro la seguridad del usuario.
- Avisar al usuario si la conexión Bluetooth con el dispositivo OBD cae durante la ejecución de la aplicación, de esta forma el usuario sabrá si la comunicación ha finalizado por algún motivo ajeno a él y podrá reiniciarla para seguir recabando datos.

- Realizar un mapeado de colores en el trazado de la ruta sobre un mapa para distinguir zonas con mayor o menor esfuerzo y consumo por parte del vehículo.

Por último, hablaremos acerca de los requisitos del terminal del usuario final. Como requisito hardware el usuario necesitará estar en posesión de un Smartphone Android con versión Ice Cream Sandwich 4.0.3 – 4.0.4 o superior. Aunque la aplicación está compilada para la versión Oreo 8.0.0 o SDK nivel 26 permite compatibilidad con dispositivos más antiguos hasta la versión 15 de SDK, con el objetivo de que el mayor número de dispositivos móviles puedan hacer uso de la aplicación.

Code name	Version	API level
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
KitKat	4.4 - 4.4.4	API level 19
Jelly Bean	4.3.x	API level 18
Jelly Bean	4.2.x	API level 17
Jelly Bean	4.1.x	API level 16
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15, NDK 8
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14, NDK 7
Honeycomb	3.2.x	API level 13

Ilustración 16 – Versiones de Android y sus SDK equivalentes

4.2.2. Historias de usuario

Historia 1: conexión Bluetooth:

- Como: usuario final
- Quiero: poder establecer conexión Bluetooth con el dispositivo OBD2
- Para: consultar los valores de los parámetros de conducción

Historia 2: geolocalización

- Como: usuario final
- Quiero: recopilar datos sobre la localización del vehículo durante el trayecto
- Para: trazar recorridos sobre un mapa

Historia 3: almacenar los datos

- Como: usuario final
- Quiero: poder almacenar los datos recabados del vehículo
- Para: un posterior análisis y estudio de los mismos

Historia 4: listar registros guardados

- Como: usuario final
- Quiero: poder listar los registros de los trayectos realizados
- Para: elegir uno de ellos y visualizar los datos que contiene

Historia 5: información cuantitativa

- Como: usuario final
- Quiero: poder visualizar los valores de los parámetros de conducción capturados
- Para: poder ver la evolución de los mismos a lo largo del trayecto realizado

Historia 6: información cualitativa

- Como: usuario final
- Quiero: poder ver el trayecto realizado sobre un mapa y comparar los valores de los parámetros de conducción
- Para: analizar la variación de esfuerzo y consumo del vehículo en diferentes zonas del recorrido

4.2.3. Utilización y programación del sistema OBD (On Board Diagnostics)

En los siguientes apartados se explicará qué es el sistema OBD, cómo funciona y cuáles son sus componentes.

4.2.3.1. Definición e historia

Como sus siglas indican, el OBD es un sistema de diagnóstico a bordo utilizado en automóviles y camiones. Una de las funciones más conocidas es la de notificar si sucede algún fallo en alguna pieza del motor mediante luces de advertencia en el cuadro de mandos.

El sistema OBD nació en Estados Unidos a finales de la década de los 80 de la mano de un organismo del Estado de California, el **CARB** o California Air Resources Board, con el objetivo de monitorizar ciertos componentes de los vehículos para controlar las emisiones de gases, reducir la contaminación y garantizar el correcto funcionamiento de los vehículos. Se implementaron sistemas de encendido y alimentación controlada de combustible con sensores que medían las prestaciones del motor y ajustaban los sistemas para una mínima contaminación, estos sistemas también permitían una cierta ayuda a la hora de reparar los vehículos notificando qué componente producía el error.

A comienzos de los 90, la legislación norteamericana estableció a los fabricantes de vehículos que era obligatorio que se incorporaran estos sistemas para la monitorización de las emisiones, nace el **OBD I**. Sin embargo, este primer sistema fue un relativo fracaso, pues no era muy efectivo a la hora de monitorizar algunos componentes del vehículo. En 1996 se crea una versión mejorada, el **OBD II**, que sí cumplía finalmente con el objetivo de monitorizar y controlar de manera efectiva las emisiones.

A partir de ese momento y hasta la actualidad, el sistema OBD ha evolucionado enormemente. Es muy utilizado por los talleres mecánicos para la detección de averías en el motor, ya que el sistema puede almacenar un registro del fallo y en las condiciones en las que se produjo. Por otro lado, el sistema también puede ser utilizado para la monitorización de cualquier parámetro del estado del vehículo, velocidad, revoluciones del motor, temperaturas del motor o de componentes, etc.

En Europa y en Japón no existe el sistema OBD II como tal, sino una variación del mismo conocida en Europa como **EOBD** o Sistema de Diagnóstico a Bordo Europeo y en Japón **JOBD** o Sistema de Diagnóstico a Bordo Japonés, aplicado a los vehículos fabricados y vendidos allí. No obstante, la gran mayoría de vehículos fabricados en los últimos 15 años incorporan una interfaz ODB, independientemente del país de origen o de la zona de comercialización.

4.2.3.2. Componentes del sistema OBD

Los componentes del sistema de diagnóstico de un vehículo son los siguientes:

- La ECU o la Unidad de Control del Motor: conocida como el ordenador del vehículo. Es un componente electrónico capaz de monitorizar el motor y la combustión a través de distintos sensores.
- Los transductores: son los sistemas encargados de la transferencia de datos desde los sensores a la ECU o viceversa.
- Las luces indicadoras en el panel de instrumentos del vehículo: son las luces que se encienden en la zona del velocímetro del vehículo. Indican posibles fallos o averías del motor y son mostradas gracias a la ECU.

Ilustración 17 – Panel de instrumentos

- El conector de diagnosis DLC o conector OBD: sirve de interfaz entre los dispositivos de diagnóstico y la ECU. A través de estos conectores podemos acoplar dispositivos que permiten comunicarse con la ECU y obtener los valores de los sensores de los componentes del vehículo.

Ilustración 18 – Conector OBD o DLC

4.2.3.3. Características del conector OBD

Inicialmente cada fabricante de vehículos tenía su propio sistema de diagnóstico y códigos de error. A partir de 1998, la **SAE** (Sociedad de Ingenieros de la Automoción) definió un conector OBD estándar de 16 pines y un conjunto de códigos de diagnóstico.

A partir de ese momento todo conector del sistema OBD II debe cumplir una serie de especificaciones basadas en la normativa **ISO 15031-3:2016**. Dicha normativa obliga a los fabricantes a situar el conector OBD cerca del asiento del conductor, a diferencia de los sistemas anteriores en los cuales el conector se encontraba en el compartimento del motor.

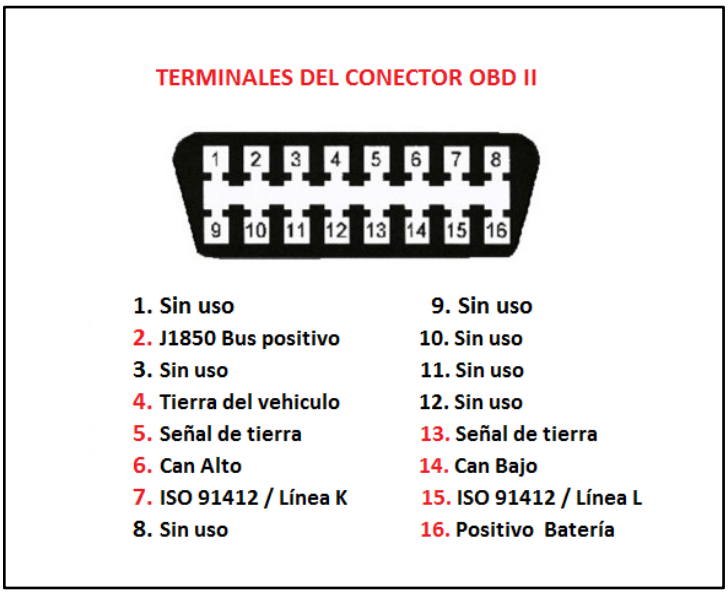


Ilustración 19 – Conector OBD y sus pines de conexión

4.2.3.4. Dispositivos de diagnóstico

Los dispositivos de diagnóstico son los dispositivos que se acoplan al conector OBD o DLC para obtener la información del vehículo. Podemos encontrar desde dispositivos muy sofisticados utilizados en talleres mecánicos, como los lectores de fallas o averías, los cuales son un dispositivo con todo incluido desde cables de conexión a pantalla para la visualización de datos. Y luego están los dispositivos que hacen de intermediario entre el vehículo y el dispositivo que analiza los datos obtenidos, es el ejemplo del dispositivo utilizado en este proyecto. Estos dispositivos son más asequibles, se acoplan al conector OBD y permiten, mediante una comunicación vía Bluetooth o Wifi, poder enviar comandos a la ECU de un vehículo desde un ordenador o hasta desde un smartphone.

Ilustración 20 – Dispositivos de diagnosis, a la izquierda el utilizado en este proyecto. A la derecha dispositivo utilizado en tallares mecánicos

4.2.3.5. Protocolos de comunicación

Existen múltiples protocolos de comunicación para el sistema OBD II y su variante europea y japonesa. Las diferencias entre los protocolos están en el sistema utilizado por el vehículo y en los escáneres usados para la obtención de la información de los sensores. Por lo general, los protocolos están distribuidos de la siguiente forma: los vehículos europeos y asiáticos utilizan el protocolo **ISO 9141**, los vehículos fabricados por General Motors utilizan el protocolo de comunicación **SAE J1850 VPW** y los vehículos fabricados por Ford usan el **SAE J1850 PWM**.

Existen muchos otros protocolos de comunicación, pero a pesar de ello, éstos se diferencian únicamente en la conexión eléctrica del sistema OBD y en pequeños detalles en la comunicación. Para que todo sea más sencillo y genérico en todos los países, se utiliza un protocolo estándar de comunicación que va ligado a los

protocolos vistos anteriormente, el **SAE J1979.** Este protocolo define un conjunto de comandos que permiten comunicarse de manera única con cada componente del vehículo.[2]

4.2.3.6. Comunicación y transferencia de datos

En la comunicación y obtención de datos destacan tres agentes principales. Por un lado tenemos el vehículo del que deseamos extraer la información y al que acoplaremos, en su puerto DLC u OBD, el dispositivo de diagnosis OBD. El segundo agente es el dispositivo de diagnosis, necesario para establecer la comunicación con la ECU del vehículo. Por último, tenemos el sistema que se comunica con el dispositivo de diagnosis y solicita consultas, este sistema normalmente es una aplicación móvil que establece un canal de comunicación con el dispositivo OBD2 mediante Bluetooth o Wifi.

Una vez establecida la comunicación, el dispositivo OBD2 nunca enviará datos a la aplicación automáticamente. Es necesario solicitar los datos desde la aplicación. Estas solicitudes o consultas se realizan mediante una serie de comandos alfanuméricos que el dispositivo OBD reenvía a la ECU del vehículo.

La comunicación con la ECU a través del sistema OBD se caracteriza principalmente por la restricción de envío de un único comando a la vez, es decir, para comunicarnos con el dispositivo sólo podemos enviar un comando en un momento dado y no podremos enviar otro hasta no haber recibido la respuesta del comando anterior.

4.2.3.7. SAE J1979 y comandos PID

Para obtener la información del vehículo y sus componentes es necesario que los dispositivos de diagnóstico envíen una serie de comandos indicados a la ECU del vehículo. Cada componente del motor tiene un comando único asociado para la petición de datos, estos comandos están definidos por el estándar que hemos explicado anteriormente, el **SAE J1979**. Los comandos son conocidos como PIDS o Parameter ID y están compuestos por cuatro dígitos hexadecimales.

Los dos primeros dígitos indican el modo de trabajo que se va a utilizar, existen los siguientes modos 01, 02, 03, 04, 05, 06, 07, 08, 09 y 0A. Nosotros únicamente

nos centraremos en el **modo 01**, que es el encargado de obtener los datos actuales, es decir, en tiempo real de los componentes. Además, el fabricante del vehículo no está obligado a implementar todos los modos de operación o códigos y tiene la libertad de añadir los suyos propios.

Los dos últimos dígitos hacen referencia al componente del vehículo del que se desea obtener información. Por ejemplo, si queremos conocer las revoluciones del motor en un instante dado, el PID asociado es 010C, 01 para indicar que queremos el valor actual y 0C para referirnos a las revoluciones. De igual forma, para la velocidad del vehículo en un momento dado enviamos 010D, para la temperatura del refrigerante del vehículo es 0105, etc.

PID (hex)	PID (Dec)	Data bytes returned	Description	Min value	Max value	Units	Formula[a]
00	0	4	PIDs supported [01 - 20]				Bit encoded [A7..D0] == [PID $01..PID $20] See below
01	1	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)				Bit encoded. See below
02	2	2	Freeze DTC				
03	3	2	Fuel system status				Bit encoded. See below
04	4	1	Calculated engine load	0	100	%	$\frac{100}{255}A$ (or $\frac{A}{2.55}$)
05	5	1	Engine coolant temperature	-40	215	°C	$A-40$
06	6	1	Short term fuel trim—Bank 1	-100 (Reduce Fuel: Too Rich)	99.2 (Add Fuel: Too Lean)	%	$\frac{100}{128}A-100$ (or $\frac{A}{1.28}-100$)
07	7	1	Long term fuel trim—Bank 1				
08	8	1	Short term fuel trim—Bank 2				
09	9	1	Long term fuel trim—Bank 2				
0A	10	1	Fuel pressure (gauge pressure)	0	765	kPa	$3A$
0B	11	1	Intake manifold absolute pressure	0	255	kPa	A
0C	12	2	Engine RPM	0	16,383.75	rpm	$\frac{256A+B}{4}$
0D	13	1	Vehicle speed	0	255	km/h	A
0E	14	1	Timing advance	-64	63.5	° before TDC	$\frac{A}{2}-64$
0F	15	1	Intake air temperature	-40	215	°C	$A-40$
10	16	2	MAF air flow rate	0	655.35	grams/sec	$\frac{256A+B}{100}$
11	17	1	Throttle position	0	100	%	$\frac{100}{255}A$
12	18	1	Commanded secondary air status				Bit encoded. See below
13	19	1	Oxygen sensors present (in 2 banks)				[A0..A3] == Bank 1, Sensors 1-4. [A4..A7] == Bank 2...
14	20	2	Oxygen Sensor 1 A: Voltage B: Short term fuel trim				
15	21	2	Oxygen Sensor 2 A: Voltage B: Short term fuel trim				
16	22	2	Oxygen Sensor 3 A: Voltage B: Short term fuel trim				
17	23	2	Oxygen Sensor 4 A: Voltage B: Short term fuel trim	0	1.275	volts	$\frac{A}{200}$
			Oxygen Sensor 5	-100	99.2	%	$\frac{100}{128}B-100$

Ilustración 21 – Listado de comandos PID y sus características extraídos de Wikipedia

El listado anterior es una breve muestra del listado real, mucho más amplio, se han omitido gran cantidad de PIDs para simplemente hacernos una idea de cómo se trabaja con estos comandos.[3]

4.2.3.8. La respuesta del vehículo

Una vez el dispositivo de diagnóstico haya enviado una petición, la ECU del vehículo la procesa, recaba la información necesaria a partir de los sensores de los componentes y envía esa información a modo de respuesta. Cada comando PID tiene una única respuesta por parte de la ECU, la respuesta suele ser un conjunto de tres o cuatro pares de dígitos hexadecimales. Por ejemplo, el comando que solicita las revoluciones del motor, "**01 0C**" recibe como respuesta "**41 0C 0F A0**". Los dos primeros dígitos hacen referencia al modo de trabajo en el que nos encontramos, el modo 01 o modo de obtener resultados o valores actuales. El siguiente par de números identifica al componente del cual solicitamos datos, en este caso 0C está asociado a las revoluciones del motor.

Los dos últimos pares de dígitos se refieren al valor de las revoluciones codificado en hexadecimal. Para cada tipo de comando debemos aplicar una fórmula diferente para la decodificación de los resultados. En el caso de las revoluciones, una vez tengamos los valores de los pares de dígitos convertidos a formato decimal, aplicamos la siguiente fórmula:

$$((A*256) + B) / 4 \qquad \textit{(Fórmula 1)}$$

Siendo A el par 0F y B el par A0 convertidos a decimal. Una vez aplicada la fórmula y realizados los cálculos, obtenemos como resultado el valor de 1.000 rpm o revoluciones por minuto del motor del vehículo en el instante en el que se consultó.

4.2.3.9. Comandos AT

El conjunto de comandos AT es una serie de comandos que permiten inicializar, configurar o modificar ciertos parámetros de la conexión entre el dispositivo de diagnóstico conectado al vehículo y el dispositivo desde el cual se pedirán los valores de los parámetros. Los comandos enviados tienen una longitud de entre 3, 4 o 5 caracteres y pueden enviarse con espacios blancos entre

caracteres o sin ellos. Todos poseen una estructura similar, los comandos de tres caracteres serían AT X, siendo X un parámetro de la conexión que se desea modificar. Por ejemplo, AT Z y AT D, el primer comando se utiliza para resetear o reinicializar el dispositivo OBD, mientras que el segundo, se utiliza para establecer por defecto todas las características de la comunicación, AT Default.

Por otro lado, tenemos los comandos de cuatro caracteres que poseen la estructura AT XX, por ejemplo AT E0 o Echo Off, se utiliza para evitar que el dispositivo OBD envíe como respuesta el comando. AT S0 o Spaces Off, para desactivar los espacios en blanco entre caracteres.

Por último, tenemos los comandos de cinco caracteres, que tienen la estructura AT XX X. En este proyecto se hará uso de un único comando de este tipo y será el AT SP 0 o Select Protocol Auto. Este comando nos será muy útil porque detectará el tipo de protocolo de comunicación que utiliza nuestro vehículo y lo establecerá automáticamente para la comunicación a posteriori.

4.2.4. Estudio acerca de la conducción eficiente

La conducción eficiente es un modo de conducir un vehículo que tiene como objetivo lograr un bajo consumo de carburante, reducir la contaminación ambiental, obtener un mayor confort en la conducción y disminuir los riesgos en la carretera, aumentando así la seguridad del conductor. Por otro lado, la conducción eficiente también permite aumentar la vida útil de los vehículos, evitando un deterioro prematuro por malas prácticas a la hora de conducir. Este tipo conducción se rige por una serie de reglas sencillas y eficaces, que tratan de aprovechar las posibilidades que ofrecen los motores de los coches actuales.

Reglas a seguir para una conducción eficiente según la DGT[14]:

- Arrancar el motor del vehículo sin pisar el pedal del acelerador.
- En vehículos con motores turboalimentados, esperar unos segundos antes de iniciar la marcha tras el arranque del motor.

- Cuando el vehículo esté parado, utilizar la primera marcha para volver a poner el vehículo en movimiento. Pasar a la segunda marcha tras unos cinco o seis metros o tras un par de segundos.

- Para reducir las marchas se recomienda hacerlo en torno a las 2.000 y 2.500 revoluciones en motores de gasolina y en torno a las 1.500 y 2.000 revoluciones en motores diesel.

- El cambio de marchas debe realizarse ágilmente, pues demorarse en el cambio provoca una pérdida en la aceleración del vehículo por rozamiento.

- El cambio general de marchas: se recomienda pasar a tercera marcha a partir de los 30 km/h, pasar a cuarta marcha a partir de los 40 km/h y a quinta a partir de los 50 km/h. Siempre acelerar tras la realización del cambio.

- Circular el mayor tiempo posible en las marchas más largas y a bajas revoluciones permite consumir menos combustible, siempre y cuando se nos permita mantener la distancia de seguridad con los vehículos que nos preceden.

- Es preferible circular en marchas largas con el acelerador pisado en mayor medida (entre el 50% y el 70% de su recorrido), que en marchas más cortas con el acelerador menos pisado.

- En ciudad, siempre que sea posible, utilizar la 4ª y la 5ª marcha, respetando siempre los límites de velocidad.

- Frenar con el motor. Buscar la fluidez en la circulación, evitando los frenazos. En el momento en que se detecte un obstáculo o una reducción de la velocidad de circulación en la vía, levantar el pie del acelerador intentando evitar la frenada brusca. Esto es, frenar con el motor. Se trata de mantener el vehículo en movimiento por su propia inercia con una marcha engranada.

- Frenar de forma suave con el pedal del freno y reducir de marcha lo más tarde posible, con especial atención en las bajadas.

- Siempre que la velocidad y el espacio lo permitan, detener el coche sin reducir previamente de marcha.

- Nunca bajar una pendiente con el coche en punto muerto pues además de incrementar el consumo y la contaminación resulta extremadamente peligroso.

- Si las pendientes fuesen ascendentes hay que procurar circular en la marcha más elevada posible, aunque tengamos que pisar más el acelerador.

- Si son descendientes levantar el pie del acelerador sin reducir de marcha y dejar bajar al coche por su propia inercia, si la aceleración no se mantuviera acelerar lo justo para conseguir la velocidad de crucero pretendida.

- Antes de entrar en una curva hay que adaptar la velocidad del vehículo, se hará exactamente igual que en cualquier deceleración. Levantar el pie del acelerador y dejar rodar el coche por su propia inercia. Si fuera necesario reducir a la marcha precisa para tomar la curva.

- Conducción en caravanas: Procurar circular en la marcha más larga posible que permita mantener la distancia de seguridad con los vehículos que nos preceden. Evitar acelerar para después tener que volver a frenar, así evitaremos desgastes innecesarios para nuestro vehículo, además de ahorrar combustible y contaminar menos. Como consecuencia de esto, los vehículos que circulan detrás podrán hacer lo mismo y habrá más fluidez en el tráfico.

4.2.5. Conducción eficiente aplicada al proyecto en desarrollo

Tras estudiar los consejos proporcionados por la DGT y otros organismos acerca de la conducción eficiente se consideraron los siguientes aspectos a tratar

para combinarlos con los datos recogidos del vehículo y plasmarlos en el proyecto en desarrollo:

- El valor de las revoluciones del motor es de vital importancia para medir el grado de conducción eficiente. Revolucionar el motor implica mayor consumo y por tanto ha de notificarse cuando los valores de las revoluciones se hayan elevado innecesariamente. Los valores de revoluciones iguales o por debajo de las 2.500 rpm serán considerados bajos y por tanto óptimos. Los valores de revoluciones entre las 2.500 rpm y las 3.000 rpm serán considerados intermedios y los valores iguales o por encima de las 3.000 rpm serán considerados elevados.

- En cuanto a la posición del acelerador, la DGT considera que de manera óptima su pisado debe estar entre el 50% y el 70% en marchas largas. Sin embargo, hay que tener en cuenta que a veces es inevitable pisar a fondo el acelerador cuando subimos pendientes en carretera con la máxima marcha. Lo ideal es combinar la medición de este parámetro con las revoluciones del motor para sacar conclusiones acerca de la eficiencia.

- La velocidad del vehículo en un instante dado por sí solo, es un parámetro muy relativo, medir la eficiencia a través de la velocidad depende del tramo de carretera por el que el vehículo se mueva, y es difícil fijar reglas. Lo ideal es combinar el valor de este parámetro junto al de las revoluciones del motor y la posición del pedal del acelerador. De esta forma sabremos a ciencia cierta el momento exacto de un acelerón comprobando el valor de los tres parámetros.

- El consumo instantáneo del vehículo nos permite conocer la distancia recorrida en función del consumo de combustible. Se mide en kilómetros recorridos por litros de combustible consumidos (Km/L), es decir, el número de kilómetros que podemos recorrer por litro de combustible consumido. Resulta muy útil combinar la medición de este parámetro con el de la posición del acelerador en un momento dado porque así sabremos qué cantidad de combustible quemamos cuando pisamos el

acelerador. Además, a posteriori nos podrá ayudar a corregir la conducción para evitar un consumo excesivo de carburante en tramos problemáticos.

- El proceso de frenado no pude medirse con la aplicación debido a que no existe ningún comando en el protocolo SAE J1979 que permita medir el índice de pisado sobre este pedal. Sin embargo, si podrá aproximarse comparando los valores de velocidad del vehículo en instantes de tiempo pequeños, identificando así frenazos por parte del conductor cuando la velocidad del vehículo se reduzca drásticamente.

- La temperatura del líquido refrigerante no nos permite estimar el mayor o menor consumo de combustible por parte del vehículo, pero sí es realmente útil para evitar graves problemas en el motor y aumentar la vida útil del automóvil. El líquido refrigerante es el encargado de eliminar parte del calor producido por el funcionamiento del motor, la eliminación de altas temperaturas permite el correcto funcionamiento de muchos componentes del motor. Por otro lado, el líquido refrigerante también permite evitar la congelación de componentes del motor a temperaturas extremadamente bajas. La temperatura ideal del motor en funcionamiento está en torno a los 90ºC, el hecho de que el valor baje o se eleve considerablemente nos indica que el motor puede no estar funcionando correctamente lo que puede derivar en problemas mucho más graves para nuestro vehículo. Podremos tener en cuenta estas situaciones conociendo en todo momento la temperatura del líquido refrigerante. Las temperaturas máxima y mínima aceptadas por el refrigerante dependen de la cantidad de etilenglicol que posea el líquido. Por ejemplo, si la cantidad de etilenglicol es de un 10% el refrigerante resistirá temperaturas entre los -2ºC y los 102ºC, si es de un 20% resistirá temperaturas entre -12ºC y 103ºC y si es de 30% resistirá entre -37ºC y 108ºC. Como es complicado saber qué refrigerante utiliza cada coche emplearemos los valores del líquido con un 20% de etilenglicol. No tendría sentido utilizar el de 30% porque en España no se alcanzan temperaturas por debajo de los -30ºC.[15][16]

- La temperatura del aire de admisión al motor es otro parámetro que ayuda a aumentar la vida útil del vehículo. Dependiendo de la temperatura del aire se regula la entrada de combustible a las cámaras de combustión. De esta forma se combina aire y combustible de manera adecuada para corregir el punto estequiométrico. La temperatura del aire debe rondar entre los 20ºC y los 80ºC, valores fuera de ese rango pueden significar averías en el sensor de entrada de aire que a su vez pueden provocar mayor consumo de combustible, rendimiento deficiente del motor o en casos muy graves cortocircuitos o fallos mecánicos. Si la temperatura se sale del rango se le notificará al usuario en la aplicación.[17]

4.2.6. El problema del consumo instantáneo

Uno de los primeros problemas encontrados estaba relacionado con la forma de obtener información acerca del consumo instantáneo del vehículo. El comando empleado por el estándar SAE J1979 para calcular los litros de combustible consumidos por hora es el 015E. Sin embargo, este comando no está implementado en la mayoría de los vehículos, por lo que al realizar la consulta no se obtiene respuesta alguna.

Tras una búsqueda exhaustiva se determinó una forma alternativa de calcular el consumo instantáneo. Consiste en utilizar los valores obtenidos por los sensores de flujo de masa de aire (MAF) y de la velocidad del vehículo (VSS)[6]. El MAF es la cantidad de aire que entra al motor para la combustión, a mayor cantidad de aire de entrada mayor será la cantidad de combustible que el motor necesite quemar y por tanto mayor será el consumo. Una vez tengamos los valores en un momento dado:

- Dividimos el valor obtenido del MAF entre un constante, 14,7, que es la cantidad ideal de gramos de aire consumida por el motor por cada gramo de combustible. Obtenemos con resultado la cantidad de gramos de combustible consumida por segundo.
- Dividimos el resultado anterior entre 454 para convertir los gramos por segundo a libras por segundo.

