TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Control de flotas

Autor

Gonzalo Montalbán Tirado

Directores

Sandra Sendra Compte

Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, Junio de 2018

Control de flotas

Autor
Gonzalo Montalbán Tirado

Directores
Sandra Sendra Compte
Jorge Navarro Ortiz

Control de flotas

Gonzalo Montalbán Tirado

Palabras clave: OBD, CANBus, I2C, ESP8266, Arduino, GPS, ECU, Internet Of Things, WLAN, OSI, PID, Electrónica, telemetría, flotas.

Resumen

El objetivo del presente proyecto es realizar un sistema de telemetría que sea capaz de obtener la información de estado de vehículos con el objetivo de realizar un diagnóstico continuo gracias a un sistema de gestión de datos remoto orientado al control de flotas vehiculares.

Para la medición de la información del vehículo se utilizará un adaptador de la empresa Freematics que hará de puente entre la interfaz de diagnóstico OBD del vehículo y un arduino Nano. Por cada medición completa del vehículo se enviará la información a un Wemos D1 Mini, el cual se encargará de enviar la información al sistema de gestión de datos mediante una tecnología de red.

De esta manera se consigue hacer una separación entre la extracción de datos y el envío de estos mediante la red inalámbrica, lo cual nos proporcionará facilidad en la modificación del envío y/o adición de sensores externos a la electrónica del vehículo cuyo estudio sea de interés.

Respecto a las tecnologías empleadas se busca que aporten valor al factor académico gracias a que estas sean de uso extenso y emergentes. De este modo a lo largo de este trabajo se presentarán y emplearán tecnologías usadas actualmente en la industria o proyección sobre ella, como es el caso del marco del Internet Of Things (IoT).

Fleets management

Gonzalo Montalbán Tirado

Keywords: OBD, CANBus, I2C, ESP8266, Arduino, GPS, ECU, Internet Of Things, WiFi, OSI, PID, WLAN, Electronic, telemetry.

Abstract

The objective of this project is to create a telemetry system that is capable of obtaining vehicle status information. With the purpose of making a continuous diagnosis thanks to a remote data management system oriented to the control of vehicle fleets.

For the measurement of vehicle information, an adapter from Freematics will be used as a bridge between the OBD diagnostic interface of the vehicle and an Arduino Nano. For each complete measurement of the vehicle, the information will be sent to a Wemos D1 Mini, which will be responsible for sending the information to the data management system through a network technology.

In this way, it is possible to make a separation between the extraction of data and the sending of these through the wireless network, which will provide us with ease in modifying the sending and / or adding external sensors to the electronics of the vehicle whose study is of interest.

Regarding the technologies used, it is sought that they contribute value to the academic factor thanks to the fact that they are widely used and emerging. In this way throughout this work will be presented and used technologies currently used in the industry or projection on it, as is the case of the Internet Of Things (IoT) framework.

Yo, **Gonzalo Montalbán Tirado**, alumno de la titulación Grado en Ingeniería en Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultad a por las personas que lo deseen.

Fdo: Gonzalo Montalbán Tirado

Granada a 18 de junio de 2018.

D. **Sandra Sendra Compte**, Profesora del Área de Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Jorge Navarro Ortiz**, Profesor del Área de Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Control de flotas***, ha sido realizado bajo su supervisión por Gonzalo Montalbán Tirado, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 18 de junio de 2018.

Los directores:

Sandra Sendra Compte **Jorge Navarro Ortiz**

Agradecimientos

En primer lugar, agradezco a todo aquel que estoy seguro de que no necesitará leer estas líneas para saber lo agradecido que estoy por haberme acompañado en este camino. Lo más importante en la vida es la gente que está aquí y ahora.

No obstante, el mayor agradecimiento es para mis padres y hermana, que sin duda han confiado en mí siempre y han hecho que nunca faltaran las ganas de seguir adelante.

Es de mención especial a todos los compañeros con los que he compartido clase, pero es de destacar a todos aquellos que son mucho más amigos que compañeros. Ellos son los que han hecho que cada día de estos años haya sido para el recuerdo.

Por último, cabe destacar a mis tutores Sandra y Jorge por el tiempo dedicado, y a la Universidad de Granada, la cual siempre formará parte de mí.

Índice general

Índice de figuras

Índice de tablas

Glosario

ACK: Acuse de recibo o asentimiento. Es un mensaje que el destino de la comunicación envía al origen de esta para confirmar la recepción de un mensaje

Android: Sistema operativo que se emplea en dispositivos móviles, por lo general con pantalla táctil. De este modo, es posible encontrar dispositivos electrónicos como teléfonos móviles y relojes equipados con Android, aunque el software también se usa en automóviles, televisores y otras máquinas.

App Blynk: Plataforma con aplicaciones iOS y Android para controlar Arduino, Raspberry Pi y otros dispositivos a través de Internet. Es un tablero digital donde puedes construir una interfaz gráfica para tu proyecto simplemente arrastrando y soltando widgets.

Big Data: Término que hace referencia a una cantidad de datos tal que supera la capacidad del software convencional para ser capturados, administrados y procesados en un tiempo razonable. El volumen de los datos masivos crece constantemente.

Bootloader: Programa sencillo que no tiene la totalidad de las funcionalidades de un sistema operativo, y que está diseñado exclusivamente para preparar todo lo que necesita para iniciar el sistema operativo.

Cableado Jumper: Es un cable con un conector en cada punta, que se usa para interconectar entre sí los componentes en una protoboard.

Cisco Systems: Empresa global, principalmente dedicada a la fabricación, venta, mantenimiento y consultoría de equipos de telecomunicaciones.

Cloud Computing: Conocida también como servicios en la nube, informática en la nube, nube de cómputo, nube de conceptos o simplemente "la nube", es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.

Dashboard: Interfaz gráfica de usuario donde este puede administrar el equipo y/o software.

Diagrama de Gantt: Herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

Drag-And-Drop: Expresión informática que se refiere a la acción de mover, con el cursor del ratón, los objetos de una ventana a otra o entre partes de una misma ventana. Los objetos arrastrados son habitualmente archivos, pero también pueden ser arrastrados otros tipos de elementos en función del software.

ELM327: Microcontrolador programado producido por ELM Electronics para traducir la interfaz de diagnóstico a bordo (OBD) que se encuentra en la mayoría de los automóviles modernos. El protocolo de comando ELM327 es uno de los estándares de interfaz de PC a OBD más populares y también lo implementan otros proveedores.

Formula Student: Competición entre estudiantes de universidades de todo el mundo que promueve la excelencia en ingeniería a través de una competición donde los miembros del equipo diseñan, construyen, desarrollan y compiten un vehículo monoplaza.

Front-End: Parte del software que interactúa con los usuarios y el back-end es la parte que procesa la entrada desde el front-end. La separación del sistema en front-ends y back-ends es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas.

Hardware: Partes físicas tangibles de un sistema informático, sus componentes eléctricos, electrónicos, electromecánicos y mecánicos. Cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado componen el hardware.

Inteligencia Artificial: Inteligencia exhibida por máquinas. En ciencias de la computación, una máquina "inteligente" ideal es un agente racional flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea

IOS: Sistema operativo móvil de la multinacional Apple Inc. Originalmente desarrollado para el iPhone), después se ha usado en dispositivos como el iPod touch y el iPad. No permite la instalación de iOS en hardware de terceros.
IoT: Sistema de dispositivos de computación interrelacionados, máquinas mecánicas y digitales, objetos, animales o personas que tienen identificadores únicos y la capacidad de transferir datos a través de una red, sin requerir de interacciones humano a humano o humano a computadora.

LoRaWAN: Especificación para redes de baja potencia y área amplia, LPWAN (en inglés, Low Power Wide Area Network), diseñada específicamente para dispositivos de bajo consumo de alimentación, que operan en redes de alcance local, regional, nacionales o globales.

Open Source: Modelo de desarrollo de software basado en la colaboración abierta. Se enfoca más en los beneficios prácticos (acceso al código fuente) que en cuestiones éticas o de libertad que tanto se destacan en el software libre.

Protoboard: Placa de pruebas o placa de inserción, es un tablero con orificios que se encuentran conectados eléctricamente entre sí de manera interna, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares.

5G: Siglas utilizadas para referirse a la quinta generación de tecnologías de telefonía móvil. Es la sucesora de la tecnología 4G. Actualmente se encuentra sin estandarizar y las empresas de telecomunicación están desarrollando sus prototipos. Está previsto que su uso común sea en 2020.

Shield: Las *shields* son placas de circuitos modulares que se montan unas encima de otras para dar funcionalidad extra.

Smart Cities: Sistema e interconectado que aplica las nuevas tecnologías para gestionar desde el correcto funcionamiento de los sistemas de transporte público y privado, hasta el uso eficiente de los recursos energéticos o hídricos, pasando por los planos de protección civil, o aspectos socio-económicos, como la vitalidad de los espacios públicos y del tejido comercial, o la comunicación de incidencias a habitantes y visitantes.

Smartphone: Tipo de ordenador de bolsillo que combina los elementos de una *tablet* con los de un teléfono celular. Sobre una plataforma informática móvil, con mayor capacidad de almacenar datos y realizar actividades, semejante a la de una minicomputadora, y con una mayor conectividad que un teléfono móvil convencional.

Software: Soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas. La interacción entre el software y el hardware hace operativo un ordenador (u otro dispositivo), es decir, el software envía instrucciones que el hardware ejecuta, haciendo posible su funcionamiento.

Tacógrafo: Dispositivo electrónico que registra diversos sucesos originados en un vehículo de transporte terrestre durante su conducción, sea de carga o de pasajeros y de carretera o ferroviario. Los sucesos que registra un tacógrafo normalmente son: distancia recorrida por el vehículo, velocidad (promedio y máxima), aceleraciones y frenadas bruscas, tiempo de ralentí (periodo durante el cual el vehículo permanece detenido con el motor en marcha), tiempos de descanso e interrupciones entre otros.

Telemedicina: Prestación de servicios médicos a distancia. Para su implantación se emplean tecnologías de la información y las comunicaciones. La telemedicina puede ser tan simple como dos profesionales de la salud discutiendo un caso por teléfono, hasta la utilización de avanzada tecnología en comunicaciones e informática para realizar consultas, diagnósticos o cirugías a distancia y en tiempo real.

WiFi: Es una tecnología que permite la interconexión inalámbrica de dispositivos electrónicos. Los dispositivos habilitados con wifi pueden conectarse entre sí o a internet a través de un punto de acceso de red inalámbrica.

Acrónimos

CAN: Controller Area Network
CARB: California Air Resources Board
MIL o Check Engine: Malfunction Indicator Lamp
CSV: Comma-Separated Values
DTC: Diagnostic Trouble Code
ECM: Engine Control Module
ECU: Engine Control Unit
EGR: Exhaust Gas Recirculation
EPA: United States Environmental Protection Agency
FET: Field-Effect Transistor
GPS: Global Positioning System
GSM/ GPRS: Global System for Mobile communications/ General Packet Radio Service
I2C: Inter-Integrated Circuit
IDE: Integrated Development Environment
IoT: Internet of Things
IPv4/6: Internet Protocol version 4/6
M2M: Machine To Machine
OBD: On Board Diagnostics
OLED: Organic light- Emitting Diode
PID: Parametrer ID
PWM: Pulse-Width Modulation
RPM: Revolutions Per Minute
SAE: Society of Automotive Engineers
SCL: Clock signal
SDA: Serial Data
SIM: Subscriber Identity Module
SoC: System on Chip
SPI: Serial Peripheral Interface
SSD: Solid-State Drive

TIC: Tecnologías de información y comunicación
TTL: Transistor-Transistor Logic
UART: Universal Asynchronous Receiver Transmitter
USART: Universal Synchronous Asynchronous Receiver Transmitter
V2I: Vehicle to infrastructure
V2V: Vehicle-to-vehicle
VPW: Variable pulse width

Capítulo 1

Introducción

En la actualidad se está incorporando la tecnología de manera exponencial en los vehículos, los cuales producen información continua de alto nivel de interés. Toda esta numerosa información es de gran utilidad siempre y cuando seamos capaces de procesarla, es por ello que este proyecto se encuadra en la creación de un sistema de adquisición de datos en remoto capaz de ofrecer en tiempo real información relevante de un vehículo.

Haciendo uso de la electrónica con la que cuentan los vehículos, podremos conocer el estado del vehículo y con ello utilizarlo tanto para un posible diagnóstico en caso de averías como para una mayor productividad, como puede ser en ahorro de combustible.

Los vehículos cuentan con un sistema de diagnóstico a bordo el cual controla continuamente el correcto funcionamiento del vehículo, este está formado por una o varias unidades de control del motor (ECU, Engine Control Unit), las cuales procesan la información aportada por un gran número de sensores de los que cuenta cada vehículo, como pueden ser de control químico, mecánico, electrónico, etc.

En caso de un funcionamiento incorrecto del vehículo, la ECU se encarga de comunicárselo al conductor por medio de La luz de servicio del automóvil (Check Engine), por tanto, conocer el estado del vehículo puede aportarnos grandes ventajas en su uso.

No obstante, mediante el cuadro de mandos, la información que obtenemos luminosa es escasa, por tanto, debemos encontrar la manera de extraer la mayor información del vehículo posible. La información de interés será el estado continuo de los sensores del vehículo que forman parte del sistema de diagnóstico a bordo de estos.

Gracias a la legislación actual sobre la obligatoriedad de implantación del sistema de diagnóstico basado en OBD-II (Variantes según país), que será expuesto posteriormente en detalle, podemos acceder a la ECU y con ello a un gran número de sensores y datos del vehículo por medio de un conector estándar y unos protocolos de comunicación normalizados.

Este acceso a la información se creó para facilitar la labor de reparación o diagnóstico en caso de averías, puesto que se tendría acceso tanto a la información en tiempo real como a la información en fallas puntuales (logs, DTC). Por tanto, accediendo a este conector, tenemos el estado continuo de un numeroso número de parámetros del vehículo, los cuales apoyándonos de la conectividad inalámbrica serán accesibles en remoto.

Esto nos lleva a observar la gran utilidad que tiene el acceso a esta información para el control en flotas de vehículos, los cuales, en caso de pertenecer a una empresa, podría aportar productividad como puede ser en ahorro de combustible, eficiencia o control sobre la posición de cada uno de los vehículos pertenecientes a la flota.

Este proyecto se enmarca en la era de la conectividad vehicular, donde los fabricantes automotrices están dotando a sus productos de la capacidad de comunicarse para transmitir información valiosa. Los últimos avances tecnológicos en este campo se centran en la tecnología Vehículo-Vehículo (*V2V- Vehicle-To-Vehicle*) como en Vehículo a la Infraestructura (*V2I, Vehicle-To-Infracture*) [1]. Lo cual demuestra que el futuro de los automóviles será la compartición de datos como es el estado del vehículo, con el exterior.

Es por tanto que la pieza fundamental de este proyecto es la extracción de información relevante del vehículo, la cual en su posterior procesado podrá determinarse los fines a los que satisface los intereses y no previo a la extracción.

De esta manera un mismo sistema de extracción será útil para infinidad de posibles usos, desde uso particular con un único vehículo hasta controlar una flota de un gran número de automóviles conectados al mismo tiempo.

1.1 Motivación del proyecto

La logística es uno de los departamentos de la empresa que más recursos consume, motivado por el aumento del comercio online y la mejora de la economía, el transporte por carretera vuelve a tomar gran importancia como es el caso de las empresas de transportes, servicios de transporte privado o alquiler de vehículos.

Estos modelos de negocio serán de gran interés en este proyecto, ya que la información que hará disponible el dispositivo al administrador de la flota puede proporcionar eficiencia o ahorro en costes, por ejemplo.

Una de las motivaciones principales de este proyecto viene dado por las escasas oportunidades que se ofrecen en el mercado actual en la gestión de flotas. En el capítulo siguiente, estado del arte, se estudiarán algunos ejemplos de productos disponibles en el mercado como soluciones al control de vehículos.

Dado que está naciendo el concepto de vehículos conectados, la solución actual más común en el control de flotas se basa en la localización GPS de los vehículos. Esta solución ha resuelto hasta ahora el interés de la industria, ya que ha aportado grandes posibilidades a las empresas mediante el estudio de la localización geográfica de cada vehículo.

En La Exposición Internacional De Vehículos Comerciales IAA [2] celebrada en Hannover (Alemania) en septiembre de 2016, se presentaron sistemas innovadores en el sector de la logística de flotas de vehículos, los cuales se basaban únicamente en la localización de estos. Una de las empresas líderes en este marco en la exposición es *LOSTnFOUND* [3], esta ganó el premio en telemática "*User Award*" dado que aportaba un gran control a los clientes de sus vehículos industriales. Esto es reflejado, por ejemplo, en que el sistema era capaz de alertar en caso de que un transportista hubiera sobrepasado el tiempo máximo de conducción continua. De esta manera una de las funcionalidades sería de tacógrafo inteligente.

Entre otras funcionalidades, podemos encontrar la de controlar que un vehículo permanezca en la ruta o zona establecida de operación. Toda esta información en manos de un gestor de flotas favorece en gran medida la planificación y actuación sobre sus vehículos, y como hemos podido observar en el caso de la empresa mencionada, todas estas funcionalidades han sido posible gracias a un gran procesado de información, pero de un único tipo de información, la localización.

Es por ello, que este proyecto quiere ir más allá de las funcionalidades mencionadas, con el objetivo de crear un dispositivo capaz de extraer numerosos tipos de información del vehículo, de manera que en el futuro sea el procesado de la información quien marque las funcionalidades según el objetivo o uso del cliente. La posición geográfica también formará parte de este proyecto, puesto que da pie, como hemos visto, a un gran número de funcionalidades.

El estado del vehículo a partir de los sensores que pertenecen a la electrónica del automóvil nos dará infinidad de funcionalidades de gran utilidad al cliente en el *front-end* como puede ser, por ejemplo, basándonos en el nivel de depósito de combustible, podrá servir a una empresa de transporte privado el tiempo de inhabilitación en el servicio para el repostaje de combustible, o determinar a partir de él una comparación del consumo entre varios conductores para un mismo kilometraje.

Dado que el procesado de los datos será quien aporte la versatilidad de la utilización del sistema, el objetivo del proyecto es extraer el mayor número de datos o información sin distinciones según uso. De esta manera, un mismo dispositivo electrónico de extracción podrá dar respuesta a los requisitos de cada cliente.

Por último, cabe destacar como motivación tanto las necesidades por parte del ahorro de las empresas como hemos mencionado anteriormente, donde la empresa TomTom Telematics asegura que con el uso de la gestión de flotas basado en *Big Data* pueden reducirse costes de hasta un 30%, como las ventajas en calidad de servicio. Este último podemos observarlo en que este año, Avis Budget Group, empresa internacional de alquiler de vehículos ha incorporado 6.000 vehículos conectados, sumados a los 5.000 con los que ya contaba, los cuales, junto con su aplicación móvil, permiten acceso ininterrumpido a información relevante del vehículo como su localización. Al igual que da acceso la empresa a una monitorización continua del estado de los vehículos y a posibles modelos de negocio según el uso de los clientes [4].

1.2 Objetivos y logros del proyecto

El objetivo principal de este proyecto es la creación de un sistema compacto de extracción de datos vehiculares orientado al control de flotas cuya instalación sea sencilla.

Para la realización será necesaria un previo estudio de las soluciones actuales que se encuentran en el mercado con el fin de encontrar mejoras a estos productos y/o que sirvan de apoyo sobre los requisitos de un cliente final. Para ello se deberá estudiar el estándar OBD y sus distintos protocolos, para una correcta conectividad e intercambio de tramas entre el sistema de telemetría a crear y la electrónica propia del vehículo.

Por otro lado, cabe destacar el diseño del software y selección de hardware de los componentes que forman el sistema, así como los sistemas electrónicos encargados de la extracción, como los de envío de datos en remoto.

Desde el punto de vista del sistema, uno de los objetivos clave es el estudio de los funciones principales y secundarias que han de cubrir los subsistemas que forman el dispositivo. Además de valorar las distintas posibilidades en diseño y en tecnología con las que cuenta este.

Este último será de gran interés, y es que el estudio de la tecnología utilizada será el que marque las futuras adaptaciones o actualizaciones de software según el hardware utilizado que han de basarse en una elección correcta de la conectividad entre los distintos componentes electrónicos del proyecto, ya que, según la elección, dotará a este de una mayor versatilidad para posibles ampliaciones o adaptaciones.

Por último, como objetivo del proyecto, será la comprobación del correcto funcionamiento del prototipo haciendo uso de un vehículo turismo comercial al igual que poner de manifiesto los conocimientos adquiridos en el Grado en Ingeniería de Tecnologías de Telecomunicación.

1.3 Estructura de la memoria

Respecto a la memoria, debe abarcar por tanto un estudio del sistema electrónico de los vehículos actuales donde se reflejen los estándares tecnológicos por los que se rigen, lo cual nos dará pie a la selección de los dispositivos necesarios para el desarrollo del proyecto, este estudio será mostrado en el capítulo 2.

Una vez estudiada la tecnología que compone la red interior del vehículo en él, pasamos a realizar la selección del hardware necesario para poder implementar el equipo de telemetría, los cuales se centran en microcontroladores y placas de desarrollo, tanto el estudio tecnológico como la selección de elementos que forman el dispositivo están definidos en el capítulo 3.

Estos dispositivos electrónicos han de comunicarse entre ellos, de manera que una vez extraída la información de interés del vehículo a través de la interfaz OBD, la información ha de ser disponible en varios dispositivos, de esta manera la estructura modular nos dará una facilidad a la hora de realizar el equipo completo. Ya que dotará al proyecto de subsistemas con funcionalidades totalmente dedicadas, como por ejemplo el módulo de envío de datos a través una red inalámbrica.

Pasaremos entonces al desarrollo del software de cada uno de los subsistemas electrónicos en el capítulo 4, donde se pueden diferenciar notablemente el sistema de extracción continua de datos y el módulo de envío de estos para su posterior procesamiento.

Por último, en el capítulo 5, se realizará un apartado de comprobación del correcto funcionamiento del equipo junto con algunas propuestas futuras que serían de interés para añadirle mejoras y funcionalidades al sistema, capítulo 6.

1.4 Planificación y presupuesto

A continuación, se presenta el diagrama de Gantt del desarrollo del proyecto en la Figura 1, el cual representa el tiempo de dedicación previsto para las diferentes tareas del proyecto. Cabe destacar que algunas de las tareas no cumplieron los plazos previstos dada la complejidad de los problemas que aparecieron durante el transcurro de estas actividades.

En este diagrama se observa que la previsión de análisis y selección de los sistemas tanto implementados en los vehículos como el hardware a utilizar, forma un tercio de la planificación total del proyecto. Eso es debido a que existen un gran número de posibilidades de acceso a la información vehicular, tanto de uso vehicular interno como externo, por tanto, el estudio de estos, forma un papel clave en las decisiones futuras de la realización del dispositivo.

Por otro lado, el bloque funcional más importante se centra en la realización del software de los subsistemas que se contemplan y las respectivas pruebas continuas, estas pruebas serán fundamentales para evitar errores físicos o lógicos en el dispositivo.

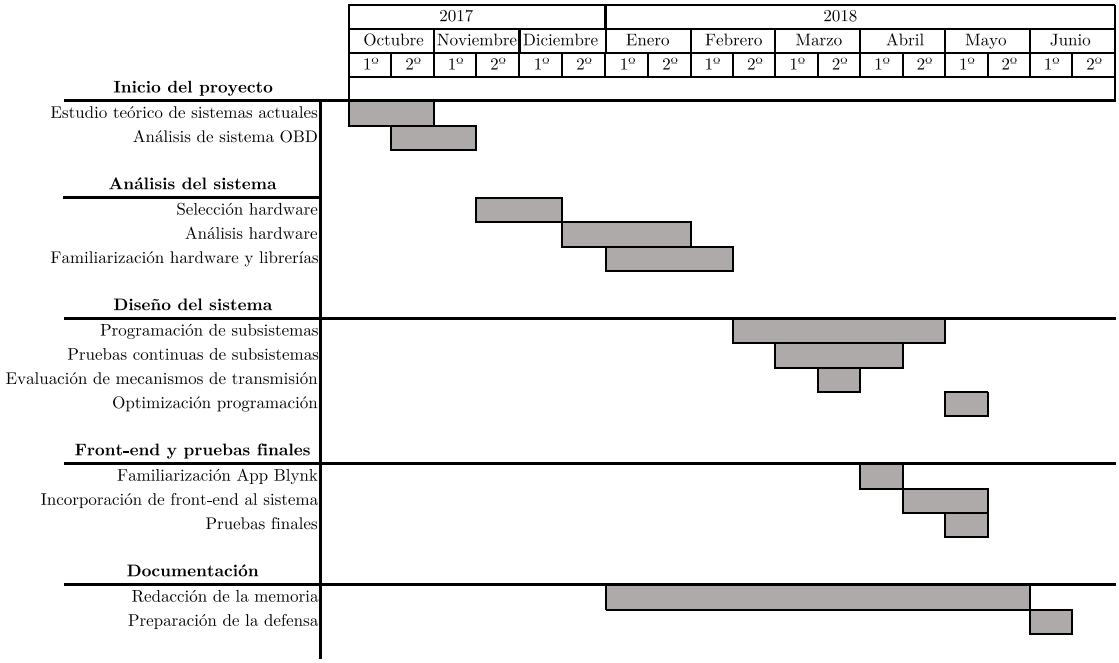


Figura 1. Diagrama de Gantt del desarrollo del proyecto

Para terminar este capítulo se exponen los costes relativos tanto al hardware, software y la aplicación móvil de representación de datos en la Tabla 1. Observamos que los mayores costes se presentan en el adaptador OBD-II I2C y en los módulos necesarios para el uso de la aplicación móvil Blynk, por tanto, en adaptaciones futuras será interesante la sustitución de estos por un desarrollo propio.

HARDWARE				
Item	**Unids. (Usado)**	**Coste/un. (€)**	**Coste (€)**	**Coste Usado (€)**
Wemos D1 Mini	3 (1)	2.12 €	6.36	2.12 €
AZDelivery Nano V3 (Arduino Nano V3)	4(1)	7.49 €	29.96 €	7.49 €
Freematics Adaptador OBD-II I2C	1(1)	34.85 €	34.85 €	34.85 €
AZDelivery pantalla OLED 128x64	1(1)	7.49 €	7.49 €	7.49 €
GPS Ublox Neo-6m	2(1)	4.02 €	8.04 €	4.02

Cableado Jumper	100(20)	0.01 €	1.00 €	0.20 €
SOFTWARE				
IDE Arduino				Licencia Gratuita
Descarga App Blynk (iOS)				Descarga Gratuita
Módulos dentro de la App Blynk (Energy)	18.000 Energy	<0.01 €	15.48 €	15.48 €
			Total	**71.65 €**

Tabla 1. Análisis de los costes de elementos Hardware y software.

Capítulo 2

Estado del arte

Una vez definida la motivación que nos lleva a realizar el proyecto, cabe enmarcarlo dentro del concepto de *Internet Of Things*. Según la empresa tecnológica Cisco, el *IoT* representa la próxima evolución de internet, que será un enorme salto en su capacidad para reunir, analizar y distribuir datos que podemos convertir en información, conocimiento y en última instancia, sabiduría [5].

Según esta empresa, *IoT* es el concepto basado en la interconexión total de diversos dispositivos, tanto entre ellos como a Internet. Esta evolución tecnológica va de la mano de avances en las telecomunicaciones como son los conceptos de *Smart Cities*, *Big Data*, *Cloud Computing*, *Inteligencia Artificial*, Telemedicina *o* despliegue de redes 5G.

Puesto el proyecto en situación en el marco del internet de las cosas, hay que mencionar la definición Máquina-A-Máquina (M2M, Machine-To-Machine) donde los dispositivos se encuentran interconectados entre ellos, y es que uno de los planteamientos futuros para este proyecto es la interconexión entre vehículos o conexión con la carretera o infraestructura.

A pesar de que ya hay numerosos proyectos en marcha de *IoT*, son varias las barreras que amenazan con retrasar el desarrollo de *IoT*, como son la transición de IPv4 a IPv6, el establecimiento de un conjunto de normas en común y el desarrollo de fuentes de energía para millones o incluso, miles de millones de sensores diminutos.

Presentado entonces el ámbito tecnológico sobre el que se rodea este proyecto, cabe presentar algunas soluciones a la gestión de flotas que se encuentran en el mercado actualmente. El objetivo de este apartado es estudiar productos comerciales que nos puedan aportar ideas sobre necesidades del cliente para el diseño del sistema de este proyecto y podamos mejorar los sistemas propuestos.

2.1 LOSTnFOUND CUMBUS

El primer dispositivo es el producto comercializado con base en la interconexión OBD de la empresa *LOSTnFOUND* [2]. Esta empresa presenta su dispositivo LnF CUMBUS como el sistema ideal para la monitorización de viajes orientado a la efectividad en costo. La Figura 2 muestra la interfaz de conexión del dispositivo.

Figura 2. *Vista del conector OBD de un dispositivo LnF CUMBUS y PinOut* [2].

- PIN2. Señal J1850 Bus+: Cable (+) de transmisión de datos en SAE J1850
- PIN4. GND: Conectado a señal de tierra.
- PIN5. GND: Conectado a señal de tierra.
- PIN6. CAN-H: Cable de diagnóstico de CAN Bus (H). ISO 15765-4
- PIN7. Señal ISO-K: Cable (-) de transmisión de datos en SAE J1850
- PIN10. J1850 Bus: Cable de diagnóstico de CAN Bus (L). ISO 15765-4
- PIN14. CAN-L: Cable de diagnóstico de CAN Bus (L). ISO 15765-4
- PIN15. ISO-L: Cable de transmisión de datos (L Cable) en ISO 9141-2
- PIN16: VIN: Entrada de alimentación de la batería (12V/24V)

El resto de pines que no se mencionan su uso, no están conectados. El sistema cuenta con conexión GSM y hace uso de GPS para obtener la localización geográfica. En la Figura 3 se muestra el dispositivo CUMBUS.

Figura 3. Dispositivo CUMBUS de la empresa LOSTnFOUND [2].

Las características técnicas proporcionadas por el fabricante LOSTnFOUND son las siguientes (ver Tabla 2):

Características	
Dimensiones (LxAxA)	49mm x 66mm x 25mm
Peso	53 g
Cualidades radio	
Frecuencia	Quad-band850/900/1800/1900MHz
Funcionalidades GSM/ GPRS	
Modo GPRS	Multi-slot Class 10
Esquema de codificado GPRS	CS1,CS2,CS3 and CS4
Antena GSM	Internal
Interfaz SIM	Tarjeta SIM 1.8V, 2.9V Soportado
Funcionalidades GPS	
Receptor	50 Canales
Sensibilidad (Tracking)	-160dBm
Tipo de antena	InternalGPSActiveAntenna,3.3V
Conector	SMA hembra
Protocolo GPS	NMEA 0183 Ver3.0
Componentes en la placa	
MCU	32-bitMicrocontroller
Memoria de datos	8MBFlash
Sensor movimiento	3-axesAccelerationSensor
Indicador LED	GPS and GSM Indicador
Eléctrica	
Fuente de alimentación	DC 8V to 30V
Consumo de potencia	145mA a 12V (Operating Mode) 10mA a 12V (Standby Mode)
Ambiente	

Temperatura de operación	-20°C a +7 0°C (sin batería de seguridad) -20 °C a +60 °C (con batería de seguridad)
Batería	
Batería seguridad recargable	130mAh, Li-Polymer

Tabla 2. Tabla de características del LnF CUMBUS [2].

Este dispositivo LOSTnFOUND CUMBUS funciona únicamente en función de la cobertura GSM (Sistema Global de Comunicación Móvil) y GPS (Satélites de Posicionamiento Global). Por lo tanto, sus funcionalidades dependen únicamente de la disponibilidad de la cobertura de red. El fabricante, sus subsidiarias, distribuidores autorizados, instaladores y distribuidores no garantizan que el funcionamiento de este dispositivo no se vea interrumpido o sin errores.

Los puntos positivos de este sistema están basados en el software, ya que la empresa cuenta con un mismo software sobre el que estudia los datos de diversos dispositivos hardware de los que dispone. Sin embargo, el dispositivo estudiado LnF CUMBUS no incrementa en gran medida sus funcionalidades respecto a otro sin conexión OBD ya que en este caso solo hace uso del estado de la batería y escasos parámetros del vehículo para determinar el consumo únicamente, sin embargo, las funcionalidades prácticas que aporta están basadas en la temperatura del vehículo como en la localización GPS. Esta última será quien aporte mayor uso al sistema de gestión de flotas proporcionado por LOSTnFOUND, de manera que gracias al software utilizado proporciona al gestor de la flota información como el tiempo de conducción y diversas alarmas, algunas de ellas son las siguientes:

- Generación de alarma en caso de bajo voltaje de la batería.
- Generación de alarmas en caso de interrupción en el suministro de energía.
- Generación de alarmas en caso de manipulación.

El coste del sistema LnF CUMBUS es de 150,00€, este puede variar en caso de aplicación de descuento si el volumen de compra supera los 10 dispositivos.

2.2 FROTCOM

Esta segunda opción que encontramos en el mercado se presenta como la solución tanto para el seguimiento de vehículos como para la gestión de flotas inteligentes. Esto se debe a que su gran potencial se encuentra en el software tanto de estudio de datos como en la presentación de estos al usuario final.

El sistema propuesto de esta empresa se basa en la instalación de un dispositivo que contiene un localizador GPS y un módulo de comunicaciones GPRS. Este dispositivo le permite controlar los movimientos de todos los vehículos: dónde están, dónde han estado, cuándo empezaron la jornada, dónde han estado parados y por cuánto tiempo, etc.

Con Frotcom es posible seguir todos los vehículos de una flota las 24 horas del día y recibir posiciones GPS y datos de sensores en tiempo real. Además, Frotcom tiene numerosos complementos que incluyen un conjunto completo de alarmas e informes, gestión de plantilla, conducta en la conducción, gestión de costes, control del combustible e identificación automática del conductor, entre otros.

Este sistema integral está enfocado al control de flotas en empresas de transporte, servicios de mensajería urgente, empresas de distribución y logística, empresas con equipos comerciales y técnicos, empresas con maquinaria para la construcción y empresas de alquiler de vehículos, entre otras.

El dispositivo se alimenta mediante el conector OBD a la batería del vehículo en cuestión. Las alimentaciones de estos son de 12V en la mayoría de coches y 24V en el caso de camiones o vehículos pesados. El acceso a la información de la flota por parte del cliente o gestor de flotas se puede realizar mediante un navegador web o su aplicación para *smartphones* (IOS y Android).

Respecto a la conexión con el vehículo, además de conectarse al conector OBD para su alimentación, lo realiza para tener acceso al CANBus. La información que obtiene es la siguiente:

- Nivel de combustible.
- RPM del motor.
- Temperatura del motor.
- Toma de fuerza.
- Tacógrafo según tiempos de encendido (utilizado en camiones europeos).

Esta información obtenida es accesible al cliente mediante su interfaz web en forma de gráficas con un histórico de hasta 7 días, la Figura 4 muestra la sección de gráficos dentro de la interfaz web.

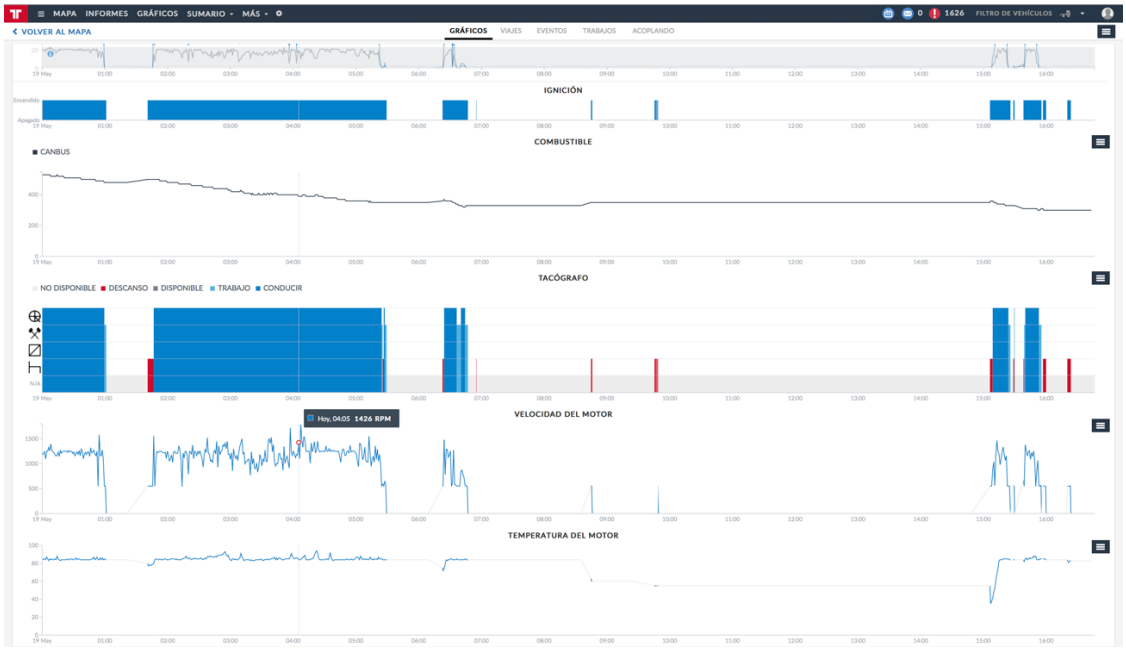


Figura 4. Gráficas de información de CANBus en interfaz web en modo simulación [6].

El sistema de seguimiento envía en torno a cada minuto la posición GPS, los datos de arranque, de los sensores periféricos y del CANBus al Centro de Datos de Frotcom mediante la comunicación GPRS.

Los datos recibidos en el Centro de Datos de Frotcom se registran y se procesan inmediatamente. Además, también se analizan para detectar posibles situaciones de alarma.

Para cada usuario se puede configurar una serie de informes (y su periodicidad). Según los parámetros configurados, los usuarios recibirán por correo electrónico los informes solicitados.

También se pueden predeterminar las situaciones de alarma, de modo que, cuando se activan, aumenta la posibilidad de una respuesta inmediata al problema. Estas forman un papel fundamental al dispositivo ya que dota de carácter particular según los intereses de cada cliente [6].

Las características funcionales del sistema propuesto por Frotcom son las siguientes:

- Solución de seguimiento GPS en tiempo real.
- Comunicaciones GPRS.
- Ignición.
- Entradas para varios sensores externos diferentes.
- Soporte en varios idiomas.
- Aplicación para smartphones para los gestores de flotas.
- Gráficos y tablas con los datos de la velocidad y de los sensores.
- Informes automáticos enviados por correo electrónico
- Alarmas automáticas por correo electrónico

En la siguiente imagen, Figura 5, se observa la interfaz gráfica de la que hace uso el gestor de flotas, en ella se observa una incidencia o alerta por exceso de velocidad.

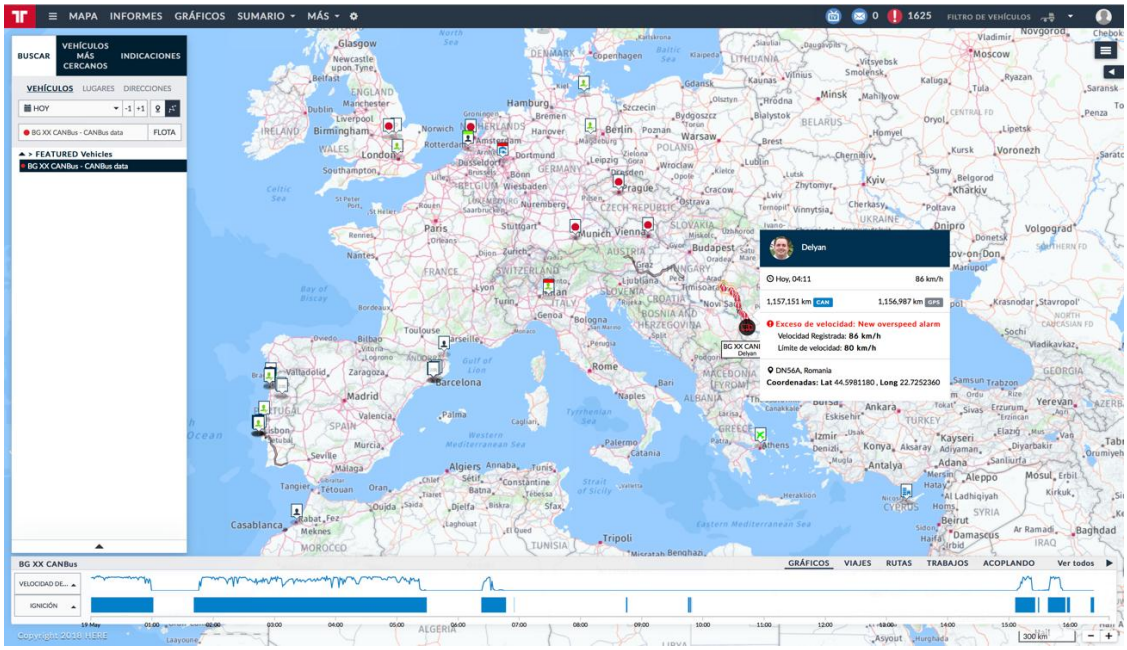


Figura 5. Interfaz gráfica del sistema Frotcom en modo simulación [6].

Es por tanto que las funcionalidades de este sistema están basadas en el estudio de los datos obtenidos, ya que, con poca información del estado de vehículo, localización de este y algunos parámetros internos es capaz de realizar una solución integral al control de flotas según diversos intereses del gestor.

2.3 WEBFLEET + LINK 410 (TomTom Telematics)

El sistema completo que aporta TomTom Telematics consta de una plataforma web llamada WEBFLEET y un conjunto de dispositivos hardware de instalación en el vehículo. Nos centraremos en el dispositivo LINK 410 dado que es el que nos muestra mayores funcionalidades orientadas a la extracción de datos del OBD.

WEBFLEET es la plataforma de software como servicio (SaaS), donde el soporte lógico y los datos que maneja se alojan en servidores de la compañía TIC, en este caso de TomTom Telematics.

La solución completa que presenta este sistema es [7]:

- Vista en tiempo real de las ubicaciones de los vehículos e información de tráfico en tiempo real, disponibles para distintos tipos de mapas, entre los que se incluyen mapas modo satélite y Google Street View.
- Planificación de rutas y envío directo a los conductores: planificación precisa de rutas por ubicación/hora de salida/llegada y tipo de vehículo.
- Mapas personalizados: Definición de zonas geográficas o grupos de vehículos y cambio de forma rápida y cómoda las vistas.
- *Dashboard*: cómoda vista general de los *KPI* para monitorizar el rendimiento en tiempo real.
- Informes: acceso instantáneo al historial de información para detectar tendencias a lo largo del tiempo.
- Mejora de los datos mediante la integración: Posibilidad de conexión de las aplicaciones empresariales con WEBFLEET o acceso a los datos de 3os desde WEBFLEET.
- Gestión de flotas durante los desplazamientos y desde cualquier dispositivo.

Las especificaciones técnicas del dispositivo LINK 410 son las siguientes (ver Tabla 3):

Dimensiones	
Unidad	121mm x 56,5mm x 21,5mm
Unidad con soporte	121mm x 56,5mm x 25,5mm
Peso y material	

Unidad	84 g, PC/ABS
Soporte	12 g, PC/ABS
Medio ambiente	
Estanqueidad	IP20
Temperatura de funcionamiento	-30 °C a +70 °C
Temperatura de almacenamiento	-40 °C a +80 °C
Eléctrica	
Fuente de alimentación	12 V / 24 V (mín. 9 V a máx. 30 V)
Consumo de corriente	
A 14V	normalmente < 50 mA
A 28V	normalmente < 30 mA
Modo de espera	normalmente < 1 mA
Durante la transmisión de datos	
A 14V	<150 mA
A 28V	<100 mA
Protección de los fusibles	
Tensión operativa con un fusible interno de 2A	9 - 30V con un fusible de un máx. de 10A
Módulos	
GSM	Integrado
GPS	Integrado, antena y receptor
Bluetooth	integrado de clase 2 para la conexión al dispositivo *TomTom PRO* y otros accesorios *TomTom*, como *Remote LINK*.

Tabla 3. Tabla características del sistema LINK105, extractor de datos OBD [7].

Este dispositivo hace uso del LINK105, mostrado en la Figura 6, como extracción de datos, ya que es el LINK 410 el que cuenta con la conectividad inalámbrica que hace posible la monitorización continua. Las interfaces con las que cuenta el Link 105 para la conexión con el sistema OBD son CAN de conforme a ISO15765, K-Line de conforme a ISO9141 y K-Line de conforme a ISO14230.

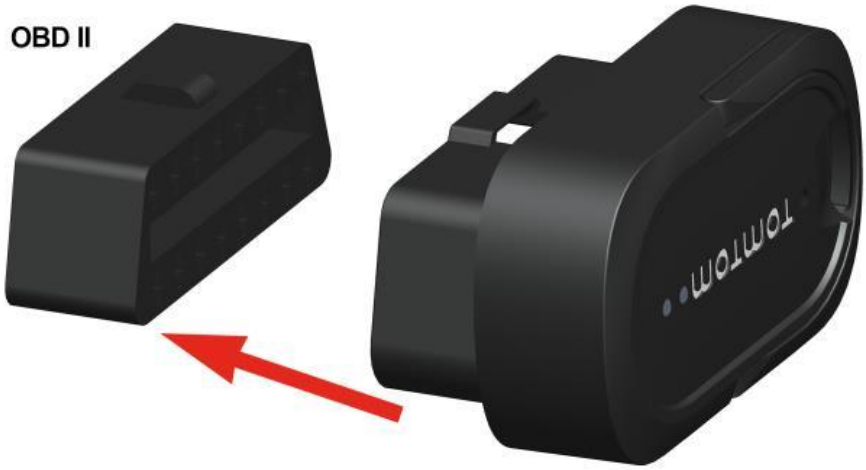


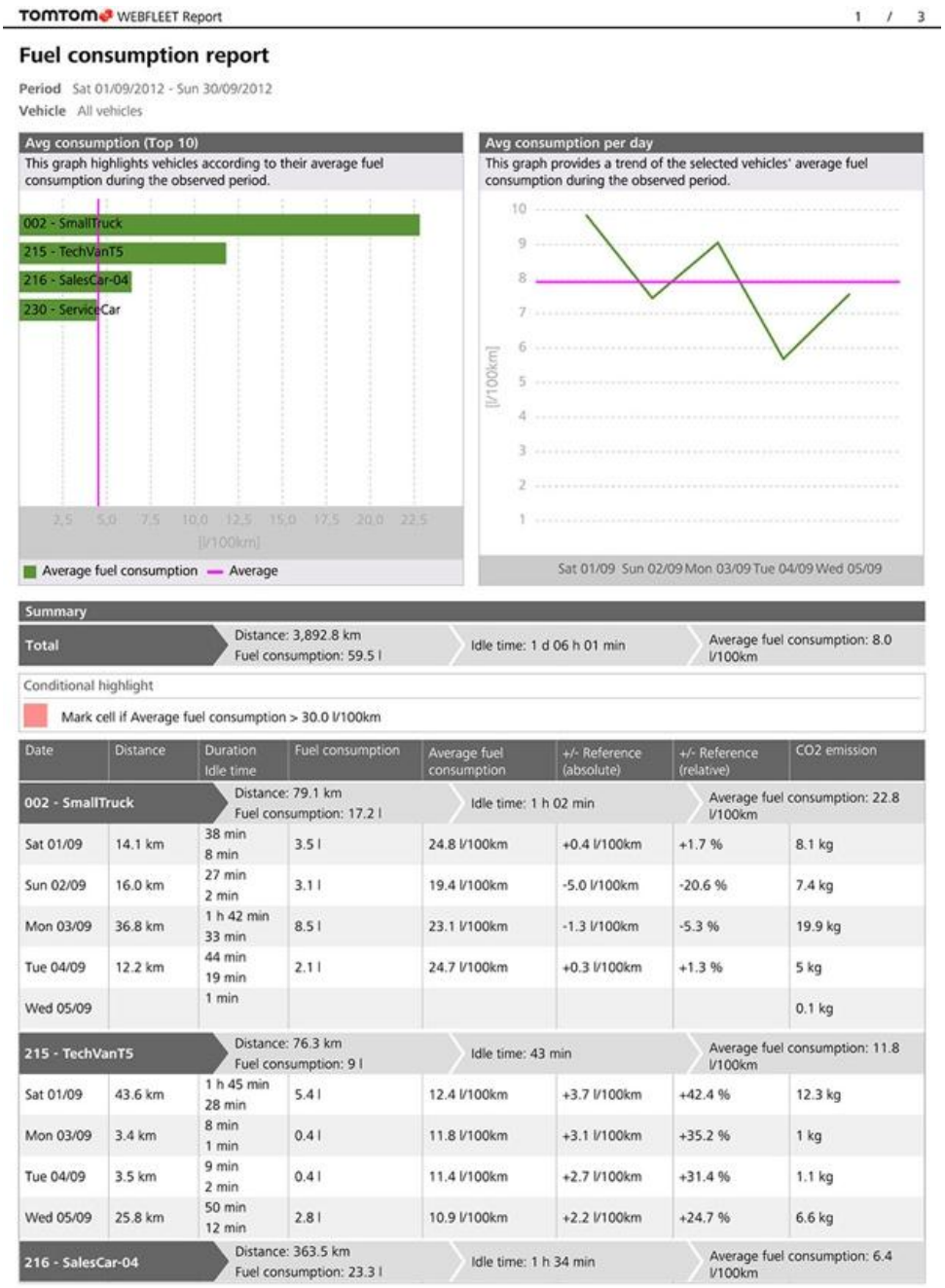
Figura 6. Conexión del Dispositivo Link 105 al puerto OBD [7].