- Ahora dividimos el resultado entre 6,407 para convertir las libras por segundo a galones por segundo.
- Multiplicamos el resultado por 3600 para obtener los galones por hora.
- Ahora tenemos que pasar de galones por hora a millas por galón y después a litros por kilómetro para obtener el resultado en el sistema de medida utilizado en España.

Todas estas operaciones pueden resumirse matemáticamente a la siguiente fórmula:

$$MPG = \frac{\text{VSS} * 7{,}718}{\text{MAF}}$$

Es decir, multiplicamos el valor de la velocidad del vehículo en un instante dado por un número que resulta de unificar todas las conversiones de medidas necesarias. El resultado lo dividimos por el flujo de masa de aire y obtenemos las millas por galón de combustible consumido, que pueden convertirse fácilmente a kilómetros por litro de combustible dividiendo MPG por 2,352.

A	B
454 gramos	1 Libra
6,701 Libras/s	1 galones/s
3.600 segundos	1 hora
0,621317 millas	1kilómetro
2,352 km/litro	1 milla/galón

Tabla 5 – Conversiones entre unidades de medida

La diferencia entre el valor del consumo instantáneo obtenido directamente a partir del vehículo y del calculado mediante la fórmula anterior es prácticamente inapreciable tal y como puede observarse en el gráfico de la ilustración 22, obtenido de un estudio realizado por HEM Data. [18]

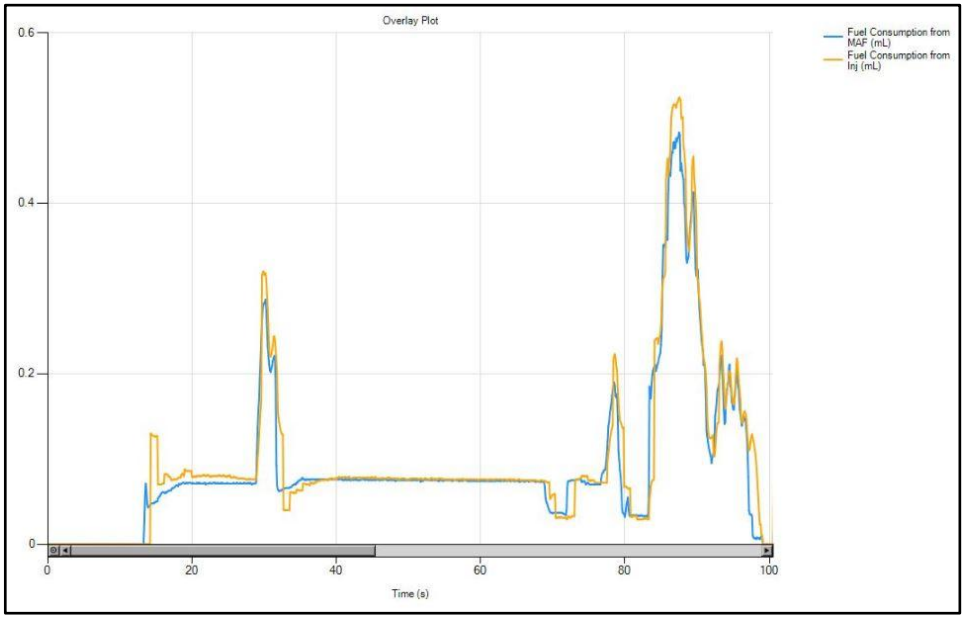


Ilustración 22 – En azul el consumo instantáneo calculado a partir del MAF y en naranja el obtenido directamente del vehículo

4.2.7. Análisis de las herramientas y librerías software disponibles

Uno de los principales objetivos de este proyecto es el de realizar un software que permita recoger información de los parámetros de un vehículo. Una vez visto cómo funciona el sistema OBD2, el siguiente paso era implementarlo.

En la propuesta de este TFG se indicaba la posibilidad de utilizar librerías de terceros siempre que fuese posible, y tras un largo proceso de búsqueda e investigación se determinó que lo mejor era tratar de implementar la comunicación con el dispositivo OBD2 partiendo desde cero. Esto fue debido a la escasa existencia de materiales en los que poder apoyarse, que se resumen a un proyecto de una app de Android disponible en la plataforma GitHub, del cual parten todas las personas que se inician en el desarrollo de software para la comunicación OBD2.

Este proyecto y otros derivados del mismo, sirven de mucha ayuda para conocer cómo ha de establecerse la comunicación con el dispositivo OBD2 y saber pequeños detalles para la estabilidad de la misma. Sin embargo, el proyecto actualmente se encuentra abandonado desde el 1 de mayo de 2017 y se conocen errores. [4][5]

4.2.8. Conclusiones obtenidas

Una vez realizados los estudios sobre el sistema de comunicación OBD2 y la conducción eficiente, se consideró obtener los datos de los siguientes parámetros mediante consultas al dispositivo OBD2:

- La velocidad del vehículo en km/h.
- Las revoluciones del motor en rpm.
- La carga del motor mediada en porcentaje de carga.
- La posición del acelerador en porcentaje, siendo el valor mínimo 0% si el pedal está libre y 100%, si está totalmente pisado.
- Temperatura del líquido refrigerante en grados centígrados.
- Temperatura del aire de entrada al motor en grados centígrados.
- El flujo de masa de aire en gramos por segundo.

Estos parámetros nos permitirán hacer cálculos en el futuro acerca de la eficiencia, consumo y mantenimiento del vehículo. En la tabla 6 observamos de manera más detallada los parámetros juntos a sus comandos para solicitarlos y sus unidades de medida.

Parámetro	Comando	Unidad de medida	Valor mínimo	Valor máximo	Fórmula
Velocidad(VSS)	010D	km/h	0	255	A
Revoluciones	010C	rpm	0	16.383,75	$\frac{(256 * A) + B}{4}$
Carga del motor	0104	%	0	100	$\frac{A * 100}{255}$
Posición del acelerador	0111	%	0	100	$\frac{A * 100}{255}$
Temperatura refrigerante	0105	ºC	-40	215	A-40
Temperatura aire entrada	010F	ºC	-40	215	A-40
Flujo de masa de aire(MAF)	0110	gr/s	0	655,35	$\frac{(256 * A) + B}{100}$
Consumo instantáneo	015E*	l/Km	0	3.276,75	$\frac{(256 * A) + B}{20}$
Consumo instantáneo (calculado)	-	Km/l	0	3.276,75	$\left(\frac{VSS * 7,718}{MAF}\right) / 2,352$

Tabla 6 – Parámetros recogidos, sus comandos, unidades de medida, valores posibles y fórmulas.

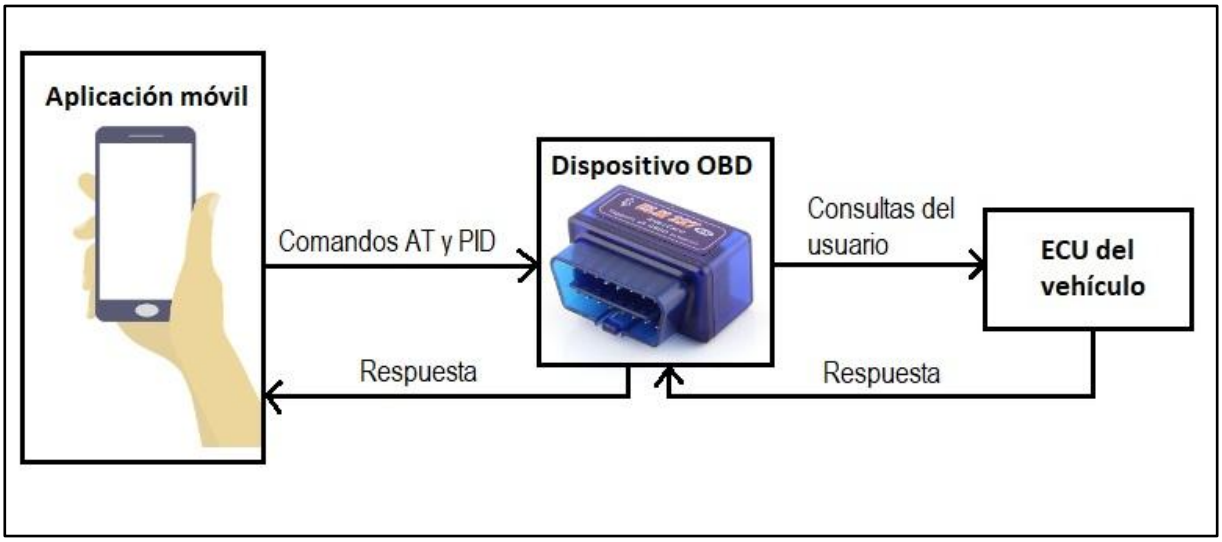


Ilustración 23 – Esquema de la comunicación

4.2.9. Reunión

En la reunión realizada con el tutor al término de la primera iteración se decidió comenzar a desarrollar un sistema de comunicación OBD2 desde su raíz, estudiando gradualmente el funcionamiento del mismo. Esta decisión fue acordada ante la incertidumbre que los proyectos y librerías para OBD2 ya existentes generaban y la posibilidad de que al desarrollar la comunicación desde cero se pudiera conocer con toda exactitud los pequeños detalles de este sistema, que podrían dar solución a futuros problemas que pudieran aparecer.

Finalmente, se estableció como objetivo para la segunda iteración el diseño e implementación de una aplicación para Android que estableciera comunicación vía Bluetooth con el dispositivo OBD2 ELM327 Mini y que enviara comandos PID para recibir los parámetros de conducción del vehículo.

4.3. Iteración 2 - Diseño e implementación de una aplicación móvil para establecer comunicación el dispositivo OBD2 vía Bluetooth.

En la segunda iteración se definió el diseño e implementación de una aplicación móvil encargada de capturar los valores de los parámetros de conducción

y componentes del vehículo mediante una conexión vía Bluetooth con el dispositivo OBD2.

4.3.1. Diagrama de clases

Los diagramas de clases permiten conocer la estructura de un sistema mostrando las clases, sus atributos, sus operaciones y las relaciones entre objetos. A continuación, se expone el diagrama de clases de la aplicación de captura de datos.

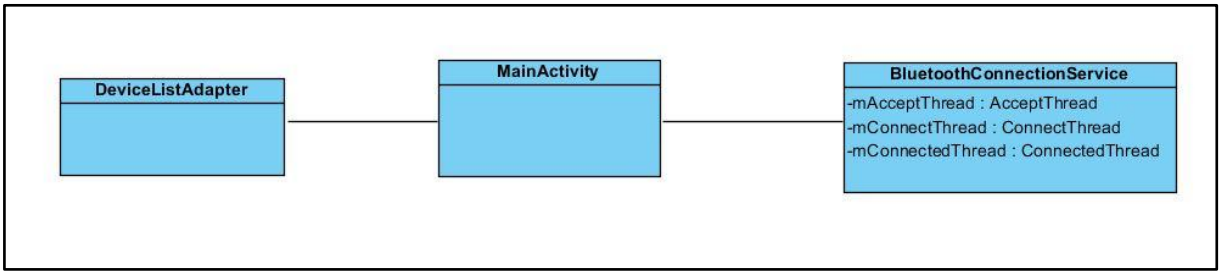


Ilustración 24 – Diagrama de clases de aplicación de captura de datos

- **MainActivity**: es la clase principal encargada de gestionar la interacción del usuario, actualizar la interfaz, enviar comandos al canal de comunicación Bluetooth, recibir y procesar las respuestas del dispositivo OBD2, etc.
- **DiviceListAdapter**: es la clase encargada de gestionar la interacción y la visualización de la lista de dispositivos Bluetooth con los que podremos establecer conexión.
- **BluetoothConnectionService**: esta clase es la encarga de establecer y mantener el canal de comunicación entre los dispositivos Bluetooth.

4.3.2. Diagrama de casos de uso

El diagrama de casos de uso define en este sentido la interacción básica y esencial que el usuario tendrá con la aplicación móvil.

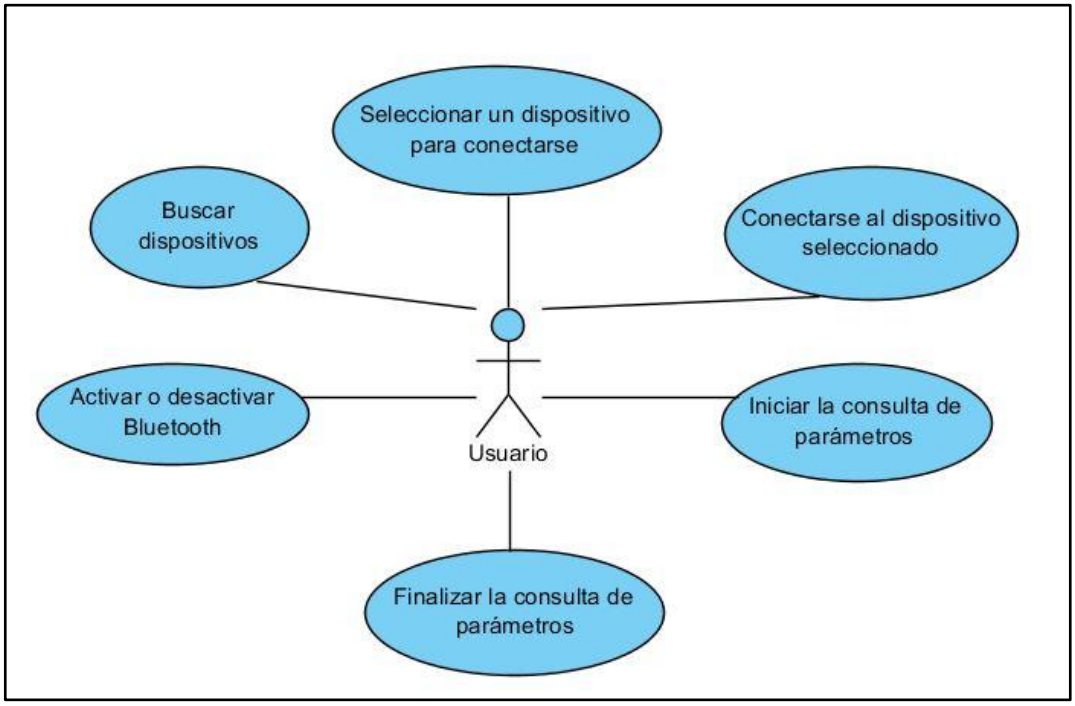


Ilustración 25 – Diagrama de casos de uso para la aplicación de captura de datos

A continuación se describe de manera más detallada cada caso de uso:

- **Activar o desactivar la conexión Bluetooth**: el usuario activa o desactiva esta característica del dispositivo móvil desde la aplicación.
- **Buscar dispositivos Bluetooth**: con el objetivo de encontrar el dispositivo OBD2 conectado al coche.
- **Seleccionar un dispositivo**: para seleccionar el dispositivo OBD2 entre todos los dispositivos encontrados por la aplicación.
- **Conectarse al dispositivo seleccionado**: para poder establecer la conexión Bluetooth con el dispositivo OBD2.
- **Iniciar la comunicación** mediante la consulta de parámetros al vehículo, así como la recepción de las respuestas por parte de este. De igual forma, el usuario podrá **finalizar la comunicación** con el dispositivo una vez finalice el trayecto.

4.3.3. Diagrama de flujo o de actividades

A través del diagrama de actividades podemos observar la secuencia principal de interacción que el usuario establecerá con la aplicación y partes del

funcionamiento de la misma que son algunas invisibles para el usuario, pero necesarios para un uso correcto.

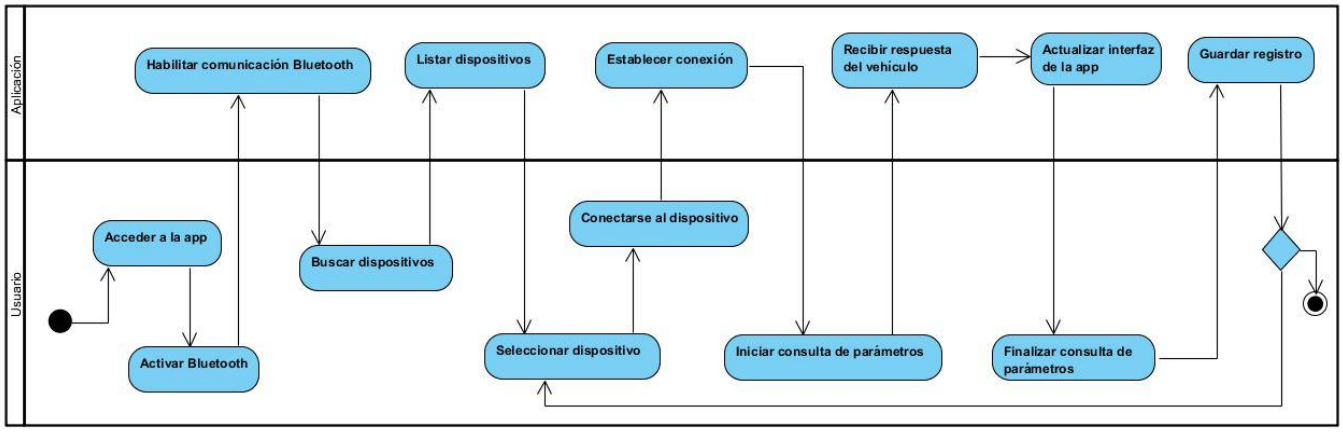


Ilustración 26 – Diagrama de actividades para la aplicación de captura de datos

El diagrama está dividido mediante un swinlane en dos carriles para identificar qué actividades realiza el usuario y qué actividades realiza la aplicación.

La descripción de cada actividad se verá de forma más detallada en el apartado de diseño con storyboards.

4.3.4. Diseño con storyboards

La aplicación cuenta con una única vista o pantalla donde el usuario tendrá que establecer la conexión Bluetooth con el dispositivo OBD2 conectado al coche e iniciar la comunicación mediante el envío automático de comandos de consulta.

Al iniciar la aplicación encontramos en la pantalla tres botones, un primer botón para activar o desactivar la conexión Bluetooth y la búsqueda de dispositivos. Un segundo botón, a la derecha del primero, para establecer la conexión Bluetooth con un dispositivo seleccionado. Para seleccionar un dispositivo, se desplegará un listado con los disponibles tras la búsqueda de dispositivos con conexión Bluetooth, bastará con seleccionar el dispositivo OBD2 en el listado y pulsar “Conectar”.

El tercer botón permitirá iniciar y finalizar la comunicación con el dispositivo OBD2.Al pulsarlo por primera vez iniciamos la comunicación mediante el envío de comandos y la recepción de las respuestas correspondientes por parte del vehículo.

A continuación se detalla la secuencia de interacción con la aplicación empezando por la pantalla de inicio:

Ilustración 27 – Prototipo de la pantalla al inicio de la aplicación

Al pulsar el botón “ON” activamos la conexión Bluetooth del dispositivo móvil y se inicia la búsqueda de dispositivos disponibles para la conexión.

Ilustración 28 – Prototipo de la pantalla al iniciar la conexión Bluetooth

El siguiente paso será conectarnos al dispositivo OBD, lo seleccionamos en el listado de dispositivos Bluetooth encontrados y pulsamos el botón “Conectar”.

Ilustración 29 – Prototipo de la pantalla al conectarse al OBD2

Por último, tendremos que iniciar la comunicación mediante el envío de comandos al vehículo. Algunos parámetros recibidos como la velocidad o las revoluciones actualizarán una serie de cuadros de texto con los valores obtenidos para que el usuario sepa en todo momento que la comunicación continúa estable.

Ilustración 30 – Prototipo de la pantalla al iniciar la comunicación con el vehículo

Al pulsar el botón de “Finalizar comunicación”, se detiene la consulta de parámetros y se cierra la conexión Bluetooth con el dispositivo OBD2.

A continuación podemos observar una secuencia completa de las acciones:

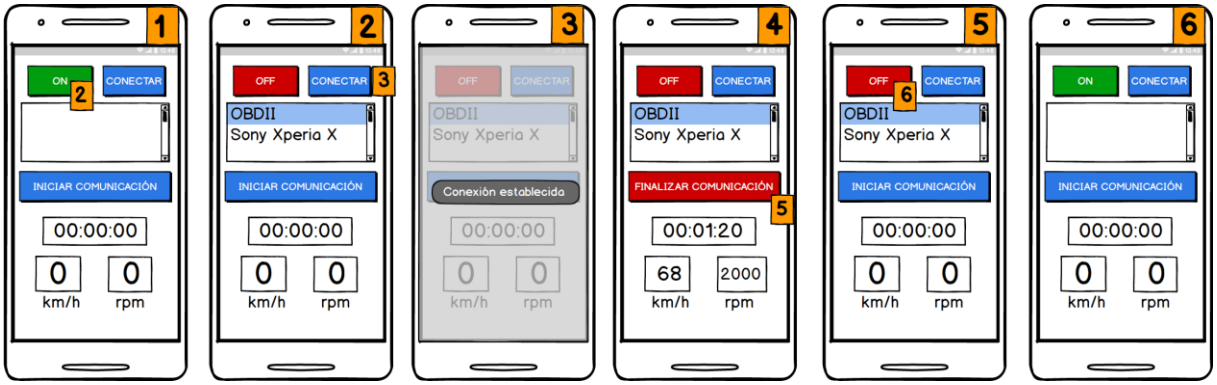


Ilustración 31 – Secuencia de acciones sobre la aplicación de captura de datos

4.3.5. Desarrollo e implementación de la aplicación

Para la implementación de la aplicación se hizo uso de los tutoriales y guías oficiales de Android que pueden encontrarse de manera gratuita en la página oficial de Android Developers. En este caso se ha hecho uso de la guía de conectividad Bluetooth y de desarrollo de un chat de texto entre dispositivos Bluetooth [7].

La guía de conectividad Bluetooth nos permite:

- Activar la conexión Bluetooth.
- Buscar otros dispositivos Bluetooth.
- Sincronizarse o conectarse a los dispositivos encontrados.
- Transferir datos entre los dispositivos.

El primer paso fue la creación de un nuevo proyecto en Android Studio con una sola Activity o pantalla. Después se llevó a cabo el diseño de la pantalla en base al diseño con storyboards previamente realizado, con la diferencia de la inclusión de un nuevo botón en la parte superior de la pantalla para la búsqueda de dispositivos Bluetooth. Este nuevo botón será el encargado de modificar el listado de dispositivos encontrados.

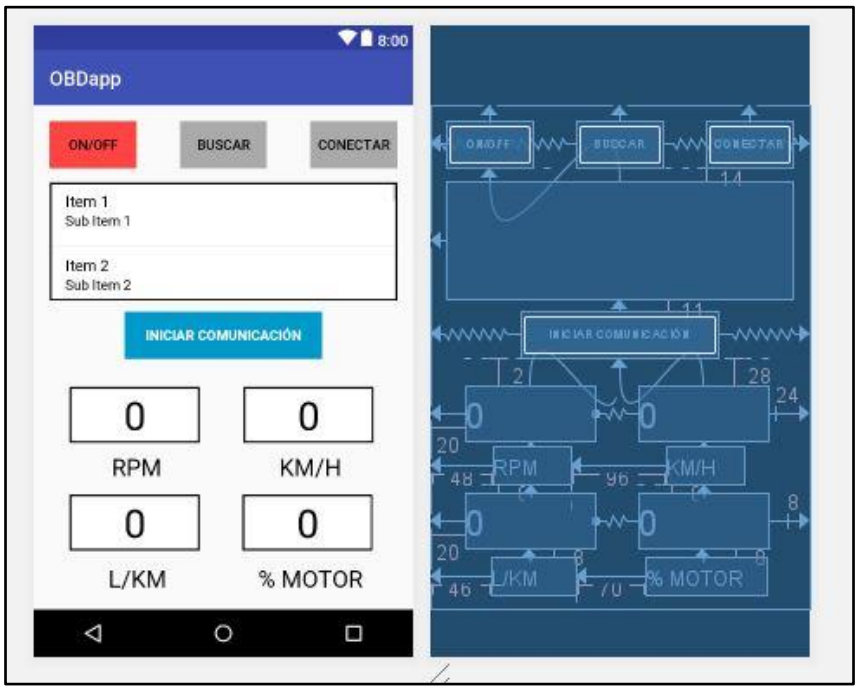


Ilustración 32 – Diseño de la aplicación desde Android Studio

Posteriormente se pasó a la implementación de la conectividad Bluetooth donde tienen vital importancia los siguientes elementos:

- **BluetoothAdapter:** un objeto de este tipo representa un adaptador local de Bluetooth y nos permite encontrar otros dispositivos, conectarnos a ellos y establecer un canal de comunicación.
- **BluetoothDevice:** los objetos de este tipo representan a los dispositivos Bluetooth remotos que poseen un nombre, una dirección MAC y un estado.
- **BluetoothSocket:** representa el canal de comunicación que se utiliza para el intercambio de información entre dispositivos.
- **Permisos Bluetooth y Bluetooth Admin:** a fin de usar las funciones Bluetooth debemos declarar estos permisos en nuestra aplicación.

```
<uses-feature android:name="android.hardware.bluetooth" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Ilustración 33 – Permisos para hacer uso de la conectividad Bluetooth en nuestra aplicación

El primer botón que encontramos en la aplicación es el encargado de activar la conectividad Bluetooth del dispositivo o desactivarla en el caso de que ya estuviese

activa y no fuera necesaria. Para realizar estas acciones simplemente tenemos que indicarle a nuestro adaptador el estado que deseamos obtener. Para activar, ACTION_REQUEST_ENABLE y para desactivar ACTION_REQUEST_DISABLE.

```
//Si el bluetooth no está activado, lo activamos
if (!mBluetoothAdapter.isEnabled()) {
    Log.d(TAG,  msg: "enableDisableBT: Activando bluetooth.");
    Intent enableBTIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivity(enableBTIntent);

    IntentFilter BTIntent = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
    registerReceiver(enableBluetoothBroadcastReceiver, BTIntent);

    btnONOFF.setBackgroundColor(Color.GREEN);
    bluetoothActivo = true;
}
```

Ilustración 34 – Código para activar la conectividad Bluetooth

```
//Si el bluetooth está activado, lo desactiva al pulsar el botón
if (mBluetoothAdapter.isEnabled()) {
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setTitle("Atención");
    builder.setMessage("¿Deseas desactivar el blueetooth?. Se perderá la conexión con el dispositivo");
    builder.setPositiveButton( text: "Aceptar", (dialogInterface, i) → {
            Log.d(TAG,  msg: "enableDisableBT: Desactivando bluetooth.");
            mBluetoothAdapter.disable();

            IntentFilter BTIntent = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(enableBluetoothBroadcastReceiver, BTIntent);

            btnONOFF.setBackgroundColor(Color.RED);
            bluetoothActivo = false;
            dispositivoOBDseleccionado = false;
            conexionOBDrealizada = false;
    });

    builder.setNegativeButton( text: "Cancelar", (dialogInterface, i) → {
            dialogInterface.cancel();
    });

    AlertDialog dialog = builder.create();
    dialog.show();
```

Ilustración 35 – Código para desactivar la conectividad Bluetooth

La búsqueda de dispositivos es muy similar, activamos la posibilidad de encontrar dispositivos y rellenamos una lista con todos los dispositivos encontrados con su nombre y su dirección MAC.

```
if(bluetoothActivo){ //El bluetooth está activado puedo iniciar la búsqueda
    Log.d(TAG,  msg: "btnDiscover: Buscando dispositivos no sincronizados");

    //Si ya se estaba buscando dispositivos, cancelamos el proceso para volver a reiniciarlo
    if (mBluetoothAdapter.isDiscovering()) {
        mBluetoothAdapter.cancelDiscovery();
        Log.d(TAG,  msg: "btnDiscover: Canceling discovery.");

        //Limpiamos la lista de dispositivos encontrados anteriormente
        mBTDevices.clear();
        lvNewDevices.deferNotifyDataSetChanged();
    }
```

Ilustración 36 – Código para buscar dispositivos Bluetooth

Al seleccionar un dispositivo de la lista guardamos sus atributos para posteriormente establecer la conexión con él. La conexión y la comunicación entre dispositivos hacen uso de tres hilos que se ejecutan en el archivo Java BluetoothConnectionService, que es el encargado de crear el socket de comunicación entre los dispositivos y mantenerlo. Los hilos de ejecución son los siguientes:

- **AcceptThread:** es un hilo que se ejecuta a la escucha de conexiones entrantes, es decir, nuestro dispositivo representa el rol de servidor, mientras que los dispositivos que intentan conectarse al nuestro emplean el rol de clientes.

- **ConnectThread:** este hilo funciona al contrario de AcceptThread, ejecutándose mientras se intenta establecer conexión con un dispositivo que se encuentra en escucha. Nuestro dispositivo adopta el rol de cliente y el que se mantiene a la escucha, el rol de servidor.

- **ConnectedThread:** una vez establecida la conexión entre dos dispositivos, este hilo se encarga de mantener la conexión y la comunicación mediante el uso de un InputStream y un OutputStream para recibir y enviar datos respectivamente al dispositivo que nos hemos conectado.

El hilo ConnectedThread posee un método para recibir los mensajes enviados desde el dispositivo Bluetooth al que nos hemos conectado y otro método para enviar mensajes al dispositivo.

```
public void run(){
    byte[] buffer = new byte[1024];  //Buffer para almacenar los streams

    int bytes; //Bytes devueltos de la lectura

    //Escuchar InputStream hasta que ocurra una excepción
    while (comunicacionActiva) {
        //Leer de InputStream
        try {
            bytes = mmInStream.read(buffer);
            String mensajeRecibido = new String(buffer,  offset: 0, bytes);
            Log.d(TAG,  msg: "InputStream: " + mensajeRecibido);

            Intent mensajeRecibidoIntent = new Intent( action: "mensajeRecibido");
            mensajeRecibidoIntent.putExtra( name: "mensaje", mensajeRecibido);
            LocalBroadcastManager.getInstance(mContext).sendBroadcast(mensajeRecibidoIntent);

        } catch (IOException e) {
            Log.e(TAG,  msg: "write: Error al leer de InputStream. " + e.getMessage() );
            Intent errorConexion = new Intent( action: "falloConexion");
            LocalBroadcastManager.getInstance(mContext).sendBroadcast(errorConexion);
            break;
        }
    }
}
```

Ilustración 37 – El método run() recibe los datos enviados por el otro dispositivo

```
//Llamar a este método desde la Main Activity para enviar datos al otro dispositivo
public void write(byte[] bytes) {
    String text = new String(bytes, Charset.defaultCharset());
    Log.d(TAG,  msg: "write: Escribiendo en OutputStream: " + text);
    try {
        mmOutStream.write(bytes);
    } catch (IOException e) {
        Log.e(TAG,  msg: "write: Error al escribir en OutputStream. " + e.getMessage() );
    }
}
```

Ilustración 38 – El método write() envía datos al otro dispositivo

Tal y como se estudió en la primera iteración, la comunicación con el dispositivo OBD2 debe iniciarla la aplicación, una vez establecida la conexión Bluetooth con el dispositivo debemos comenzar con el envío de comandos para las consultas. El envío de los comandos no se hace de manera simultánea, se inicia la comunicación con el envío de un solo comando y no se envía el siguiente hasta no

haber recibido una respuesta por parte del dispositivo OBD2. El método en el código encargado de ello se divide en dos partes, una primera parte se utiliza para el envío de comandos AT para la configuración del dispositivo OBD2 y la comunicación que éste establecerá con la ECU del vehículo. La segunda parte será la encargada de enviar los comandos PID de consulta sobre los parámetros del vehículo tales como las revoluciones, la velocidad, el consumo, etc.

```
private void enviarComando() {
    String command;
    byte[] bytes;

    if (!configuracionRealizada) {
        command = commandsAT[indiceAT];
        command += "\r";
        indiceAT++;
        bytes = command.getBytes();

        if (indiceAT >= 7) {
            configuracionRealizada = true;
        }
    } else {
        if(indicePID == commandsPID.length){
            indicePID = 0;
        }
        command = commandsPID[indicePID];
        command += "\r";
        indicePID++;
        bytes = command.getBytes();
    }
    mBluetoothConnection.write(bytes);
}
```

Ilustración 39 – Método para enviar comandos al dispositivo OBD2

El método que envía tanto los comandos AT y PID se ejecuta cada vez que se recibe una respuesta del OBD2 y lo hace para enviar la siguiente consulta. Los comandos AT únicamente tienen que enviarse al inicio de la comunicación, mientras que los comandos PID se van enviando por ráfagas. Éstos últimos se almacenan en una estructura tipo vector o array en el que cada posición almacena un comando diferente. El array se va recorriendo posición a posición cada vez que se ejecuta el método de enviar comandos. Cuando llega al tope, es decir, al final del array, vuelve a iniciarse el recorrido. Así, hasta finalizar la comunicación con el dispositivo OBD2.

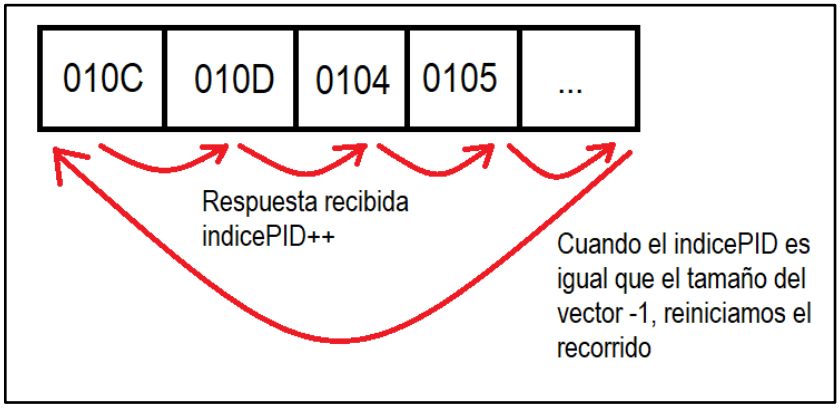


Ilustración 40 – Recorrido del vector de comandos PID durante la comunicación

```
String[] commandsAT = {"AT D", "AT Z", "AT E0", "AT L0", "AT S0", "AT H0", "AT SP 0"};
String[] commandsPID = {"010C", "0104", "010D","0105","010F","0110","0111"};
```

Ilustración 41 – Vectores de comandos AT y PID

Una vez enviado un comando PID al dispositivo OBD2, éste lo reenvía a la ECU del vehículo, comprueba los estados de los sensores de los componentes requeridos y envía una respuesta. La respuesta llega a través del InputStream hasta nuestra aplicación y será necesario comprobar que ésta se trata de una respuesta válida. Para ello comprobamos que la respuesta posee un tamaño superior a 4 y que los dos primeros caracteres son los números “41”, esto indica que el mensaje que acabamos de recibir es una respuesta a la consulta de un componente del vehículo. Cualquier respuesta que no posea esos dos primeros caracteres será rechazada por no ser válida.

```
private final BroadcastReceiver mensajeRecibidoBroadcastReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String respuesta = intent.getStringExtra( name: "mensaje");
        String subcadena = "";
        String valorRespuesta = "";

        respuesta = comprobarFormatoRespuesta(respuesta);

        //Conversión a hexadecimal de la respuesta
        if (configuracionRealizada) {
            if (respuesta.length() > 4) {
                if (respuesta.substring(0, 2).equals("41")) {
                    subcadena = respuesta.substring(0, 4);
                    valorRespuesta = conversionRespuesta(subcadena, respuesta);

                    valorRespuesta = "," + valorRespuesta;
                    cadenaSoluciones += valorRespuesta;
                }
            }
        }
        enviarComando();
    }
};
```

Ilustración 42 – Código ejecutado al recibir un mensaje del dispositivo OBD2

El siguiente paso sería convertir la respuesta recibida de formato hexadecimal a decimal para poder realizar los cálculos pertinentes. El método encargado de ello es “conversionRespuesta” que recibe como parámetros la respuesta y una subcadena con los cuatro primeros caracteres de la misma para identificar a qué comando pertenece la respuesta. Este método primero identifica si la respuesta es de longitud seis u ocho, hay que recordar tal y como vimos en la primera iteración, que existen comandos que reciben respuestas de longitud seis, como por ejemplo el comando que solicita el valor de la velocidad del vehículo y otros que reciben respuestas de longitud ocho, como el de las revoluciones del vehículo. Esta identificación nos servirá para la conversión de los caracteres hexadecimales a decimal, después se identificará a qué comando pertenece la respuesta atendiendo al valor de la subcadena pasada como parámetro al método. Por ejemplo, si el mensaje recibido es una respuesta al comando de la velocidad, la subcadena sería “410D”, si lo fuera de la temperatura del líquido refrigerante, “4105”. Una vez identificada la respuesta aplicamos la fórmula correspondiente para obtener el valor real del componente.

```
private String conversionRespuesta(String subcadena, String respuesta) {

    String solucion = "";
    Integer A = 0;
    Integer B = 0;
    Float formula;

    try{
        A = Integer.parseInt(respuesta.substring(4, 6),  radix: 16);
        if(respuesta.length() == 8) { //Diferenciar respuestas de tamaño 6 u 8
            B = Integer.parseInt(respuesta.substring(6, 8),  radix: 16);
        }
    }catch (NumberFormatException nFE) {
        Log.d(TAG,  msg: "conversionRespuesta: error al convertir la respuesta a enteros");
        return solucion;
    }

    switch (subcadena) {
        case "410C": //ENGINE RPM
            formula = ((A * 256f) + B) / 4f;
            solucion = Float.toString(formula);
            rpm.setText(solucion);
            break;

        case "4104": //ENGINE LOAD
            formula = (A * 100f) / 255f;
            solucion = Float.toString(formula);
            engineLoad.setText(solucion);
            break;

        case "410D": //VEHICLE SPEED
            VSS = (float)A;
```

Ilustración 43 –Conversión de la respuesta a decimal e identificación de la misma

Una vez procesada la respuesta se actualizan las correspondientes casillas en la interfaz de la aplicación y se vuelve a llamar al método de enviar el siguiente comando.

Otro aspecto muy importante para el correcto funcionamiento de esta aplicación es el hecho de que, al igual que aplicaciones como juegos o reproductores de videos y películas, debe evitar que la pantalla del móvil se bloquee. El usuario en todo momento ha de saber que la aplicación está funcionando correctamente y que la comunicación con el dispositivo OBD2 sigue activa. Esto lo consigue gracias a las actualizaciones de la interfaz de la aplicación con los datos que llegan desde el vehículo. Si el móvil se bloquea y la pantalla se apaga el usuario no podrá saber si la aplicación continúa funcionando o si ha dejado de funcionar. Además, aunque el móvil tenga la pantalla continuamente encendida, lo que deriva en un mayor consumo de batería, es totalmente necesario evitar que el usuario

intente desbloquear el teléfono móvil mientras conduce para comprobar la aplicación, pues podría derivar en una distracción y provocar un accidente.

Para conseguir que la pantalla no se bloquee simplemente añadimos en el fichero layout XML de la actividad el siguiente trozo de código[13]:

```
android:keepScreenOn="true">
```

Ilustración 44 – Código para evitar que la pantalla del móvil se bloquee

4.3.6. Pruebas realizadas

Tras el desarrollo de la aplicación se llevaron a cabo una serie de pruebas para probar el funcionamiento de la misma. Las pruebas realizadas permitieron comprobar el correcto funcionamiento de la aplicación y corregir pequeños errores en el código y en la conectividad Bluetooth. Sin embargo, hubo un error que no pudo corregirse en esta iteración. Se trataba de un fallo de la aplicación que ocurría tras unos minutos de correcto funcionamiento, debido a que nuestro dispositivo móvil recibía del OBD2 respuestas con caracteres no ASCII intercalados. Estas respuestas eran totalmente válidas, pero sucedía que contenían caracteres que no eran esperados y por tanto debidamente tratados.

Por otro lado, se pudo comprobar el correcto funcionamiento de la comunicación con la ECU del vehículo y las respuestas recibidas de ésta como puede observarse en la ilustración 45.

```
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C1200
D/MainActivity: Se va a enviar un nuevo comando: 1
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 410455
D/MainActivity: Se va a enviar un nuevo comando: 2
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010D
D/BluetoothConnectionServ: InputStream: 410D00
D/MainActivity: Se va a enviar un nuevo comando: 3
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C1233
D/MainActivity: Se va a enviar un nuevo comando: 1
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 41045B
D/MainActivity: Se va a enviar un nuevo comando: 2
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010D
D/BluetoothConnectionServ: InputStream: 410D00
D/MainActivity: Se va a enviar un nuevo comando: 3
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C127D
D/MainActivity: Se va a enviar un nuevo comando: 1
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
```

Ilustración 45 – Captura de la consola de Android Studio durante la ejecución de la aplicación

```
D/BluetoothConnectionServ: InputStream: 410D00
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C0D4D
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 410►►435
D/AndroidRuntime: Shutting down VM
E/AndroidRuntime: FATAL EXCEPTION: main
                  Process: com.example.juancarlos.obdapp, PID: 18022
                  java.lang.StringIndexOutOfBoundsException: length=7; regionStart=6; regionLength=2
                      at java.lang.String.substring(String.java:1931)
                      at com.example.juancarlos.obdapp.MainActivity.conversionRespuesta(MainActivity.java:600)
                      at com.example.juancarlos.obdapp.MainActivity.access$800(MainActivity.java:50)
                      at com.example.juancarlos.obdapp.MainActivity$8.onReceive(MainActivity.java:349)
                      at android.support.v4.content.LocalBroadcastManager.executePendingBroadcasts(LocalBroadcastManager.java:308)
                      at android.support.v4.content.LocalBroadcastManager.access$000(LocalBroadcastManager.java:46)
                      at android.support.v4.content.LocalBroadcastManager$1.handleMessage(LocalBroadcastManager.java:118)
                      at android.os.Handler.dispatchMessage(Handler.java:102)
                      at android.os.Looper.loop(Looper.java:154)
                      at android.app.ActivityThread.main(ActivityThread.java:6119) <1 internal calls>
                      at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:886)
                      at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:776)
D/BluetoothConnectionServ: InputStream: >
Application terminated.
```

Ilustración 46 – Error al recibir del OBD2 respuestas con caracteres no ASCII

4.3.7. Reunión

En la reunión realizada al término de la segunda iteración se mostró al tutor los avances realizados y los resultados obtenidos. El tutor valoró positivamente el desarrollo de la aplicación y sugirió la necesidad de buscar el origen del error de las respuestas con caracteres no ASCII y de encontrar una solución. Además, se estableció como nuevo objetivo para la siguiente iteración la necesidad de

almacenar los valores de las respuestas obtenidas del vehículo para un posterior procesamiento de las mismas.

Como último objetivo, se acordó el inicio de la segunda parte del proyecto que consistiría en el desarrollo de una aplicación que permitiese utilizar los datos obtenidos del vehículo para representarlos en gráficas y trazar sobre un mapa el trayecto realizado por el vehículo, teniendo como objetivo adicional medir, en la aplicación desarrollada en la iteración 2, la geolocalización del vehículo durante el trayecto realizado mediante los valores de la longitud y latitud.

4.4. Iteración 3 - Localización y almacenamiento de los datos recogidos.

4.4.1. Corrección de las respuestas erróneas

Como se pudo comprobar en las pruebas realizadas en la iteración anterior, ocurría que a veces el dispositivo OBD2 respondía a nuestra aplicación con respuestas que contenían valores no ASCII intercalados. Tras investigar acerca del origen de este posible error en la documentación proporcionada por ELM Electronics y en la red no se encontró un caso similar a este, por lo que se decidió corregir el problema comprobando el valor de cada respuesta recibida, recorriendo ésta carácter a carácter, eliminando aquellos caracteres erróneos que no fueran ni números ni letras o que simplemente no fueran caracteres ASCII. Para este procedimiento se emplean expresiones regulares, si se localiza una ocurrencia de la expresión regular en la respuesta, el carácter o los caracteres en cuestión se sustituyen por un carácter vacío.

```
public String comprobarFormatoRespuesta(String respuesta){
    try{
        //Eliminamos símbolos o respuestas no válidas
        respuesta = respuesta.replace( target: "null",  replacement: "");
        respuesta = respuesta.replaceAll( regex: "\n",  replacement: "");
        respuesta = respuesta.replaceAll( regex: "\r",  replacement: "");
        respuesta = respuesta.replaceAll( regex: "[^\\p{ASCII}]",  replacement: "");

    }catch(NullPointerException e){
        respuesta = "";
        return respuesta;
    }

    //Comprobamos que todos los caracteres son números o letras
    if(respuesta.matches( regex: "[a-zA-Z0-9]*")){
        return respuesta;
    }else{
        for(int i=0;i<respuesta.length();i++){
            char caracter = respuesta.charAt(i);
            if(!Character.isLetter(caracter) && !Character.isDigit(caracter)){
                String cambio = Character.toString(caracter);
                respuesta = respuesta.replaceAll(cambio, replacement: "");
            }
        }
        return respuesta;
    }
}
```

Ilustración 47 – Función que elimina caracteres no válidos de la respuesta

4.4.2. Geolocalización del dispositivo

Para medir la geolocalización del dispositivo cuando la aplicación estuviera en ejecución se hizo uso de los servicios de Google para la localización, en concreto la API **FusedLocationProviderClient**[8]. Para ello es necesario establecer como dependencia en nuestro proyecto el uso de este servicio y añadir los permisos adecuados.

```
implementation 'com.google.android.gms:play-services-location:15.0.1'
```

Ilustración 48 – Nueva dependencia para el uso de la localización

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Ilustración 49 – Permisos para la localización

Para la gestión de la localización serán muy importantes los siguientes elementos:

- **LocationRequest:** es un objeto con el que podremos definir algunas características en la recepción de los valores de geolocalización, como son el intervalo de tiempo entre cada medición o la precisión de la medida.

```
//Inicialización de los parámetros de la localización GPS
fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);
locationRequest = new LocationRequest();
locationRequest.setInterval(1000); //Intervalo de tiempo entre cada actualización de la loca
locationRequest.setFastestInterval(500);
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

Ilustración 50 – Inicialización del LocationRequest y los intervalos entre medidas

- **LocationCallback:** actúa como un listener o una función que se ejecuta cada vez que se recibe un nuevo valor de la localización del dispositivo.

```
locationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
        super.onLocationResult(locationResult);
        for (Location location : locationResult.getLocations()) {
            if (location != null) {
                latitud = location.getLatitude();
                longitud = location.getLongitude();
            }
        }
    }
};
```

Ilustración 51 – Listener que se ejecuta cada vez que cambian las coordenadas

- Los métodos **StartLocationUpdates** y **StopLocationUpdates:** para iniciar y finalizar las mediciones de localización.

```
//Activa la actualización automática del GPS
private void startLocationUpdates() {
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        fusedLocationProviderClient.requestLocationUpdates(locationRequest, locationCallback,  looper: null);
    }else{
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            requestPermissions(new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, MY_PERMISSION_REQUEST_FINE_LOCATION);
        }
    }
}
```

Ilustración 52 – Método para iniciar la medición de la localización del dispositivo

```
//Desactiva la actualización automática del GPS
private void stopLocationUpdates() {
    fusedLocationProviderClient.removeLocationUpdates(locationCallback);
}
```

Ilustración 53 – Método para finalizar la medición de la localización del dispositivo

Cada vez que se obtenga una nueva localización para el dispositivo el LocationCallback guardará las nuevas coordenadas en ese momento dado.

4.4.3. Almacenamiento de los datos tomados del vehículo

En la reunión que se produjo al finalizar la segunda iteración del proyecto se acordó establecer un método para almacenar los valores de los parámetros del vehículo obtenidos durante las peticiones para poder consultarlos y procesarlos a posteriori.

El método diseñado se complementa con el funcionamiento del método que envía comandos PID al dispositivo OBD2. Cada vez que enviamos un comando, se almacena la respuesta en una cadena que se concatena posteriormente con la respuesta del siguiente comando. Cuando enviamos una ráfaga de comandos completa, es decir, cuando llegamos al tope del array de comandos PID, se crea una cadena con el tiempo desde que se inició la comunicación con el dispositivo OBD2, la longitud y la latitud del dispositivo móvil y la cadena con las respuestas a los comandos.

```
tiempo transcurrido, longitud, latitud, RPM, velocidad, carga motor, temperatura, ....
```

Ilustración 54 – Cadena formada al finalizar una ráfaga de comandos

Las cadenas con el tiempo transcurrido, la localización y las respuestas a los comandos se almacenan en una estructura tipo vector que luego nos servirá para conservar los resultados una vez terminada la comunicación con el dispositivo OBD2.

```
if(indicePID == commandsPID.length){
    long tiempoActual = System.currentTimeMillis();
    long tiempoTranscurrido = tiempoActual - tiempoInicio;
    String entrada = Long.toString(tiempoTranscurrido) + "," + Double.toString(longitud) + "," + Double.toString(latitud) + cadenaSoluciones;
    registros.add(entrada);
    indicePID = 0;
    cadenaSoluciones = "";
```

Ilustración 55 – Creación de una nueva medición completa tras enviar una ráfaga de comandos

Cuando el usuario pulse el botón de finalizar la comunicación, además de cerrar la conexión Bluetooth con el OBD2, se creará un fichero en formato txt donde escribiremos todas las cadenas obtenidas durante el envío de las ráfagas de comandos. Este fichero se almacenará en nuestro dispositivo móvil para un uso posterior.

A la función que crea y escribe este fichero se le proporciona un directorio donde almacenar el fichero, se crea una instancia del mismo y se recorre el vector donde se habían almacenado las cadenas con las mediciones guardando cada cadena en una nueva línea del fichero.

```
public void guardarArchivoRegistro(){
    date = new Date();
    fechaString = dateFormat.format(date);
    String nombreArchivo = "registro" + fechaString + ".txt";

    //Crear el directorio donde de almacenará el archivo
    File folder = new File( pathname: Environment.getExternalStorageDirectory().getPath() + "/Documents/carpeta_registros");
    if (!folder.mkdirs()) {
        Log.d(TAG , msg: "guardarArchivo: directorio no creado");
    }

    try {
        //Crear archivo
        File file = new File(folder,nombreArchivo);
        file.createNewFile();

        FileOutputStream fos = new FileOutputStream(file);
        OutputStreamWriter outputStream = new OutputStreamWriter(fos);

        for (int i=0; i<registros.size(); i++) {
            outputStream.append(registros.get(i));
            outputStream.append("\n\r");
        }
        outputStream.close();
        fos.close();
    }catch (FileNotFoundException e){
        Log.d(TAG,  msg: "guardarArchivo: archivo no encontrado");
        e.printStackTrace();
    }catch(IOException e){
        Log.d(TAG,  msg: "guardarArchivo: error al guardar");
```

Ilustración 56 – Método que almacena un fichero txt con los datos obtenidos del vehículo

El fichero se crea con el nombre “registro” más la fecha exacta en la que se creó, por ejemplo, “registro2018-07-21-10.23.55.txt” y se almacena en una carpeta creada específicamente para la aplicación, llamada **carpeta_registros** dentro del directorio **Documents** en la memoria interna de Android.

Es muy importante especificar que la fecha que se concatena al nombre del fichero no puede contener caracteres como “/”, porque sería interpretado como una ruta en vez del nombre de un fichero. De ahí que se haya utilizado guiones para separar la fecha.[9]

4.4.4. Pruebas realizadas

Durante las pruebas realizadas en esta iteración se consiguió probar de manera satisfactoria los cambios realizados en la aplicación tanto en la medición de la localización como en el almacenamiento de los parámetros captados del vehículo.

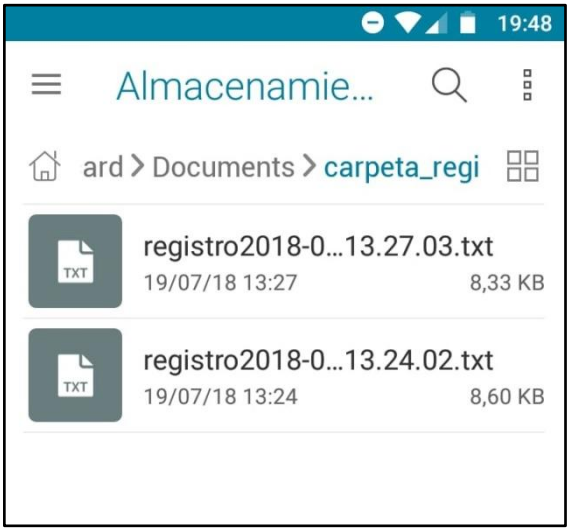


Ilustración 57 – Almacenamiento de los ficheros creados por la aplicación

```
2018-07-19 13.26.51,-3.7791613,37.7899064,911,27,5
2018-07-19 13.26.51,-3.7791613,37.7899064,955,13,5
2018-07-19 13.26.52,-3.7791613,37.7899064,903,23,5
2018-07-19 13.26.52,-3.7791676,37.7898988,887,20,5
2018-07-19 13.26.52,-3.7791676,37.7898988,880,20,4
2018-07-19 13.26.52,-3.7791676,37.7898988,872,21,4
2018-07-19 13.26.52,-3.7791676,37.7898988,864,21,3
```

Ilustración 58 – Contenido de los ficheros guardados

Los nuevos cambios introducidos corrigieron el problema de las respuestas con caracteres no ASCII que provocaban fallos y la caída de la aplicación. Las nuevas pruebas realizas permitieron también conocer qué tiempos tomaban la comunicación entre la aplicación y el dispositivo OBD2 y se pudo observar que se realizaban hasta cuatro ráfagas de consultas por segundo, lo que provocó la necesidad de establecer un retardo en el envío de los comandos al dispositivo OBD2 para que no se enviaran tantas peticiones en poco tiempo, ya que eso resultaría de poco provecho y generaría ficheros muy extensos con muchas entradas.

4.4.5. Reunión

Tras la reunión realizada al término de la tercera iteración el tutor valoró positivamente los avances desarrollados y secundó la necesidad de establecer un retardo a la hora de enviar los comandos al dispositivo OBD2 para evitar realizar mediciones innecesarias.

Por otro lado, el tutor sugirió continuar con el desarrollo de la aplicación que procesara los datos recabados del vehículo. De esta forma se estableció como nuevo objetivo desarrollar una aplicación que mostrara en gráficas los valores obtenidos y trazara sobre un mapa el trayecto realizado por el vehículo.

4.5. Iteración 4 - Procesamiento de los datos recogidos

Durante esta iteración se corrigieron los errores encontrados en las pruebas realizadas en la tercera iteración y se inició el diseño y la implementación de la aplicación encargada de procesar los datos recabados del vehículo y que los mostrara de manera cualitativa y cuantitativa a través de gráficos y con el trazado de la ruta realizada sobre un mapa.