Y las funcionalidades que aporta este sistema de extracción de información del vehículo son:

- Control del combustible en tiempo real.
- Informes de las emisiones de dióxido de carbono.
- Comunicación de códigos de avería del vehículo y del motor.
- Compatible con *Active Driver Feedback*
- Rápida instalación en el puerto OBD-II del vehículo
- Funciona en combinación con un dispositivo de seguimiento de vehículos LINK

Los datos más relevantes a los que tiene acceso el sistema son el consumo de combustible, las emisiones de CO_2, RPM del motor y los códigos de avería del motor del vehículo.

La Figura 7, muestra un informe detallado sobre el consumo de varios vehículos de una flota durante un periodo de tiempo de observación, de manera que destaca el gran poder de procesado del que posee este sistema en comparación con los dos anteriores. Este informe muestra el gran poder de procesado que posee el dispositivo y su gran detalle gracias a los datos obtenidos sobre OBD.



TOMTOM WEBFLEET Report 1 / 3

Fuel consumption report

Period Sat 01/09/2012 - Sun 30/09/2012
Vehicle All vehicles

Avg consumption (Top 10)
This graph highlights vehicles according to their average fuel consumption during the observed period.

002 - SmallTruck
215 - TechVanT5
216 - SalesCar-04
230 - ServiceCar

2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0 22.5
[l/100km]

Average fuel consumption — Average

Avg consumption per day
This graph provides a trend of the selected vehicles' average fuel consumption during the observed period.

10 9 8 7 6 5 4 3 2 1
[l/100km]

Sat 01/09 Sun 02/09 Mon 03/09 Tue 04/09 Wed 05/09

Summary

Total	Distance: 3,892.8 km Fuel consumption: 59.5 l	Idle time: 1 d 06 h 01 min	Average fuel consumption: 8.0 l/100km

Conditional highlight

Mark cell if Average fuel consumption > 30.0 l/100km

Date	Distance	Duration Idle time	Fuel consumption	Average fuel consumption	+/- Reference (absolute)	+/- Reference (relative)	CO2 emission
002 - SmallTruck	Distance: 79.1 km Fuel consumption: 17.2 l		Idle time: 1 h 02 min			Average fuel consumption: 22.8 l/100km	
Sat 01/09	14.1 km	38 min 8 min	3.5 l	24.8 l/100km	+0.4 l/100km	+1.7 %	8.1 kg
Sun 02/09	16.0 km	27 min 2 min	3.1 l	19.4 l/100km	-5.0 l/100km	-20.6 %	7.4 kg
Mon 03/09	36.8 km	1 h 42 min 33 min	8.5 l	23.1 l/100km	-1.3 l/100km	-5.3 %	19.9 kg
Tue 04/09	12.2 km	44 min 19 min	2.1 l	24.7 l/100km	+0.3 l/100km	+1.3 %	5 kg
Wed 05/09		1 min					0.1 kg
215 - TechVanT5	Distance: 76.3 km Fuel consumption: 9 l		Idle time: 43 min			Average fuel consumption: 11.8 l/100km	
Sat 01/09	43.6 km	1 h 45 min 28 min	5.4 l	12.4 l/100km	+3.7 l/100km	+42.4 %	12.3 kg
Mon 03/09	3.4 km	8 min 1 min	0.4 l	11.8 l/100km	+3.1 l/100km	+35.2 %	1 kg
Tue 04/09	3.5 km	9 min 2 min	0.4 l	11.4 l/100km	+2.7 l/100km	+31.4 %	1.1 kg
Wed 05/09	25.8 km	50 min 12 min	2.8 l	10.9 l/100km	+2.2 l/100km	+24.7 %	6.6 kg
216 - SalesCar-04	Distance: 363.5 km Fuel consumption: 23.3 l		Idle time: 1 h 34 min			Average fuel consumption: 6.4 l/100km	

Figura 7. Informe del consumo de combustible [7].

El precio del dispositivo Link 410 es de 264,99 € por unidad y el precio del complemento Link 105 es de 173,47 €, este segundo dispositivo, incluye la versión lite del software WEBFLEET.

Presentados tres dispositivos o sistemas representativos del mercado, se observa que no hay un buen compendio entre la cantidad de información de distinto carácter, la capacidad de procesamiento y muestra de este y precio. Ya que, el dispositivo con mayor adquisición de datos no presenta un procesado de estos tan completo como las otras opciones. Mientras que la versión más completa que compensa la adquisición de datos y el procesado muestra un notable aumento del precio.

A parte de la comparación entre ellos, cabe destacar la escasa versatilidad a modificaciones que tienen los sistemas, ya que están orientados a utilizar una única tecnología de transmisión de la información, orientados a la red móvil GSM/GPRS.

Es por ello, que este proyecto capta la idea de una amplia adquisición de diversos datos del vehículo, los cuáles serán procesados posteriormente e incorpora la idea de modularidad al dispositivo completo, creando subsistemas con funciones localizadas, de manera que sea muy factible la sustitución de alguno de ellos. Esto nos lleva a la posibilidad de incorporar tecnología inalámbrica de diversas características según las necesidades a las que se deba el uso del dispositivo.

Algunas de las opciones podrían ser tecnologías inalámbricas de larga alcance y baja potencia como son las redes LoRaWAN con las cuales sería interesante su uso en entornos urbanos, o de corto alcance para la representación de información vehicular en el interior del vehículo.

Capítulo 3

Requisitos técnicos de la propuesta y tecnologías empleadas

En este capítulo se presentan los requisitos técnicos del sistema que darán pie al análisis de las diversas tecnologías que nos pueden proporcionar soluciones para conseguir los objetivos funcionales de este proyecto. Para ello se presentará los requisitos que enmarquen los subsistemas. El análisis se realiza atendiendo al carácter software y hardware.

Para estudiar el sistema completo, se realiza una evaluación desde la electrónica del vehículo hacia el subsistema de envío de datos mediante tecnología inalámbrica. Será de interés dividir según el hardware usado, entre el sistema de adquisición de datos, encargado de la extracción y estructuración de la información obtenida del vehículo, y el sistema de transmisión de esta información hacia el servidor.

Para la realización del sistema de adquisición de datos debemos conocer cómo es accesible la información a estudiar, y es que los automóviles en la actualidad incorporan un gran número de sensores, debido a las exigencias gubernamentales de disminuir los niveles de emisión de contaminantes, las exigencias de seguridad por parte de los usuarios y de los gobiernos (como por ejemplo, llevan a la mejora del sistema de frenado electrónico) y el ahorro en combustible que lleva a remplazar el carburador y los sistemas mecánicos por sistemas electrónicos.

Estas son algunas de los hechos que favorecen la integración de sistemas electrónicos en la industria automotriz.

Para combatir los problemas de las emisiones de gases, en los vehículos posteriores al 1970, se creó una EPA (Agencia de Protección Medioambiental) en los Estados Unidos, esta EPA inició el desarrollo de una serie de estándares para las emisiones de gases y unos requerimientos para el mantenimiento de los vehículos con el fin de reducir la contaminación y aumentar la vida útil de los mismos.

Para ampliar estos estándares se implementaron sistemas de encendido y alimentación controlada de combustible con sensores que median las prestaciones del motor y ajustaban los sistemas para reducir la contaminación, estos también permitían una cierta ayuda a la hora de reparar los vehículos.

En el año 1985 un organismo del Estado de California llamado CARB (Junta de recursos medioambientales de California) aprobó una regulación para un sistema de diagnóstico a bordo llamado OBD (On-Board Diagnostic). Esta regulación determina que el Modulo de Control del Motor llamado ECM (Engine Control Module) tiene que monitorizar ciertos componentes del vehículo relacionados con las emisiones de gases para asegurar un correcto funcionamiento. También se tiene que iluminar una luz indicadora de fallo o avería llamada MIL (Luz indicadora de mal funcionamiento) en el cuadro de instrumentos cuando se detecte algún problema.

3.1 Estándar OBD, OBD-II y EOBD.

3.1.1 OBD

El sistema OBD aporta una serie de códigos de error DTC (Diagnostic Trouble Codes) para ayudar a los técnicos mecánicos a determinar las causas más probables de las averías en los motores y problemas en las emisiones de gases.

Como se ha mencionado en la introducción de este capítulo, los objetivos principales de este sistema son el reforzar el cumplimiento de la normativa referente a la regulación de las emisiones, notificación al conductor en caso de fallos en el vehículo y favorecer a los técnicos frente a las averías del vehículo. La autodiagnosis a partir del sistema OBD testea principalmente los mayores responsables del aumento de las emisiones de gases de escape en caso de averías, por ello se centra en comprobar el estado de los sensores principales del motor, en el sistema de medición del combustible y en la función de recirculación de gases de escape (EGR).

3.1.2 OBD-II

Este nuevo estándar, incorporado en 1990, no solo se centra en el control de emisiones como objetivo principal, si no que amplía sus posibilidades implementando un chequeo exhaustivo del motor y otras partes del vehículo, como el chasis.

Dado que cada fabricante utilizaba sus propios sistemas y códigos de error, la Sociedad de Ingenieros de Automoción (SAE) define en 1998 el conector estandarizado OBD de 16 pines y un conjunto de códigos de diagnóstico. OBD-II incluía en su estándar los siguientes elementos [8]:

- Unidad de control del motor (ECU).
- Los transductores, los cuales le envían los datos a la ECU.
- Luz indicadora de fallo (MIL).
- El conector de diagnosis DLC, interfaz entre ECU y dispositivo de diagnosis.

Este conector debe cumplir las especificaciones de la normativa ISO 15031-3:2016, el cual debe estar situado en la zona del conductor, a diferencia de la primera versión del estándar, donde estaba colocado en el compartimento del motor.

El conector físico usado en OBD-II se define en el estándar SAE J1962, mostrado en la Figura 8, no todos los pines de este son usados.

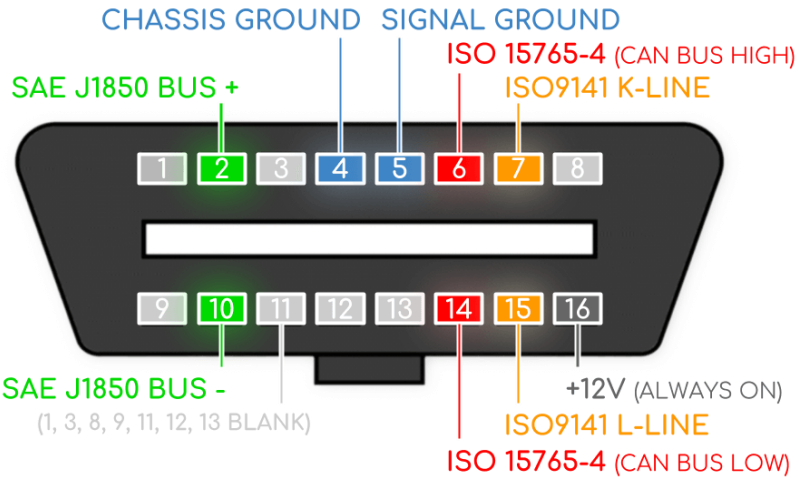


Figura 8. Definición del conector OBD-II y uso de los pines [9].

3.1.3 EOBD

En Europa se implanta esta ampliación o adaptación del sistema OBD-II adaptado a los requisitos del mercado europeo. Los controles que se obtienen son los siguientes [8]:

Motores de gasolina:

- Rendimiento del catalizador.
- Diagnóstico de sonda lambda.
- Prueba de tensión de sonda lambda.
- Sistema de aire secundario.
- Sistema de recuperación de vapores de combustible.
- Diagnóstico de fugas.
- Verificación del sistema de alimentación de combustible.
- Fallos de la combustion.
- Control sistema gestión electronica.
- Sensores del sistema electrónico que intervienen en la gestión del motor.
- Sensores relacionados con las emisiones de escape

Motores diésel:

- Fallos de la combustion.
- Regulación del comienzo de la inyección.
- Regulación de la presión de sobrealimentación.
- Recirculación de gases del escape.
- Funcionamiento de los sistemas de comunicación entre unidades. de mando como el CANBus.
- Sistema de gestión electrónica.
- Sensores del motor y gestión de las emisiones de escape.

3.2 Protocolos de comunicación en OBD.

Gracias al conector estándar se puede identificar fácilmente cuál de los protocolos es usado en cada vehículo según los pines que estén activos [10].

SAE J1850 VPW y PWM: es un estándar para la velocidad de transmisión de datos a baja y media velocidad. VPW es la versión de comunicación más lenta alcanzando 10,4 Kbit/s y la transmisión a través de un solo cable a masa. PWM, es la versión de comunicación alcanzando los 41,6 Kbit/s y transmitiendo en modo diferencial con dos cables.

ISO 9141-2: utilizado en vehículos europeos, asiáticos y americanos, intercambio de información serial asincrónico, es decir intercambio de palabras clave mediante códigos.

CAN ISO 15765: Este protocolo CAN fue desarrollado por Robert Bosch Gmbh que se extiende al campo del control industrial, la especificación de un bus CAN se muestra en la Figura 9. Existen dos versiones básicas de este protocolo que son las siguientes:

CAN 1.0 es el método empleado para permitir una velocidad de transmisión de hasta 125 Kbit/s lo que permite con esta velocidad realizar funciones de control.

CAN 2.0, es una versión del protocolo CAN compatible con el estándar CAN 1.0 cuya función es aumentar la velocidad a 500 Kbit/s al estandarizase bajo SAE J2284-500 y a 1 Mbit/s bajo la ISO 11898 (Velocidad de transmisión alta).

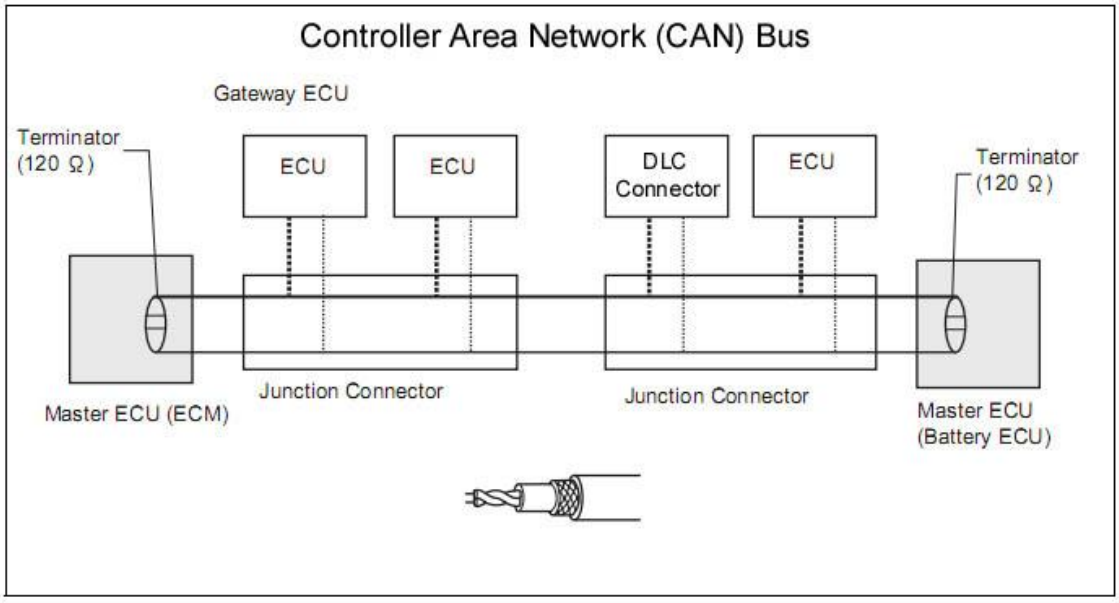


Figura 9. Defínción del cableado y sistema de una red CAN Bus [11].

ISO 14230-4 / KWP2000: Este estándar es un protocolo de comunicación muy similar al ISO 9141 basado en Bus bidireccional sobre una única línea llamada K-Line, ocasionalmente puede haber una segunda línea llamada L-Line para las señales presentes.

Para concluir la presentación de los protocolos de comunicación en OBD, la Figura 10, muestra el uso de los pines sobre un conector DLC según la comunicación empleada.

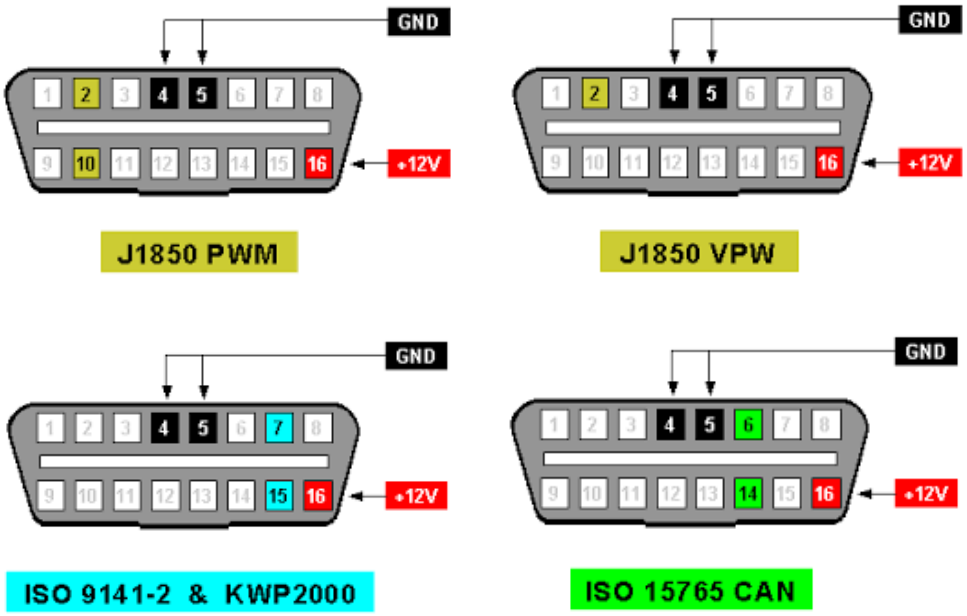


Figura 10. Conectores DLC respectivos a los protocolos de comunicación de OBD [11] .

3.3 Protocolos de comunicación en el dispositivo de diagnóstico.

Se ha presentado la electrónica correspondiente a un vehículo y determinado las características o especificaciones técnicas que forman el sistema de diagnóstico OBD. Por tanto, ahora cabe presentar los protocolos posibles a utilizar en el sistema de diagnóstico para la transmisión de información.

En el siguiente capítulo se detallarán las elecciones tomadas de hardware y software en base a la utilización de varios de los protocolos siguientes [12].

3.3.1 I2C

Es un protocolo síncrono. I2C usa solo 2 cables, uno para el reloj (SCL) y otro para el dato (SDA). Esto significa que el maestro y el esclavo envían datos por el mismo cable, el cuál es controlado por el maestro, que crea la señal de reloj. I2C no utiliza selección de esclavo, sino direccionamiento.

Dos o más señales a través del mismo cable pueden causar conflicto, y ocurrirían problemas si un dispositivo envía un 1 lógico al mismo tiempo que otro envía un 0.

Por tanto, el bus es "cableado" con dos resistencias para poner el bus a nivel alto, y los dispositivos envían niveles bajos. Si quieren enviar un nivel alto simplemente lo comunican al bus. La velocidad es de 100 kbit/s en el modo estándar, aunque también permite velocidades de 3.4 Mbit/s.

Los dispositivos conectados al bus I2C tienen una dirección única para cada uno. También pueden ser maestros o esclavos. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo, esta característica se la pueden ir pasando los dispositivos que tengan esa capacidad. Esta característica hace que al bus I2C se le denomine bus multi-maestro. Las líneas SDA y SCL son del tipo drenaje abierto, es decir, un estado similar al de colector abierto, pero asociadas a un transistor de efecto de campo (FET). Se deben polarizar en estado alto (conectando a la alimentación por medio de resistores "pull-up") lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas. El proceso de transmisión es el siguiente:

- El maestro comienza la comunicación enviando un patrón llamado "*start condition*". Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transacción.
- El maestro se dirige al dispositivo con el que quiere hablar, enviando un byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con el que se quiere comunicar, y el octavo bit (A0) de menor peso se corresponde con la operación deseada (L/E), lectura=1 (recibir del esclavo) y escritura=0 (enviar al esclavo).
- La dirección enviada es comparada por cada esclavo del bus con su propia dirección, si ambas coinciden, el esclavo se considera direccionado como esclavo-transmisor o esclavo-receptor dependiendo del bit R/W.
- Cada byte leído/escrito por el maestro debe ser obligatoriamente reconocido por un bit de ACK por el dispositivo maestro/esclavo.
- Cuando la comunicación finaliza, el maestro transmite una "stop condition" para dejar libre el bus.

En principio, el número de dispositivos que se puede conectar al bus no tiene límites, aunque hay que observar que la capacidad máxima sumada de todos los dispositivos no supere los 400 pF. El valor de los resistores de polarización no es muy crítico, y puede ir desde 1K8 (1.800 ohms) a 47K (47.000 ohms).

Un valor menor de resistencia incrementa el consumo de los integrados, pero disminuye la sensibilidad al ruido y mejora el tiempo de los flancos de subida y bajada de las señales. Los valores más comunes en uso son entre 1K8 y 10K (ohms).

Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I2C, ya que un número de 7 bits puede ir desde 0 a 127. Cuando se envían las direcciones de 7 bit, de cualquier modo, la transmisión es de 8 bits. El bit extra se utiliza para informarle al dispositivo esclavo si el dispositivo maestro va a escribir o va a leer datos desde él. Si el bit de lectura/escritura (R/W) es cero, el dispositivo maestro está escribiendo en el esclavo. Si el bit es 1 el maestro está leyendo desde el esclavo. La dirección de 7 bit se coloca en los 7 bits más significativos del byte y el bit de lectura/escritura es el bit menos significativo.

3.3.2 SPI

El Bus SPI (Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).

Incluye una línea de reloj, dato entrante, dato saliente y un pin de *chip select*, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

Muchos sistemas digitales tienen periféricos que necesitan existir, pero no ser rápidos. Las ventajas de un bus serie es que minimiza el número de conductores, pines y el tamaño del circuito integrado. Un bus de periféricos serie es la opción más flexible cuando se tiene tipos diferentes de periféricos serie. El hardware consiste en señales de reloj, *data in*, *data out* y *chip* select para cada circuito integrado que tiene que ser controlado. Casi cualquier dispositivo digital puede ser controlado con esta combinación de señales.

Los dispositivos se diferencian en un número predecible de formas. Unos leen el dato cuando el reloj sube otros cuando el reloj baja. Algunos lo leen en el flanco de subida del reloj y otros en el flanco de bajada. Escribir es casi siempre en la dirección opuesta de la dirección de movimiento del reloj.

3.3.3 UART Y USART

Las siglas de estos buses significan "Universal Asynchronous Receiver/Transmitter" y “Universal synchronous Asynchronous Receiver/Transmitter” respectivamente. En general las dos diferencias que hay entre ellos es que el USART puede trabajar en modo síncrono y asíncrono y puede transmitir a mayor velocidad mientras que el UART sólo funciona de manera síncrona. Es uno de los protocolos más usados ya que la mayoría de controladores disponen de hardware UART. Usa una línea de datos simple para transmitir y otra para recibir.

Estos buses están compuestos por 3 líneas fundamentales: TX, RX y GND. De tal manera que el pin TX de un dispositivo esté conectado RX del otro dispositivo y viceversa y que las masas de ambos estén conectadas. Si sólo conectamos un TX a un RX, la comunicación puede existir de manera unidireccional, es decir, que sólo 1 envía y sólo 1 recibe. No es el mejor bus para comunicar más de 2 elementos, es mejor abrir 1 puerto UART distinto para cada dispositivo al que quieras conectarte. Lo que hace muy útil a este puerto es que trabaja con niveles lógicos TTL (transistor-transistor logic) lo cual hace que dos micros se puedan comunicar sin necesidad de convertir los voltajes con transreceptores /drivers.

Comúnmente, 8 bits de datos son transmitidos de la siguiente forma: un bit de inicio (a nivel bajo), 8 bits de datos y un bit de parada (nivel alto). El bit de inicio a nivel bajo y el de parada a nivel alto indican que siempre hay una transmisión de alto a bajo para iniciar la transmisión. Eso es lo que describe a UART. Puede ser a 3.3V o 5V dependiendo de la alimentación del microcontrolador. Hay que fijar la velocidad de transmisión, la tasa de bits, ya que sólo disponen del bit de flanco de bajada para sincronizar. Esto es llamado comunicación asíncrona.

.