4.5.1. Retardos en el envío de los comandos

Para la corregir el envío de múltiples ráfagas de comandos al dispositivo OBD2 en un mismo segundo se estableció un retardo de un segundo para asegurarnos de que se enviara una única ráfaga de comandos por segundo. De esta forma sólo escribiremos una línea en nuestro fichero del registro del trayecto por cada segundo de comunicación con el OBD2.

```
if(indicePID == commandsPID.length){
    long tiempoActual = System.currentTimeMillis();
    long tiempoTranscurrido = tiempoActual - tiempoInicio;
    String entrada = Long.toString(tiempoTranscurrido) + "," + Double.t
    registros.add(entrada);
    indicePID = 0;
    cadenaSoluciones = "";

    //Delay de 1 segundo
    try {
        Thread.sleep( millis: 1000);
    }catch(InterruptedException ex) {
        Log.d(TAG,  msg: "mensajeRecibido: el delay falló");
    }
}
command = commandsPID[indicePID];
command += "\r";
indicePID++;
bytes = command.getBytes();
```

Ilustración 59 – Retardo añadido tras enviar una ráfaga de comandos

4.5.2. Diagrama de clases

A continuación se muestra el diagrama de clases para la aplicación de visualización y monitorización de los datos recabados del vehículo.

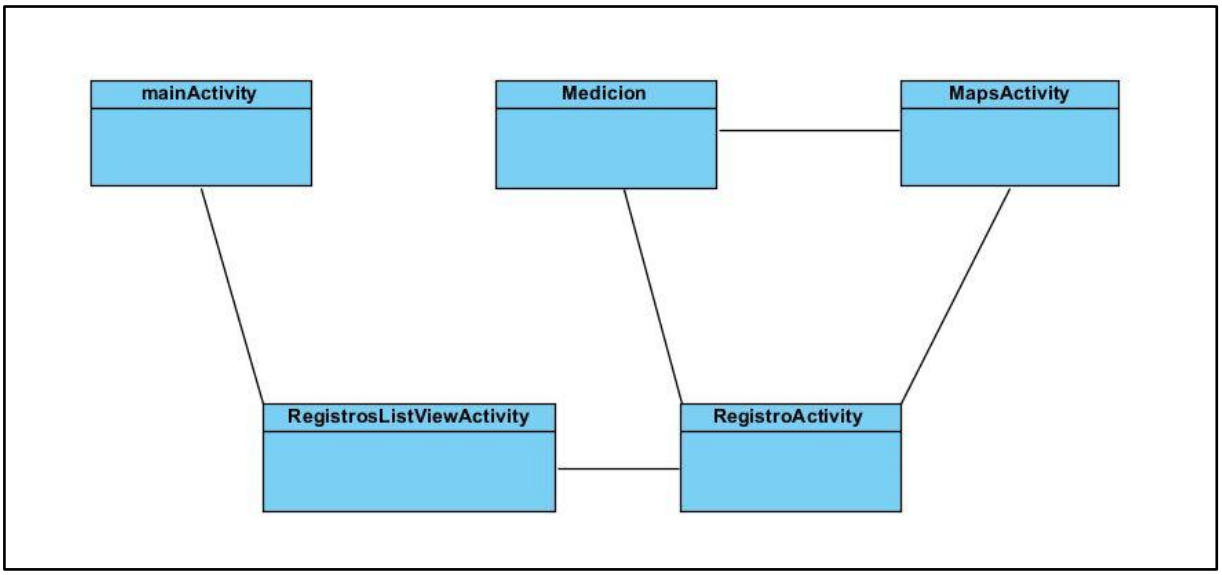


Ilustración 60 – Diagrama de clases de la aplicación de visualización de datos

- **MainActivity**: es la clase que implementa la funcionalidad de la pantalla inicial de la aplicación.

- **RegistrosListViewActivity**: esta clase es la encargada de listar todos los ficheros almacenados en el dispositivo móvil por la aplicación de captura de datos con el fin de que el usuario pueda escoger uno.
- **Medicion**: es una clase auxiliar que nos permitirá almacenar los valores de los datos leídos de un fichero. Cada objeto Medicion representará una línea de dicho fichero.
- **RegistroActivity**: es la clase que gestiona la pantalla donde se muestran las gráficas de valores e información relacionada con el fichero seleccionado.
- **MapsActivity**: esta última clase es la encargada de gestionar la pantalla en la que se muestra el mapa y de realizar el trazado de rutas.

4.5.3. Diagrama de casos de uso

El diagrama de casos de uso define la interacción básica y esencial que el usuario tendrá con la aplicación móvil.

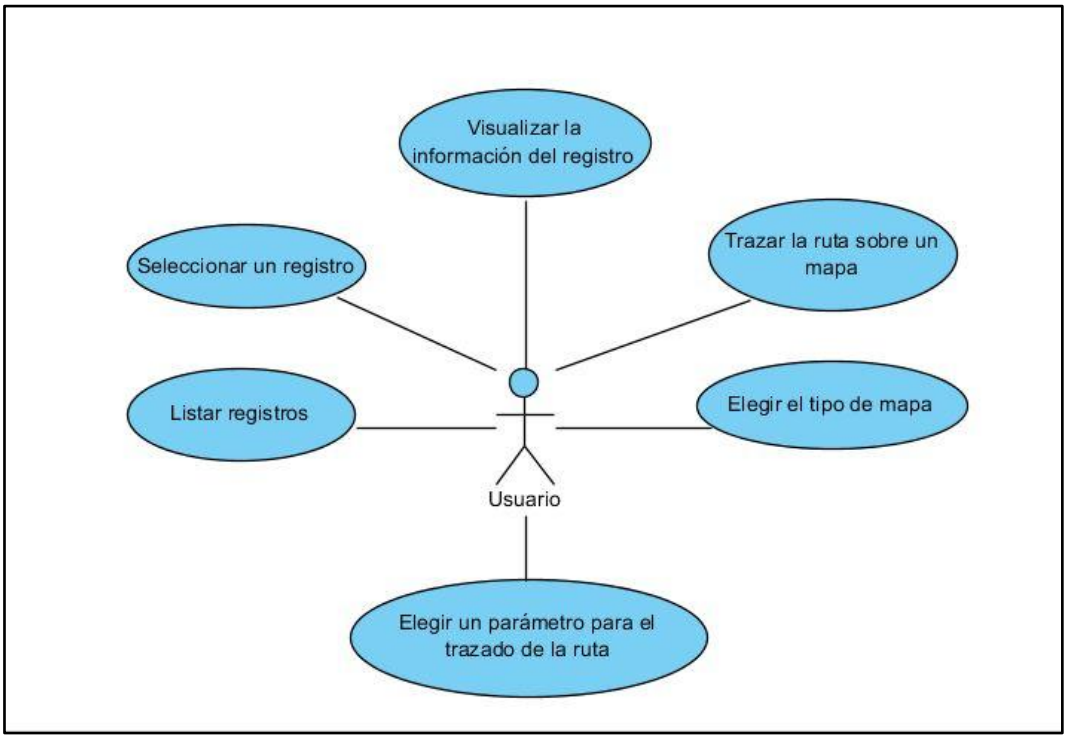


Ilustración 61 – Diagrama de casos de uso

A continuación se describe de manera detallada cada caso de uso:

- **Listar registros:** el usuario puede comprobar en un listado todos los ficheros con datos tomados de trayectos realizados con el vehículo.
- **Seleccionar un registro:** el usuario selecciona uno de los registros listados.
- **Visualizar la información del registro:** la información que almacena el registro se representa en gráficas.
- **Trazar la ruta sobre el mapa:** se traza sobre un mapa de Google el trayecto realizado por el vehículo en basándose en las coordenadas capturadas por la aplicación.
- **Elegir el tipo de mapa:** permite cambiar el estilo del mapa sobre el que se traza la ruta, por ejemplo, mapa tipo carretera, tipo satélite, híbrido, etc.
- **Elegir un parámetro para el trazado de la ruta:** permite colorear el trazado de la ruta con diferentes colores en función de los distintos valores de un parámetro del vehículo. Sería de especial utilidad para marcar las zonas de mayor o menor esfuerzo o consumo por parte del vehículo.

4.5.4. Diagrama de flujo o actividades

A través del diagrama de actividades podemos observar la secuencia principal de interacción que el usuario establecerá con la aplicación y partes del funcionamiento de la misma que son algunas invisibles para el usuario, pero necesarios para un uso correcto.

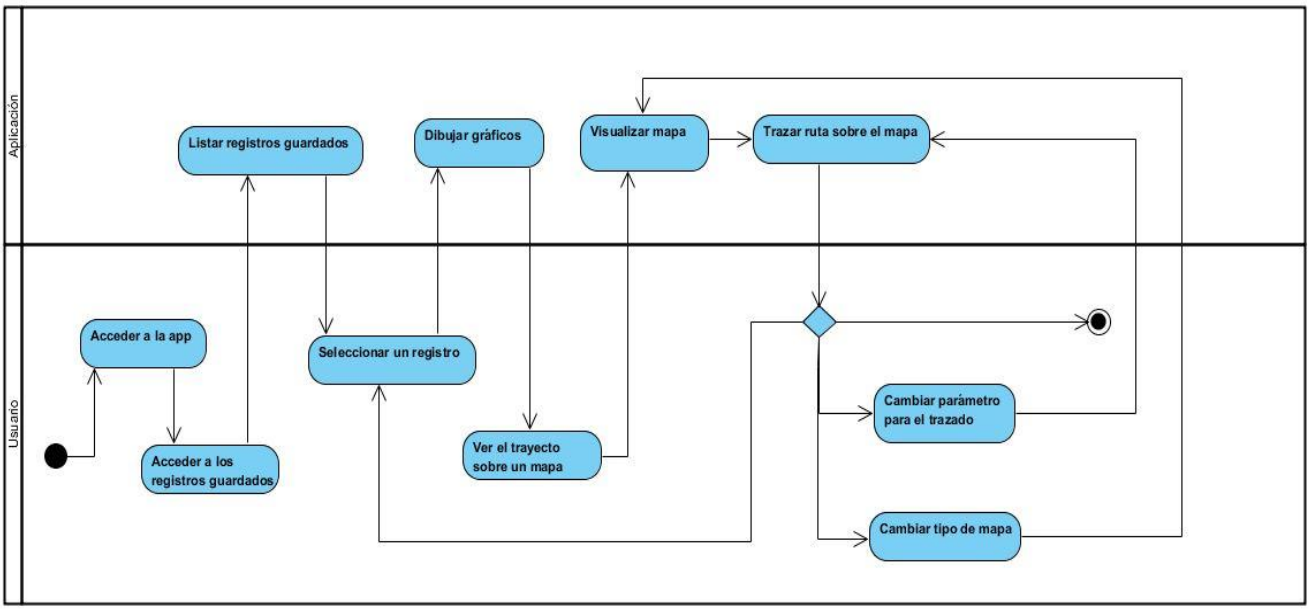


Ilustración 62 – Diagrama de actividades

El diagrama está dividido mediante un swinlane en dos carriles para identificar qué actividades realiza el usuario y qué actividades realiza la aplicación. La descripción de cada actividad se verá de forma más detallada en el apartado de diseño con storyboards.

4.5.5. Diseño con storyboards

La primera pantalla que encuentra el usuario mostrará un texto introductorio y un botón para acceder a los distintos registros grabados por la aplicación de captura de datos.

Ilustración 63 – Prototipo de la pantalla de inicio

Al pulsar el botón de registros accedemos a una nueva pantalla donde se listan todos los trayectos guardados. Los trayectos se guardan cada uno en un fichero en formato txt que almacena los parámetros recogidos del vehículo.

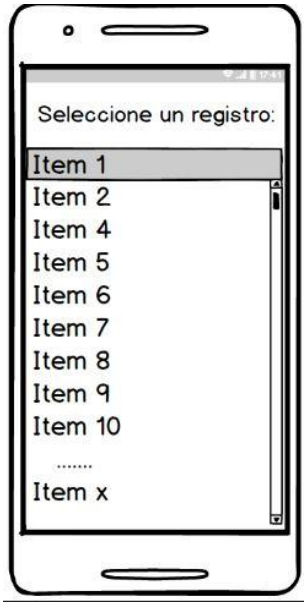


Ilustración 64 – Prototipo de la pantalla con el listado de registros/ficheros almacenados

Al seleccionar cualquiera de los registros en la lista, se pasará a una nueva pantalla donde se mostrarán en gráficas de manera cuantitativa los valores de los parámetros del vehículo almacenados en el registro seleccionado.

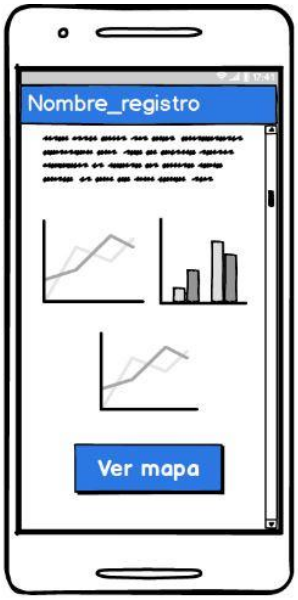


Ilustración 65– Prototipo de la pantalla donde se muestra la información cuantitativa del trayecto

La pantalla con información sobre el registro poseerá un botón para pasar a una nueva pantalla, donde se trazará en el mapa de Google el trayecto realizado por el vehículo por las distintitas calles y carreteras.

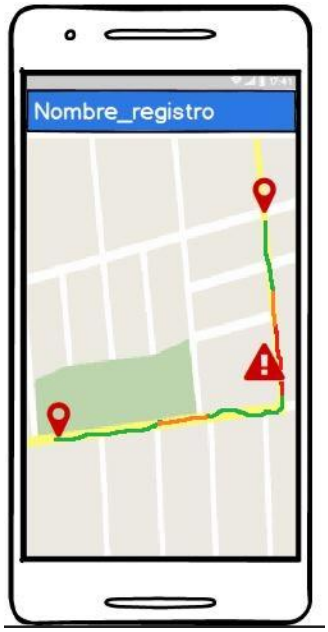


Ilustración 66 – Prototipo de la pantalla donde se muestra la información cualitativa del trayecto

En el trazado de la ruta del vehículo se señalizarán las zonas de mayor y menor consumo/esfuerzo del vehículo mediante un mapeado de colores. Las zonas clave donde se puede haber producido un cambio drástico de los valores, ya sea por pegar un acelerón o revolucionar demasiado el coche, se marcarán con iconos interactivos.

Ilustración 67 – Ejemplo de iconos interactivos dentro del mapa

Al pulsar un icono interactivo en la ruta se mostrará en pantalla un texto descriptivo de la razón por la cual se ha marcado ese punto para que sirva de incentivo al usuario en futuros trayectos con el objetivo de mejorar la conducción.

En la pantalla del mapa, el usuario dispondrá también de un botón para cambiar el estilo del mapa mostrado y otro botón para el elegir el tipo de parámetro con el cual queremos representar el grado de eficiencia de la conducción.

En la siguiente imagen podemos observar la secuencia de acciones que deberá realizar el usuario para navegar por las distintas pantallas de la aplicación:

Ilustración 68 – Secuencia de navegación por la aplicación

4.5.6. Desarrollo e implementación

El primer paso para la implementación de esta aplicación fue la creación de una pantalla que accediera a la memoria del teléfono, buscara la carpeta con los registros guardados y los mostrara en un listado en la aplicación.

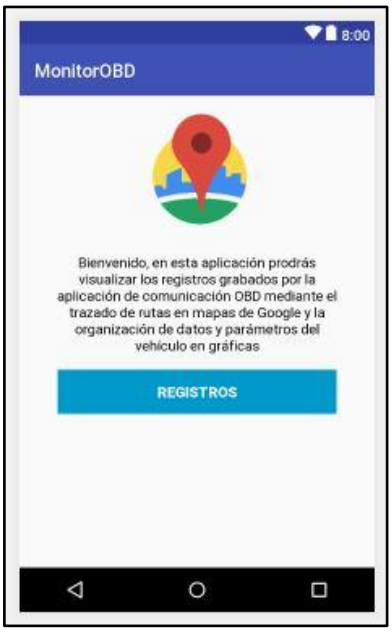


Ilustración 69– Pantalla de inicio de la aplicación

En la pantalla con el listado de registros guardados cada elemento de la lista se identificaría con el nombre del archivo. En el caso de que no hubiera ningún archivo guardado en la carpeta se mostraría un mensaje informando de ello al usuario.

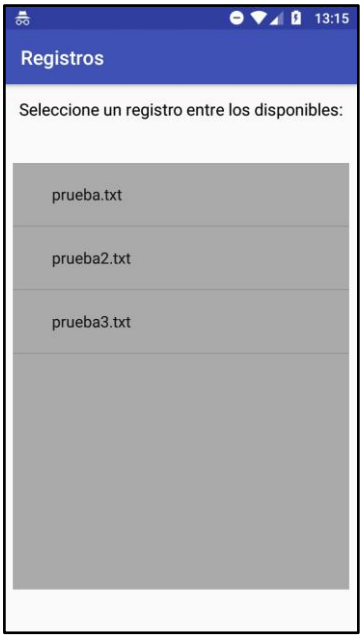


Ilustración 70 – Pantalla con el listado de registros guardados

Ilustración 71 – Pantalla cuando no hay registros almacenados

Para poder leer el contenido de la carpeta donde se almacena los ficheros desde nuestra aplicación necesitamos proporcionar el permiso de lectura, lo hacemos simplemente añadiéndolo al fichero **AndroidManifest** del proyecto.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Ilustración 72 – Permiso de lectura

Al código asociado a esta pantalla le pasamos un directorio y guardamos el nombre de los ficheros almacenados en él en una estructura tipo vector, que será utilizada luego para rellenar el listado y mostrarlo en pantalla.

```
public void leerCarpetaRegistros(){
    String ruta = "/storage/671B-1A10/carpeta_registros";
    File directorio= new File(ruta);
    File[] archivos = directorio.listFiles();

    if(archivos.length > 0){
        for(int i=0;i<archivos.length;i++){
            registros.add(archivos[i].getName());
        }
    }else{
        listaVacia.setVisibility(View.VISIBLE);
    }
}
```

Ilustración 73 – Método que lee los ficheros almacenados en la carpeta

Para rellenar la lista con los archivos encontrados le pasamos al objeto que representa la lista, un **ListView**, la estructura tipo vector con los nombres de los archivos.

```
final ArrayAdapter<String> adaptador = new ArrayAdapter<~>( context: this,android.R.layout.simple_expandable_list_item_1,registros);
listaRegistros.setAdapter(adaptador);
```

Ilustración 74 – Proceso de modificación del listado de registros

Cuando el usuario pulse en uno de los elementos de la lista pasará a una nueva pantalla donde se cargará la información contenida en el registro asociado a la entrada de la lista pulsada. Para ello se crea un objeto **Intent** que permite transferir datos entre distintas pantallas de una aplicación. En este caso el objeto Intent transferirá el nombre del registro pulsado para luego leer su contenido completamente y cargar sus datos en la nueva pantalla.

```
listaRegistros.setOnItemClickListener((adapterView, view, i, l) → {
        Intent intent = new Intent(getApplicationContext(),RegistroActivity.class);
        intent = intent.putExtra( name: "Registro_seleccionado", adaptador.getItem(i));
        startActivity(intent);
});
```

Ilustración 75 – Código ejecutado al pulsar en uno de los elementos del listado de registros

En la nueva pantalla se muestran los datos del registro de manera cuantitativa en gráficas de valores. Para cada parámetro medido se dibuja una gráfica diferente en la cual el eje de coordenadas Y representa el parámetro del vehículo medido y el eje de coordenadas X el transcurso del tiempo. De esta manera el usuario puede comprobar en cada gráfica la evolución de los valores medidos a lo largo del trayecto realizado.

Para el diseño de esta pantalla se utilizó un **ScrollView** para poder desplazarse por la misma debido a la imposibilidad de colocar todas las gráficas para cada uno de los parámetros en una pantalla estática.

Para el diseño y trazado de las gráficas se utilizó la librería **MPAndroidChart** disponible en GitHub, esta librería permite la representación de datos en múltiples tipos de gráficas, como gráficas de líneas, barras, sectores y un largo etc. Además, la librería permite personalizar totalmente el aspecto y diseño de las gráficas como

los colores, los bordes, las leyendas y otros pequeños detalles. **MPAndroidChart** dispone de una wiki en GitHub con toda la documentación necesaria que nos sirve a modo de manual para comenzar con el desarrollo de las gráficas.[11]

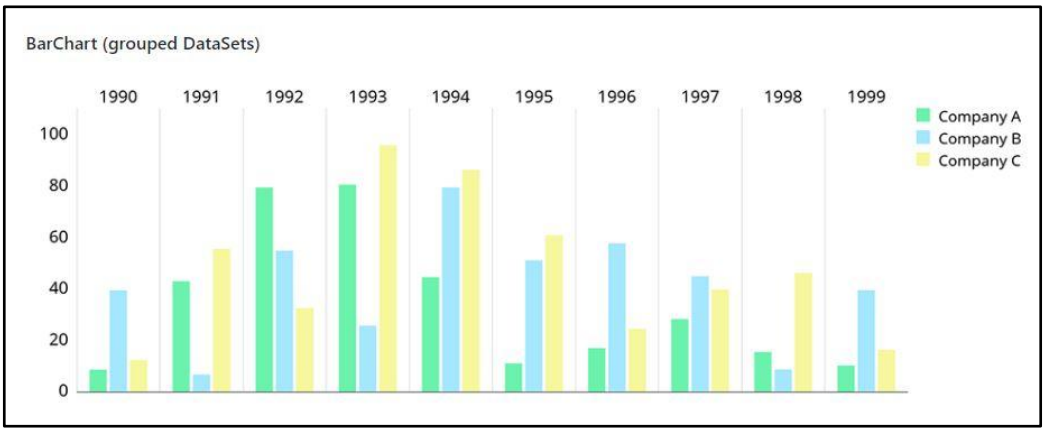


Ilustración 76 – Ejemplo de gráfica que se puede diseñar con MPAndroidChart

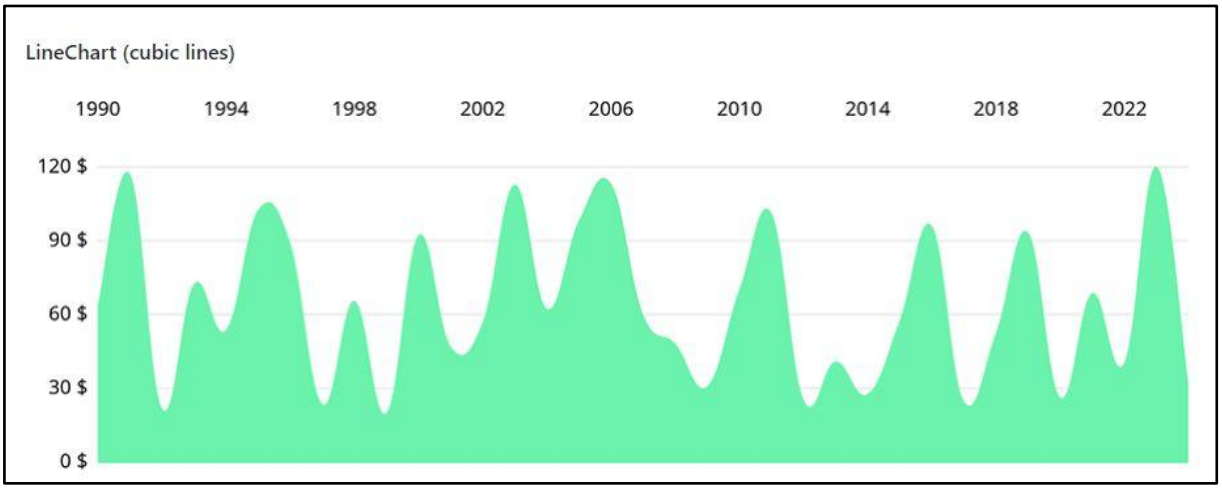


Ilustración 77 – Otro ejemplo de gráfica que se puede diseñar con MPAndroidChart

Para el uso de esta librería en nuestra aplicación debemos especificar en el Gradle del proyecto la URL de la misma y añadirla además como dependencia.

```
repositories {
    maven { url 'https://jitpack.io' }
}
```

Ilustración 78 – URL del repositorio de la librería

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.0.3'
```

Ilustración 79 – Dependencia añadida al Gradle del proyecto

Para dibujar las gráficas necesitamos leer del fichero del registro los valores del tiempo y de cada parámetro del vehículo en cada medición realizada, es decir, en cada línea del fichero. Reutilizando el método que permite acceder a la memoria y leer ficheros, recorremos línea a línea el fichero del registro y para cada línea creamos un objeto **Medición** compuesto por los siguientes atributos:

- **Tiempo:** instante de tiempo en el que se realizó la medición, nos servirá para definir el eje de coordenadas X de las gráficas.
- **Longitud:** valor de la coordenada longitud en el instante de la medición.
- **Latitud:** valor de la coordenada latitud en el instante de la medición
- **Rpm:** valor de las revoluciones del motor en el instante de la medición.
- **Velocidad:** valor de la velocidad del vehículo en el instante de la medición.
- **PosicionAcelerador:** valor de la posición del acelerador en el instante de la medición.
- **ConsumoInstantaneo:** valor del consumo instantáneo del vehículo en el instante de la medición.
- **TempRefrigerante:** valor de la temperatura del líquido refrigerante en el instante de la medición.
- **TempMotor:** valor de la temperatura del aire de admisión al motor en el instante de la medición.

```
public class Medicion {

    private String tiempo;
    private String longitud;
    private String latitud;
    private String velocidad;
    private String rpm;
    private String posicionAcelarador;
    private String consumoInstantaneo;
    private String tempRefrigerante;
    private String tempMotor;
```

Ilustración 80 – Objeto Medición y sus atributos

Los objetos Medición creados por cada línea leída del fichero se almacenan en una estructura tipo vector para poder ser utilizados posteriormente al dibujar las gráficas. Como hemos mencionado anteriormente, el atributo tiempo definirá el eje de coordenadas X de cada una de las gráficas. El resto de los atributos exceptuando la longitud y la latitud se emplearán para definir los ejes coordenadas Y de cada una de sus respectivas gráficas.

Todos las gráficas que van ser mostradas en pantalla serán gráficas de líneas, para crear gráficas de este tipo necesitamos crear un objeto **LineChart** proporcionado por la librería. Este objeto referenciará en todo momento a la gráfica que vamos a dibujar y podremos editar el estilo de ciertas características de la misma, como por ejemplo dibujar bordes alrededor de la gráfica o presentar una descripción o una leyenda.

```
//Método que dibuja un gráfico de líneas para las revoluciones
private void dibujarGraficoRPM(){
    Description description = new Description();
    description.setText("Gráfica de revoluciones");
    description.setTextSize(10);
    graficoRPM.setDescription(description);
    graficoRPM.setData(getLineData(valoresRPM, leyenda: "Revoluciones del motor"));
    graficoRPM.invalidate();
    graficoRPM.setDrawBorders(true);
    graficoRPM.setBorderColor(Color.BLACK);
    graficoRPM.setBorderWidth(2);
    graficoRPM.setScaleMinima( scaleX: 120f, scaleY: 1f);

    axisX(graficoRPM.getXAxis());
    axisLeft(graficoRPM.getAxisLeft());
    axisRight(graficoRPM.getAxisRight());
}
```

Ilustración 81 – Método que permite dibujar una gráfica de líneas en pantalla y editar ciertas opciones de personalización de la misma

Otro tipo de elementos que se pueden personalizar son los ejes X e Y donde se van mostrar los valores numéricos. Podemos establecer que los valores del X aparezcan en la parte superior de la gráfica o en la parte inferior. De la misma forma podemos elegir si los valores del eje Y aparecen a la izquierda de la gráfica o a la derecha. En la siguiente ilustración podemos observar que el eje X recibe la

estructura donde se han almacenado los valores del tiempo leídos del registro seleccionado.

```
//Método que personaliza el eje de coordenadas X
private void axisX(XAxis axis){
    axis.setGranularityEnabled(true);
    axis.setPosition(XAxis.XAxisPosition.BOTTOM);
    axis.setValueFormatter(new IndexAxisValueFormatter(tiempo));
}

//Método que personaliza el eje de coordenadas Y de la izquierda
private void axisLeft(YAxis axis){
    axis.setSpaceTop(30);
    axis.setAxisMinimum(0);
}
```

Ilustración 82 – Personalización de los ejes de coordenadas

Una vez tenemos los ejes definidos y personalizados debemos configurar la información que se mostrará en la gráfica, para ello haremos uso de un objeto conocido como **LineDataSet** o lo que es lo mismo, el conjunto de datos para el gráfico de líneas. A este objeto se le pasará la estructura que almacena el conjunto de valores de un parámetro específico del vehículo. Por ejemplo, si queremos dibujar la gráfica de las revoluciones del motor al **LineDataSet** le pasaremos la estructura que contiene todos los valores de las revoluciones leídos del registro que hemos seleccionado. El objeto LineDataSet también nos permitirá editar la forma en la que se visualizará la línea de valores de múltiples maneras.

```
//Método que genera una estructura tipo LineData con los datos que queremos mostrar
private LineData getLineData(Float[] parametro, String leyenda){
    LineDataSet lineDataSet = (LineDataSet)getData(new LineDataSet(getEntries(parametro),leyenda));

    lineDataSet.setLineWidth(3);
    lineDataSet.setCircleRadius(4);
    lineDataSet.setCircleColor(Color.BLACK);
    lineDataSet.setCircleColorHole(Color.BLACK);
    lineDataSet.setDrawCircleHole(true);
    lineDataSet.setDrawCircles(true);
    lineDataSet.setDrawHorizontalHighlightIndicator(false);
    lineDataSet.setDrawHighlightIndicators(false);
    lineDataSet.enableDashedLine( lineLength: 15f, spaceLength: 6f, phase: 0);

    LineData lineData = new LineData(lineDataSet);
    return lineData;
}
```

Ilustración 83 – Configuración del LineDataSet

El resultado tras configurar las opciones de personalización, definir los ejes y el LineDataSet sería similar al mostrado en la ilustración 84.

Ilustración 84 – Pantalla con los valores de los parámetros representados en gráficas

La pantalla que muestra las gráficas contiene también un botón para iniciar una nueva pantalla donde se cargará un mapa y se trazará la ruta realizada por el vehículo en función de las coordenadas recogidas. Para crear la pantalla en la que se muestra el mapa se utilizó una nueva Activity de Android Studio conocida como **MapsActivity**, ésta permite cargar mapas y mostrarlos en pantalla haciendo uso de la API de Google Maps. Esta API se conecta automáticamente con los servidores de Google Maps, muestra el mapa en pantalla y además permite editarlo pudiendo añadir polígonos, trazar líneas o colocar iconos y marcas interactivas. Para el uso de esta API necesitamos pasarle a nuestro código una clave que se genera en la web de **Google API Console**[10].En ella crearemos un nuevo proyecto al cual se le asociará una clave para el uso de la API. Posteriormente tendremos que agregar algunos servicios para mapas al proyecto. Los servicios habilitados para este proyecto fueron los siguientes:

- **Maps SDK Android**: para poder cargar los mapas en la aplicación.

- **Google Maps Geocoding API:** para convertir coordenadas geográficas en direcciones.
- **Google Maps Geolocation API:** para obtener datos de localización.
- **Google Maps Directions API:** para el trazado de rutas entre distintos puntos del mapa.
- **Places SDK para Android:** para obtener información detallada de miles de lugares. Se utilizaría para mostrar en el mapa nombres de lugares emblemáticos, monumentos, edificios institucionales, etc.

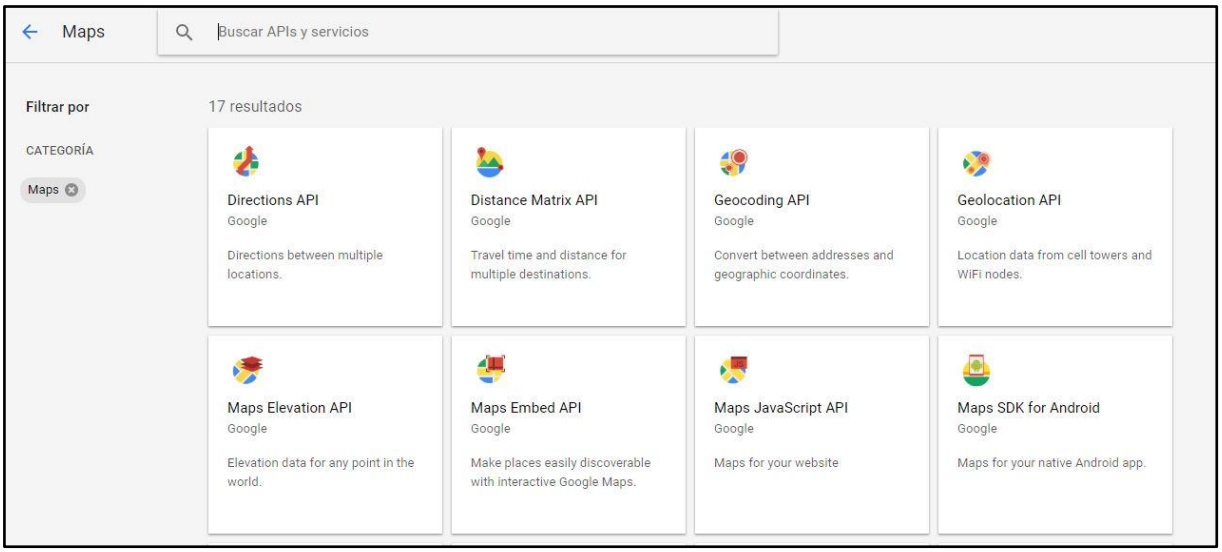


Ilustración 85 – Servicios y APIs que podemos añadir a nuestro proyecto

Una vez elegidos los servicios con los que complementaremos nuestro proyecto podremos tomar la clave proporcionada y añadirla a nuestro proyecto de Android Studio. Para ello buscaremos en la carpeta “values” del proyecto de Android Studio un archivo xml que se creó automáticamente al añadir la actividad MapsActivity. El archivo xml posee el nombre de “google_maps_api.xml” y en su interior debemos colocar la clave que la web de Google nos proporcionó para el uso de la API, tal y como podemos comprobar en las ilustraciones 86, 87 y 88:

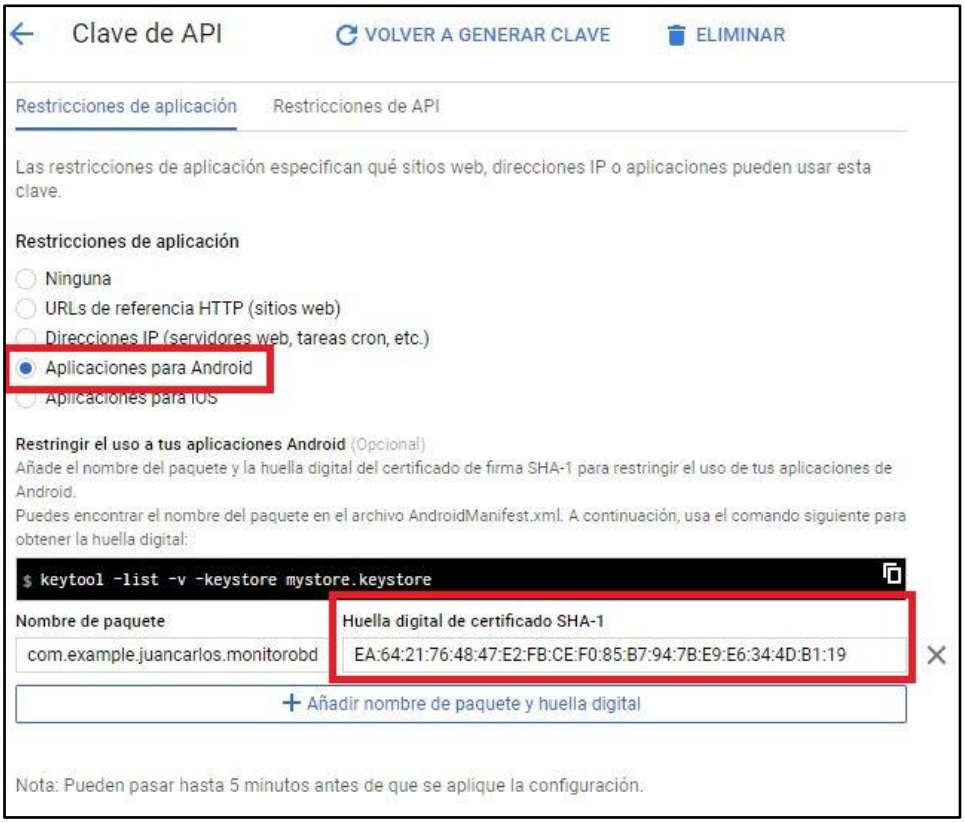


Ilustración 86 – En el cuadro rojo, la clave proporcionada para el uso de la API

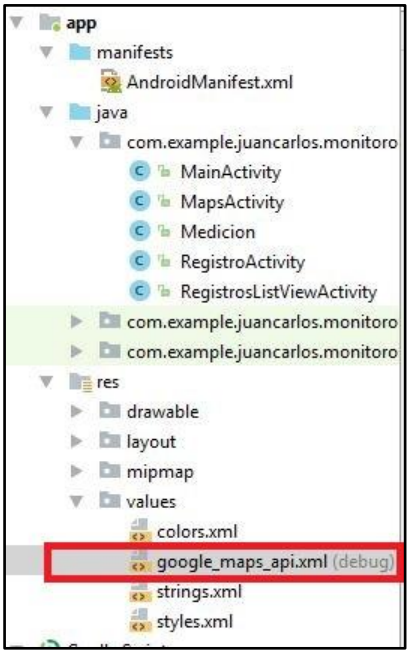


Ilustración 87 – En el cuadro rojo, archivo donde debemos añadir la clave de la API

```
<resources>
    <!--
    TODO: Before you run your application, you need a Google Maps API key.

    To get one, follow this link, follow the directions and press "Create" at the end:

    https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=EA:64:21:76:48:47:E2:FB:CE:

    You can also add your credentials to an existing key, using these values:

    Package name:
    EA:64:21:76:48:47:E2:FB:CE:F0:85:B7:94:7B:E9:E6:34:4D:B1:19

    SHA-1 certificate fingerprint:
    EA:64:21:76:48:47:E2:FB:CE:F0:85:B7:94:7B:E9:E6:34:4D:B1:19

    Alternatively, follow the directions here:
    https://developers.google.com/maps/documentation/android/start#get-key

    Once you have your key (it starts with "AIza"), replace the "google_maps_key"
    string in this file.
    -->
    <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">AIzaSyAyYk3RtKh48zysEM20bWzEI4TwDOBRvXs</string>
</resources>
```

Ilustración 88 – En el cuadro rojo, clave de la API añadida a nuestro proyecto

El mapa ya está totalmente disponible en nuestra aplicación, el siguiente paso es el trazado de la ruta sobre el mismo. Para ello haremos uso de las coordenadas geográficas obtenidas al elegir el registro. Estas coordenadas se encuentran almacenadas en una estructura tipo vector donde están almacenados todos los valores leídos del fichero del registro. Iremos recorriendo elemento a elemento el vector y por cada posición tomaremos los valores de longitud y latitud. Crearemos una línea uniendo las coordenadas geográficas de la posición actual del vector que estamos recorriendo con las coordenadas obtenidas de la posición anteriormente recorrida, creando finalmente una polyline, es decir, una secuencia de líneas conectadas.

```
for(int i=1;i<registro.size();i++){
    Double latActual = Double.parseDouble(registro.get(i).getLatitud());
    Double lonActual = Double.parseDouble(registro.get(i).getLongitud());
    LatLng puntoActual = new LatLng(latActual, lonActual);

    Double latAnterior = Double.parseDouble(registro.get(i-1).getLatitud());
    Double lonAnterior = Double.parseDouble(registro.get(i-1).getLongitud());
    LatLng puntoAnterior = new LatLng(latAnterior, lonAnterior);

    PolylineOptions polyOptions = new PolylineOptions()
            .add(puntoAnterior,puntoActual)
            .color(Color.parseColor(color));
```

Ilustración 89 – Código que traza una secuencia de líneas en el mapa en función de las coordenadas geográficas

Además, añadiremos iconos para señalizar el lugar donde se inició y donde finalizó el trayecto realizado por el vehículo. Para ello simplemente añadimos un **Marker** al mapa y le proporcionamos las coordenadas donde queremos que sea dibujado.

```
Double latInicio =   Double.parseDouble(registro.get(0).getLatitud());
Double lonInicio = Double.parseDouble(registro.get(0).getLongitud());
LatLng puntoDePartida = new LatLng( latInicio, lonInicio);
mMap.addMarker(new MarkerOptions().position(puntoDePartida).title("Inicio").snippet("Punto de partida"));
```

Ilustración 90 – Código que añade un icono para señalizar el punto de partida del trayecto

Esta nueva pantalla posee también dos botones con menús desplegables para la selección de opciones del mapa. Estos botones son conocidos en Android Studio como **Spinners**[12], al pulsarlos se muestra en pantalla un menú con distintas opciones de las que únicamente se puede seleccionar una entre las disponibles. Este tipo de botones serán utilizados para la elección del estilo del mapa y la elección del parámetro del vehículo con el cual queremos medir la eficiencia de la conducción.

La idea sería poder elegir entre diferentes opciones de visualización del mapa mostrado, por ejemplo, mostrar un mapa tipo satélite o que únicamente muestre las calles y carreteras. La implementación de los Spinners es muy simple tan solo necesitamos pasarle al objeto que representa el botón una estructura que contenga los elementos que van ser visualizados cuando se despliegue el menú al pulsarlo.

```
tipoMapa = (Spinner)findViewById(R.id.SpinnerTipoMapa);
tipoParametro = (Spinner)findViewById(R.id.SpinnerParametro);

String [] opcionesMapas = {"Normal","Satélite","Híbrido","Carretera"};
String [] opcionesParametros = {"RPM","Velocidad","Posición acelerador","Velocidad y posición acelerador"};

ArrayAdapter<String> spinnerMapaAdapter = new ArrayAdapter<~>( context: this,R.layout.spinner_item_google_maps,opcionesMapas);
tipoMapa.setAdapter(spinnerMapaAdapter);

ArrayAdapter<String> spinnerParametroAdapter = new ArrayAdapter<~>( context: this,R.layout.spinner_item_google_maps,opcionesParametro
tipoParametro.setAdapter(spinnerParametroAdapter);
```

Ilustración 91 – Definición de los Spinners y los menús de opciones asociados

Al pulsar cualquiera de las opciones que muestra el Spinner en su menú desplegable, se ejecutará un código que permitirá efectuar cambios sobre el mapa. En caso del estilo del mapa, si pulsamos sobre la opción del menú “Satélite”, el

código comprobará el ítem seleccionado y cambiará el estilo del mapa en función de la opción escogida.

```
tipoMapa.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id){
        String itemSeleccionado = parent.getItemAtPosition(position).toString();

        if(itemSeleccionado == "Satélite"){
            mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
        }else if(itemSeleccionado == "Híbrido"){
            mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
        }else if(itemSeleccionado == "Carretera"){
            mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
        }else if(itemSeleccionado == "Normal"){
            mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
        }
    }
```

Ilustración 92 – Código ejecutado al pulsar una opción del menú desplegable del Spinner

El aspecto de la pantalla donde se muestra el mapa sería similar al mostrado en la siguiente ilustración.

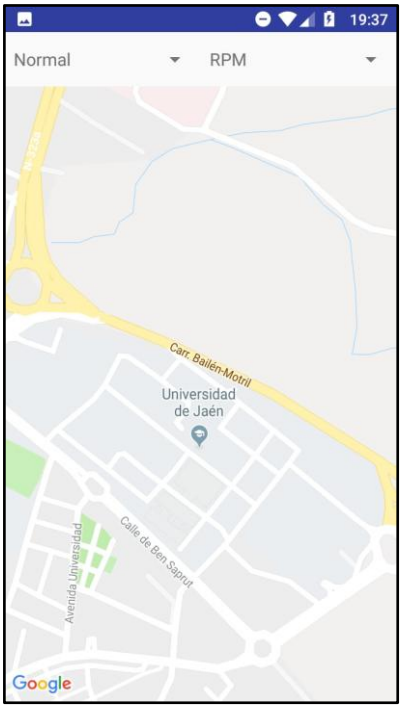


Ilustración 93 – Pantalla con visualización del mapa de Google

Al pulsar el Spinner de tipo de mapa se desplegaría el menú con las distintas opciones como puede observarse en las siguientes ilustraciones.

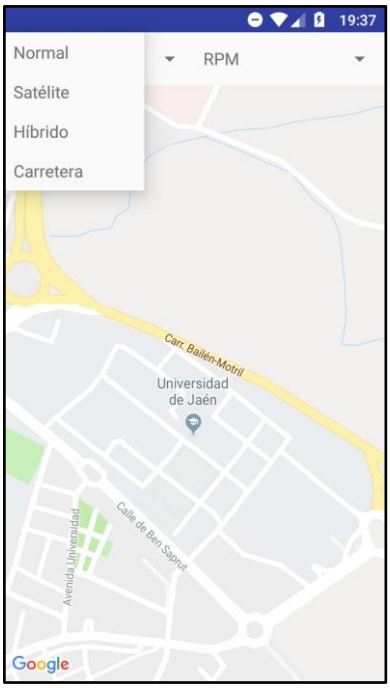


Ilustración 94 – Menú desplegado al pulsar el Spinner

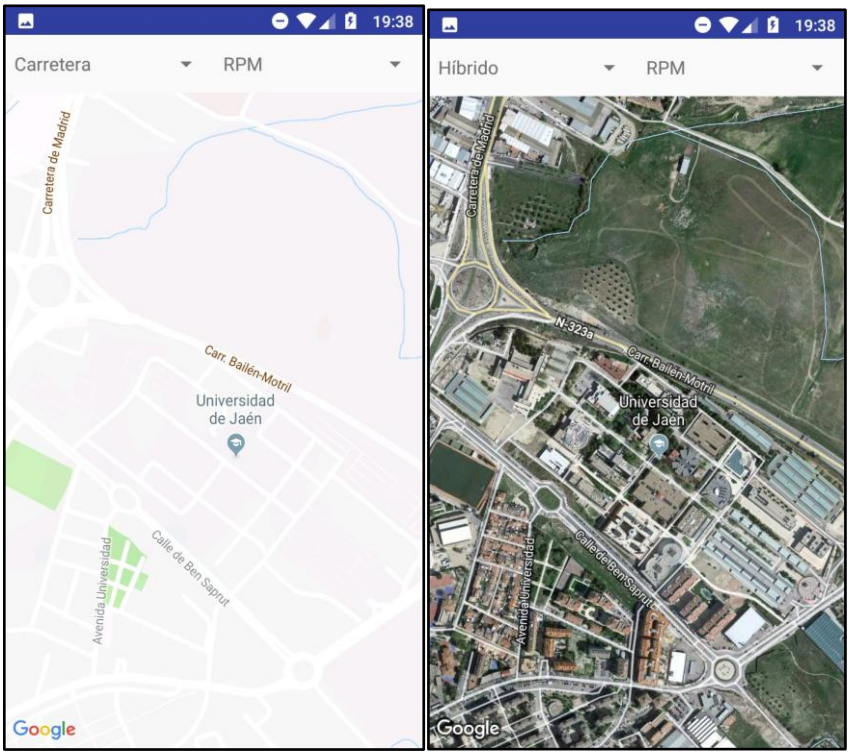


Ilustración 95 y 96 - Diferentes estilos del mapa

4.5.7. Pruebas realizadas

Las diferentes pruebas realizadas durante esta iteración permitieron comprobar de manera satisfactoria el funcionamiento de la aplicación de monitorización de datos recabados del vehículo, la correcta visualización en pantalla de las gráficas de valores y el trazado de las rutas sobre el mapa. Sin embargo, un importante detalle a comentar fue que debido a la escasa precisión con la que a veces se captaban las coordenadas geográficas, impedían dibujar con total exactitud el trayecto realizado por el vehículo, mostrando en ciertas ocasiones que el trazado sobre el mapa se salía de las carreteras o atravesaba edificaciones.

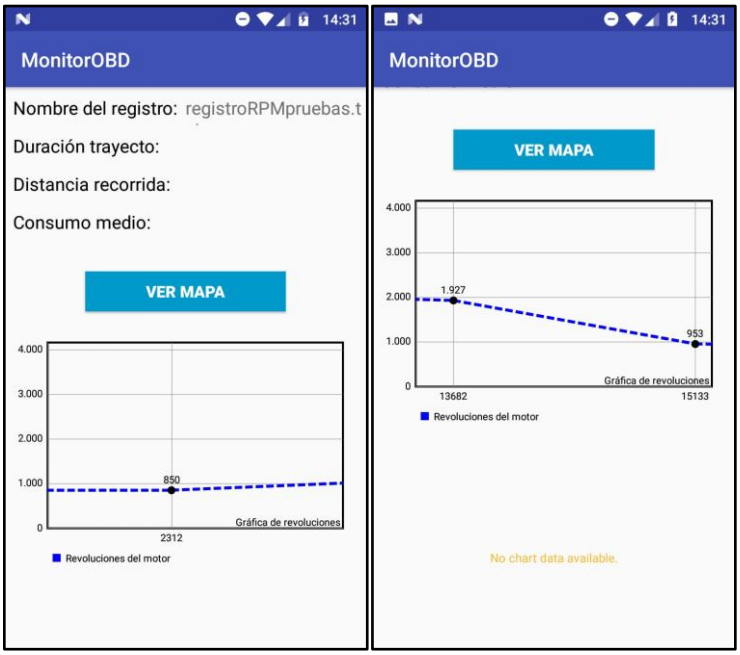


Ilustración 97 y 98 - Gráficas con los valores capturados y detalle de interacción sobre las gráficas

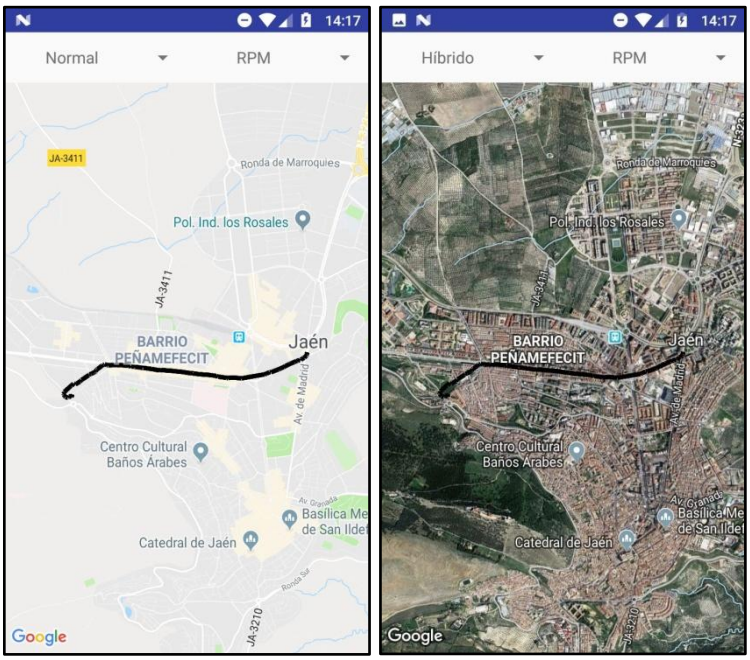


Ilustración 99 y 100 - Trazado del trayecto realizado en distintos tipos de mapas

4.5.8. Reunión

Tras la reunión realizada al término de la cuarta iteración, el tutor valoró positivamente el trabajo y los avances realizados y se acordaron los siguientes pasos a realizar en el proyecto. Para la quinta iteración el tutor sugirió aplicar las técnicas sobre conducción eficiente, estudiadas previamente, al proyecto en desarrollo y a los datos captados del vehículo. Se estableció como principal objetivo la necesidad de colorear la ruta trazada en el mapa en función de un mayor o menor esfuerzo o consumo por parte del vehículo y de esta forma notificar al usuario acerca de los mejores y peores tramos del trayecto realizado en términos de eficiencia.

4.6. Iteración 5 - Conducción eficiente aplicada al proyecto

4.6.1. Trazado de rutas con diferentes colores en función del esfuerzo o consumo del vehículo

Partiendo como base de las conclusiones tomadas en el estudio acerca de la conducción eficiente, los parámetros de conducción con los que podemos medirla y los datos recabados del vehículo, el objetivo ahora es modificar el trazado de las rutas sobre el mapa para mostrar información cualitativa mediante el empleo de diferentes colores a la hora de dibujarlas.

A través de los botones tipo Spinners implementados en la iteración anterior podremos trazar dos tipos de rutas diferentes.

- Para el primer caso utilizaremos los valores de los parámetros de las revoluciones del motor, la velocidad y la posición del acelerador. Trazaremos la ruta sobre el mapa utilizando las coordenadas geográficas y aplicaremos un color diferente en cada tramo en función de los valores de esos parámetros. Por ejemplo, si los valores de las revoluciones del motor están dentro de los límites normales de eficiencia, es decir, el coche va a unas revoluciones con valores entre 1.000 o 2.500 rpm, el color de la ruta trazada sería verde, si las revoluciones pasan de 2.500 el color de la línea dibujada pasaría a amarillo y por último, si las revoluciones pasan de 3.000, utilizaríamos el color rojo. En el último caso también podemos colocar un icono interactivo que muestre en pantalla un mensaje con los valores de las revoluciones, velocidad y posición del acelerador. De esa forma conseguimos que el usuario sepa dónde realizó una maniobra incorrecta y cuáles fueron los valores de los parámetros medidos. Así se consigue evitar que usuario vuelva a cometer nuevamente el fallo, mejorando su conducción eficiente.

- El segundo caso sería colorear la ruta en función de los parámetros que permitan medir el consumo del coche, en este caso utilizaremos los valores del consumo instantáneo en ese instante, la posición del acelerador y el consumo instantáneo medio de todo el trayecto. El color de la ruta dependerá de los valores del consumo instantáneo de manera similar al empleado con las revoluciones del motor. La posición del acelerador nos permitirá conocer su valor registrado en un momento el que el consumo instantáneo se elevó por encima de la media. De esta forma el usuario sabrá en que tramo del trayecto pisó el acelerador demasiado y obtendrá el porcentaje de pisada que le servirá para corregir el hábito en el futuro.

Tras implementarlo en la aplicación el resultado de pulsar el botón Spinner para elegir el tipo de ruta sería similar al de la siguiente ilustración:

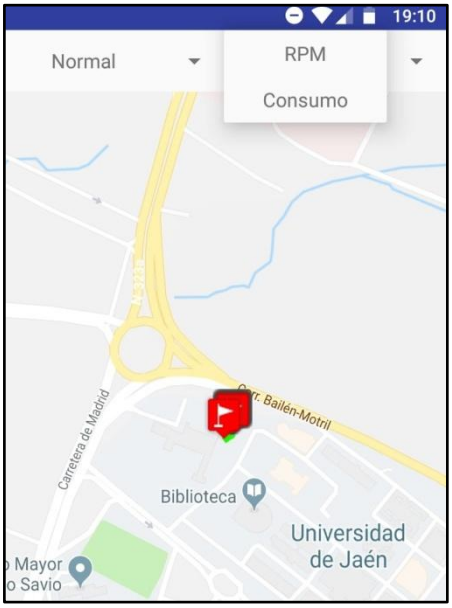


Ilustración 101 - Resultado del botón con el menú desplegable

Para la implementación de este nuevo objetivo tenemos que modificar el código que trazaba la ruta sobre el mapa. Como se explicó en la cuarta iteración, el método recorría una estructura tipo vector donde se almacenaban todas las mediciones realizadas de un registro. El método tomaba las coordenadas geográficas de la posición por la que iba la iteración más las coordenadas de la posición anterior y trazaba una línea. Ahora, cada vez que se vaya a trazar un segmento del trayecto se hará una comprobación de los valores de los parámetros de conducción para determinar el color de la línea a trazar.

El caso de las revoluciones del motor puede observarse en la siguiente ilustración:

```
public String comprobarValorRPM(Double rpm, LatLng punto, Integer pos){
    String color = "#22FF00"; //Color por defecto verde

    if(rpm >= 3000.0){
        color = "#FFF400"; // Color rojo
        Marker m = mMap.addMarker(new MarkerOptions().position(punto).title("Advertencia").snippet("RPM elevadas").icon(BitmapDescripto
        marcadores.put(pos,m);
    }else if(rpm >= 2500.0){
        color = "#FF0000"; //Color amarillo
    }

    return color;
}
```

Ilustración 102 - Método que comprueba el valor de las revoluciones y establece el color de la línea a trazar

Por defecto, se mostrará la ruta con colores en función de las revoluciones del motor. Si el usuario elige en el Spinner la opción del combustible, la ruta volverá a trazarse, pero esta vez con los colores en función de la cantidad de combustible consumido.

En cuanto al uso de iconos y mensajes en pantalla para mostrar información importante, por ejemplo, cuando se excedan las revoluciones del motor se mostrará un icono de advertencia en el tramo del trayecto correspondiente. Al pulsar el icono se mostrará en pantalla un cuadro de texto con el detalle de la acción ocurrida y con los valores de los parámetros del vehículo involucrados, en este caso serían las revoluciones, la velocidad y la posición del acelerador. De la misma manera mostraríamos los iconos también cuando en un momento dado del trayecto el consumo instantáneo se eleve por encima de la media. Pero en este último caso mostraríamos los valores de consumo instantáneo en ese instante, el consumo instantáneo medio del trayecto completo y la posición del acelerador.

Los iconos empleados para los marcadores en el trazado de la ruta se pueden observar en la siguiente ilustración[20]. De izquierda a derecha, el icono con la bandera se utiliza para señalizar en el mapa los puntos de inicio y fin del trayecto. El símbolo de advertencia se utiliza cuando las revoluciones del motor sobrepasan el valor de las 3.000 rpm. El símbolo del surtidor del combustible se emplea cuando se registra un valor de consumo instantáneo por encima de la media general del todo el trayecto. El icono del velocímetro se emplea para de señalizar los puntos donde el vehículo excedió los 120 kilómetros por hora de velocidad. Por último, el icono del termómetro se utiliza señalizar en el mapa los tramos en los que la temperatura del líquido refrigerante y del aire de admisión al motor han registrado un valor anormal fuera de su rango característico. En el caso del refrigerante serían valores de temperatura inferiores a -12ºC o superiores 103ºC. Mientras que en la temperatura del aire de admisión, serían valores inferiores a 20ºC o superiores a 80ºC.

Ilustración 103 - Iconos para los marcadores empleados en la aplicación

4.6.2. Marcadores interactivos y mensajes descriptivos en el mapa

Otra de las nuevas funcionalidades introducidas a la aplicación fue hacer que los iconos colocados en el mapa fueran interactivos, de tal forma que al pulsar sobre ellos se muestre en pantalla un cuadro de texto con información más detallada de por qué se ha colocado un icono en dicho lugar.

Podemos implementar esta funcionalidad de dos formas diferentes, por un lado implementando el método **OnMarkerClickListener** que es una función que se ejecuta cuando se pulsa sobre el icono de un marcador o podemos utilizar **OnInfoWindowClickListener,** una función que se ejecuta cuando se pulsa sobre el pequeño cuadro de descripción de un marcador. En nuestro caso haremos uso de la segunda función puesto que al pulsar cualquiera de los marcadores colocados sobre nuestro mapa se mostrará un pequeño cuadro de texto con una breve descripción del marcador. De esta forma al pulsar sobre el pequeño cuadro descriptivo se mostrará en pantalla uno más amplio con información más detallada.

Ilustración 104 - Al pulsar sobre el icono de un marcador se muestra un breve texto descriptivo tal y como se aprecia en la imagen

Para mostrar la información con más detalle haremos uso de un objeto **Dialog**. Este objeto recibirá en su constructor dos campos un título y un cuerpo con el texto descriptivo.

```
public static ComentarioFragment newInstance(String title, String fullSnippet){
    ComentarioFragment fragment = new ComentarioFragment();
    Bundle b = new Bundle();
    b.putString(ARGUMENTO_TITLE,title);
    b.putString(ARGUMENTO_FULL_SNIPPET,fullSnippet);
    fragment.setArguments(b);

    return fragment;
}
```

Ilustración 105 - Constructor del cuadro descriptivo

Una vez llamado el constructor y creado el objeto, se muestra el Dialog en pantalla con los dos campos pasados al constructor.

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState){
    Dialog dialog = new AlertDialog.Builder(getActivity())
            .setTitle(title)
            .setMessage(fullSnippet)
            .create();

    return dialog;
}
```

Ilustración 106 - Constructor del Dialog

Ahora necesitamos implementar el método **OnInfoWindowClick** que será el encargado de llamar al constructor del Dialog, es decir, del cuadro de texto que se mostrará en pantalla.

Al ser la selección de los iconos una interacción posterior al trazado de la ruta en pantalla no podremos, a priori, acceder a la información de los parámetros del vehículo al seleccionar los marcadores. Para poder acceder a la información de los parámetros en el momento en el que se colocó el marcador en el mapa haremos uso de una tabla hash que identificará de manera única a cada marcador colocado. Recordemos que para colocar los marcadores en pantalla se tenía que recorrer de manera iterativa el vector donde se almacenaba toda la información leída del registro del trayecto. Almacenar el índice de iteración en el mapa nos permitirá

posteriormente acceder a los valores de los parámetros de conducción en esa posición para determinar por qué se colocó el marcador en el mapa.

Se implementó una estructura tipo mapa que tenía como clave el índice de iteración sobre el vector de datos y como valor de entrada el objeto que representa al marcador en sí, un Marker. Sin embargo, la función OnInfoWindowClick, que recibe como parámetro el marcador que el usuario ha pulsado, no conseguía identificar a cada marcador colocado a través del objeto Marker. Así que se implementó otra estructura tipo mapa pero esta vez se puso como clave el título del mensaje descriptivo que poseía el marcador concatenado con el índice de iteración y como valor de la entrada se puso el índice de iteración. De esta forma conseguimos que todos los marcadores sean únicos.

```
Marker m = mMap.addMarker(new MarkerOptions().position(puntoActual).title("Temperatura aire admisión"+"("+Integer.toString(i)+")").snippe
marcadores.put(m.getTitle(),i);
```

Ilustración 105 - Inserción de los marcadores a la tabla hash

En la función OnInfoWindowClick tomaremos el título del marcador que se ha pulsado para buscarlo en el mapa. Una vez localizado tendremos que comprobar de qué tipo se trata el marcador, si es un marcador asociado a las revoluciones, al consumo, a la velocidad, etc. Para ello comprobamos si el título contiene palabras clave como “Revoluciones” en el caso de marcadores asociados a las revoluciones del motor, “Consumo” para los asociados al consumo instantáneo, “refrigerante” para los asociados a la temperatura del líquido refrigerante, etc. Estas comprobaciones nos servirán para definir la información que aparecerá sobre el cuadro del objeto Dialog.

```
@Override
public void onInfoWindowClick(Marker marker) {
    String key = marker.getTitle();

    //Comprobamos si el marcador está en la tabla hash
    if(marcadores.containsKey(key)){

        //Marcadores para las revoluciones
        if(key.contains("Revoluciones")){
            String comentario = "Revoluciones elevadas por encima de 3.000 rpm\n";
            comentario += "Valor medido: " + registro.get(marcadores.get(key)).getRpm() + " rpm\n";
            comentario +=  "Velocidad: " + registro.get(marcadores.get(key)).getVelocidad() + " km/h\n";
            comentario +=  "Posición acelerador: " + registro.get(marcadores.get(key)).getPosicionAcelarador() + "%\n";
            ComentarioFragment.newInstance(marker.getTitle(),comentario)
                    .show(getSupportFragmentManager(), tag: null);
        }

        //Marcadores para el cosnumo instantáneo
        if(key.contains("Consumo")){
            String comentario = "Consumo instántaneo por encima de la media\n";
            comentario += "Valor medido: " + registro.get(marcadores.get(key)).getConsumo() + " l/km\n";
            comentario +=  "Posición acelerador: " + registro.get(marcadores.get(key)).getPosicionAcelarador() + "%\n";
            ComentarioFragment.newInstance(marker.getTitle(),comentario)
                    .show(getSupportFragmentManager(), tag: null);
        }

        //Marcadores para la velocidad
        if(key.contains("Velocidad")){
            String comentario = "Velocidad por encima de los 120 km/h\n";
            comentario += "Valor medido: " + registro.get(marcadores.get(key)).getVelocidad() + " km/h\n";
            ComentarioFragment.newInstance(marker.getTitle(),comentario)
                    .show(getSupportFragmentManager(), tag: null);
```

Ilustración 108 - Método ejecutado al pulsar sobre el cuadro descriptivo de un marcador.

4.6.3. Diagrama de clases

La nueva funcionalidad implementada en la aplicación provocó un cambio en el diagrama de clases expuesto en la cuarta iteración cuando se diseñó la aplicación de visualización de datos.

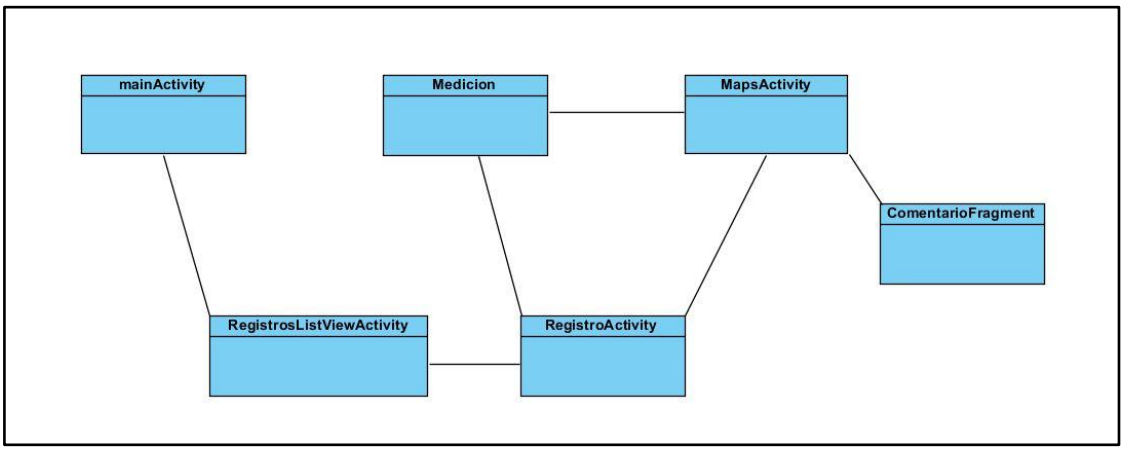


Ilustración 109 - Nuevo diagrama de clases

El único cambio introducido fue la inclusión de la clase **ComentarioFragment** donde se implementa el código del objeto Dialog explicado en el apartado anterior.

4.6.4. Cambios en la interfaz de la aplicación de monitorización de datos

Algunos cambios fueron introducidos en la interfaz de la aplicación de monitorización de datos. Al seleccionar un registro la aplicación mostrará a partir de ahora información relevante del mismo como son la duración del trayecto, la distancia total recorrida en kilómetros o el consumo instantáneo medio del trayecto en kilómetros/litro.

Ilustración 110 - Nuevo aspecto de la interfaz al seleccionar un registro

Para el cálculo de la duración total del trayecto simplemente tenemos en cuenta el valor del tiempo en la última entrada del fichero asociado el registro que hemos seleccionado. Recordemos que cada vez que escribíamos una línea en el fichero cuando nos comunicábamos con el OBD2 almacenábamos el tiempo desde que se inició la comunicación con el mismo, así que la duración total del trayecto corresponde al valor de la variable temporal de la última entrada o línea del fichero.

```
public void calcularDuracionTrayecto(){
    duracionTrayecto.setText(tiempo[tiempo.length-1]);
}
```

Ilustración 111 - Cálculo de la duración total del trayecto

Para el cálculo de la distancia total recorrida durante el trayecto hacemos uso de los valores de longitud y latitud de cada entrada del fichero. Recorremos la estructura que las almacena, tomamos dos puntos y calculamos la distancia entre ellos encada iteración. Luego, la distancia obtenida se va sumando en una variable que contendrá finalmente la distancia total.

```
public void calcularDistanciaRecorrida(){
    Float distanciaTotal = 0.0f;
    for(int i=1;i<registro.size();i++){
        Location locationA = new Location( provider: "punto A");
        locationA.setLatitude(Double.parseDouble(registro.get(i-1).getLatitud()));
        locationA.setLongitude(Double.parseDouble(registro.get(i-1).getLongitud()));

        Location locationB = new Location( provider: "punto B");
        locationB.setLatitude(Double.parseDouble(registro.get(i).getLatitud()));
        locationB.setLongitude(Double.parseDouble(registro.get(i).getLongitud()));

        Float distancia = locationA.distanceTo(locationB);
        distanciaTotal += distancia;
    }
    distanciaTotal = distanciaTotal/1000f; //Pasamos al distancia total a kilómetros
    distanciaRecorrida.setText(Float.toString(distanciaTotal) + " km");
}
```

Ilustración 112 - Cálculo de la distancia total recorrida en kilómetros

Finalmente, para el cálculo del consumo instantáneo medio, recorremos la estructura que almacena todos los consumos leídos del fichero, sumamos todos los valores y calculamos la media.

```
public void calcularConsumoMedio(){
    Float totalConsumo = 0.0f;
    for(int i=0;i<valoresConsumo.length;i++){
        totalConsumo += valoresConsumo[i];
    }

    Float media = totalConsumo/valoresConsumo.length;
    consumoMedio.setText(Float.toString(media) + " l/km");
}
```

Ilustración 113 - Cálculo del consumo instantáneo medio del trayecto

4.6.5. Pruebas realizadas

Durante la fase de pruebas correspondiente a esta iteración se comprobó el correcto funcionamiento de las nuevas características implementadas. A continuación, se mostrarán los resultados obtenidos en un trayecto realizado por la ciudad de Jaén con el trazado de la ruta en función de los valores de las revoluciones del motor.

Ilustración 114 - Información del registro y gráficas de valores

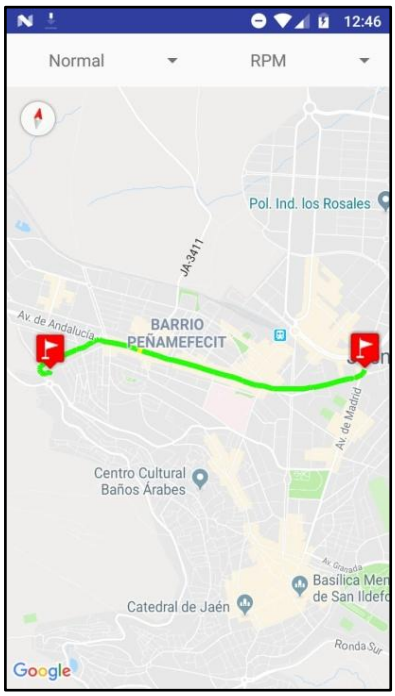


Ilustración 115 - Trayecto completo

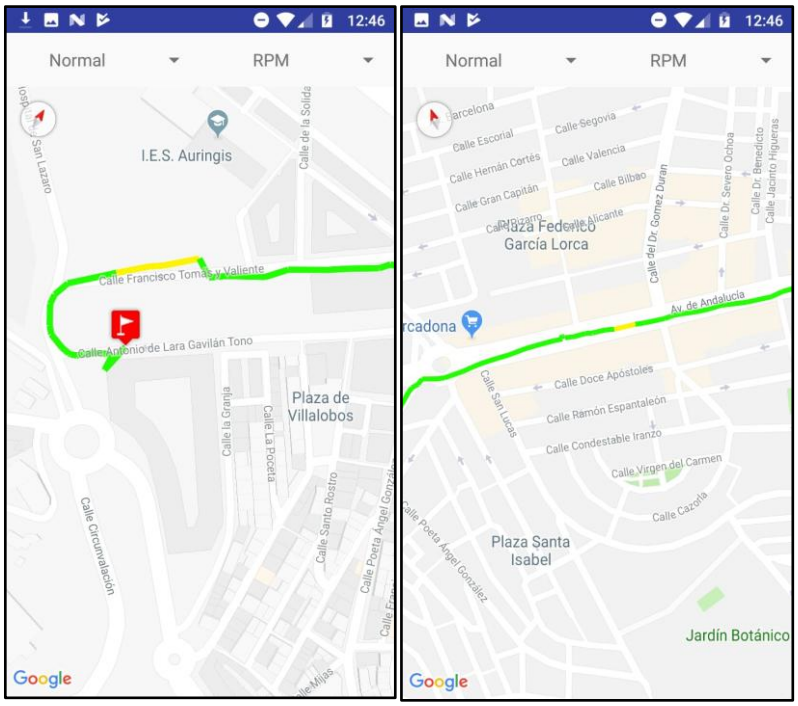


Ilustración 116 y 117 - Tramos del recorrido con diferentes colores en función de los valores de las revoluciones del motor

Durante este trayecto no se superaron las 3.000 revoluciones del motor, así que se modificaron algunos valores de las revoluciones en el fichero guardado para

hacerlos superiores a 3.000 rpm y comprobar los resultados obtenidos al visualizar el mapa.

Al detectar la aplicación valores elevados de revoluciones cambia el color del trazado a rojo y coloca un marcador para notificar al usuario la razón del cambio de color.

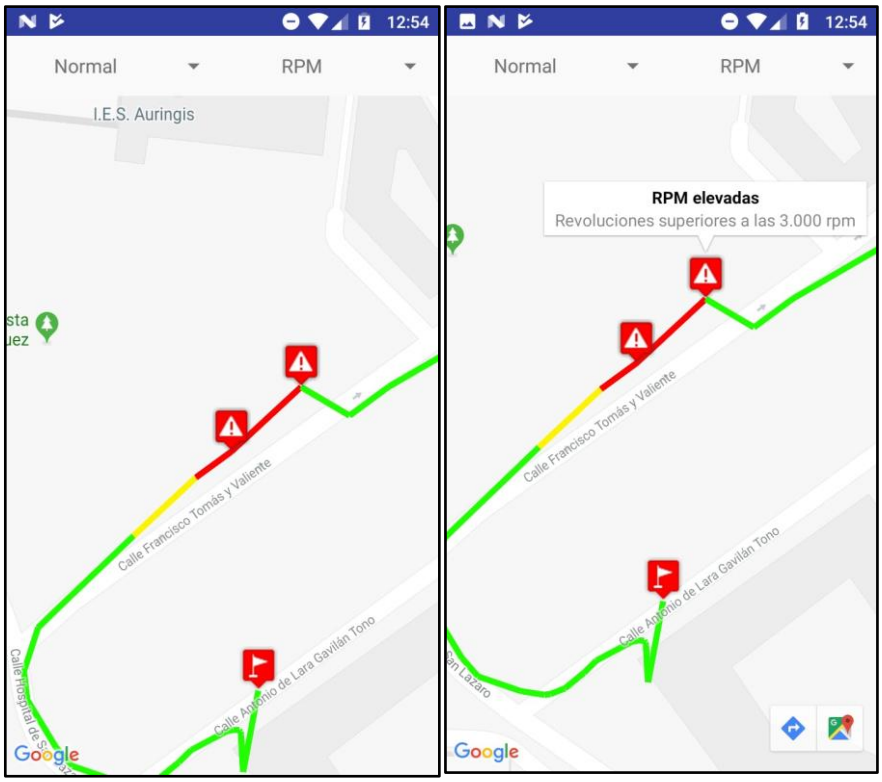


Ilustración 118 y 119 - Tramos del recorrido con revoluciones del motor superiores a 3.000 rpm

Si pulsamos sobre cualquier marcador asociado a los parámetros de conducción se desplegará un pequeño bocadillo sobre el icono del marcador con información descriptiva del mismo, tal y como puede observarse en la ilustración 119. Al pulsar el bocadillo se muestra un cuadro de texto más amplio con los valores de los parámetros de conducción capturados en ese punto del trayecto.

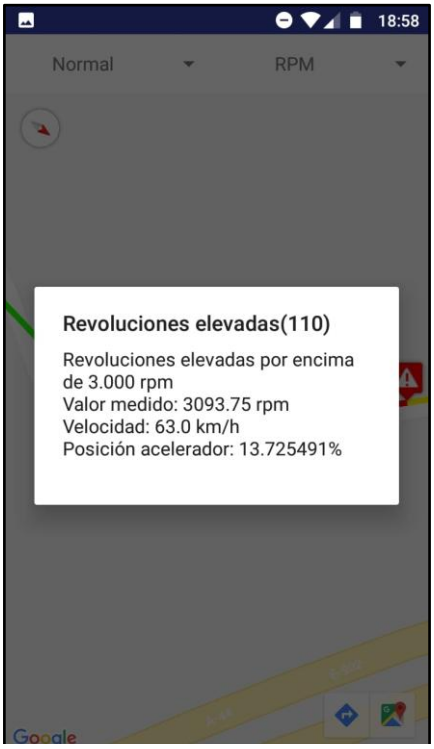


Ilustración 120 – Cuadro descriptivo mostrado al pulsar sobre la descripción de un marcador

Las nuevas pruebas realizadas también nos sirvieron para comprobar con detalle el problema de la imprecisión de algunas coordenadas geográficas medidas.

Otros nuevos problemas fueron detectados en la aplicación de comunicación con el dispositivo OBD2 durante esta etapa de pruebas. Los comandos enviados para las consultas sobre los parámetros de conducción fueron introducidos en las pruebas de manera incremental. En la segunda iteración cuando se diseñó la aplicación de captura de datos se emplearon solamente los comandos de las revoluciones, la velocidad y la carga del motor. En las siguientes iteraciones se ampliaron el número de comandos utilizados hasta llegar a esta iteración donde se utilizaron todos. El problema se detectó al analizar los ficheros con las respuestas recibidas. En total se enviaban siete comandos para consultas sobre los componentes del vehículo o sensores, sin embargo, la aplicación solo recibía seis respuestas a estos comandos.

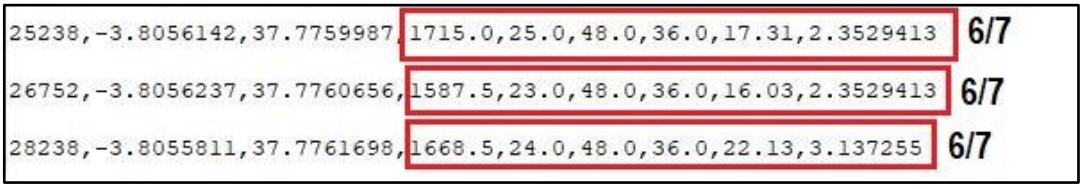


Ilustración 121 – Del envío de siete comandos por ráfaga se obtenían seis respuestas

Así que se tuvo que modificar la aplicación de captura para saber a qué comando pertenecía cada respuesta y de este modo conocer cuál era el sensor del que no se podía obtener información porque no estaba soportado en el vehículo de pruebas. La solución fue simple, al almacenar las repuestas de cada comando, se concatenó al valor de la respuesta el comando en sí para determinar el componente al que pertenecía.

```
switch (subcadena) {
    case "410C": //ENGINE RPM
        formula = ((A * 256f) + B) / 4f;
        solucion = Float.toString(formula);
        rpm.setText(solucion);
        solucion += "(0C)";
        break;

    case "4104": //ENGINE LOAD
        formula = (A * 100f) / 255f;
        solucion = Float.toString(formula);
        engineLoad.setText(solucion);
        solucion += "(04)";
        break;

    case "410D": //VEHICLE SPEED
        VSS = (float)A;
        solucion = Float.toString(VSS);
        speed.setText(solucion);
        solucion += "(0D)";
        break;

    case "4105": //COOLANT TEMPERATURE
        formula = (A - 40f);
        solucion = Float.toString(formula);
        solucion += "(05)";
        break;
```

Ilustración 122 – Concatenación de los comandos a las respuestas

Este problema se sumó a la recepción de respuestas sin valor alguno desde el dispositivo OBD2 cuando la comunicación llevaba varios minutos en funcionamiento, provocando un desajuste en la recepción de las respuestas de los comandos. Este desajuste se observó en las respuestas almacenadas en los ficheros en formato txt. La recepción de respuestas sigue un orden preestablecido y se almacenan en el

fichero en ese orden, al alterarse, las respuestas se escribían desordenadas o incluso en algunas faltaban valores de algunos componentes.

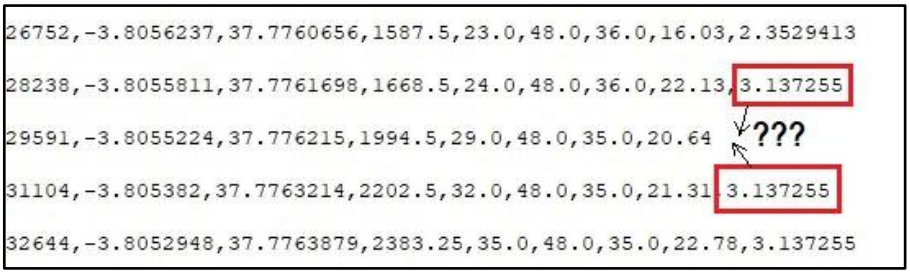


Ilustración 123–Respuestas en las que faltan los valores de algunos componentes

Se pensó que el origen de este problema podría estar en un carácter recibido desde el OBD2 que se pudo detectar en iteraciones anteriores, pero no se le dio mayor importancia hasta las pruebas realizadas en esta iteración, las cuales fueron de trayectos más largos y de mayor duración. El carácter en cuestión es “>”, según la documentación de ELM Electronics este carácter lo envía el dispositivo OBD2 tras enviar la respuesta a un comando consultado para notificar al sistema de diagnóstico que el canal de comunicación vuelva a estar libre y que el dispositivo está listo para recibir un nuevo comando de consulta. Sin embargo, en pruebas realizadas en iteraciones anteriores, tal y como puede observarse en la ilustración 125, el carácter no era siempre recibido por la aplicación o tardaba mucho tiempo en hacerlo, por lo que en esas iteraciones se optó por obviarlo y no tenerlo en cuenta.