La dependencia de la coordinación es uno de los puntos negros de UART, y la solución es USART (recepción-transmisión síncrona/asíncrona universal). Puede actuar igual que UART, pero también como un protocolo síncrono. Sincronización de datos y del reloj. En cada pulso de reloj se indica al receptor que capte el bit. Los protocolos sincronizados necesitan de una conexión extra para el reloj, como es el caso de SPI o I2C. La principal diferencia de protocolos como el SPI o el I2C, y la comunicación serie que hemos visto anteriormente es que la UART o USART no necesita un pin de señal de reloj entre ambos dispositivos para realizar la comunicación. Cada dispositivo se pone de acuerdo en la velocidad a la que se va a realizar la transmisión (se fija la velocidad), y después de una condición de inicio cada dispositivo muestrea los bits en el tiempo acordado, por lo que sólo son necesarios dos pines para que el micro pueda enviar y recibir datos.

3.4 Hardware.

Como requisitos principales debemos contar con los dispositivos hardware capaces de recopilar la información del vehículo a partir de los requisitos técnicos presentados en este capítulo.

Según lo visto en el apartado 3.1, tenemos acceso al sistema OBD gracias al conector estándar ubicado en la zona del vehículo por tanto necesitamos un adaptador el cual nos haga accesible la información de la ECU en un microcontrolador externo. Para ello hago uso del dispositivo creado por la empresa Freematics, el cual como punto a favor es que aportan gran soporte respecto a la programación y consta de una extendida comunidad de desarrolladores. La elección principal entre los dispositivos con los que cuenta esta empresa, se basan en la conectividad de sus buses digitales.

Freematics OBD-II UART Adapter V2.1 (for Arduino) X

Este producto funciona como un puente de datos entre el puerto OBD de un automóvil y Arduino (o hardware similar) con una biblioteca de código abierto dedicada. Proporciona acceso a datos OBD-II de alta velocidad e integra el sensor de movimiento 9-DOF con algoritmo de fusión de sensor incorporado. El adaptador se alimenta directamente desde el puerto OBD y emite un voltaje regulado de 5 V para alimentar los dispositivos conectados.

La Figura 11 muestra el Adaptador OBD-II UART de Freematics, el cual es analizado, pero no será empleado en el dispositivo desarrollado en el proyecto.

Figura 11. Dispositivo Freematics adaptador entre interfaz OBD y UART [13].

Características:

- Acceso a todos los PID estándar OBD-II con el comando ELM327 AT.
- Interfaz de datos serial UART comatible con microcontroladores de 3.3V y 5V.
- Lectura y borrado de códigos de diagnóstico de problemas del vehículo (solo motor y tren motriz).
- Medición del voltaje de la batería del automóvil.
- Sensor de movimiento MPU-9250 9-DOF incorporado.
- Suministro de energía para el dispositivo conectado (5 V hasta 2.1 A).
- Puerto micro USB para ordenador con a acceso OBD-II y actualización de firmware.
- Biblioteca Arduino y de ejemplo.
- Modo de baja potencia a 6mA.

Los protocolos soportados son CAN (250/500 Kbps y 11/29 bits), KWP2000 (Modo *fast*/5Kbps) y parcialmente algunos vehículos con definición de ISO 9141-2.

Freematics OBD-II I2C Adapter (for Arduino) √

Se trata del producto homólogo al anterior, pero en vez de comunicación serial UART, este dispositivo trata comunicación I2C, mostrado en la Figura 12.

Figura 12. Dispositivo Freematics adaptador entre interfaz OBD y UART [13].

Características:

- Acceso a todos los PID estándar OBD-II con el comando extendido ELM327 AT.
- Interfaz de datos serie I2C para microcontroladores.
- Lectura y borrado del código de diagnóstico de problemas del vehículo (DTC).
- Medidor de voltaje incorporado para medir el voltaje de la batería del automóvil.
- Módulo MEMS MPU-6050 incorporado (acelerómetro, giroscopio, brújula y temperatura).
- Alimentando dispositivos conectados con DC 5V hasta 2A.
- Biblioteca Arduino y ejemplos disponibles.

Los protocolos soportados son CAN (250/500 Kbps y 29 bits), KWP2000 (Modo *fast*/5Kbps).

Este último dispositivo es el empleado en el proyecto dado que gracias al uso de I2C, ya que nos introduce escalabilidad a la hora de incluir otros dispositivos, sobre todo para ampliaciones futuras, como es el caso de sensores externos.

Arduino Nano

Dada la elección encontramos que la compatibilidad es máxima con las placas de desarrollo de la compañía de *open source* y open hardware Arduino. Por tanto, será una de sus placas la que optemos como microcontrolador encargado de la adquisición de datos. Este irá conectado y alimentado al adaptador de Freematics.

En primer lugar, cabe destacar que se busca un diseño modular del sistema, lo que nos lleva a que no es objetivo principal que la placa de desarrollo seleccionada para la adquisición de datos cuente con conectividad inalámbrica.

Basado en la capacidad de procesamiento y el tamaño se ha optado una de las siguientes placas de desarrollo. A continuación, se desarrolla una comparativa de las placas de desarrollo pertenecientes a Arduino que pueden ser interesantes para su uso en este proyecto. La comparativa de entre varias placas de desarrollo de arduino es (véase tabla 4):

	Arduino MEGA	Arduino UNO	Arduino NANO
Características			
Microcontrolador	ATmega2560	ATmega328P	ATmega328
Dimensiones	101.52 x 53.3 mm	68.6 x 53.4 mm	45 x 18 mm
Peso	37 g	25g	7 g
Eléctrica			
Tensión operación	5 V	5 V	5 V
Voltaje entrada (recomendado)	7-12 V	7-12 V	7-12 V
Voltaje entrada (límite)	6-20 V	6-20 V	6-20 V
Corriente CC por pin E/S	20 mA	20 mA	40 mA
Corriente CC por pin 3.3V	50 mA	50 mA	
Pines			
Pines digitales E/S	54 (15 de ellos PWM)	14 (6 de ellos PWM)	22 (6 de ellos PWM)
Pines analógicos	16	6	8
LED_BUILTIN	13	13	13
Memoria			
Memoria Flash	256 KB (8 KB para bootloader)	32 KB (0.5 KB para bootloader)	32 KB (2 KB para bootloader)
SRAM	8 KB	2 KB	2 KB
EEPROM	4 KB	1KB	1 KB

Tabla 4. Tabla comparativa Arduino [14].

Las características presentadas en la comparativa son:

Flash: Memoria de programa. Usualmente desde 1 Kb a 4 Mb. Donde se guarda el sketch ya compilado. Sería el equivalente al disco duro de un ordenador. En la memoria flash también se almacena del *bootloader*. Se puede ejecutar un programa desde la memoria flash, pero no es posible modificar los datos, sino que es necesario copiar los datos en la SRAM para modificarlos. La memoria flash usa la misma tecnología que las tarjetas SD, los pendrives o algunos tipos de SSD, esta memoria tiene una vida útil de unos 100.000 ciclos de escritura.

SRAM (static random access memory): Variables locales, datos parciales. Usualmente se trata como banco de registros (PIC) y memoria volátil. Es la zona de memoria donde el sketch crea y manipula las variables cuando se ejecuta. Es un recurso limitado y necesita supervisión para evitar agotarlo.

EEPROM: Memoria no volátil para mantener datos después de un *reset*. Se puede grabar desde el programa del microcontrolador, usualmente, constantes de programa. Las EEPROMs tienen un número limitado de lecturas/escrituras, tener en cuenta a la hora de usarla.
Esta memoria solo puede leerse byte a byte, de menor velocidad frente a la SRAM. La vida útil de la EEPROM es de unos 100.000 ciclos de escritura

Como podemos observar, las condiciones de operación de estas placas son prácticamente similares, nos lleva a que, la placa de la que haremos uso será el modelo basado en Arduino NANO, ya que cumple con las especificaciones de procesamiento y memoria, y las dimensiones son notablemente menores. La placa basada usada basada en este se muestra en la Figura 13.

Figura 13. AzDelivery 3 basado en Arduino NANO v3.

A este subsistema se le ha incluido una pantalla OLED de 0.96" con dimensiones 25 x 14 mm, resolución de 128x64 pixeles, controlador SDD1306 y posibilidad de conexión I2C o SPI, la pantalla usada se presenta en la Figura 14.

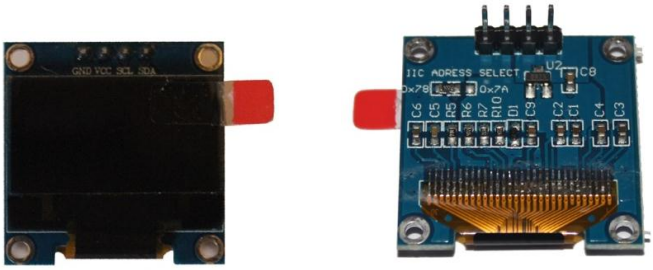


Figura 14. Pantalla OLED de 128x64 píxeles.

Queda determinado el subsistema de adquisición de datos, ahora cabe obtener solución a los requisitos propuestos sobre la transmisión de la información. Como veremos en el siguiente capítulo de diseño del dispositivo, en base a la programación se favorece la inclusión de nuevas tecnologías de transmisión inalámbrica. Utilizamos entonces una placa de desarrollo que incluya alguna tecnología inalámbrica.

En este subsistema no se necesita una alta capacidad de procesado ni de almacenamiento, para futuras adaptaciones sería interesante incluir un *shield micro-sd* que permita un almacenaje de la información obtenida. Según estas especificaciones, se ha elegido la siguiente placa de desarrollo.

WEMOS D1 MINI

Es una tarjeta de desarrollo similar a Arduino, especialmente orientada al Internet de las cosas (*IoT*). Está basado en el *SoC* (System on Chip) ESP8266**,** un chip altamente integrado, diseñado para las necesidades de un mundo conectado. Incorpora un potente procesador con arquitectura de 32 bits y conectividad WiFi.

Para el desarrollo de aplicaciones se puede elegir entre los lenguajes Arduino y Lua. Es posible trabajar desde el IDE de Arduino, al igual que las placas de desarrollo de este, existe una amplia información sobre librerías y ejemplos de proyectos en internet gracias a la amplia comunidad. En la imagen siguiente, Figura 15, se visualizan las posibles funciones de cada pin del Wemos D1 Mini.

Wemos está diseñado especialmente para trabajar en *protoboard* o soldado sobre una placa de circuito impresa. Posee un regulador de voltaje en placa que le permite alimentarse directamente del puerto USB. Los pines de entradas/salidas trabajan a 3.3V. El chip CH340G se encarga de la comunicación USB-Serial.

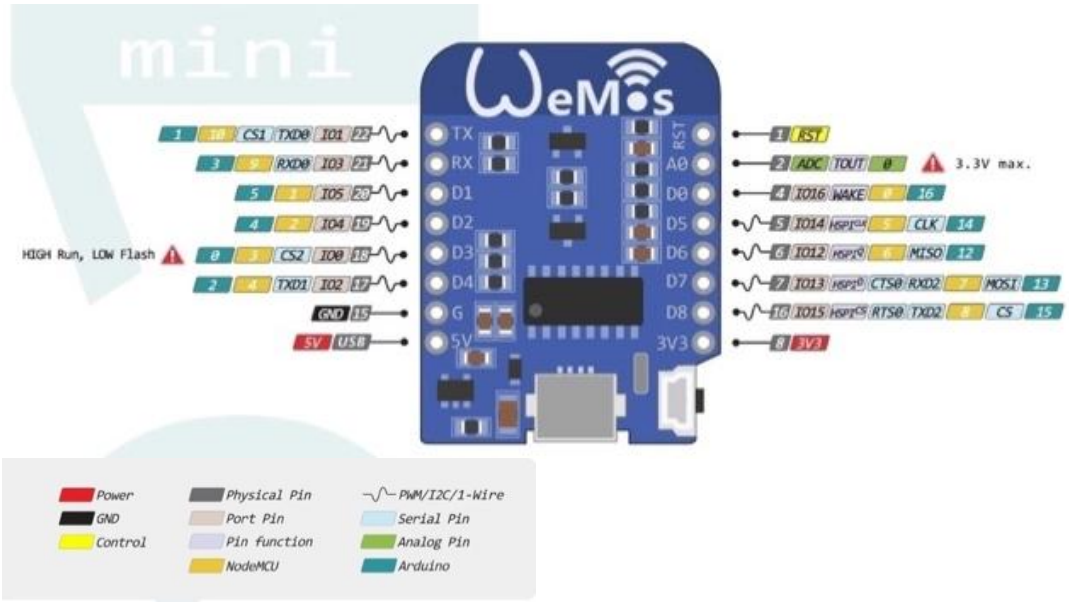


Figura 15. *PinOut* de la placa Wemos D1 Mini según el fabricante [15].

Las especificaciones técnicas son las siguientes (véase Tabla 5):

Características	
Dimensiones	32.2 x 25.6mm
Peso	10 g
Reloj	80/160 MHz
Módulos	
SoC	ESP8266 (Módulo ESP-12E)
CPU	Tensilica Xtensa LX3 (32 bit)
Chip USB	CH340G
Eléctrica	
Tensión de alimentación	5 V
Tensión pines E/S	3.3 V
Corriente promedio	70 mA
Corriente de pico	400 mA
Corriente en standby	40 uA
Potencia en standby	< 1.0 mW
Memoria	
Flash	4 MB
Data RAM	96 KB
Instruction RAM	32 KB

Pines	
Pines digitales E/S	11 (los 11 pueden ser PWM a 3.3V)
Pines analógicos	1 (0-1 V)
Conectividad	
Estándar Wifi	802.11 b/g/n
Stack de protocolo TCP/IP	integrado
Procesador MAC/Baseband	integrado
Módulos WEB, TKIP, AES Y WAPI	integrados
Potencia de salida (en modo 802.11b)	+19.5 dBm
Antena	STBC, 1x1 MIMO, 2x1 MIMO
Interfaces	I2C, UART, SPI y SDIO 2.0

Tabla 5. Tabla característica Wemos D1 Mini [15].

En este subsistema, de transmisión de datos a través de la red, hay que incluir un módulo GPS. De esta manera quedaría resuelto el dispositivo al completo, este último elemento se muestra en la Figura 16.

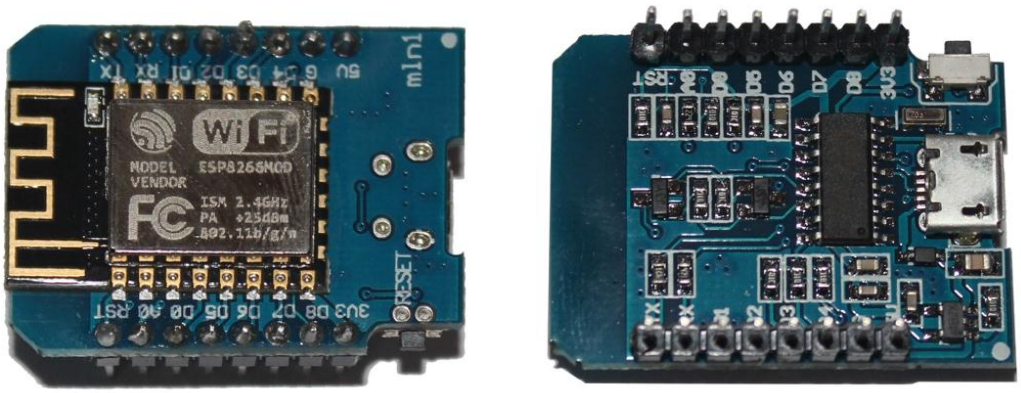


Figura 16. D1 Mini - mini NodeMcu basado en ESP8266 Wemos D1 Mini.

MÓDULO GPS SERIAL NEO-6M

Módulo GPS para Arduino y microcontroladores, mostrado en Figura 17, basado en el receptor de la marca Ublox modelo NEO 6M, el módulo incluye su antena cerámica para colocarse directamente sobre el PCB, por lo que ya viene listo para operar sin requerir más accesorios [17].

El GPS puede funcionar con un voltaje de alimentación en el rango de 3.0 a 5.0 V, mientras que las señales en los pines de entrada y salida son de 3.3 V. La interfaz de comunicación que implementa es UART, como vimos en el capítulo anterior, asíncrona.

Figura 17. Ublox NEO-6M módulo GPS NEO MV2 GY-GPS6MV2.

3.5 Software.

Arduino proporciona un entorno de programación sencillo y potente para programar, pero además incluye las herramientas necesarias para compilar el programa y "quemar" el programa ya compilado en la memoria flash del microcontrolador. Además, el IDE nos ofrece un sistema de gestión de librerías y placas muy práctico. Las herramientas necesarias para programar los microcontroladores AVR de Atmel son avr-binutils, avr-gcc y avr-libc y ya están incluidas en el IDE de Arduino, pero cuando compilamos y cargamos un sketch estamos usando estas herramientas.

Un programa de Arduino se denomina sketch o proyecto y tiene la extensión ".ino", la estructura básica de un sketch se compone de al menos dos partes. Estas dos partes son *setup()* y *loop()*, obligatorios y que encierran bloques que contienen declaraciones, estamentos o instrucciones. *setup()* es la parte encargada de recoger la configuración y *loop()* es la que contiene el programa que se ejecuta cíclicamente.

Aunque se hable de que hay un lenguaje propio de programación de Arduino, no es cierto, la programación se hace en C++ pero Arduino ofrece unas librerías, también llamado *core*, que facilitan la programación de los pines de entrada y salida y de los puertos de comunicación, así como otras librerías para operaciones específicas. El propio IDE ya incluye estas librerías de forma automática y no es necesario declararlas expresamente. La estructura del programa es una de las diferencias frente a C++ standard, ya que, no usa la función *main()*, sino que usa las funciones *setup()* y *loop()*.

Una variable puede ser declarada al inicio del programa antes de la parte de configuración *setup()*, a nivel local dentro de las funciones, y, a veces, dentro de un bloque. Una variable global es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de *setup()*.

Una variable global, está en un espacio en memoria permanente, en la zona de *static data* y su excesivo uso puede suponer un uso ineficiente de la memoria. Una variable local es aquella que se define dentro de una función o como parte de un bucle. Sólo es visible y sólo puede utilizarse dentro de la función en la que se declaró.

El modificador de variable *static*, es utilizado para crear variables que solo son visibles dentro de una función, sin embargo, al contrario que las variables locales que se crean y destruyen cada vez que se llama a la función, las variables estáticas mantienen sus valores entre las llamadas a las funciones.

Tipos de variables (véase Tabla 6):

- Boolean: 1 Byte. True (1) o False(0).
- char: 1 Byte. Caracter ASCII o valor con signo entre -128 y 127.
- unsigned char, byte, unit8_t: Caracter ASCII o valor sin signo entre 0 y 255.
- int, short: 2 Bytes. Valor con signo entre -32.768 y 32.767.
- unsigned int, word, unit16_t: 2 Bytes. Valor sin signo entre 0 y 65.535.
- long: 4 Bytes. Valor con signo entre -2.147.483.648 y 2.147.483.647.
- unsigned long, unit32_t: 4 Bytes. Valor sin signo entre 0 y 4.294.967.295.
- float, double: 4 Bytes. Valor con signo entre -3.4028235E+38 y 3.4028235E+38

Variables		
Tipo	**Memoria que ocupa**	**Rango de valores**
Boolean	1 Byte	True (1) o False(0).
char	1 Byte	-128 y 127
unsigned char, byte, unit8_t	1 Byte	0 a 255
int, short	2 Bytes	-32.768 a 32.767.
unsigned int, word, unit16_t	2 Bytes	0 a 65.535
long	4 Bytes	-2.147.483.648 a 2.147.483.647

unsigned long, unit32_t	4 Bytes	0 y 4.294.967.295
Float, double	4 Bytes	-3.4028235E+38 a 3.4028235E+38

Tabla 6. Tipos de variables, memoria y valores

Un *array* es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un *array* tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado.

Un *string (char array)* es un *array* de *chars*. Cuando se trabaja con grandes cantidades de texto, es conveniente usar un *array* de *strings*. Un *string* (objeto) es una clase que permite usar y manipular cadenas de texto de una forma más sencilla. Es posible, por ejemplo, concatenar, añadir o buscar usando los métodos/funciones que ofrece esta clase. Los *strings* tienen un uso intensivo de memoria. Al ser una clase, tienes unas funciones asociadas (métodos), operadores y unas propiedades, es una abstracción del dato.

Los operadores que pueden ser usados y las estructuras de control son las siguientes:

Aritméticos
- De asignación, +, -, *, / o módulo

Compuestos:
- ++, –:, += , -= , *= , /= :

Comparación:
- ==, !=, <, >, <=, >=:

Booleanos:
- &&

Estructuras de decisión:
- if:
- else:
- switch case:

Estructuras de repetición:
- for:
- while:
- do .. while:

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas cuando se la llama.

Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Segmentar el código en funciones permite crear piezas de código que hacen una determinada tarea y volver al área del código desde la que han sido llamadas.

Junto con los compiladores de C y C++, se incluyen ciertos archivos llamadas librerías. Las librerías contienen el código objeto de muchos programas que permiten hacer cosas comunes, como leer el teclado, escribir en la pantalla, manejar números, realizar funciones matemáticas, etc.

La declaración de librerías, tanto en C como en C++, se debe hacer al principio de todo nuestro código, antes de la declaración de cualquier función o línea de código, debemos indicarle al compilador que librerías usar, para el saber qué términos están correctos en la escritura de nuestro código y cuáles no. La sintaxis es la siguiente: *#include* <nombre de la librería> o alternativamente *#include* "nombre de la librería".

Capítulo 4

Diseño e implementación

Establecidos los elementos hardware que vamos a utilizar en el dispositivo, pasamos en este capítulo al análisis del diseño e implementación de este. Para ello será fundamental la distinción de cada subsitema según sus funcionalidades. Antes de comenzar, cabe recordar que el potencial de este proyecto recae en la versatilidad que aporta la modularización de los sistemas que forman el dispositivo, permitiendo la inclusión de otras tecnologías inalámbricas. En este proyecto se hará uso de la tecnología WiFI, cuya conectividad es posible gracias a la Wemos D1 Mini como hemos visto en el capítulo anterior.

Para comenzar se presenta la arquitectura completa en computación. Esta arquitectura se visualiza según el subsistema o bloque funcional, observamos en esta arquitectura que cualquier sistema con conectividad UART programado con la secuencia de estados del actual subsistema de recepción de datos y transmisión, podría sustituir a este.

A pesar de que no sea objeto de ampliación la modificación del subsistema de adquisición de datos sobre OBD, en caso de que se requiera disminuir la velocidad de refresco de los datos obtenidos, sería necesaria una modificación de este, como podría ser incorporando prioridades sobre PIDs.

El mapa conceptual de la arquitectura es el representado por la Figura 18.

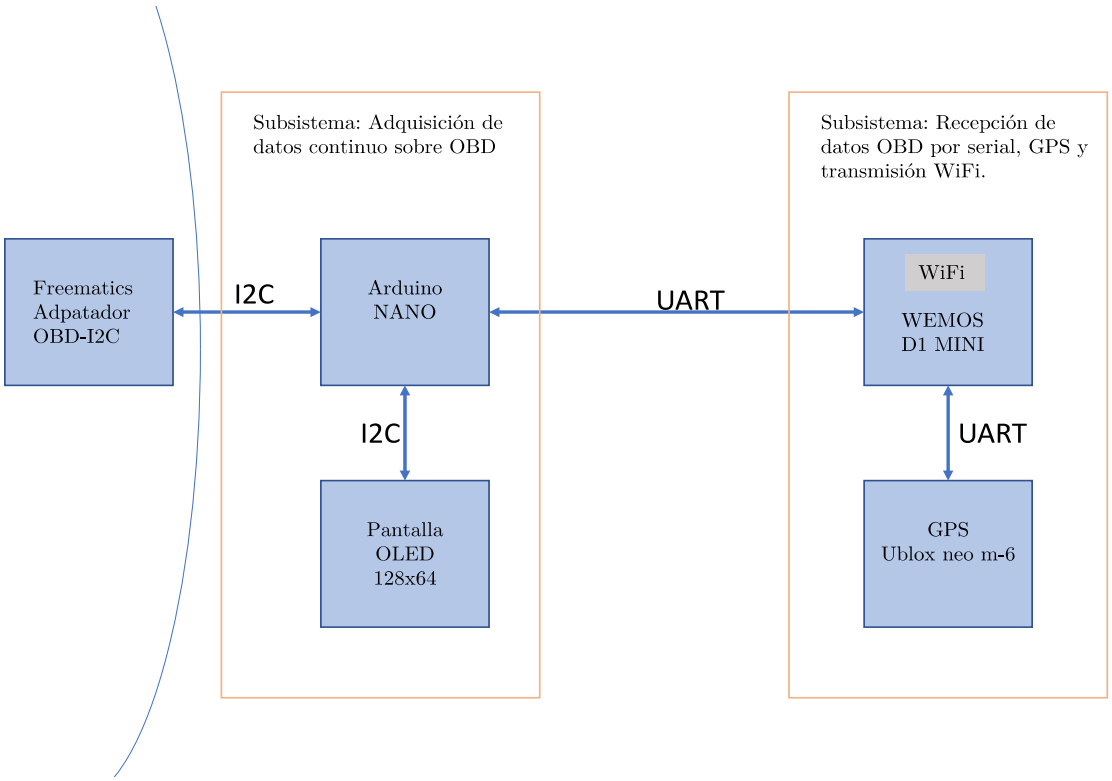


Figura 18. Mapa conceptual de la arquitectura

4.1 Funcionamiento.

A continuación, se expone los pasos que sigue el dispositivo completo observado en cada subsistema y las condiciones en que se encuentra.