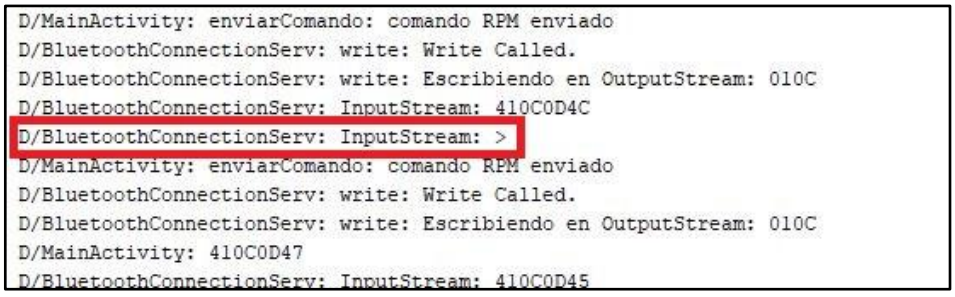


Ilustración 124 – Recepción del carácter “>” durante la comunicación con el dispositivo OBD2

```
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: AT L0
D/BluetoothConnectionServ: InputStream: OK>
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: AT S0
D/BluetoothConnectionServ: InputStream: OK>
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: AT H0
D/BluetoothConnectionServ: InputStream: OK>
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: AT SP 0
D/BluetoothConnectionServ: InputStream: OK>
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: SEARCHING...
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 410C0D45
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010D
D/BluetoothConnectionServ: InputStream: 410D00
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C0D4A
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 410435
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010D
D/BluetoothConnectionServ: InputStream: 410D00
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C0D4E
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 410435
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010D
D/BluetoothConnectionServ: InputStream: 410D00
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C0D4A
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
D/BluetoothConnectionServ: InputStream: 410435
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010D
D/BluetoothConnectionServ: InputStream: 410D00
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 010C
D/BluetoothConnectionServ: InputStream: 410C0D4D
D/BluetoothConnectionServ: write: Escribiendo en OutputStream: 0104
```

Ilustración 125 – Comunicación completa con el dispositivo OBD2 sin recibir el carácter “>”

Al no tener en cuenta ese carácter, la recepción del mismo por parte de la aplicación provocaba el envío de un nuevo comando de consulta a la cola del canal de comunicación, generando el desajuste en el orden de recibimiento de las respuestas pues ese comando no debería ponerse en cola hasta no recibir la respuesta del comando anterior a él.

4.6.6. Reunión

En la reunión realizada al término de la quinta iteración el tutor del TFG valoró positivamente los avances realizados en la aplicación de visualización de datos y los resultados obtenidos en la misma. El tutor también sugirió la necesidad de corregir el desorden de las respuestas a los comandos de consulta provocado por la recepción de caracteres sin valor que no eran debidamente tratados. Una vez finalizada la reunión se dio por iniciada la última iteración del proyecto.

4.7. Iteración 6 - Pruebas finales

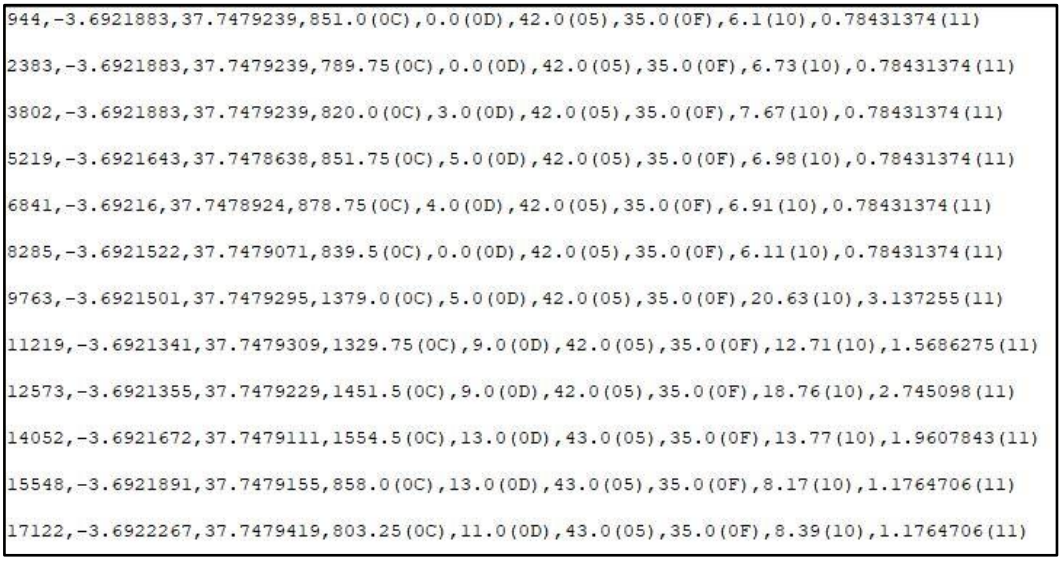
Esta última iteración sirvió para corregir los problemas encontrados en las pruebas realizadas en la quinta iteración y exponer los resultados finales obtenidos por las aplicaciones.

El primer problema a solucionar fue en la aplicación de captura de datos, en concreto el desajuste en la recepción de respuestas por parte del dispositivo OBD2. Al término de la quinta iteración se añadió un nuevo cambio con el objetivo de conocer a qué componente del vehículo correspondía cada respuesta recibida concatenando a ésta el comando del componente correspondiente.

En las nuevas pruebas realizadas en esta iteración se pudo comprobar que realmente sí se recibían respuestas para todos los comandos enviados al vehículo y que el orden de llegada de las respuestas era correcto. El problema estaba en que el comando correspondiente a la carga del motor, el 0104, se estaba enviando en cada medición, pero sólo se recibía respuesta por parte del vehículo en unas pocas ocasiones. Por este motivo solo se hallaban seis respuestas a los comandos en el fichero con los datos capturados. En las pocas ocasiones en las que se recibía la respuesta de este comando tampoco se obtenían todas las respuestas a los comandos enviados. Lo que sucedía era que el comando de la carga de motor "machacaba" la recepción de otro comando por lo que todas las mediciones quedaban incompletas.

```
39236,-3.6919256,37.7481272,852.75(0C),10.0(0D),45.0(05),36.0(0F),8.75(10),29.80392(11)
40670,-3.6919189,37.7481258,809.0(0C),10.0(0D),45.0(05),36.0(0F),8.59(10),1.1764706(11)
42125,-3.69188,37.7481074,844.5(0C),5.0(0D),45.0(05),36.0(0F),8.5(10),1.1764706(11)
43582,-3.6918653,37.7481002,1645.75(0C),7.0(0D),45.0(05),36.0(0F),24.53(10),3.5294118(11)
45049,-3.6918343,37.7480829,1091.0(0C),9.0(0D),45.0(05),36.0(0F),9.89(10),1.1764706(11)
46470,-3.6918153,37.7480726,867.25(0C),34.117645(04),6.0(0D),45.0(05)
47949,-3.6917839,37.7480513,854.5(0C),0.0(0D),45.0(05),36.0(0F),8.84(10),1.1764706(11)
49401,-3.6917731,37.7480439,856.25(0C),0.0(0D),45.0(05),36.0(0F),8.54(10),1.1764706(11)
```

Ilustración 126 – La respuesta al comando 0104 es recibida en ocasiones contadas

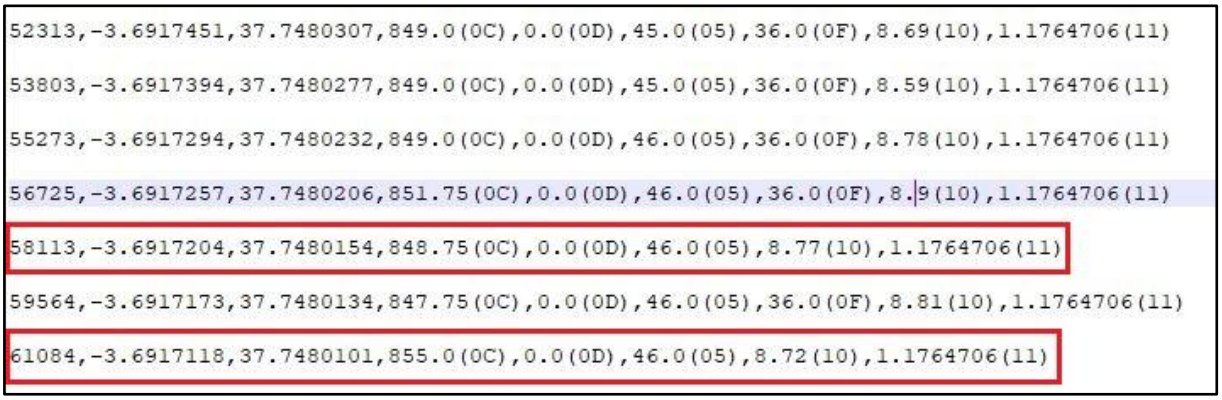


```
944,-3.6921883,37.7479239,851.0(0C),0.0(0D),42.0(05),35.0(0F),6.1(10),0.78431374(11)
2383,-3.6921883,37.7479239,789.75(0C),0.0(0D),42.0(05),35.0(0F),6.73(10),0.78431374(11)
3802,-3.6921883,37.7479239,820.0(0C),3.0(0D),42.0(05),35.0(0F),7.67(10),0.78431374(11)
5219,-3.6921643,37.7478638,851.75(0C),5.0(0D),42.0(05),35.0(0F),6.98(10),0.78431374(11)
6841,-3.69216,37.7478924,878.75(0C),4.0(0D),42.0(05),35.0(0F),6.91(10),0.78431374(11)
8285,-3.6921522,37.7479071,839.5(0C),0.0(0D),42.0(05),35.0(0F),6.11(10),0.78431374(11)
9763,-3.6921501,37.7479295,1379.0(0C),5.0(0D),42.0(05),35.0(0F),20.63(10),3.137255(11)
11219,-3.6921341,37.7479309,1329.75(0C),9.0(0D),42.0(05),35.0(0F),12.71(10),1.5686275(11)
12573,-3.6921355,37.7479229,1451.5(0C),9.0(0D),42.0(05),35.0(0F),18.76(10),2.745098(11)
14052,-3.6921672,37.7479111,1554.5(0C),13.0(0D),43.0(05),35.0(0F),13.77(10),1.9607843(11)
15548,-3.6921891,37.7479155,858.0(0C),13.0(0D),43.0(05),35.0(0F),8.17(10),1.1764706(11)
17122,-3.6922267,37.7479419,803.25(0C),11.0(0D),43.0(05),35.0(0F),8.39(10),1.1764706(11)
```

Ilustración 127 – Finalmente el orden de captura era correcto

El hecho de que fuera el comando de la carga del motor el que originaba los problemas fue una sorpresa pues se habían realizado pruebas en anteriores iteraciones y siempre se recibía la respuesta a este comando de forma correcta.

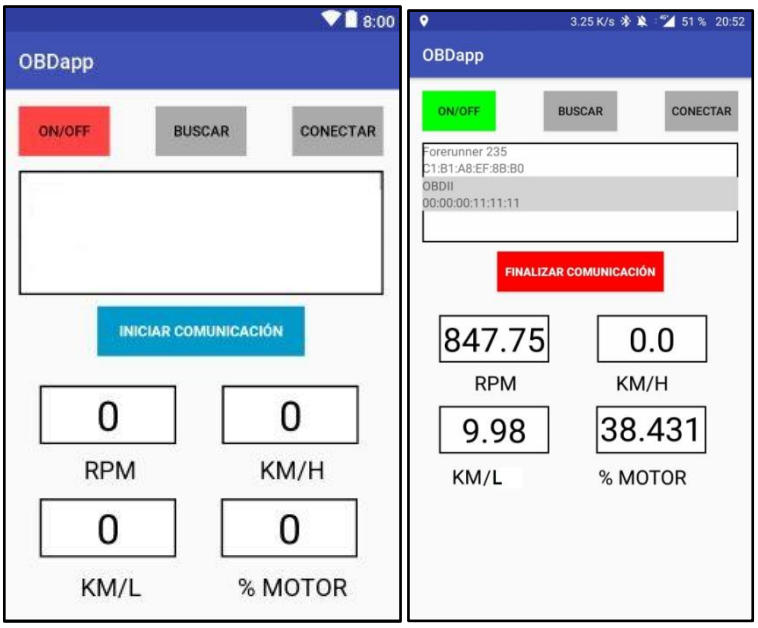
Además, en las nuevas pruebas realizadas se detectó que en ciertas ocasiones en un mismo trayecto, la aplicación móvil no recibía respuesta a determinados comandos que en otras muchas ocasiones sí hacía. Este problema no fue posible corregirlo pues no se debía a la aplicación implementada sino al dispositivo de diagnóstico o al vehículo con el que se realizaron las pruebas. El no poder solucionar este problema nos obliga a revisar manualmente los ficheros creados por la aplicación de captura para que las mediciones no queden incompletas ya que de la otra forma provocarían errores en la aplicación de visualización de los datos recabados.



```
52313,-3.6917451,37.7480307,849.0(0C),0.0(0D),45.0(05),36.0(0F),8.69(10),1.1764706(11)
53803,-3.6917394,37.7480277,849.0(0C),0.0(0D),45.0(05),36.0(0F),8.59(10),1.1764706(11)
55273,-3.6917294,37.7480232,849.0(0C),0.0(0D),46.0(05),36.0(0F),8.78(10),1.1764706(11)
56725,-3.6917257,37.7480206,851.75(0C),0.0(0D),46.0(05),36.0(0F),8.9(10),1.1764706(11)
58113,-3.6917204,37.7480154,848.75(0C),0.0(0D),46.0(05),8.77(10),1.1764706(11)
59564,-3.6917173,37.7480134,847.75(0C),0.0(0D),46.0(05),36.0(0F),8.81(10),1.1764706(11)
61084,-3.6917118,37.7480101,855.0(0C),0.0(0D),46.0(05),8.72(10),1.1764706(11)
```

Ilustración 128 - En los cuadros rojos, mediciones a las que les falta una respuesta a un comando, en concreto al comando de la temperatura del aire de admisión al motor, 010F

A continuación se mostrarán mediante capturas los resultados obtenidos por las aplicaciones desarrolladas durante este proyecto. En primer lugar la aplicación de captura de datos:



Ilustraciones 129 y 130– Aplicación de captura de datos al inicio y tras establecer comunicación con el dispositivo OBD2

Captura del directorio donde se almacenan los datos:

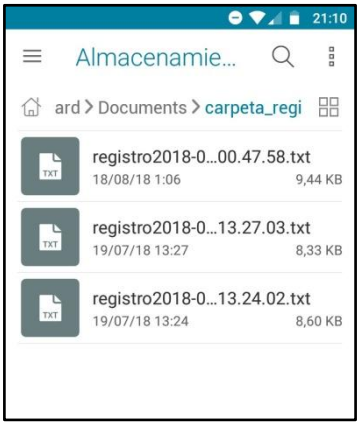
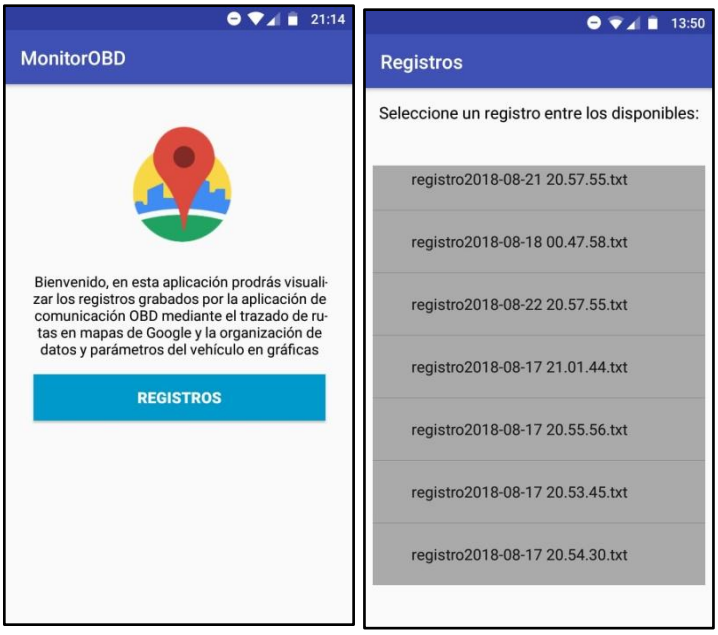


Ilustración 131 – Directorio con ficheros de datos capturados

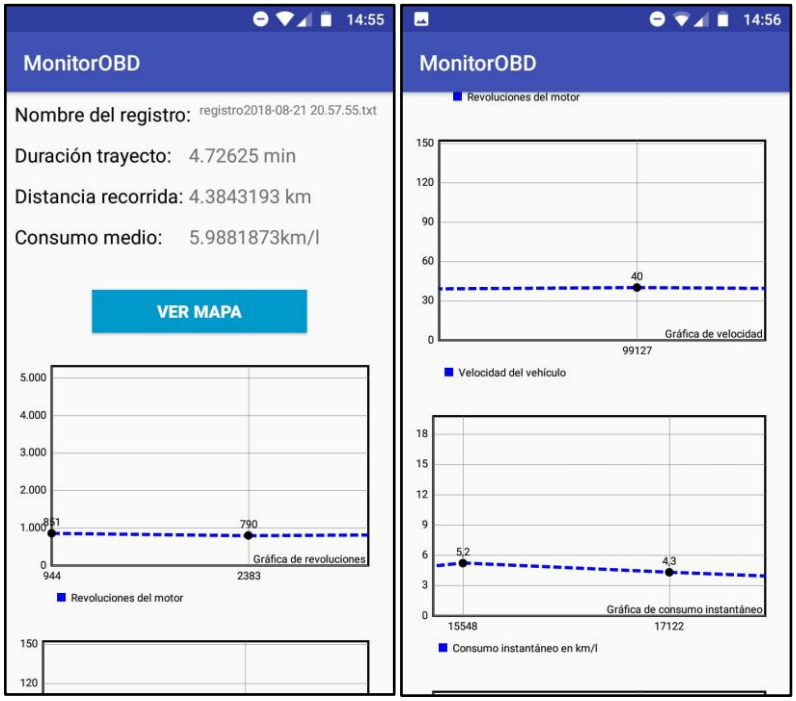
```
944,-3.6921883,37.7479239,851.0(0C),0.0(0D),42.0(05),35.0(0F),6.1(10),0.78431374(11)
2383,-3.6921883,37.7479239,789.75(0C),0.0(0D),42.0(05),35.0(0F),6.73(10),0.78431374(11)
3802,-3.6921883,37.7479239,820.0(0C),3.0(0D),42.0(05),35.0(0F),7.67(10),0.78431374(11)
5219,-3.6921643,37.7478638,851.75(0C),5.0(0D),42.0(05),35.0(0F),6.98(10),0.78431374(11)
6841,-3.69216,37.7478924,878.75(0C),4.0(0D),42.0(05),35.0(0F),6.91(10),0.78431374(11)
8285,-3.6921522,37.7479071,839.5(0C),0.0(0D),42.0(05),35.0(0F),6.11(10),0.78431374(11)
9763,-3.6921501,37.7479295,1379.0(0C),5.0(0D),42.0(05),35.0(0F),20.63(10),3.137255(11)
11219,-3.6921341,37.7479309,1329.75(0C),9.0(0D),42.0(05),35.0(0F),12.71(10),1.5686275(11)
12573,-3.6921355,37.7479229,1451.5(0C),9.0(0D),42.0(05),35.0(0F),18.76(10),2.745098(11)
14052,-3.6921672,37.7479111,1554.5(0C),13.0(0D),43.0(05),35.0(0F),13.77(10),1.9607843(11)
15548,-3.6921891,37.7479155,858.0(0C),13.0(0D),43.0(05),35.0(0F),8.17(10),1.1764706(11)
17122,-3.6922267,37.7479419,803.25(0C),11.0(0D),43.0(05),35.0(0F),8.39(10),1.1764706(11)
```

Ilustración 132 – Datos capturados

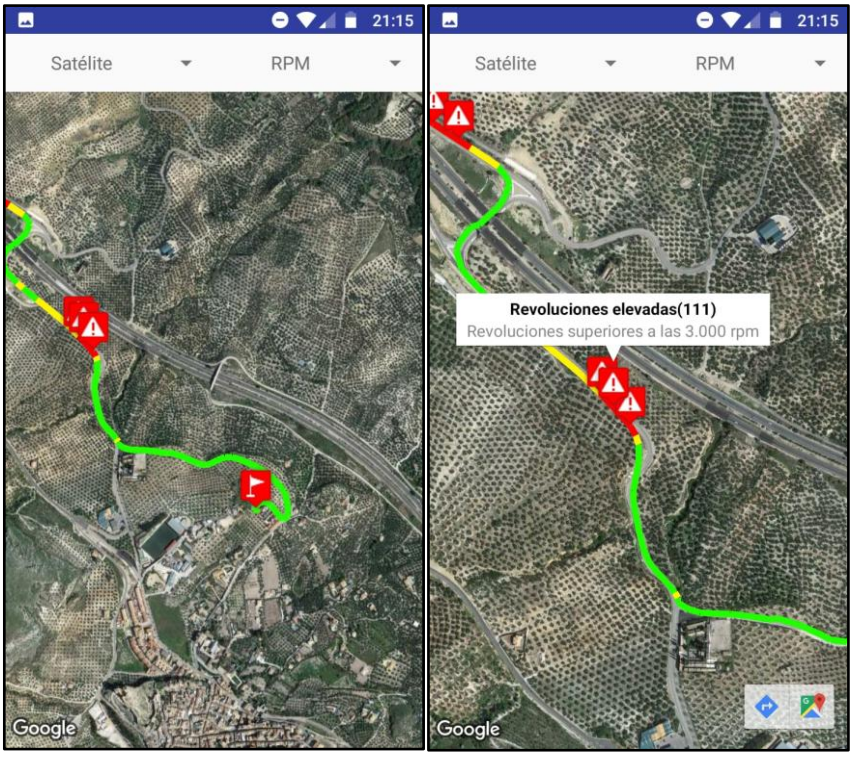
A continuación las capturas de la aplicación de visualización y monitorización de datos:



Ilustraciones 133 y 134 – Pantalla de inicio y pantalla con el listado de ficheros guardados



Ilustraciones 135 y 136 – Gráficas mostradas al seleccionar un fichero del listado



Ilustraciones 137 y 138 – Trazado de la ruta en función de las revoluciones del motor

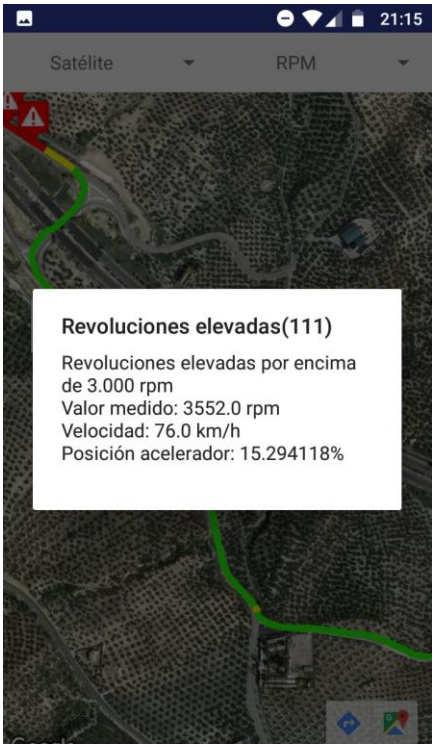
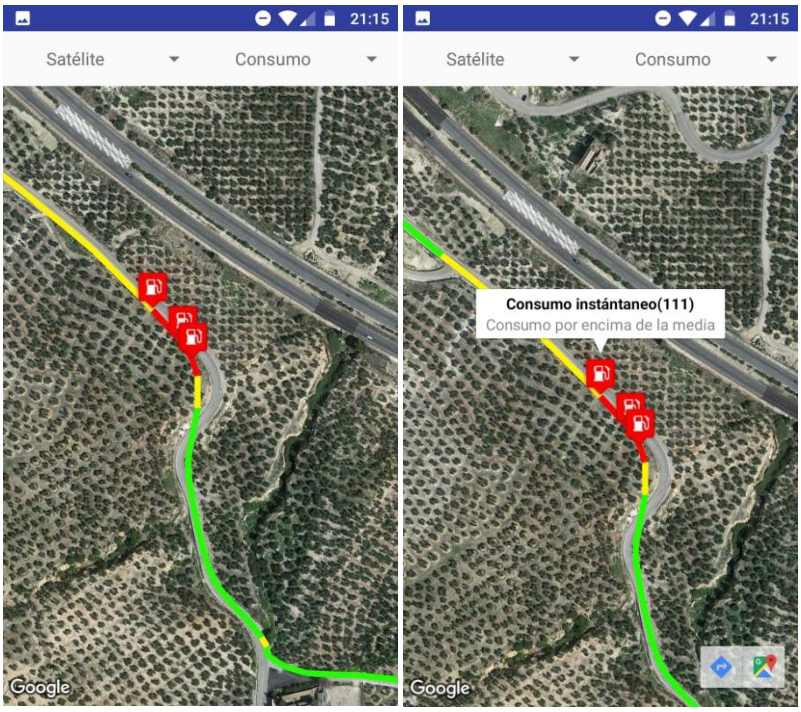


Ilustración 139 – Cuadro con texto informativo mostrado al pulsar el texto descriptivo de un marcador del mapa



Ilustraciones 140 y 141 – Trazado de la ruta en función del consumo instantáneo

5. CONCLUSIONES OBTENIDAS Y TRABAJO FUTURO

En este apartado analizaremos los resultados obtenidos tras finalizar el desarrollo y la implementación de las aplicaciones de captura y visualización de datos. También haremos un repaso a los problemas que no han podido solucionarse y al trabajo futuro para mejorar la funcionalidad y el alcance del proyecto.

En primer lugar, con la aplicación de captura de datos hemos conseguido una aplicación totalmente operativa que establece conexión Bluetooth con el dispositivo ELM327 Mini, configura la comunicación con el mismo e inicia un bucle de consultas sobre los parámetros de conducción. Los parámetros consultados finalmente son las revoluciones del motor, la velocidad del vehículo, la carga del motor, la temperatura del aire de admisión al motor, la temperatura del líquido refrigerante, la posición del acelerador y el flujo de masa de aire que entra al motor para el cálculo del consumo instantáneo. Se ha conseguido que la aplicación guarde los valores de estos parámetros junto a la ubicación geográfica del vehículo y el tiempo desde que se inició la comunicación con el dispositivo de diagnóstico.

El principal problema que queda pendiente en esta aplicación sería determinar por qué en ciertos momentos del trayecto el dispositivo de diagnóstico no devuelve los valores de algunos parámetros quedando las mediciones incompletas. Habría que estudiar si el problema lo originan los vehículos donde se han realizado las pruebas o el dispositivo de medida ELM327. El hecho de no haber podido corregir este problema obliga a revisar manualmente los ficheros generados en busca de mediciones que hayan quedado incompletas y rellenarlas con valores nulos o valores válidos de mediciones anteriores. Si no realizamos esta revisión manual la aplicación de visualización de datos fallará en su ejecución.

En cuanto a la aplicación de visualización hemos conseguido mostrar la información recogida del vehículo de manera cuantitativa a través de las gráficas trazadas y de manera cualitativa a través del trazado del trayecto sobre los mapas y la posibilidad de colorear las rutas en función de los valores de las revoluciones o el consumo. Además de colocar marcadores por el mapa cuando haya que notificar información importante al usuario como la captura de un parámetro con un valor

fuera de su rango normal, temperaturas, velocidad o revoluciones excesivas y de estar forma mostrar tramos del trayecto problemáticos.