Hay que estudiar el sistema en los estados paralelos de cada bloque funcional. Dado el carácter secuencial, es vital la alimentación simultánea de ambos bloques, esto se consigue alimentando desde el adaptador conectado al puerto OBD.

Alimentado el sistema, inicialmente la Wemos D1 Mini abre tanto el puerto serie por defecto, el cual servirá de comunicación con el arduino NANO, como el puerto serial por software con el cual recibimos los datos del GPS.

Este, espera a recibir un símbolo de estado del arduino indicándole que está listo para enviarle el vector de booleanos que indica los sensores disponibles en el vehículo, por ello, una vez abierto el puerto serie que se comunicará con el arduino NANO, la Wemos usa la función *flush*() sobre el serial para borrar cualquier dato en la entrada que pudiera haber. En esta inicialización, el bloque de transmisión se conecta mediante WiFi gracias a la librería proporcionada por la app que usaremos para la visualización de datos, Blynk.

En este punto la Wemos está esperando a recibir por la conexión serial, el símbolo correspondiente a *ReadyRunning*, el cual indica que el arduino NANO está listo para enviarle el primer vector. Entonces en este estado la Wemos ha abierto los dos puertos seriales a utilizar, ha vaciado la entrada del serial conectado al arduino nano, inicializado las variables necesarias y se encuentra esperando la señal de indicación de que el arduino nano está listo para enviarle el primer vector.

Ahora observamos el arduino, este comienza abriendo el puerto serie de los pines por defecto para comunicarse con la Wemos y borra cualquier dato que haya.

Haciendo uso de la librería proporcionada por el fabricante OBD, creado el objeto obd a partir de la clase COBDI2C que pertenece a la librería mencionada. Se inicializa este mediante la función *begin()*, el cual establece la conexión. Una vez establecida, se presentan por la pantalla OLED algunos mensajes de inicio del sistema, y se pasa a realizar un sondeo de los sensores o información útil que se puede obtener.

Llegado a este punto cabe destacar las variables y vectores fundamentales de los que cuenta cada microcontrolador. Ambos han declarado los siguientes:

- static byte pids_vector[ ]= { PID_ABSOLUTE_ENGINE_LOAD, … , PID_THROTTLE, … , PID_AMBIENT_TEMP};
- int pids_1_elementos = sizeof(pids_vector) / sizeof(pids_vector[0]);
- bool pids_1_running[sizeof(pids_vector) / sizeof(pids_vector[0])];
- int pids_1_valores[sizeof(pids_vector) / sizeof(pids_vector[0])];

El *pids_vector* contiene los PID en formtato byte para cada uno de los parámetros que serán objeto de solicitud a la ECU. El valor en hexadecimal de cada PID se caracteriza en la librería OBD.

pids_1_elementos contiene el número de elementos del *pids_vector.*

pids_1_running es un vector de tipo booleano, el cual indicará según su posición, si el PID de una posición dada según el *pids_vector* es un dato útil.

pids_1_valores será el vector de valores enteros que iremos sobrescribiendo con los datos que nos proporcione la ECU.

Retomando la situación del arduino nano, este se encuentra en la secuencia de rellenar el vector de booleanos que indica los PID útiles. Para ello solicita el valor de cada uno de los PIDs del *pids_vector* y determina si un PID es útil si el tiempo de obtención del dato no supera un TIMEOUT establecido (TIMEOUTcorto=100 ms).

Esto se realiza gracias a la función *SondeoPID()* que devuelve un booleano si el tiempo de obtención para un PID ha sido inferior al TIMEOUT indicado [19]. Realizado este sondeo con todos los PIDs del vector pids_vector[ ], tenemos completo el vector pids_1_running[]. La situación de los vectores mencionados es la siguiente:

Arduino NANO

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- **pids_1_running[] = Completo con true o false.**
- pids_1_valores[] = Completo con valor 0.

WEMOS D1 MINI

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- **pids_1_running[] = - .**
- pids_1_valores[] = Completo con valor 0.

Entonces el arduino nano le envía el símbolo *ReadyRunning* indicándole a la WEMOS que ya ha realizado el sondeo y está listo para enviarle el vector pids_1_running[ ]. La WEMOS le devuelve en *ReadyRunning* y entonces se transifiere dicho vector por la comunicación serial por defecto ya que ambos conocen la longitud del vector y cada dato es tabulado para una fácil recepción.

Una vez concluida la transmisión, la Wemos envía *ReadyRunningACK* y ambos ya han ejecutado completamente su función *setup()*. La situación de los vectores es:

Arduino NANO

- pids_vector[ ]= Completo con PIDs en hexadecimal.
- pids_1_running[]=Completo con true o false;
- pids_1_valores[]=Completo con valor 0.

WEMOS D1 MINI

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- pids_1_running[] = Completo con true o false;
- pids_1_valores[] = Completo con valor 0.

La función loop de cada uno de los dispositivos se repite cíclicamente, de esta manera el ciclo de cada uno seguirá la siguiente estructura.

El arduino comienza a rellenar el vector de valores pids_1_valores[] propio de esa iteración del ciclo, únicamente solicitando el valor a de los PIDs que marca el vector pids_1_running[] como útiles.

Mientras esto ocurre que puede llevar hasta unos segundos, según la cantidad de PIDs útiles, la WEMOS ejecuta la función de conexión con la app Blynk y espera a recibir el mensaje de estado del arduino. Terminado el llenado del vector de valores pids_1_valores[] por parte del arduino los vectores serían:

Arduino NANO

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- pids_1_running[] = Completo con true o false;
- **pids_1_valores[] = Completo con valores enteros de iteración actual.**

WEMOS D1 MINI

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- pids_1_running[] = Completo con true o false;
- **pids_1_valores[] = Completo con valor 0.**

Por tanto la WEMOS ha de completar su vector de valores, con los datos que ha obtenido el arduino de la ECU. Para ello, recibe del arduino el símbolo *ReadyVector* y lo devuelve para que el arduino le comience a transmitir este vector. El envío se realiza semejante al envío del vector pids_1_running[], por la conexión serial. Finalizada la recepción, envía *ReadyVectorACK* y entonces el ARDUINO termina su ciclo mientras la WEMOS modifica estos parámetros en los objetos relacionados con la app móvil Blynk para que estos sean representados al comienzo de la iteración siguiente y la WEMOS comienza un ciclo nuevo.

Arduino NANO

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- pids_1_running[] = Completo con true o false;
- **pids_1_valores[] = Completo con valores enteros de iteración actual.**

WEMOS D1 MINI

- pids_vector[ ] = Completo con PIDs en hexadecimal.
- pids_1_running[] = Completo con true o false;
- **pids_1_valores[] = Completo con valores enteros de iteración actual.**

De esta forma se consigue que mientras se ejecuta el proceso de obtención de datos por parte del arduino, el cual es de los más costosos en tiempo, la WEMOS se encuentra modificando y enviando los datos de la iteración anterior a la app de representación. En la función de modificación de estos parámetros a partir de los valores del vector pids_1_valores[] también se incluye la modificación de los parámetros de localización a partir del GPS.

Además, en el estado de obtención de datos en el arduino se encuentra la representación por la pantalla OLED de estos. Para que estos datos consigan un menor tiempo de respuesta desde la obtención hasta la visualización. Por cada dato que se solicite al OBD se solicitan los cuatro datos que serán representados en la pantalla. Esto hace que el tiempo de escaneo y llenado total del vector de valores aumente considerablemente, pero a su vez los datos representados en pantalla serán inmediatos.

Este funcionamiento se refleja en figuras siguientes. En la Figura 19, se muestra el diagrama de secuencia correspondiente al intercambio de información y comunicación de los elementos más importantes que forman el dispositivo [18]. El bloque superior representa a la función *setup()* por parte tanto del Arduino Nano como de la Wemos D1 Mini, mientras que a partir de él se representa una iteración de la función *loop()*.

En la Figura 20 encontramos el diagrama de flujo del código implementado tanto en el Arduino Nano como en la Wemos D1 mini en la fase de iniciación y ejecución de la función *setup()*. Mientras que en la Figura 21 se muestra la continuación del diagrama de flujo en la ejecución de la función *loop()*. Visto que el sistema contiene un alto carácter secuencial debido al intercambio de mensajes de estado, esto se ha reflejado en el dimensionamiento de los estados en el diagrama, de manera que la altura en el diagrama corresponde al estado de cada elemento en el tiempo. Diagrama completo en Figura 22.

Es observable en este que la conexión con la app Blynk se realiza antes de obtener los valores de la adquisición de los datos sobre OBD. Puesto que nos encontramos en la función *loop()*, esto nos lleva a que en la primera iteración los datos mostrados en la app serán los valores con los que se ha iniciado el vector pids_1_valores[], pero en las posteriores iteraciones la transmisión de los datos a la app Blynk será correcta, enviando los datos de la iteración anterior.

El hecho de realizar esta implementación es la optimización en tiempo, apoyándose en tareas paralelas, de manera que mientras el arduino Nano está adquiriendo datos nuevos, la Wemos D1 Mini está actualizando los datos visualizados en la app con los valores de la iteración anterior.

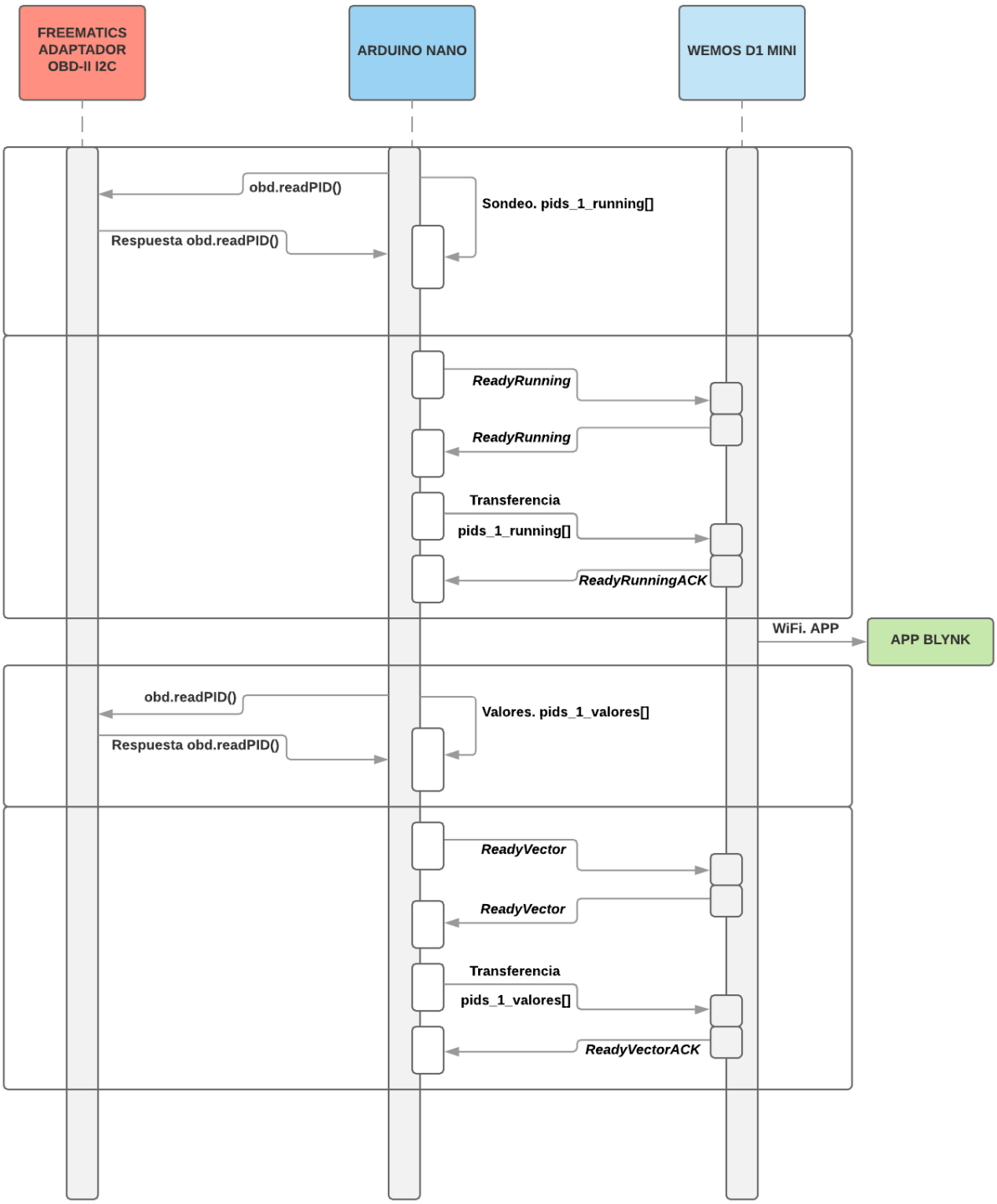


Figura 19. Diagrama de secuencia del sistema.

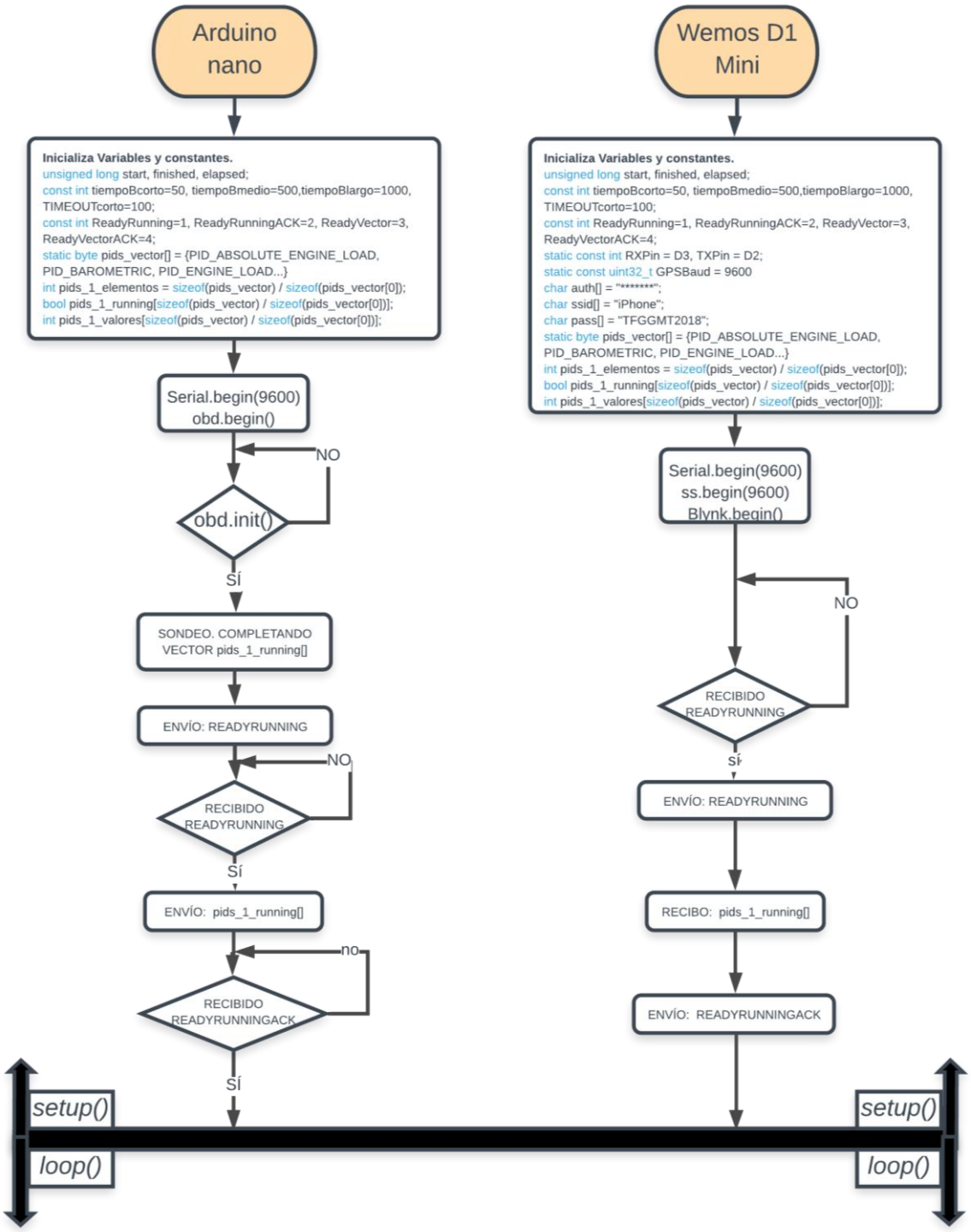


Figura 20. Diagrama de flujo del código en el Arduino Nano (izquierda) y en la Wemos D1 mini (derecha) en la inicialización y ejecución de la función *setup()*.

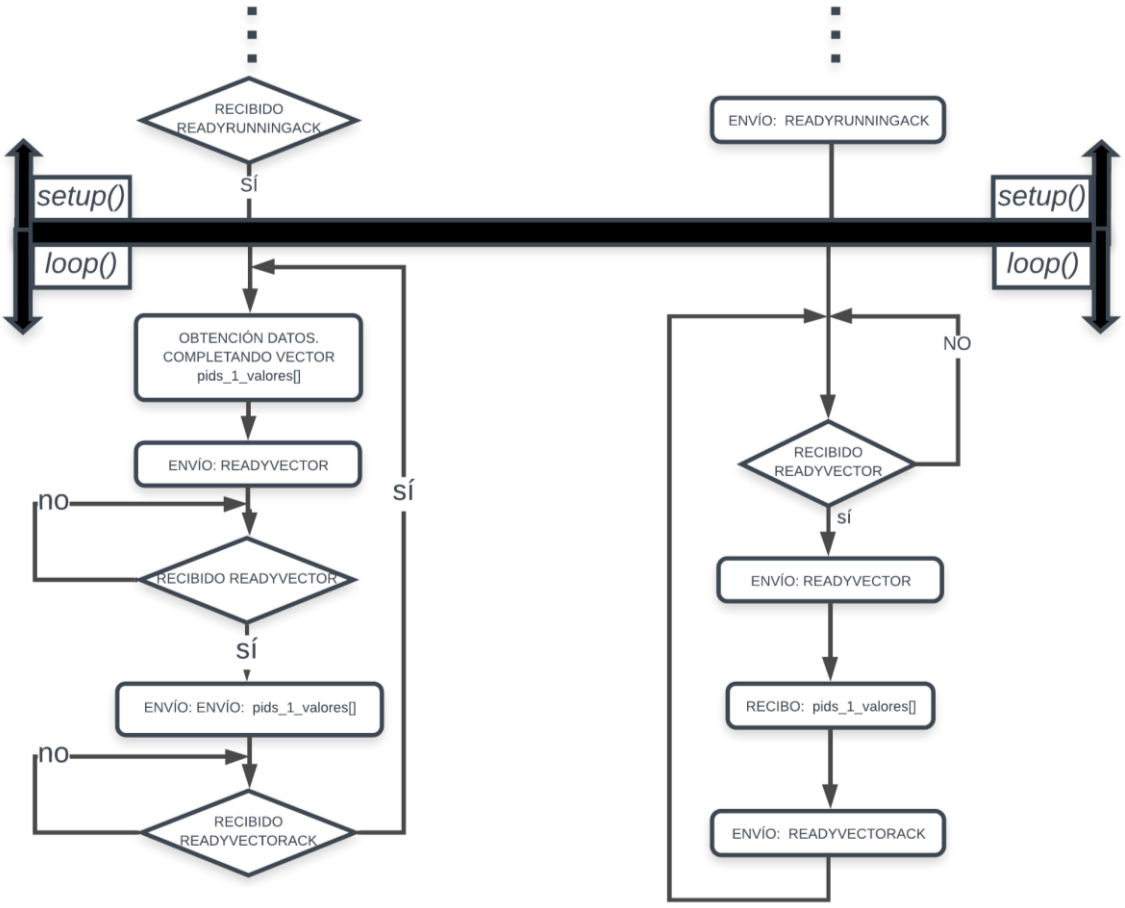


Figura 21. Diagrama de flujo del código en el Arduino Nano (izquierda) y en la Wemos D1 mini (derecha) en la ejecución de la función *loop()*.

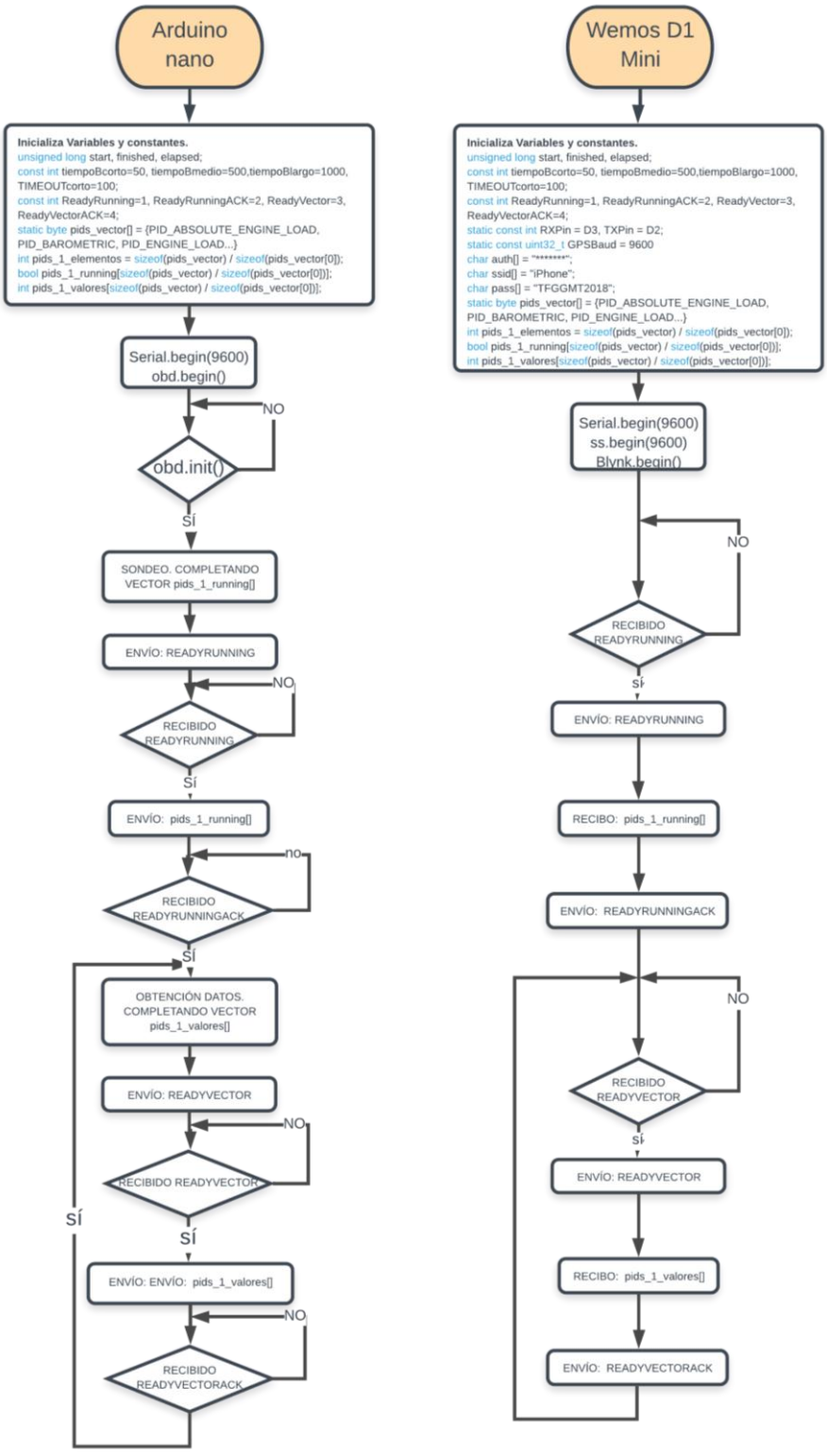


Figura 22. Diagrama de flujo completo del código en el Arduino Nano y en la Wemos D1 mini .

4.2 Esquema y montaje.

Para el montaje se ha hecho uso de:

- 1 Freematics OBD-II I2C Adapter (for Arduino).
- 1 Arduino nano.
- 1 Wemos D1 mini.
- 1 GPS uBlox NEO-6m.
- 1 Pantalla OLED 128X64 píxeles.
- 2 Protoboards.
- Cableado tipo *jumper*.

El esquema de conexión es el siguiente (véase Figura 23):

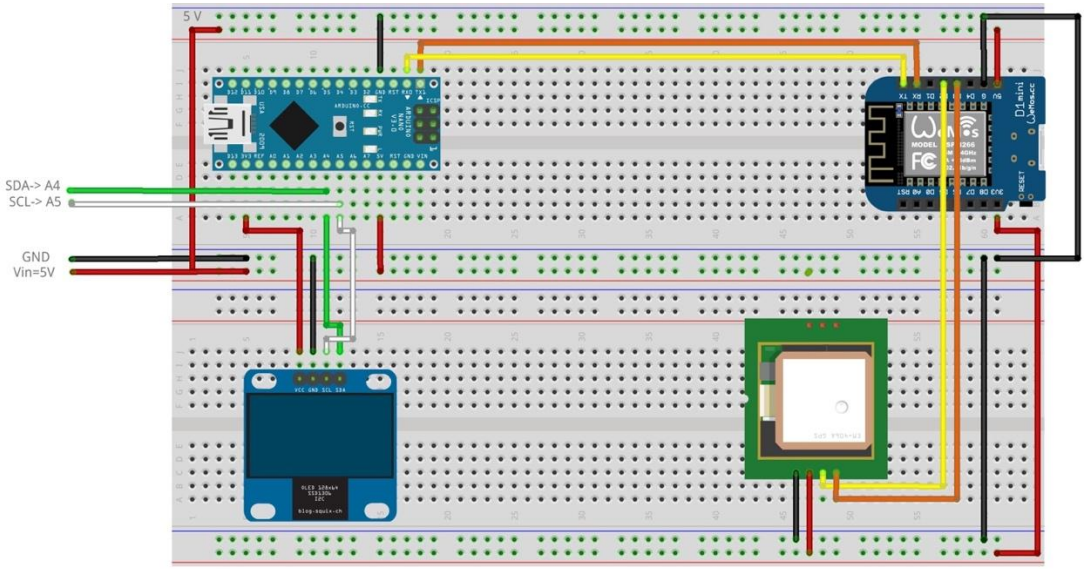


Figura 23. Esquema de conexión del sistema.

El montaje real del sistema sobre el que se han realizado las pruebas, basado en el esquema mostrado en la Figura 23, es el siguiente (véase Figura 24):

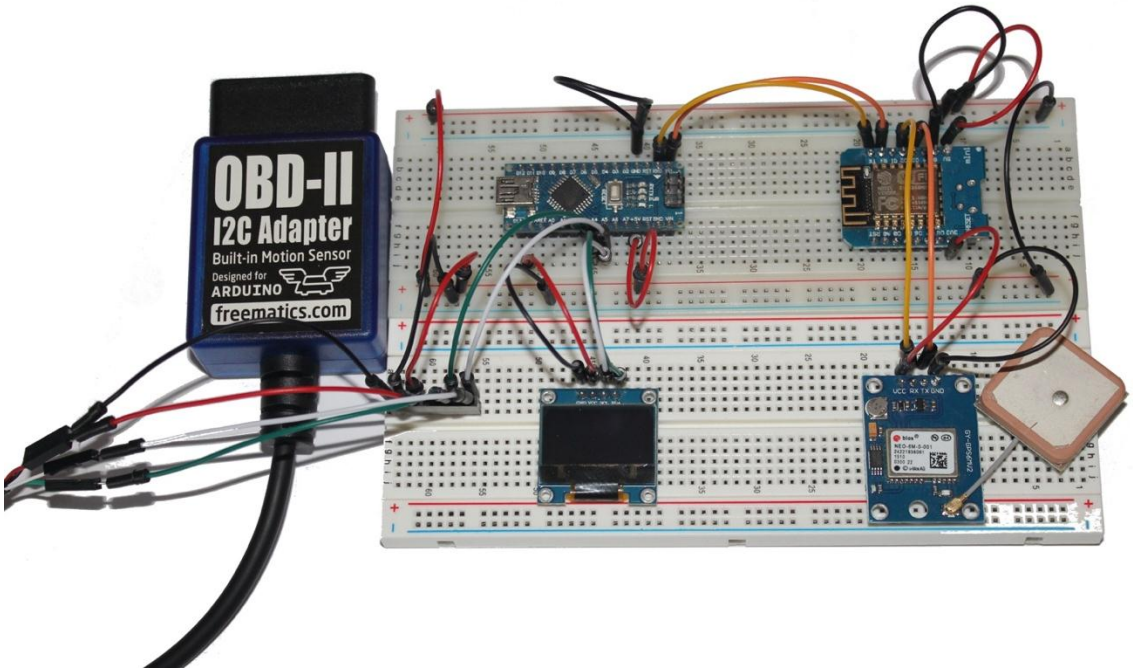


Figura 24. Montaje real del sistema completo y conexionado.

Los pines empleados en los elementos que forman el subsistema de adquisición de datos son (véase Tabla 7):

Conexionado (Pines)	
Arduino Nano	**Adaptador Freematics OBD-II I2C**
5 V	Vout 5 V (Cable rojo)
GND	GND (Cable negro)
A4 (SDA)	SDA (Cable verde)
A5 (SCL)	SCL (Cable blanco)
Arduino Nano	**Pantalla OLED 128x64**
3.3 V	VCC
GND	GND
A4 (SDA)	SDA
A5 (SCL)	SCL
Arduino Nano	**Wemos D1 Mini**
RX0	TX
TX1	RX

Tabla 7. Conexiones del subsistema de adquisición de datos sobre OBD.

Por otro lado, el conexionado del sistema de recepción y envío de estos datos es el siguiente (véase Tabla 8):

Conexionado (Pines)	
Wemos D1 Mini	**GPS Ublox Neo m6**
3.3 V	VCC
GND	GND
D2	RX
D3	TX
Wemos D1 Mini	**Adaptador Freematics OBD-II I2C**
5V	Vout 5 V (Cable rojo)
GND	GND (Cable negro)
Wemos D1 Mini	**Arduino Nano**
TX	RX0
RX	TX1

Tabla 8. Conexiones del subsistema de adquisición de datos sobre OBD.

Observamos entonces la agrupación según la funcionalidad de los diferentes dispositivos. El adaptador de la marca Freematics junto con el arduino nano y la pantalla OLED forman el bloque de extracción y estructura de datos en tiempo real. El Wemos D1 mini junto con el GPS forman el bloque de comunicaciones externas.

Ambos bloques se comunican mediante una comunicación serie entre el arduino nano y la Wemos D1 mini, de manera que el arduino nano envíe continuamente los datos extraidos del vehículo a la Wemos.

Esta estructura del sistema modular favorece el control de errores y las posibles adaptaciones o modificaciones, ya que, en caso de necesitar incorporar nuevas tecnologías inalámbricas, el sistema de extracción no necesitaría modificación alguna, si no que esta sería incluida en el bloque que queda comandado por la Wemos.

4.3 Modos y PIDs empleados.

Dadas las características del sistema OBD, se incluyen varios modos de operación como se presentaron en el capítulo 3, el objeto de estudio principal de este proyecto se basa en el modo 1.

El modo 1 hace referencia al acceso a datos en vivo de valores analógicos o digitales de salidas y entradas a la ECU dado por los identificadores de Parámetro (PID, Process Identification). Este modo es también llamado flujo de datos.

La siguiente tabla, Tabla 9, muestra los PIDs que han sido empleados en sistema en los que se basa la adquisición de datos. Estos PIDs están recogidos en el estándar SAE J1979 y el equivalente técnico europeo ISO 15031-5, estos son mostrados en la Tabla 10 del Apéndice C.

PID (hex)	**Bytes de respuesta**	**Descripción**	**Valor mín.**	**Valor max.**	**Unidad**	**Fórmula**
04	1	Carga calculada del motor	0	100	%	$\frac{100}{255}A$
05	1	Temperatura del líquido de enfriamiento del motor	-40	215	°C	$A - 40$
06	1	Ajuste de combustible a corto plazo—Banco 1	-100 (Reducción de combustible: muy rico)	99.2 (Aumento de combustible: muy magro)	%	$\frac{100}{128}A - 100$
07	1	Ajuste de combustible a largo plazo—Banco 1	-100	99.2	%	$\frac{100}{128}A - 100$
0C	2	RPM del motor	0	16,383.75	rpm	$\frac{256A + B}{4}$
0D	1	Velocidad del vehículo	0	255	km/h	A
0E	1	Avance del tiempo	-64	63.5	° antes TDC	$\frac{A}{2} - 64$
0F	1	Temperatura del aire del colector de admisión Velocidad del flujo del aire MAF	-40	215	°C	$A - 40$
11	1	Posición del acelerador	0	100	%	$\frac{100}{255}A$
1F	2	Tiempo desde que se puso en marcha el motor	0	65,535	sec	$256A + B$
21	2	Distancia recorrida con la luz indicadora de falla (Malfunction Indicator	0	65,535	km	$256A + B$

		Lamp, MIL) encendida				
2E	1	Purga evaporativa comandada	0	100	%	$\frac{100}{255}A$
2F	1	Nivel de entrada del tanque de combustible	0	100	%	$\frac{100}{255}A$
30	1	Cantidad de calentamientos desde que se borraron los fallas	0	255	cuenta	A
31	2	Distancia recorrida desde que se borraron los fallas	0	65,535	km	$256A + B$
33	1	Presión barométrica absoluta	0	255	kPa	A
3C	2	Temperatura del catalizador: Banco 1, Sensor 1	-40	6,513.5	°C	$\frac{256A + B}{10} - 40$
42	2	Voltaje del módulo de control	0	65.535	V	$\frac{256A + B}{1000}$
43	2	Valor absoluto de carga	0	25,700	%	$\frac{100}{255}(256A + B)$
44	2	Relación equivaliente comandada de combustible - aire	0	< 2	prop.	$\frac{2}{65536}(256A + B)$
45	1	Posición relativa del acelerador	0	100	%	$\frac{100}{255}A$
46	1	Temperatura del aire ambiental	-40	215	°C	$A - 40$
47	1	Posición absoluta del acelerador B	0	100	%	$\frac{100}{255}A$
49	1	Posición del pedal acelerador D	0	100	%	$\frac{100}{255}A$
4A	1	Posición del pedal acelerador E	0	100	%	$\frac{100}{255}A$
4C	1	Actuador comandando del acelerador	0	100	%	$\frac{100}{255}A$

Tabla 9. Conexiones del subsistema de adquisición de datos sobre OBD.

4.4 Representación de la información.

Para la presentación final de la información se ha elegido hacer uso de Blynk, la cual una plataforma con aplicaciones tanto para iOS como Android para controlar Arduino, Raspberry Pi y similares a través de Internet [20].

Esta app es notablemente utilizada en el campo del *Intenet Of Things* ya que permite el conocimiento de estados del dispositivo y control sobre ellos. De esta manera,

generando pines virtuales podremos visualizar en cualquier dispositivo móvil la información contenida en el sistema de telemetría. Según la web de la aplicación, los sistemas soportados son:

- Arduino: Uno, Nano, Mini, Pro Mini, Pro Micro, Mega, YÚN (Bridge), Due
- Raspberry Pi
- Particle (ex Spark Core)
- ESP8266
- TinyDuino (CC3000)
- Wicked WildFire (CC3000)

Shields y conexiones:

- USB, conectado a tu ordenador (no requiere shield)
- Ethernet shield (W5100)
- Adafruit CC3000 WiFi
- Official Arduino WiFi shield
- ENC28J60

Esta app nos da el carácter de control de flotas ya que nos facilita la visualización conjunta que está generando un conjunto de dispositivos de telemetría. Por ejemplo, en ella podremos visualizar la localización de cada uno de los vehículos de la flota o comparar el nivel de combustible de cada uno de ellos. Esta app incluye representación gráfica para cualquier tipo de información del dispositivo que le indiquemos como fuente de datos y permite la descarga de estos en formato CSV. Esto nos da un post estudio detallado de la información visualizable en tiempo real.

Además, permite obtener el control y la información en varios dispositivos a la vez y en remoto, lo cual nos permite realizar un control de flotas activo sobre los vehículos.

La implementación de la interfaz de visualización de datos se facilita gracias a que la app Blynk se basa en *Drag-And-Drop*, de manera que seleccionaremos los módulos necesarios para la representación de datos, estos serán de tipo calibre, pantalla de valores, gráfico y mapa, entre los más importantes.

Por tanto, una vez creamos un proyecto en la app, introducimos él todos los elementos, estos serán de tipo *display*, excepto el mapa. Para que puedan acceder a los datos de los valores contenidos en la Wemos D1 Mini hay que asignarle un pin virtual a cada elemento. Solo es posible usar un pin virtual por elemento. En caso de que una

información del dispositivo de telemetría requiera ser representada de varias formas, como por ejemplo en valor actual en una pantalla y en un gráfico que muestre histórico, sería necesario el uso de dos pines virtuales. La modificación de los valores de cada pin virtual se realiza mediante la secuencia: Blynk.virtualWrite(V1, val) para el caso del pin virtual 1.

La inclusión de cualquier módulo en el *dashboard*, requiere únicamente la selección de este en el menú selector de la app, siempre y cuando dispongamos de suficiente "energy". Algunos de los módulos sólo son accesibles una vez por proyecto. De ahí que los elementos de la sección "*Device management*" sean de especial interés y su precio traducido en "energy" sea elevado. La visualización del acceso a los módulos de la app móvil se muestra en la Figura 25.

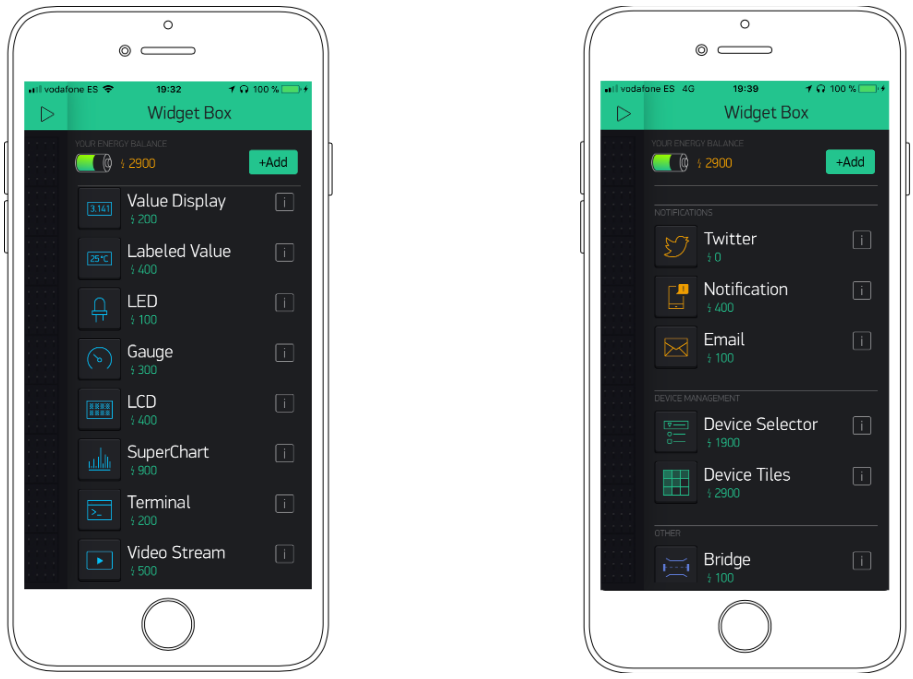


Figura 25. Módulos utilizables en la app Blynk de tipo *display* (izquierda) y de tipo, notificación y gestión de dispositivos (derecha).

El elemento más utilizado es el "*labeled value*" ya que proporciona la representación del valor de cada sensor del vehículo monitorizado añadiendo un etiquetado de las unidades en que es representado, este es por tanto el elemento de mayor uso. Por otro lado, el módulo "*gauge*" será utilizado para la representación tanto de la velocidad como de las revoluciones por minuto del motor, debido a la similitud con la representación en los vehículos comerciales, lo hace más intuitivo.

A continuación, se presenta la información aportada en la app para el uso del módulo "*gauge*" (véase Figura 26):

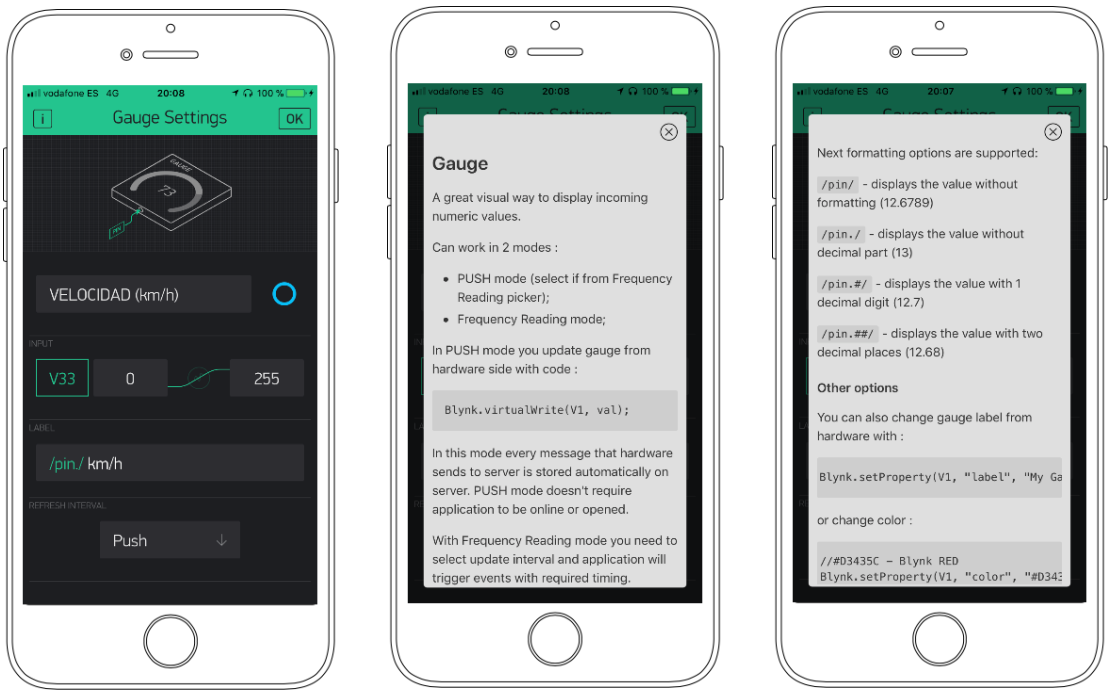


Figura 26. Configuración e información del módulo "*gauge*" perteneciente a la app Blynk.

Para cada uno de los valores a representar será necesario hacer uso de la tabla 9, donde observar los valores mínimos, máximos e unidades de la información obtenida en a partir de cada PID, para la correcta visualización.

Llegados a este punto, cabe destacar el carácter de la gestión de flotas vehiculares del proyecto en este apartado, ya que según se ha presentado, los módulos de la sección "*Device management*" están orientados a la gestión de varios dispositivos. En este proyecto cada dispositivo representa un vehículo, por lo tanto, es aquí donde se introduce el control de flotas.

En primer lugar, el módulo "*Device selector*" es de gran interés cuando se tienen varios dispositivos con las mismas funcionalidades, ya que este módulo permite seleccionar el dispositivo sobre el que se está obteniendo la información.

Incorporando el módulo *device selector* podríamos intercambiar en cualquier instante entre dispositivos activos y de esta manera reduciríamos notablemente el coste invertido en la app, ya que este es elevado para cada uno de los *dashboard* creado para cada vehículo.

No obstante, usando el módulo anterior no es posible visualizar en la misma pantalla el estado de dos dispositivos o vehículos a la vez, esto es implementado gracias al móculo "*Device tiles*", este módulo es el más orientado a la gestión de flotas ya que crea una cuadrícula en la que representa el estado de un pin virtual de cada dispositivo o vehículo activo.

Presentados los módulos que pueden ser utilizados, pasamos al siguiente capítulo en el que comprobamos el correcto funcionamiento del sistema sobre un vehículo comercial.

Capítulo 5

Pruebas y resultados

En este capítulo se muestran las pruebas llevadas a cabo, las cuales presentan el correcto funcionamiento del dispositivo. A lo largo del desarrollo de este, se han ido realizando pruebas continuas con las que se han ido solucionando los diversos problemas que han sucedido.

Para la realización de las pruebas continuas se ha hecho uso notable tanto de la pantalla OLED como de la conexión serie con el PC. Desde el IDE de arduino es posible visualizar el intercambio de mensajes que se producen en el puerto serie, por tanto, haciendo uso de la salida por pantalla se han ido logrando superar distintos estados de control, con los que verificar el funcionamiento de subsistemas.

Algunos de los problemas más sensibles han sido los referentes al control del tiempo de los distintos procesos que realiza el sistema, el intercambio de mensajes entre el subsistema de adquisición de datos OBD y el subsistema de transmisión de estos, y la interpretación de los datos obtenidos.

Por último, las pruebas se han centrado en la correcta visualización de los datos obtenidos en la App móvil, tanto en el dispositivo en el interior del vehículo en que se realizan las pruebas como en remoto.

Retomando brevemente los requisitos que ha de cumplir el sistema, los cuales han de observarse en las pruebas finales.

En primer lugar, el subsistema de adquisición ha de comunicarse correctamente con el adaptador OBD-II I2C, una vez que este ha sido conectado al vehículo y realizar un sondeo de los PIDs útiles. En la pantalla OLED se mostrará la iteración de sondeo y el número total, al final de este. Al mismo tiempo, el sistema de extracción establece la conexión WiFi con el punto de acceso. Entonces se realiza un intercambio de mensajes, los cuales son mostrados en la pantalla, estos mensajes son mostrados desde el punto de vista del arduino, lo cual nos dará información concreta de su estado. Por otro lado, el funcionamiento correcto de la adquisición y envío se comprueba mediante la actualización de los datos en la pantalla principal de la app de representación.

5.1 Verificación estados del dispositivo.

Para comprobar que el dispositivo se encuentre en un estado correcto es fundamental seguir la secuencia expuesta en el diagrama de la Figura 22. Los elementos de la Figura 27, corresponden a los estados secuenciales con los que inicia el sistema de adquisición de datos OBD, que dan una clara muestra del estado completo del sistema, ya que, debido a la recepción de los ACK, podemos estimar el estado del subsistema de transmisión de estos datos.

Figura 27. Visualización por el periférico OLED de los estados iniciales.

Estos estados únicamente se observan al comienzo del sistema, ya que se produce el intercambio del vector pids_1_running[], el cual es transmitido por el arduino Nano en su función *setup()*. A partir de este, entramos en la secuencia en bucle que define el estado continuo del dispositivo.

En la Figura 28, se observan los distintos estados en los que se encuentra el sistema de adquisición de datos definidos por su función *loop()*. Estos estados se repetirán continuamente debido al carácter estacionario del sistema.

Figura 28. Visualización por el periférico OLED de los estados estacionarios.

No obstante, esta prueba nos da información de estado, pero no es posible conocer si los datos que se están obteniendo son correctos.

5.1 Pruebas de transferencia de datos OBD.

Uno de los grandes problemas que se encontraron en la realización del dispositivo es que, a pesar de la correcta transmisión de información entre subsistemas, es que los valores representados no eran coherentes. Esta observación se realiza gracias al puerto serie del Wemos D1 Mini y en contraste de los valores esperados, observable en la tabla 9 o en el apéndice C.

Este error lógico podría haber derivado de cualquier instante en que los datos son utilizados y/o manipulados, por lo tanto, para la solución era necesario conocer el estado de los datos a lo largo de su recorrido en el sistema.

Los puntos críticos más sensibles de error en el procesado de los datos son:

- Obtención por parte del adaptador OBD-II I2C.
- Interpretación o normalización de los datos según la librería OBD.
- Almacenamiento por parte del sistema de adquisición de datos (Arduino Nano).
- Transferencia de estos datos entre ambos subsistemas.
- Almacenamiento por parte del sistema de transferencia de datos (Wemos D1 Mini).
- Modificación de parámetros relacionados con la app Blynk en Wemos D1 Mini.
- Configuración de los pines virtuales que relacionan los módulos de la app Blynk con el sistema de transferencia de datos.

Estos son entonces los puntos que requieren mayor observación en el recorrido de la información, por tanto, para obtener el objeto de error se establecieron valores fijos simulados de manera que se acotaran las posibles situaciones de error.

Los valores fijados contienen un carácter gradual en aumento que indique el PID en cuestión esté relacionado con la posición en el vector pids_vector[ ], de esta manera se visualizó en la app que los datos se representaban correctamente pero eran interpretados con un desfase. De manera que para cualquier PID representado se estaban representando los valores del PID anterior y de ahí las incongruencias en los resultados obtenidos.

El problema era causado por el bucle *for* sobre el que se realiza la lectura de los datos a partir de la comunicación serie y el almacenamiento en el vector pids_1_valores[] por parte de la Wemos D1 mini.