Además, el trazado con colores nos permite observar los puntos del trayecto donde el coche ha realizado un mayor esfuerzo o ha consumido más combustible de lo normal e incluso se pueden apreciar los momentos en los que el coche se ha puesto en marcha y ha acelerado para ponerse a la velocidad adecuada de la vía. Por ejemplo, en la siguiente ilustración podemos apreciar como el vehículo realizó un stop o un ceda el paso antes de incorporase a una vía, en el primer tramo de incorporación podemos ver la ruta de color amarillo lo que significa que el coche se aceleró para ponerse en marcha y adecuarse la velocidad de la nueva vía. Una vez el coche se aceleró lo suficiente se cambió de marcha y vehículo volvió a circular a unas revoluciones menores a 2.500 rpm

Ilustración 142– Incorporación del vehículo a una vía

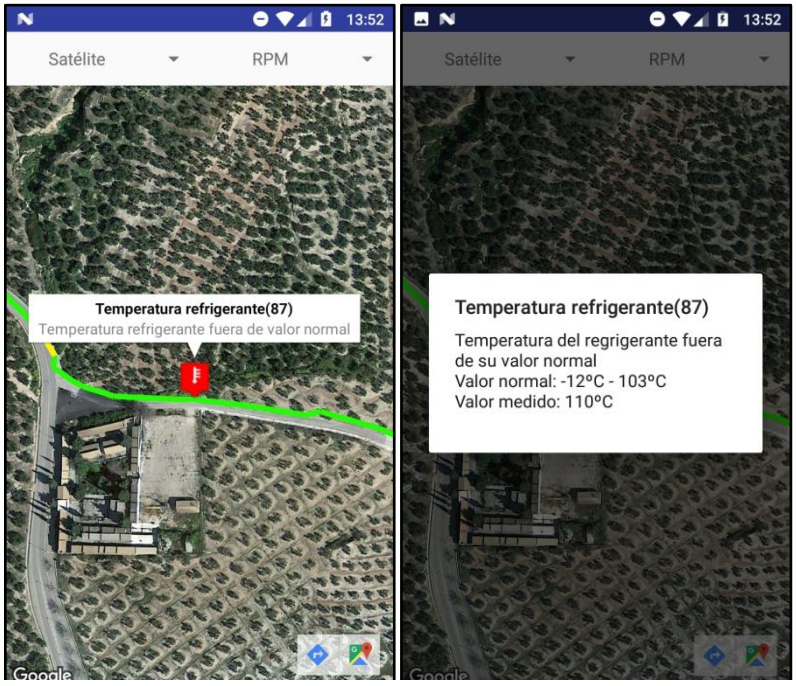
Lo mismo sucede en las situaciones de frenado tal y como podemos observar en la siguiente imagen, en este caso el vehículo se aproxima a una curva y tiene que reducir la velocidad, el conductor cambia de marcha pero la velocidad sigue siendo algo alta para la nueva marcha por lo que el motor se revoluciona un poco coloreando la ruta de amarillo.

Ilustración 143 – Aproximación a una curva

En este último caso las revoluciones del motor no alcanzaron valores como para trazar la ruta en rojo y colocar un marcador de advertencia pero, un mal hábito como puede ser frenar un poco y reducir la marcha para después volver a frenar podría reflejarse aquí por obtener valores de revoluciones elevados. Aplicando las prácticas de conducción eficiente, el usuario observaría el mal hábito gracias a la aplicación e intentaría en un trayecto futuro frenar todo lo posible antes de cambiar de marcha para hacerlo en las revoluciones del motor adecuadas.

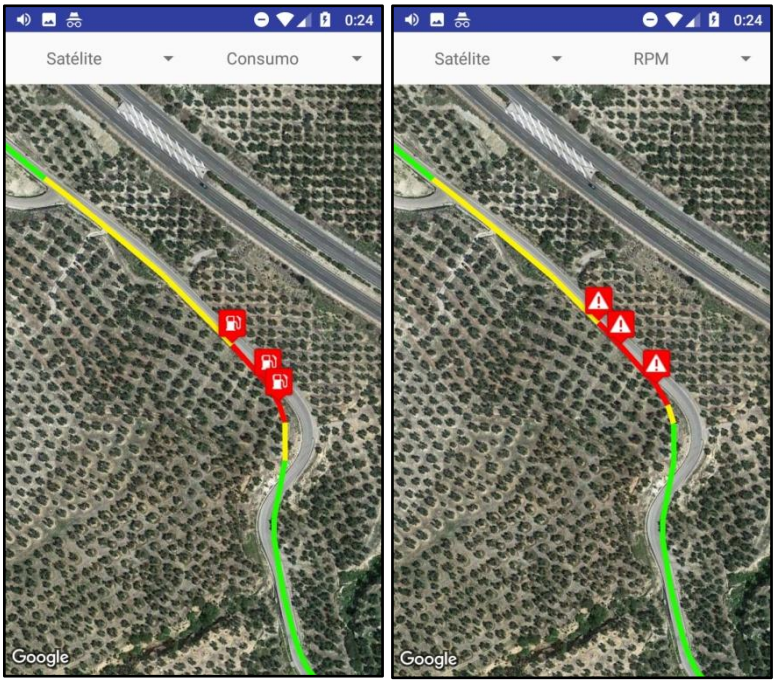
Lo cierto es que en estas situaciones sería muy útil poder conseguir los valores del pedal del freno para saber si el conductor está frenando con el pedal o con el freno del motor, el cual es más recomendable. Pero tras el estudio del sistema OBD2 no se encontró ningún comando asociado a este componente.

Por otro lado la posibilidad de medir las temperaturas del líquido refrigerante y del aire de admisión del motor permite notificar en la aplicación momentos del trayecto donde han alcanzado valores fuera del rango normal de los mismos. De esta forma se le notifica al usuario que probablemente exista un problema en el motor que debe solucionarse inmediatamente.



Ilustraciones 144 y 145 – Notificación de temperatura del refrigerante fuera de valor normal

Por último, los valores del consumo instantáneo por sí mismos nos permiten conocer en qué tramos del trayecto el consumo se ha elevado por encima de la media. Pero si comparamos las rutas trazadas en función de las revoluciones y en función del consumo instantáneo notaremos cierta similitud.



Ilustraciones 146 y 147 – A la izquierda ruta trazada sobre el mismo tramo en función del consumo, a la derecha en función de las revoluciones del motor

Como podemos observar en las dos ilustraciones anteriores los valores del consumo instantáneo van acorde con los valores de las revoluciones, si éstas se elevan por encima de las 3.000 o 3.200 rpm se incrementa el consumo porque el motor necesita más combustible en ese momento determinado.

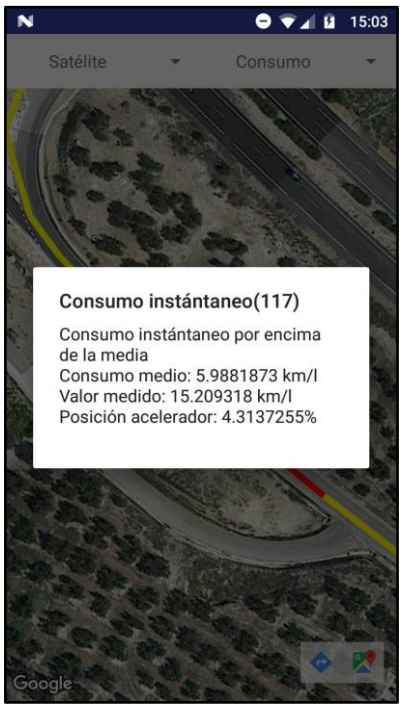


Ilustración 148 – Consumo instantáneo por encima de la media

Por otro lado, en la pantalla con las gráficas podemos evaluar los parámetros captados a lo largo del trayecto. Gracias a la librería utilizada para las gráficas, éstas son totalmente interactivas y podemos desplazarnos a lo largo y ancho de los ejes. En total se muestran cuatro gráficas, una para las revoluciones del motor, otra para la velocidad, una tercera para el consumo instantáneo y una cuarta para las temperaturas del líquido refrigerante y del aire de admisión al motor.

En cuanto a las mejoras que pueden aplicarse a esta aplicación de visualización de datos el primer paso sería la posibilidad de poder medir los valores de las marchas y el frenado porque permitirían más opciones de personalización y estudio de la conducción eficiente.

Como hemos podido comprobar las aplicaciones desarrolladas en este proyecto son un pequeño prototipo que sirve de aproximación a algo mucho mayor y más novedoso como es la posibilidad de implementar una aplicación de las mismas características que la desarrollada en el proyecto, pero que permita poder registrar usuarios. Cada usuario podría tener un perfil, varios vehículos asociados y sus trayectos realizados. La parte del trazado de rutas se podría implementar de tal forma que el usuario pudiese comparar varias rutas realizadas por el mismo o por otros usuarios. Esta aplicación mucho más avanzada podría no solo servir para mejorar la conducción del usuario y hacerla más eficiente, sino que, como se explicó en el apartado introductorio, podría servir de mucha utilidad para aquellas empresas que posean una flota de vehículos, de esta manera las empresas podrían comparar varias rutas con un mismo punto de inicio y destino o comparar la conducción de sus conductores.

6. ANEXOS

6.1. Manual de la aplicación de captura de datos

Esta aplicación consta únicamente de una sola pantalla donde se establecerá la conexión Bluetooth con el dispositivo de diagnóstico OBD2 y posteriormente se iniciará la comunicación con el mismo mediante el envío de consultas sobre los parámetros de conducción del vehículo.

Tras iniciar la aplicación encontramos tres botones en la parte superior de la pantalla. El primero, el de la izquierda, se utiliza para activar o desactivar la conexión Bluetooth de nuestro dispositivo móvil.

Ilustración 149 – Estado inicial de la aplicación

El segundo botón, “BUSCAR”, situado al centro, se utiliza para la búsqueda de dispositivos Bluetooth una vez hayamos activado esta característica en nuestro smartphone. Tras pulsar este botón se creará una lista con todos los dispositivos Bluetooth encontrados en el cuadro blanco que hay justo debajo de los tres botones superiores. En esta lista debemos buscar el dispositivo OBD2 y seleccionarlo, tras hacerlo pulsaremos el botón “CONECTAR” que permitirá establecer la conexión Bluetooth entre nuestro dispositivo móvil y el dispositivo OBD2.

Ilustración 150 – Estado de la aplicación tras buscar dispositivos Bluetooth

Una vez establecida la conexión, la aplicación mostrará en pantalla un mensaje notificando al usuario del éxito de la misma. Solamente cuando sea mostrado este mensaje el usuario podrá iniciar la comunicación mediante el envío de comandos al dispositivo OBD2 pulsando el botón “INICIAR COMUNICACIÓN”. Tras pulsar el botón, éste cambia a color rojo y su texto cambia a “FINALIZAR COMUNICACIÓN”. Inmediatamente la aplicación actualiza los datos de los parámetros del vehículo recibidos durante la comunicación modificando la interfaz de la aplicación.

Ilustración 151 – Estado de la aplicación tras iniciar la comunicación con el dispositivo OBD2

Cuando el usuario desee finalizar con el envío de consultas debe pulsar el botón de “FINALIZAR COMUNICACIÓN”, de esta forma la aplicación volverá al estado anterior a establecer la conexión Bluetooth con el dispositivo OBD2. Además, tras finalizar la comunicación la aplicación guardará un archivo con todos los datos recibidos del vehículo en una carpeta que será creada la primera vez que se utilice la aplicación. La carpeta llamada “carpeta_registros” se encuentra dentro de la carpeta “Documents” situada en la raíz de la memoria interna del teléfono móvil.

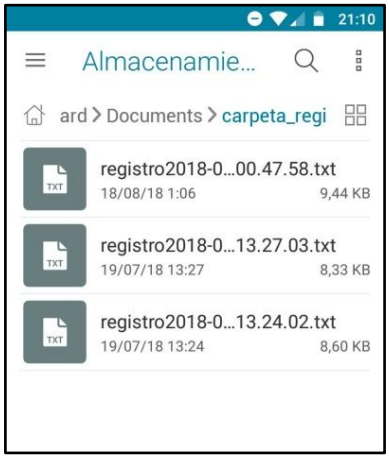


Ilustración 152 – Carpeta con los registros almacenados por la aplicación

6.2. Manual de la aplicación de visualización de datos

Una vez iniciada la aplicación, la primera pantalla que encontramos es una pantalla de bienvenida con un texto descriptivo sobre las características de la aplicación y un botón para acceder al listado de ficheros que crea la aplicación de captura de datos.

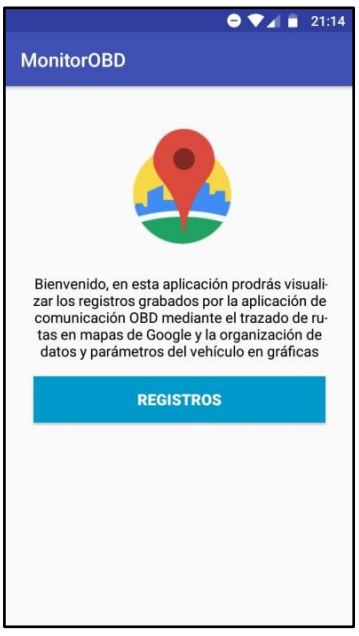


Ilustración 153 – Pantalla de bienvenida

Una vez pulsado el botón de “REGISTROS”, pasamos a una nueva pantalla donde se listan los ficheros con los datos recabados del vehículo almacenados en el

teléfono móvil. Si no existe ningún fichero almacenado se mostrará un mensaje para notificárselo al usuario.

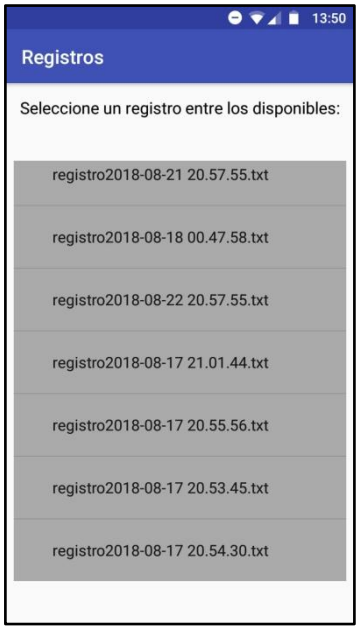


Ilustración 154 – Pantalla con el listado de ficheros almacenamos en el dispositivo móvil

Ilustración 155 – Pantalla con el listado de ficheros cuando no existen ficheros almacenados en el dispositivo móvil

Al pulsar en cualquiera de los ficheros que aparecen en la lista se pasará a otra nueva pantalla donde se podrán visualizar en gráficas los valores de algunos parámetros del vehículo recogidos en el fichero seleccionado. También encontraremos información acerca de la duración del trayecto, la distancia recorrida, el consumo instantáneo medio, etc.

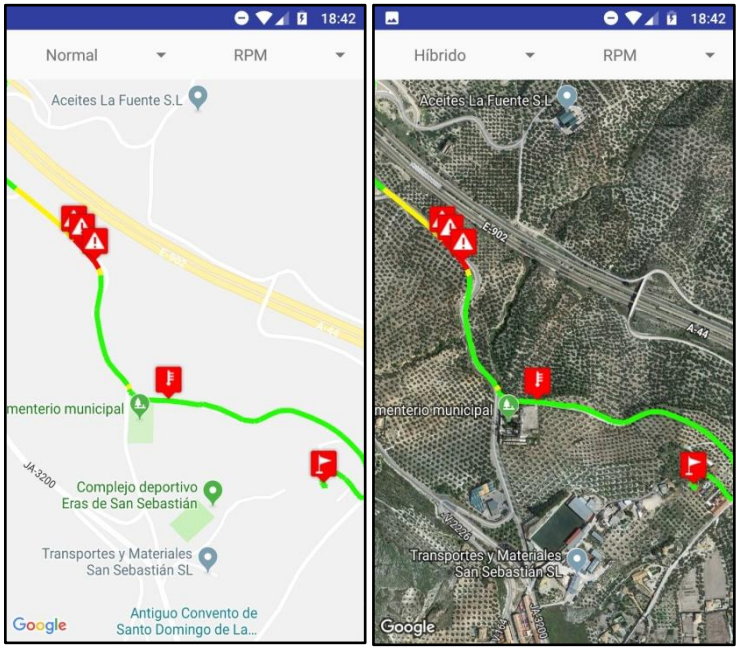
Ilustración 156 – Pantalla con las gráficas e información relacionada con el fichero seleccionado

En esta pantalla con información y gráficas se dispondrá de un botón para poder acceder al mapa donde se trazará la ruta realizada durante el trayecto del fichero seleccionado. Se trata del botón “VER MAPA”.

Una vez pulsado este botón pasaremos a la última pantalla de la aplicación. En ella se traza por defecto la ruta realizada sobre el mapa en función de los valores de las revoluciones del motor aunque, también encontraremos información acerca de la velocidad, la posición del acelerador o las temperaturas del líquido refrigerante y del aire de admisión al motor.

Ilustración 157 – Ruta trazada en función de las revoluciones del motor

En la parte de arriba de la nueva pantalla encontramos también dos botones que al pulsarlos muestran en pantalla menús desplegables con distintas opciones. El primero, el de la izquierda, permite cambiar el tipo de mapa que se mostrará en pantalla. Por defecto, está configurado para que se muestre el mapa “Normal”, las otras opciones que podemos elegir al pulsar el botón son “Satélite”, “Híbrido” y “Carretera”. Pulsando cualquiera de estas opciones el mapa cambia automáticamente.



Ilustraciones 158 y 159 – Diferentes tipos de visualización del mapa

El botón de arriba a la derecha es el encargado de trazar la ruta en el mapa en función de distintos parámetros medidos. Como se ha explicado anteriormente, por defecto se traza la ruta en función de las revoluciones del motor por lo que el botón aparecerá con el texto “RPM”. Al pulsarlo, el menú muestra la otra opción disponible para el trazado la del consumo instantáneo. Al pulsar la nueva opción, automáticamente se realiza el nuevo trazado de la ruta en función de los valores del consumo instantáneo.

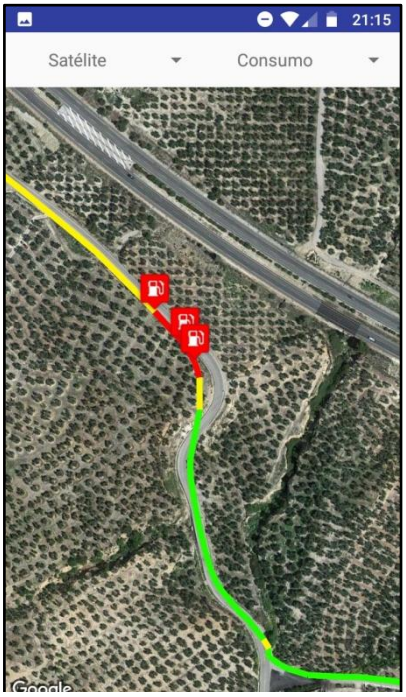
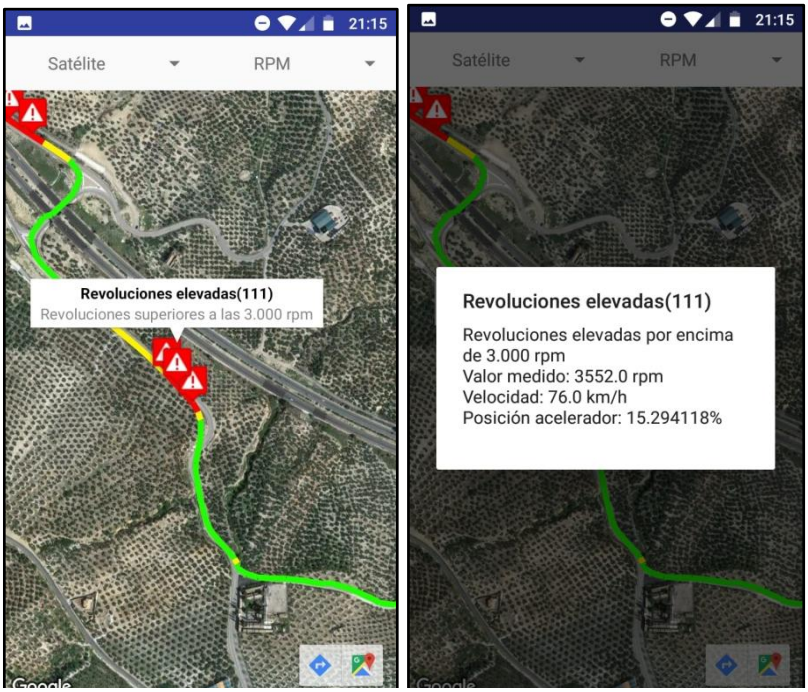


Ilustración 160 – Trazado de la ruta en función del consumo instantáneo

Por último, durante los trazados de las rutas sobre el mapa la aplicación coloca una serie de marcadores donde se han producido eventos que son necesarios de notificar al usuario. Los marcadores se identifican con distintos tipos de iconos. Si el usuario pulsa sobre un icono del mapa se muestra sobre el icono un pequeño texto descriptivo con información acerca del marcador. Este pequeño texto descriptivo se enmarca dentro de un bocadillo que aparece en pantalla justo encima del icono del marcador. Si el usuario pulsa sobre el bocadillo de texto se mostrará en pantalla un cuadro de texto con información descriptiva ampliada sobre el evento.



Ilustraciones 161 y 162 – Mensajes descriptivos con información de los eventos

Bibliografía

[1] - Metodología SCRUM, definición, componentes, detalles e iteraciones. Obtenido de https://proyectosagiles.org/que-es-scrum/

Conociendo el sistema OBD2 desde cero. Obtenido de https://learn.sparkfun.com/tutorials/getting-started-with-obd-ii

[2] - Sistema OBD2, protocolos y estándares. Obtenido de https://mdautomotive.es/los-sistemas-diagnostico-obd-obdii-conector-protocolos-estandares/

Leer datos a tiempo real desde un dispositivo OBD. Obtenido de https://www.obdsol.com/knowledgebase/obd-software-development/reading-real-time-data/

[3] - Comandos PID, descripciones, valores y fórmulas. Obtenido de https://en.wikipedia.org/wiki/OBD-II_PIDs

[4] - Librería de comunicación OBD2 disponible en GitHub. https://github.com/pires/obd-java-api

[5] - Proyecto OBD Reader de Android Studio basado en la anterior librería de GitHub. https://github.com/pires/android-obd-reader

Inicializando el dispositivo OBD2 con comandos AT. Obtenido de https://stackoverflow.com/questions/13764442/initialization-of-obd-adapter

Manual de usuario del dispositivo OBD2 el ELM327. Obtenido de https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf

Comunicación OBD2 con el dispositivo ELM327. Obtenido de http://dthoughts.com/blog/2014/11/06/obd-scanner-using-elm327/

[6] - Calcular el consumo instantáneo a través del flujo de masa de aire (MAF). http://www.windmill.co.uk/obdii.pdf

ELM327 para su compra en la web Amazon. https://www.amazon.es/Interfaz-Bluetooth-herramienta-an%C3%A1lisis-diagn%C3%B3stico/dp/B00WU97RH8/ref=sr_1_fkmr0_1?ie=UTF8&qid=1516814624&sr=8-1-fkmr0&keywords=ELM327+AT

Guía para iniciarse en Android Studio. Obtenida de https://developer.android.com/training/basics/firstapp/creating-project?hl=es-419

Curso introductorio a Android Studio. Obtenido de https://www.youtube.com/watch?v=pO7STLF82Ro&list=PLpOqH6AE0tNh5rvbCb03w8ORR8bOoftZ6

[7] - Uso de Bluetooth en el desarrollo de aplicaciones para Android. Obtenido de https://developer.android.com/guide/topics/connectivity/bluetooth

Android Bluetooth Chat Service. Obtenido de https://github.com/googlesamples/android-BluetoothChat/#readme

Proyecto en GitHub para el desarrollo de aplicaciones Android con uso de comunicación Bluetooth. Obtenido de https://github.com/mitchtabian/Sending-and-Receiving-Data-with-Bluetooth/tree/master/Bluetooth-Communication/app/src/main/java/com/example/user/bluetooth_communication

Obtener localización en aplicaciones Android. Obtenido de https://stackoverflow.com/questions/33415033/getting-current-location-in-android-studio-app

Obtener localización en aplicaciones Android (fuente alternativa). Obtenido https://medium.com/@victor.garibayy/obteniendo-mi-ubicaci%C3%B3n-en-android-studio-377226910823

[8] - Android FusedLocationProviderClient. Obtenido de https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

[9] - Guardar archivos en la memoria del teléfono móvil desde una aplicación Android. Obtenido de http://www.zoftino.com/saving-files-to-internal-storage-&-external-storage-in-android

Leer y escribir en memoria del teléfono desde una aplicación Android. Obtenido de https://www.dev2qa.com/android-read-write-external-storage-file-example/

Implementación de Google Maps en aplicaciones Android (Tutorial). Obtenido de https://www.youtube.com/watch?v=Z4X381D-_ZY&list=PL2LFsAM2rdnyWHxGwFrsTXxrGpadj3dP_

Google Maps en aplicaciones Android. Obtenido de https://developers.google.com/maps/documentation/android-sdk/start

[10] - Google API Console. https://console.developers.google.com/projectselector/apis/dashboard

Trazar rutas sobre mapas de Google Maps. Obtenido de https://developers.google.com/maps/documentation/android-sdk/polygon-tutorial

Ajustes de la cámara y el zoom de Google Maps. Obtenido de https://developers.google.com/maps/documentation/android-sdk/views?hl=es

[11] -Librería MPAndroidChart para el uso de gráficas en Android. https://github.com/PhilJay/MPAndroidChart

Wiki de la librería MPAndroidChart. Obtenida de https://github.com/PhilJay/MPAndroidChart/wiki

Gráficos de líneas con MPAndroidChart. Obtenido de https://www.numetriclabz.com/android-line-chart-using-mpandroidchart-tutorial/

[12] -Spinners en Android Studio. Obtenido de https://developer.android.com/guide/topics/ui/controls/spinner?hl=es-419

Listados en Android con ListViews. Obtenido de http://www.vogella.com/tutorials/AndroidListView/article.html

[13] -Guía para crear aplicaciones que mantengan el dispositivo móvil desbloqueado. Obtenido de https://developer.android.com/training/scheduling/wakelock

[14] -Guía de la DGT para conducción eficiente. http://www.dgt.es/PEVI/documentos/catalogo_recursos/didacticos/did_adultas/Conduccion_eficiente.pdf

Conducción eficiente según la Secretaría de Estado para la energía http://www.controlastuenergia.gob.es/consumo-inteligente/Paginas/conduccion-eficiente.aspx

Conducción eficiente según la CEA (Comisariado Europeo del Automóvil). https://www.cea-online.es/blog/139-las-10-claves-de-la-conduccion-eficiente

Manual de conducción eficiente. Ministerio de Fomento del Gobierno de España. http://www.idae.es/uploads/documentos/documentos_10297_TREATISE_ConduccionEficienteVehIndustriales_A2005_2ad0233c.pdf

[15] -Sistema de refrigeración de un vehículo. https://conducirseguro.wordpress.com/tag/circuito-refrigerante/

[16] -Refrigerante y temperatura del motor. https://www.autofit-spain.es/sirve-cualquier-liquido-refrigerante-o-anticongelante-para-mi-coche/

Refrigerante y temperatura del motor (fuente alternativa).https://www.autopista.es/trucos-y-consejos/articulo/temperatura-idonea-aceite-refrigerante-motor-no-falle-sin-averias

[17] -Sensor de temperatura de aire de admisión. https://www.hella.com/techworld/es/Tecnica/Sensores-y-actuadores/Sensor-de-temperatura-del-aire-de-admision-4326/#

[18] -Estudio sobre las distintas formas de obtener el consumo instantáneo y las diferencias entre los resultados obtenidos.http://hemdata.com/products/dawn/enews/fuel-economy-obd

Fuel Consumption Gauge. Circuit Cellar. http://www.lightner.net/lightner/bruce/Lightner-183.pdf

[20] - Iconos para los marcadores de los mapas. Obtenido de https://mapicons.mapsmarker.com/about/iconos-para-mapas/