5.2 Comprobación del correcto funcionamiento del sistema.

Por último, resuelta la comunicación total del sistema y la transmisión de datos a través de este, el sistema responde al objetivo de este, realizando un estudio continuo de los parámetros del vehículo gracias a la información obtenida desde el sistema OBD y permite la visualización en remoto del estado de uno o varios vehículos gracias a la app móvil Blynk.

En la Figura 29, se presenta el resultado final del sistema en fase de pruebas finales sobre un vehículo comercial. Cabe destacar que, para el control de una flota, se ha optado por la opción de obtener el mayor número de parámetros sin distinción alguna entre ellos y con transferencia única por cada sondeo completo.

De esta manera, se reduce notablemente el establecimiento con la app de visualización y por tanto reduce el consumo de estos debido a la transferencia de los datos por parte del *smartphone* en el vehículo.

El consumo experimental observado bajo una prueba continua del sistema es de 5 MBytes por hora.

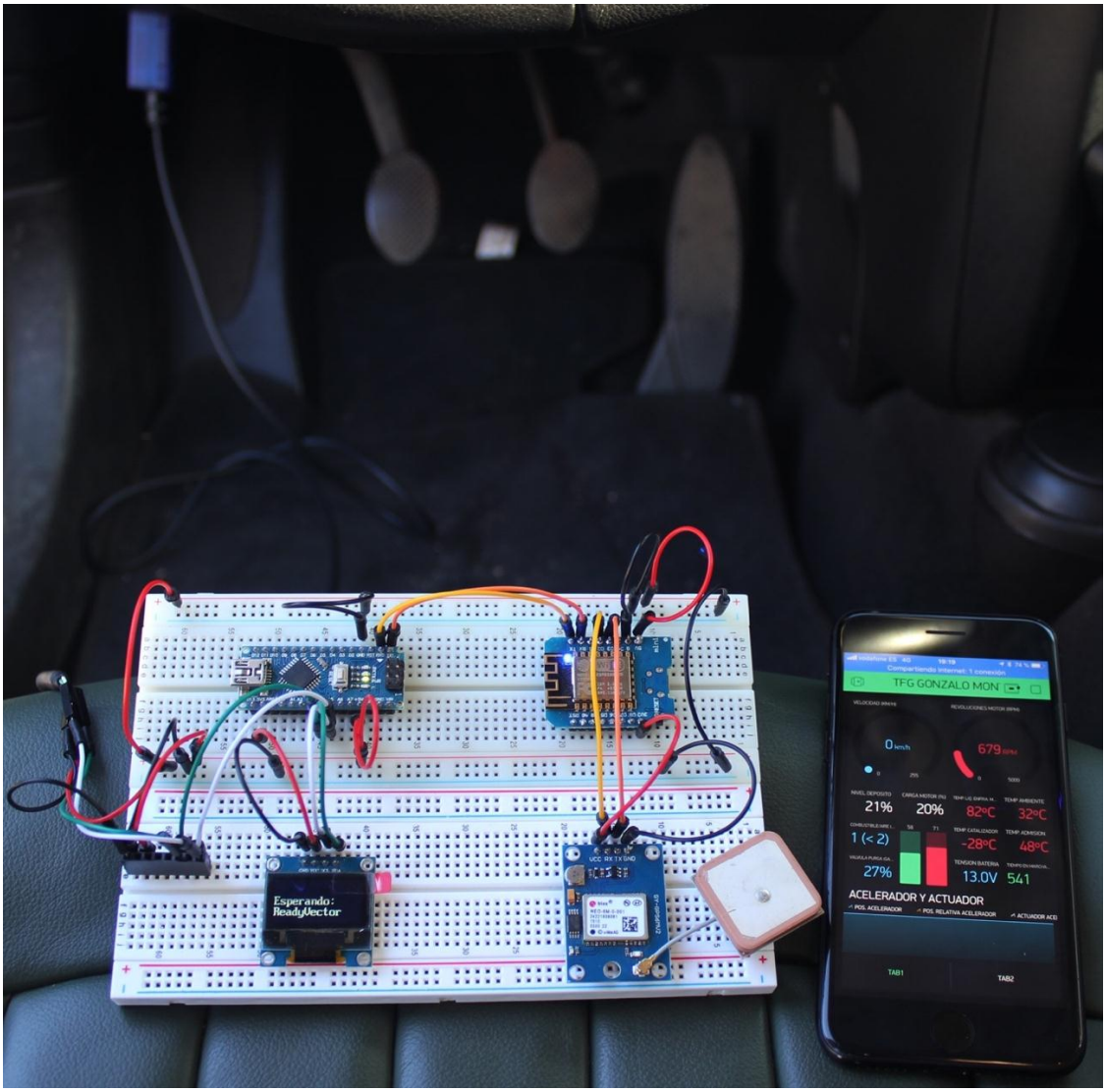


Figura 29. Sistema completo en funcionamiento en un vehículo comercial.

En la Figura 30, se muestra la interfaz de usuario que observaría un gestor de flotas en un *smartphone* gracias a la app Blynk. En esta se observa el estado de un vehículo, donde según lo presentado la sección 4.4 del capítulo 4, podemos seleccionar la información a visualizar entre varios dispositivos activos.

Figura 30. Visualización del funcionamiento en remoto del sistema.

Capítulo 6

Conclusiones y trabajos futuros

En este último capítulo se recogen las conclusiones y objetivos alcanzados en la realización de este proyecto, junto con algunos problemas y adaptaciones que se pueden realizar como líneas futuras. A lo largo del desarrollo del sistema se han ido generando problemas, los cuales algunos serán expuestos en este capítulo.

Para concluir las futuras ampliaciones o adaptaciones se expondrá un caso real en uso de una adaptación del sistema creado.

6.1 Conclusiones.

En este TFG se ha desarrollado un sistema de telemetría orientado al control de flotas haciendo uso de tecnología *open-source* obteniendo información a distancia del estado de uno o varios vehículos a la vez. Todos los parámetros e información obtenida es visualizable en la plataforma Blynk a través de la app correspondiente a iOS o Android.

En la realización se ha profundizado en el estudio de la electrónica que forma parte del sistema OBD de los vehículos comerciales actuales, además del estudio del funcionamiento de los dispositivos hardware utilizados junto con la transmisión de información.

Los principales aportes que se han obtenido en el proyecto son:

- Estudio completo de los sistemas OBD implementados en los vehículos actuales junto con la electrónica correspondiente que sustenta este sistema.

- Recopilación de sistemas actuales de telemetría orientados al control de flotas vehiculares. Con ello, se ha pretendido establecer unas mejoras o ventajas en el dispositivo desarrollado.

- Uso del IDE de arduino para la implementación del software correspondiente a los elementos hardware que forman el dispositivo. Ello ha mostrado los conocimientos adquiridos a lo largo de estos años universitarios con especial estudio en la transmisión de información.

- Estudio de la adquisición de datos externa sobre OBD, tratamiento de esta y transmisión. Incorporación de una plataforma orientada al IoT con la cual viasualizar la información recogida por el sistema.

6.2 Problemas a lo largo del desarrollo.

A lo largo del desarrollo del sistema han aparecido problemas que han provocado el retraso de algunas de las etapas, los de mayor importancia han sido los relacionados con el software del sistema.

En primero lugar, los problemas nacieron en el estudio de la librería proporcionada por la empresa Freematics a la que pertenece el adaptador OBD-II I2C, lo cual facilitaba la obtención de datos procedentes de la ECU del vehículo.

No obstante, el problema que mayor tiempo en resolución ha sido causado por la resolución de la transmisión de información entre los dos subsistemas que forman el dispositivo completo, esto se ha debido a la interpretación de los datos bajo la comunicación serie. Estos datos que se necesitan transmitir son de tipo entero. La solución a la transferencia de información se ha basado en la conversión a tipo *string,* cadena de caracteres, y al uso del carácter especial que hace referencia a una línea nueva para diferenciar entre datos.

6.3 Trabajos futuros.

Llegado a este punto cabe destacar que se han completado los objetivos planteados sobre los que nace este proyecto. A lo largo de la realización de este, se han encontrado posibles ampliaciones o modificaciones que podrían hace el sistema más versátil y robusto frente a errores.

Esto último es de vital importancia en futuras ampliaciones ya que en este no se han contemplado los posibles errores que se pudieran ocasionar en el sistema, los cuales provocarían un bloqueo de este.

Ya que los subsistemas contienen un carácter secuencial, requiere la coordinación de ellos para el correcto funcionamiento, por tanto, sería interesante el uso de *timeouts* que controlen el estado del sistema. Habría que contemplar los problemas dados por una desconexión con el punto de acceso inalámbrico WiFi o con el vehículo. Cualquiera de estos fallos llevaría a un bloqueo del sistema.

Por otro lado, en este proyecto se han tratado todos los datos obtenidos del vehículo con igual prioridad, en cada sondeo de obtención completa de información de la ECU se solicitan todos los PIDs que estén disponibles de los seleccionados. Desde el punto de vista de tiempo de respuesta es poco eficiente ya que puede ascender a varios segundos el tiempo de sondeo, el cual incluye información estable que varía en periodos mucho mayores. Cabe destacar que una de las adaptaciones será la inclusión de contadores con los cuales facilitar un sondeo reducido de la información crítica y determinar cada cuanto tiempo o cada cuántas iteraciones realizar un sondeo completo.

Desde el punto de vista tecnológico cabe presentar que dado el carácter modular que se ha conseguido se pretende que en futuras modificaciones se incorporen tecnologías de red de largo alcance como es LoRaWAN, con las cuales se podría realizar un control vehicular bajo una zona urbana sin necesidad de incorporar un punto de acceso en el interior del vehículo, como es el caso en el uso de WiFi. Además, será interesante la introducción de sistemas de comunicación móvil como es GSM para el control de vehículos que no se encuentren bajo una zona controlada de red.

Es de mención que una adaptación del sistema orientada a la extracción prioritaria de datos y representación de estos, se está ejecutando en la actualidad con el objetivo de conformar el sistema de telemetría, tanto para el control de los ingenieros de pista como para la visualización de datos en el volante de un vehículo de competición que participará en la Formula Student.

Referencias

[1] Vehicle-to-Everything Technology Will Be a Life Saver. (Última visita: 13/06/2018). Disponible en http://www.iotevolutionworld.com/smart-transport/articles/420596-vehicle-to-everything-technology-will-be-life-saver.htm

[2] IAA Vehículos Comerciales Hanóver 2017. (Última visita: 13/06/2018). Disponible en https://www.iaa.de

[3] LOSTnFOUND. (Última visita: 13/06/2018). Disponible en https://www.lostnfound.com/

[4] Blog de gestión de flotas. (Última visita: 13/06/2018). Disponible en http://www.4gflota.com/blog/los-nuevo-retos-del-sector-del-transporte/

[5] Cisco Systems, connected Transportation. (Última visita: 13/06/2018). Disponible en https://www.cisco.com/c/en/us/solutions/industries/transportation.html

[6] Frotcom. (Última visita: 13/06/2018). Disponible en http://www.frotcom.com

[7] TOMTOM Telematics. (Última visita: 13/06/2018). Disponible en https://telematics.tomtom.com

[8] El OBDII Completo. (Última visita: 13/06/2018). Disponible en https://es.wikibooks.org/wiki/El_OBDII_Completo

[9] El protocolo OBD/ OBD2/ EOBD. (Última visita: 13/06/2018). Disponible en http://www.grupocircuit.com/el-protocolo-obd2-eobd/

[10] Sistema OBD-II. (Última visita: 13/06/2018). Disponible en http://obd2-elm327.es/sistema-obd2-historia-descripcion-futuro

[11] OBD (ON BOARD DIAGNOSIS). (Última visita: 13/06/2018). Disponible en http://www.aficionadosalamecanica.com/obd2.htm

[12] Aprendiendo a manejar Arduino en profuncidad. (Última visita: 13/06/2018). Disponible en https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/

[13] Freematics Company . (Última visita: 13/06/2018). Disponible en https://www.freematics.com

[14] Arduino Language Reference . (Última visita: 13/06/2018). Disponible en https://www.arduino.cc/reference/en/

[15] D1 Mini (Wemos Electronics). (Última visita: 13/06/2018). Disponible en https://www.wemos.cc

[16] AZ-Delivery Electronics . (Última visita: 13/06/2018). Disponible en https://www.az-delivery.de

[17] uBlox electronics . (Última visita: 13/06/2018). Disponible en https://www.u-blox.com/en

[18] Cacoo, software online de diagramas. (Última visita: 13/06/2018). Disponible en https://cacoo.com

[19] Repositorio GitHub oficial de Freematics. (Última visita: 13/06/2018). Disponible en https://github.com/stanleyhuangyc/ArduinoOBD

[20] Plataforma Blynk . (Última visita: 13/06/2018). Disponible en https://www.blynk.cc

[21] YouTube . (Última visita: 13/06/2018). Disponible en https://www.youtube.com

Apéndice A
Código de programación del Arduino Nano

Este apéndice muestra el código de programación implementado en el subsistema de adquisición de datos continuo sobre OBD. Este código completa el conocimiento del desarrollo de software que presenta el capítulo 4.

```
/*****************************************************************************
*
*          TRABAJO DE FIN DE GRADO:      CONTROL DE FLOTAS
*          GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN
*          UNIVERSIDAD DE GRANADA
*
*          CÓDIGO 1. ARDUINO NANO. ADQUISICIÓN DE DATOS CONTINUO SOBRE OBD.
*
*           2018  GONZALO MONTALBÁN TIADO <gonzalomontalban@correo.ugr.es>
*
*****************************************************************************/

/*
 *      Subsistema Extractor de datos OBD-II I2C
 *
 *      Este programa hace un primer sondeo de la mayoría de PIDs del modo 1 de
 *       operación del sistema OBD-II, con fin de determinar los sensores cuyo tiempo
 *       de respuesta es inferior a 100 ms. El vector booleano que indica el
 *       resultado del sondeo será transmitido por el puerto serie por defecto.
 *
 *      Una vez obtenidos, se tendrá un vector con los PIDs, otro vector booleano
 *      que indica qué PIDs son disponibles con retardo menos a 100 ms y otro vector
 *      de valores, el cual se irá rellenando completamente en cada ciclo de la
 *      función loop.
 *
 *      Este vector de valores que se rellena encada ciclo de la función loop es
 *       enviado por el puerto serie junto con unos mensajes de confirmación de
 *       estado, como los ACK.
 *
 */
```

```
#include <Wire.h>          // Librería que posibilita la conexión I2C
#include <MicroLCD.h>      // Librería que incluye el objeto que controla la OLED
                              128x64
#include <OBD.h>           // Librería otorgada por Freematics para comunicación con
                              el adaptador OBD


LCD_SSD1306 lcd;           //Creación del objeto lcd sobre la clase LCD_SSD1306 en
                              librería MicroLCD
COBDI2C obd;               //Creación del objeto obd sobre la clase COBDI2C en
                             librería OBD

/*
 * Declaración de variables y constantes tanto de tiempo como de estado.
 * ReadyRunning/ACK y ReadyVector/ACK son fundamentales para la correcta
 * transmisión de los vectores de información entre el Arduino Nano y la Wemos D1
 * mini, ya que servirán para comenzar la transmisión entre estos y comprobación
 * de la recepción por parte de la Wemos.
 */
unsigned long start, finished, elapsed;
const int tiempoBcorto=50, tiempoBmedio=500,tiempoBlargo=1000, TIMEOUTcorto=100;
const int ReadyRunning=1, ReadyRunningACK=2, ReadyVector=3, ReadyVectorACK=4;

/*
 *  static byte pids_vector[] es el vector que contiene los PIDs en hexadecimal,
 *  cuyo estado será solicitado a la ECU. No todos los PID han de ser objeto
 *  de estudio tanto por tiempo en recepcion por parte de la ECU como por
 *  disponibilidad. Por tanto, será necesaria una función que compruebe cuales
 *  de ellos serán los que se solicitarán a lo largo del funcionamiento.
 *
 */


// Primera declaración de pids_vector[]. Contiene todos los PIDS posibles cuyo
// valor en hexadecimal está contenido en la libreria OBD

/*
   static byte pids_vector[] = {
     PID_ENGINE_LOAD, PID_COOLANT_TEMP, PID_SHORT_TERM_FUEL_TRIM_1,
     PID_LONG_TERM_FUEL_TRIM_1, PID_SHORT_TERM_FUEL_TRIM_2,PID_LONG_TERM_FUEL_TRIM_2,
     PID_FUEL_PRESSURE, PID_INTAKE_MAP, PID_RPM, PID_SPEED, PID_TIMING_ADVANCE,
     PID_INTAKE_TEMP, PID_MAF_FLOW, PID_THROTTLE, PID_AUX_INPUT, PID_RUNTIME,
     PID_DISTANCE_WITH_MIL, PID_COMMANDED_EGR, PID_EGR_ERROR,
     PID_COMMANDED_EVAPORATIVE_PURGE, PID_FUEL_LEVEL, PID_WARMS_UPS, PID_DISTANCE,
     PID_EVAP_SYS_VAPOR_PRESSURE, PID_BAROMETRIC, PID_CATALYST_TEMP_B1S1,
     PID_CATALYST_TEMP_B2S1, PID_CATALYST_TEMP_B1S2, PID_CATALYST_TEMP_B2S2,
     PID_CONTROL_MODULE_VOLTAGE, PID_ABSOLUTE_ENGINE_LOAD, _AIR_FUEL_EQUIV_RATIO,
     PID_RELATIVE_THROTTLE_POS, PID_AMBIENT_TEMP, PID_ABSOLUTE_THROTTLE_POS_B,
     PID_ABSOLUTE_THROTTLE_POS_C, PID_ACC_PEDAL_POS_D, PID_ACC_PEDAL_POS_E,
     PID_ACC_PEDAL_POS_F, PID_COMMANDED_THROTTLE_ACTUATOR, PID_TIME_WITH_MIL,
     PID_TIME_SINCE_CODES_CLEARED, PID_ETHANOL_FUEL, PID_FUEL_RAIL_PRESSURE,
     PID_HYBRID_BATTERY_PERCENTAGE, PID_ENGINE_OIL_TEMP, PID_FUEL_INJECTION_TIMING,
     PID_ENGINE_FUEL_RATE, PID_ENGINE_TORQUE_DEMANDED, PID_ENGINE_TORQUE_PERCENTAGE,
     PID_ENGINE_REF_TORQUE
   };

*/


// Segunda declaración de pids_vector[]. contiene los PID ordenados según tiempo de
// extracción de datos. A partir del  PID_BAROMETRIC, el tiempo de cada uno es
// superior a los 6 segundos o no está disponible. Contiene todos los PIDS posibles
// cuyo  valor en hexadecimal está contenido en la libreria OBD.
 /*
static byte pids_vector[] = {
    PID_SHORT_TERM_FUEL_TRIM_1, PID_COOLANT_TEMP, PID_RELATIVE_THROTTLE_POS,
```

```
    PID_ACC_PEDAL_POS_E, PID_LONG_TERM_FUEL_TRIM_1, PID_ABSOLUTE_ENGINE_LOAD,
    PID_AIR_FUEL_EQUIV_RATIO, PID_CATALYST_TEMP_B1S1, PID_TIMING_ADVANCE,
    PID_RUNTIME, PID_SPEED, PID_INTAKE_TEMP, PID_ENGINE_LOAD, PID_DISTANCE_WITH_MIL,
    PID_AMBIENT_TEMP, PID_ABSOLUTE_THROTTLE_POS_B, PID_FUEL_LEVEL, PID_WARMS_UPS,
    PID_DISTANCE, PID_COMMANDED_THROTTLE_ACTUATOR, PID_THROTTLE,
    PID_COMMANDED_EVAPORATIVE_PURGE, PID_CONTROL_MODULE_VOLTAGE,
    PID_ACC_PEDAL_POS_D, PID_RPM, PID_BAROMETRIC
    };
*/

// Tercera declaración de pids_vector[]. contiene los PID de menor tiempo de
// obtención ordenados según su función en el vehículo.

static byte pids_vector[] = {
    PID_ABSOLUTE_ENGINE_LOAD, PID_BAROMETRIC, PID_ENGINE_LOAD,
    PID_SPEED, PID_RPM,
    PID_THROTTLE,
    PID_RELATIVE_THROTTLE_POS, PID_ABSOLUTE_THROTTLE_POS_B, PID_ACC_PEDAL_POS_D,
    PID_ACC_PEDAL_POS_E, PID_COMMANDED_THROTTLE_ACTUATOR,
    PID_AIR_FUEL_EQUIV_RATIO, PID_COMMANDED_EVAPORATIVE_PURGE,
    PID_CATALYST_TEMP_B1S1,
    PID_INTAKE_TEMP,
    PID_FUEL_LEVEL,
    PID_CONTROL_MODULE_VOLTAGE,
    PID_RUNTIME,
    PID_COOLANT_TEMP,
    PID_SHORT_TERM_FUEL_TRIM_1, PID_LONG_TERM_FUEL_TRIM_1,
    PID_WARMS_UPS,
    PID_DISTANCE, PID_DISTANCE_WITH_MIL,
    PID_TIMING_ADVANCE,
    PID_AMBIENT_TEMP
};


/*
 *  pids_1_elementos: Entero que contiene el numero de elementos del
 *  vector pids_vector.
 *
 *  pids_1_running: Vector de booleanos que indica cuales de los PIDs incluidos
 *  en el vector pids_vector son útiles en el vehículo correspondiente.
 *
 *  pids_1_valores: Vector que se irá sobre escribiendo con los valores obtenidos
 *  para cada PID.
 */
 int pids_1_elementos = sizeof(pids_vector) / sizeof(pids_vector[0]);
 bool pids_1_running[sizeof(pids_vector) / sizeof(pids_vector[0])];
 int pids_1_valores[sizeof(pids_vector) / sizeof(pids_vector[0])];
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//                                                                             -
//                                  VOID SETUP                                 -
//                                                                             -
//------------------------------------------------------------------------------


/*
 *    FUNCION SETUP:
 *                    1° Se inicia la conexión Serial y se elimina cualquier dato
 *                       posible en recepción
 *                    2° Se comienza la conexión con el OBD y se espera hasta que
 *                       esta se establezca.
 *                    3° Se muestra por la pantalla OLED, información de estado.
 *                    4° Se realiza un sondeo de los datos útiles obtenidos a partir
 *                       de cada PID, con el objetivo de rellenar el vecotor bool
 *                       pids_1_running[ ].
 *                    5° Se envía mensaje ReadyRunning, listo para enviar
```

```
 *                      pids_1_running[].
 *                    6° Se espera a recibir ReadyRunning que le indica que la Wemos
 *                      espera el
 *                       vector pids_1_running[].
 *                    7° Le envía el vector completo por el puerto serie por defecto
 *                      y espera a recibir ReadyRunningACK.
 */

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);
  lcd.begin();  lcd.setFontSize(FONT_SIZE_MEDIUM); lcd.println("Gonzalo M. TFG");
  delay(1000);

  Serial.begin(9600);
  Serial.flush();
  obd.begin();

  lcd.println(); lcd.println("Iniciando:"); lcd.print("Conexion OBD");  delay(500);

  while (!obd.init()) {
    lcd.clear();
    lcd.setFontSize(FONT_SIZE_MEDIUM); lcd.println("Sin Conexion ");
    lcd.println("Puerto OBD");
    blinkLED(tiempoBcorto);
  }

  lcd.clear();  lcd.println("Conectado"); delay(1000);


//------------------------------->  SONDEO PIDs ÚTILES  <----------------------------
  for(int i=0; i<pids_1_elementos; i++){
      pids_1_running[i]= SondeoPID(  pids_vector[i], TIMEOUTcorto, i  );
      pids_1_valores[i]=0;
  }
  MostrarStringOLED( "PIDs VALIDOS");
  lcd.print(NRunnin(pids_1_running,pids_1_elementos)); delay(2000);  lcd.clear();
//-------------------------------------------------------------------------------------

//------------------------------->  TRANSMISIÓN pids_1_running[ ]  <-----------------
  Serial.print('\n');  Serial.print(ReadyRunning);   Serial.print('\n');
  MostrarStringOLED("Esperando: ");     lcd.println("running");
  //Esperando hasta que WEMOS esté listo para recibir el vector pids_1_running[];
  while(!Serial.available()){}
  while(Serial.parseInt()!=ReadyRunning){};
  MostrarStringOLED("Recibido: ");     lcd.println("ReadyRunning ");delay(1000);

    for(int i=0;i<pids_1_elementos;i++){
        if(pids_1_running[i]){
          Serial.print(1);    Serial.print('\n');
        }
        else{
          Serial.print(0);    Serial.print('\n');
        }
    }

  MostrarStringOLED("Esperando: ");     lcd.println("Running ACK");
  //Esperando hasta que WEMOS mande ACK del vector pids_1_running[];
  while(!Serial.available()){}
  while(Serial.parseInt()!=ReadyRunningACK){};

  MostrarStringOLED("Recibido: ");     lcd.println("Running ACK");delay(1000);
//-------------------------------------------------------------------------------------
}


//-------------------------------------------------------------------------------------
```

```
//                                                                                -
//                                      VOID LOOP                                 -
//                                                                                -
//--------------------------------------------------------------------------------


/*
 *    FUNCION LOOP:
 *                     1° Incicia la pantalla OLED con los campos de estado del
 *                        vehículo.
 *                     2° Se rellena el vector pids_1_valores. Solo son completados
 *                        los correspondientes según lo indique el vector
 *                        pids_1_running[].
 *                     3° Se envía mensaje ReadyVector, listo para enviar
 *                        pids_1_valores[].
 *                     4° Se espera a recibir ReadyVector que le indica que la Wemos
 *                        espera el
 *                         vector pids_1_valores[].
 *                     5° Le envía el vector completo por el puerto serie por defecto
 *                        y espera a recibir ReadyVectorACK.
 */

void loop() {

    initScreen1();

//-----------------> OBTENCIÓN DATOS DE ECU GUARDADO EN pids_1_valores <----------
    for (int indice=0; indice < pids_1_elementos; indice++) {

        if (pids_1_running[indice]) {
          obd.readPID(pids_vector[indice], pids_1_valores[indice]);
        }

        MostrarOLEDPIDs(PID_SPEED, PID_FUEL_LEVEL,PID_THROTTLE,PID_RPM);
    }
//--------------------------------------------------------------------------------


//-------------------------------> TRANSMISIÓN pids_1_valores[ ] <----------------
    Serial.print(ReadyVector);   Serial.print('\n');
    MostrarStringOLED("Esperando: ");     lcd.println("ReadyVector");
    //Esperando hasta que WEMOS esté listo para recibir el vector pids_1_running[];
    while(!Serial.available()){}
    while(Serial.parseInt()!=ReadyVector){};
    MostrarStringOLED("Recibido: ");     lcd.println("ReadyVector ");delay(50);

        for(int i=0;i<pids_1_elementos;i++){
            MostrarStringOLED("ENVIADO");
            lcd.print("Num: "); lcd.println(i+1);
            Serial.print(String(pids_1_valores[i]));    Serial.print('\n');
        }

    MostrarStringOLED("Esperando: ");   lcd.println("ReadyVectorACK");
    //Esperando hasta que WEMOS mande ACK del vector pids_1_running[];
    while(!Serial.available());{}
    while(Serial.parseInt()!=ReadyVectorACK){};
    MostrarStringOLED("Recibido: ");     lcd.println("ReadyVectorACK");
//--------------------------------------------------------------------------------

}



//--------------------------------------------------------------------------------
//                            SUBPROGRAMA initScreen1()                          -
//--------------------------------------------------------------------------------
/*
```

```
 *    FUNCION initScreen1:
 *                          Borra la pantalla OLED y fija las posiciones en esta de
 *                          las unidades de los PIDs que serán representados con la
 *                          función MostrarOLEDPIDs();
 */
void initScreen1(){
  lcd.clear();
  lcd.setFontSize(FONT_SIZE_SMALL);

  lcd.setCursor(24, 3);   lcd.print("km/h");
  lcd.setCursor(110, 3);  lcd.print("rpm");
  lcd.setCursor(0, 7);    lcd.print("DEPOSITO %");
  lcd.setCursor(80, 7);   lcd.print("ACELER.%");
}
//--------------------------------------------------------------------------------

//--------------------------------------------------------------------------------
//                           SUBPROGRAMA MostrarOLEDPIDs()                       -
//--------------------------------------------------------------------------------
/*
 *    FUNCION MostrarOLEDPIDs:
 *                          Muestra en la OLED el valor correspondiente en su
 *                          posición según el PID indicado.
 */
void MostrarOLEDPIDs(byte PID1, byte PID2, byte PID3,byte PID4){

    int valor=0;

      obd.readPID(PID1, valor);   showData(PID1,valor);
      obd.readPID(PID2, valor);   showData(PID2,valor);
      obd.readPID(PID3, valor);   showData(PID3,valor);
      obd.readPID(PID4, valor);   showData(PID4,valor);
}
//--------------------------------------------------------------------------------



//--------------------------------------------------------------------------------
//                             SUBPROGRAMA MostrarStringOLED()                   -
//--------------------------------------------------------------------------------
/*
 *    FUNCION MostrarStringOLED:
 *                                Muestra en la pantalla OLED el string indicado.
 *
 *
 */
void MostrarStringOLED( String palabra){
    lcd.clear(); lcd.setFontSize(FONT_SIZE_MEDIUM);lcd.setCursor(0, 3);
    lcd.println(palabra);
    delay(10);
}
//--------------------------------------------------------------------------------


//--------------------------------------------------------------------------------
//                                    SUBPROGRAMA showData()                     -
//--------------------------------------------------------------------------------
/*
 *    FUNCION showData:
 *                          Muestra en la OLED el valor correspondiente en su
 *                          posición según el PID indicado.
 */
void showData(byte pid, int value){
  switch (pid) {
    //---------------> Valores de PIDs correspondientes a unidades de initScreen1().
    case PID_RPM:
      lcd.setFontSize(FONT_SIZE_SMALL);
```

```
      lcd.setCursor(110, 3);  lcd.print("rpm");

      lcd.setFontSize(FONT_SIZE_XLARGE);
      lcd.setCursor(64, 0);   lcd.printInt((unsigned int)value % 10000, 4);
      break;

    case PID_SPEED:
      lcd.setFontSize(FONT_SIZE_SMALL);
      lcd.setCursor(24, 3);   lcd.print("km/h");

      lcd.setFontSize(FONT_SIZE_XLARGE);
      lcd.setCursor(0, 0);    lcd.printInt((unsigned int)value % 1000, 3);
      break;

    case PID_THROTTLE:
      lcd.setFontSize(FONT_SIZE_SMALL);
      lcd.setCursor(80, 7);   lcd.print("ACELER.%");

      lcd.setFontSize(FONT_SIZE_MEDIUM);
      lcd.setCursor(88, 5);   lcd.printInt(value % 100, 3);
      break;
    //Valores de PIDs correspondientes a unidades de initScreen1().<----------------
    case PID_ENGINE_LOAD:
      lcd.setFontSize(FONT_SIZE_SMALL);
      lcd.setCursor(0, 7);    lcd.print("CARGA MOTOR %");

      lcd.setFontSize(FONT_SIZE_MEDIUM);
      lcd.setCursor(12, 5);   lcd.printInt(value, 3);
      break;

    case PID_FUEL_LEVEL:
      lcd.setFontSize(FONT_SIZE_SMALL);
      lcd.setCursor(0, 7);    lcd.print("DEPOSITO %");

      lcd.setFontSize(FONT_SIZE_MEDIUM);
      lcd.setCursor(12, 5);   lcd.printInt(value, 3);
      break;
     //-------------------------------------------------------------------------------
  }
 }
 //-------------------------------------------------------------------------------



 //-------------------------------------------------------------------------------
 //                                  SUBPROGRAMA blinkLED()                      -
 //-------------------------------------------------------------------------------
 /*
  *    FUNCION blinkLED:
  *                        Función que enciende y apaga el led integrado en el tiempo
  *                        indicado.
  *
  */
 void blinkLED(int tiempoB) {

    digitalWrite(LED_BUILTIN, HIGH);
    delay(tiempoB);
    digitalWrite(LED_BUILTIN, LOW);
    delay(tiempoB);

 }
 //-------------------------------------------------------------------------------


 //-------------------------------------------------------------------------------
 //                                  SUBPROGRAMA SondeoPID()                     -
//-------------------------------------------------------------------------------
```

```
/*
 *    FUNCION SondeoPID:
 *                          Función que solicita el valor de un PID a la ECU y mide el
 *                           tiempo que este tarda en devolver el dato. Esta devuelve
 *                           un booleano de estado según si ha superado un tiempo
 *                           establecido como TIMEOUT o no
 *
 *
 */
bool SondeoPID(byte PIDsondeo, int TIMEOUTsondeo, int nSondeado){
    bool resultado;   int valor;

    MostrarStringOLED("SONDEANDO");
    lcd.print("Num: "); lcd.println(nSondeado+1);

    start= millis();
    obd.readPID(PIDsondeo, valor);
    finished=millis();

    elapsed= finished - start;
    if (  elapsed < TIMEOUTsondeo  ) {
      resultado = true;
    }
    else {
      resultado = false;
    }

    return resultado;
}
//-------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------------------------
//                                      SUBPROGRAMA NRunnin()                                     -
//-------------------------------------------------------------------------------------------------
/*
 *    FUNCION NRunnin:
 *                        Funcion que indica el número de tipo true que existen dentro
 *                         de un vector indicado.
 *
 */
int NRunnin(bool Vector_run[],int NElementos){
  int numero=0;

  for(int i=0; i<NElementos; i++){
      if(Vector_run[i]){
        numero++;
      }
  }
  return numero;
}
//-------------------------------------------------------------------------------------------------
```

Apéndice B
Código de programación del Wemos D1 Mini

Al igual que en el apéndice A, el código siguiente completa el desarrollo del software creado. En este apéndice se trata el código implementado en la Wemos D1 Mini que pertenece al subsitema de recepción de datps OBD por serial, GPS y transmisión vía WiFi.

```
/*****************************************************************************
    *
    *           TRABAJO DE FIN DE GRADO:      CONTROL DE FLOTAS
    *           GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN
    *           UNIVERSIDAD DE GRANADA
    *
    *           CÓDIGO 2. WEMOS D1 MINI. RECEPCIÓN DE DATOS OBD POR SERIAL, GPS Y
    *            TRANSMISION VÍA WIFI.
    *
    *           2018  GONZALO MONTALBÁN TIADO <gonzalomontalban@correo.ugr.es>
    *
    *****************************************************************************/

    /*
     *      Subsistema Transmisión de datos OBD-II
     *
     *      Este programa hace un primer sondeo de la mayoría de PIDs del modo 1 de
     *      operación del sistema OBD-II, con fin de determinar los sensores cuyo tiempo
     *       de respuesta es inferior a 100 ms. El vector booleano que indica el
     *       resultado del sondeo será transmitido por el puerto serie por defecto.
     *
     *      Una vez obtenidos, se tendrá un vector con los PIDs, otro vector booleano
     *      que indica qué PIDs son disponibles con retardo menos a 100 ms y otro vector
     *       de valores, el cual se irá rellenando completamente en cada ciclo de la
     *       función loop.
     *
     *      Este vector de valores que se rellena encada ciclo de la función loop es
     *      enviado por el puerto serie junto con unos mensajes de confirmación de
     *       estado, como los ACK.
     *
     */


    #include <OBD.h>
    #include <SoftwareSerial.h>
    #include <Wire.h>
    #include <TinyGPS++.h>
    #include <ESP8266WiFi.h>
```

```
#include <BlynkSimpleEsp8266.h>

/*
 * Declaración de variables y constantes tanto de tiempo como de estado.
 * ReadyRunning/ACK y ReadyVector/ACK son fundamentales para la correcta
 * transmisión de los vectores de información entre el Arduino Nano y la Wemos D1
 * mini, ya que servirán para comenzar la transmisión entre estos y comprobación
 * de la recepción por parte de la Wemos.
 *
 * RXPin t TXPin serán los pines usados para la comunicación serie por software.
 * GPSBaud indica la tasa de transferencia con el GPS
 *
 * auth, ssid y pass son características de la conexión Wifi y la app Blynk
 */
unsigned long start, finished, elapsed;
const int tiempoBcorto=50, tiempoBmedio=500,tiempoBlargo=1000, TIMEOUTcorto=100;
const int ReadyRunning=1, ReadyRunningACK=2, ReadyVector=3, ReadyVectorACK=4;

static const int RXPin = D3, TXPin = D2;//Pines 2 y 3 a modulo Ublox 6m GPS
static const uint32_t GPSBaud = 9600;

char auth[] = "cf12fa01eff04b37bf688accbe650ea4";
char ssid[] = "iPhone";
char pass[] = "TFGGMT2018";

/*
 * Objetos creados a partir de las clases contenidas en las librerías incluidas.
 */
TinyGPSPlus gps;
SoftwareSerial ss(RXPin, TXPin);

WidgetMap MapaBlynk(V3);

/*
 *  static byte pids_vector[] es el vector que contiene los PIDs en hexadecimal,
 *  cuyo estado será solicitado a la ECU. En este código podría no ser necesario
 *  pero en futuras implementaciones la Wemos podría solicitar el valor de un
 *  PID concreto, y podría ser de interés su valor en hexadecimal.
 */

// Primera declaración de pids_vector[]. Contiene todos los PIDS posibles cuyo
// valor en hexadecimal está contenido en la libreria OBD
/*
  static byte pids_vector[] = {
    PID_ENGINE_LOAD, PID_COOLANT_TEMP, PID_SHORT_TERM_FUEL_TRIM_1,
    PID_LONG_TERM_FUEL_TRIM_1, PID_SHORT_TERM_FUEL_TRIM_2,PID_LONG_TERM_FUEL_TRIM_2,
    PID_FUEL_PRESSURE, PID_INTAKE_MAP, PID_RPM, PID_SPEED, PID_TIMING_ADVANCE,
    PID_INTAKE_TEMP, PID_MAF_FLOW, PID_THROTTLE, PID_AUX_INPUT, PID_RUNTIME,
    PID_DISTANCE_WITH_MIL, PID_COMMANDED_EGR, PID_EGR_ERROR,
    PID_COMMANDED_EVAPORATIVE_PURGE, PID_FUEL_LEVEL, PID_WARMS_UPS, PID_DISTANCE,
    PID_EVAP_SYS_VAPOR_PRESSURE, PID_BAROMETRIC, PID_CATALYST_TEMP_B1S1,
    PID_CATALYST_TEMP_B2S1, PID_CATALYST_TEMP_B1S2, PID_CATALYST_TEMP_B2S2,
    PID_CONTROL_MODULE_VOLTAGE, PID_ABSOLUTE_ENGINE_LOAD, _AIR_FUEL_EQUIV_RATIO,
    PID_RELATIVE_THROTTLE_POS, PID_AMBIENT_TEMP, PID_ABSOLUTE_THROTTLE_POS_B,
    PID_ABSOLUTE_THROTTLE_POS_C, PID_ACC_PEDAL_POS_D, PID_ACC_PEDAL_POS_E,
    PID_ACC_PEDAL_POS_F, PID_COMMANDED_THROTTLE_ACTUATOR, PID_TIME_WITH_MIL,
    PID_TIME_SINCE_CODES_CLEARED, PID_ETHANOL_FUEL, PID_FUEL_RAIL_PRESSURE,
    PID_HYBRID_BATTERY_PERCENTAGE, PID_ENGINE_OIL_TEMP, PID_FUEL_INJECTION_TIMING,
    PID_ENGINE_FUEL_RATE, PID_ENGINE_TORQUE_DEMANDED, PID_ENGINE_TORQUE_PERCENTAGE,
    PID_ENGINE_REF_TORQUE
  };

*/
/*
 *    pids_vector[]: Vector que contiene los PID ordenados según tiempo de
 *    extracción de datos probado en el mini en una vez.
 *    A partir del PID_BAROMETRIC, el tiempo de cada uno es superior a
```

```
 *    los 6 segundos.
 *
 */

// Segunda declaración de pids_vector[]. contiene los PID ordenados según tiempo de
// extracción de datos. A partir del PID_BAROMETRIC, el tiempo de cada uno es
// superior a los 6 segundos o no está disponible. Contiene todos los PIDS
// posibles cuyo valor en hexadecimal está contenido en la librería OBD.
/*
static byte pids_vector[] = {
    PID_SHORT_TERM_FUEL_TRIM_1, PID_COOLANT_TEMP, PID_RELATIVE_THROTTLE_POS,
    PID_ACC_PEDAL_POS_E, PID_LONG_TERM_FUEL_TRIM_1, PID_ABSOLUTE_ENGINE_LOAD,
    PID_AIR_FUEL_EQUIV_RATIO, PID_CATALYST_TEMP_B1S1, PID_TIMING_ADVANCE,
    PID_RUNTIME, PID_SPEED, PID_INTAKE_TEMP, PID_ENGINE_LOAD, PID_DISTANCE_WITH_MIL,
    PID_AMBIENT_TEMP, PID_ABSOLUTE_THROTTLE_POS_B, PID_FUEL_LEVEL, PID_WARMS_UPS,
    PID_DISTANCE, PID_COMMANDED_THROTTLE_ACTUATOR, PID_THROTTLE,
    PID_COMMANDED_EVAPORATIVE_PURGE, PID_CONTROL_MODULE_VOLTAGE,
    PID_ACC_PEDAL_POS_D, PID_RPM, PID_BAROMETRIC
    };
*/

// Tercera declaración de pids_vector[]. contiene los PID de menor tiempo de
   obtención
// ordenados según su función en el vehículo.

static byte pids_vector[] = {
    PID_ABSOLUTE_ENGINE_LOAD, PID_BAROMETRIC, PID_ENGINE_LOAD,
    PID_SPEED, PID_RPM,
    PID_THROTTLE,
    PID_RELATIVE_THROTTLE_POS, PID_ABSOLUTE_THROTTLE_POS_B, PID_ACC_PEDAL_POS_D,
    PID_ACC_PEDAL_POS_E, PID_COMMANDED_THROTTLE_ACTUATOR,
    PID_AIR_FUEL_EQUIV_RATIO, PID_COMMANDED_EVAPORATIVE_PURGE,
    PID_CATALYST_TEMP_B1S1,
    PID_INTAKE_TEMP,
    PID_FUEL_LEVEL,
    PID_CONTROL_MODULE_VOLTAGE,
    PID_RUNTIME,
    PID_COOLANT_TEMP,
    PID_SHORT_TERM_FUEL_TRIM_1, PID_LONG_TERM_FUEL_TRIM_1,
    PID_WARMS_UPS,
    PID_DISTANCE, PID_DISTANCE_WITH_MIL,
    PID_TIMING_ADVANCE,
    PID_AMBIENT_TEMP
};


/*
 *  pids_1_elementos: Entero que contiene el número de elementos del
 *  vector pids_vector.
 *
 *  pids_1_running: Vector de booleanos que indica cuales de los PIDs incluidos
 *  en el vector pids_vector son útiles en el vehículo correspondiente.
 *
 *  pids_1_valores: Vector que se irá sobre escribiendo con los valores obtenidos
 *  para cada PID.
 */

  int pids_1_elementos = sizeof(pids_vector) / sizeof(pids_vector[0]);
  bool pids_1_running[sizeof(pids_vector) / sizeof(pids_vector[0])];
  int pids_1_valores[sizeof(pids_vector) / sizeof(pids_vector[0])];
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//                                                                             -
//                                    VOID SETUP                               -
//------------------------------------------------------------------------------
```

```
/*
 *    FUNCION SETUP:
 *                    1° Se inicia la conexión Serial en pines por defecto y se
 *                       elimina cualquier dato posible en recepción
 *                    2° Se inicia la conexión Serial por software para conexión con
 *                       GPS
 *                    3° Se comienza la conexión WIFI a partir de auth ssid y pass
 *                    3° Se muestra por la pantalla OLED, información de estado.
 *                    5° Recibe mensaje ReadyRunning y lo vuelve a enviar. listo
 *                       para enviar pids_1_running[].
 *                    7° Recibe el vector completo por el puerto serie por defecto y
 *                       envía ReadyRunningACK.
 */

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(9600);
  Serial.flush();
  ss.begin(GPSBaud);

  Blynk.begin(auth, ssid, pass);

//-------------------------------->  RECEPCIÓN PIDs ÚTILES  <------------------------
  while(!Serial.available()){};
  while(Serial.parseInt()!=ReadyRunning){}
  Serial.print('\n');  Serial.print(ReadyRunning);   Serial.print('\n');

  for(int i=0; i<pids_1_elementos;i++){
    while(!Serial.available()){};
    if(Serial.parseInt()==1){
      pids_1_running[i]=true;
    }
    else{
      pids_1_running[i]=false;
    }
    pids_1_valores[i]=0;
  }

  Serial.print(ReadyRunningACK);   Serial.print('\n');
//--------------------------------------------------------------------------------

 }

//--------------------------------------------------------------------------------
//                                                                              -
//                                    VOID LOOP                                 -

//                                                                              -
//--------------------------------------------------------------------------------


/*
 *    FUNCION LOOP:
 *                    1° Emplea la función run de la librería Blynk para enviar los
 *                       datos a la app
 *                    2° Se rellena el vector pids_1_valores. Solo son completados
 *                       los correspondientes según lo indique el vector
 *                       pids_1_running[].
 *                    4° Se espera a recibir ReadyVector y reenviarlo que indica que
 *                       puede comenzar la
 *                       transferencia de pids_1_valores[].
 *                    5° Recibe el vector completo por el puerto serie por defecto y
 *                       espera a recibir ReadyVectorACK.
 */
```

```
void loop() {

  Blynk.run();

//----------------->  RECEPCIÓN DE DATOS DE ECU GUARDADO EN pids_1_valores  <-------
  while(!Serial.available()){};
  while(Serial.parseInt()!=ReadyVector){}
  Serial.print(ReadyVector);   Serial.print('\n');

  while(!Serial.available()){};
  Serial.readStringUntil('\n');

  for(int i=0; i<pids_1_elementos;i++){
    while(!Serial.available()){};
    pids_1_valores[i]=(Serial.readStringUntil('\n')).toInt();
  }

  Serial.print(ReadyVectorACK);   Serial.print('\n');
//-------------------------------------------------------------------------------------

  DatosBlynk(pids_1_valores);
}


//-------------------------------------------------------------------------------------
//                                                                                    -
//                                    SUBPROGRAMA DatosBlynk()                        -
//-------------------------------------------------------------------------------------
/*
 *    FUNCION DatosBlynk:
 *                           Modifica los valores asociados a los pines virtuales V__
 *                           los cuales hacen referencia a las entradas de los
 *                           módulos de representación
 *                           en la app Blynk.
 *
 *                           Los pines virtuales de los datos recibidos del OBD
 *                           comienzan en el pin 30.
 */
 void DatosBlynk(int pids_valores_vector[] ){

    //Blynk.virtualWrite(V30, pids_valores_vector[0]);
    //Blynk.virtualWrite(V31, pids_valores_vector[1]);
    Blynk.virtualWrite(V32, pids_valores_vector[2]);
    Blynk.virtualWrite(V33, pids_valores_vector[3]);
    Blynk.virtualWrite(V34, pids_valores_vector[4]);
    Blynk.virtualWrite(V35, pids_valores_vector[5]);
    Blynk.virtualWrite(V36, pids_valores_vector[6]);
    //Blynk.virtualWrite(V37, pids_valores_vector[7]);
    //Blynk.virtualWrite(V38, pids_valores_vector[8]);
    //Blynk.virtualWrite(V39, pids_valores_vector[9]);
    Blynk.virtualWrite(V40, pids_valores_vector[10]);
    Blynk.virtualWrite(V41, pids_valores_vector[11]);
    Blynk.virtualWrite(V42, pids_valores_vector[12]);
    Blynk.virtualWrite(V43, pids_valores_vector[13]);
    Blynk.virtualWrite(V44, pids_valores_vector[14]);
    Blynk.virtualWrite(V45, pids_valores_vector[15]);
    Blynk.virtualWrite(V46, pids_valores_vector[16]);
    Blynk.virtualWrite(V47, pids_valores_vector[17]);
    Blynk.virtualWrite(V48, pids_valores_vector[18]);
    //Blynk.virtualWrite(V49, pids_valores_vector[19]);
    //Blynk.virtualWrite(V50, pids_valores_vector[20]);
    //Blynk.virtualWrite(V51, pids_valores_vector[21]);
    //Blynk.virtualWrite(V52, pids_valores_vector[22]);
    //Blynk.virtualWrite(V53, pids_valores_vector[23]);
    //Blynk.virtualWrite(V54, pids_valores_vector[24]);
```

```
   Blynk.virtualWrite(V55, pids_valores_vector[25]);

   GPSBlynk();
}
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//
//                          SUBPROGRAMA GPSBlynk()
//                                  -
//-----------------------------------------------------------------------------
/*
*    FUNCION GPSBlynk:
*                          Modifica los valores asociados a los pines virtuales V__
*                          los cuales hacen referencia de la latitud y longitud
*                          dada por el GPS a la app Blynk.
*/
void GPSBlynk( ){
   int index = 1;
   float latitud = (gps.location.lat());
   float longitud = (gps.location.lng());

   MapaBlynk.location(index, latitud, longitud, "TFG GMONTALBAN");
}
//-----------------------------------------------------------------------------
```

Apéndice C
Todos los PIDs

En el Capítulo 4 se exponen los PIDs objetos de adquisición de sus datos en el dispositivo creado en este proyecto. La Tabla 10 muestra todos los posiles PIDs que son pueden ser objeto de estudio, empleando correctamente su identificador y la fórmula de conversión de los datos obtenidos, estos están recogidos en el estándar J1939.

PID (hex)	Bytes de respuesta	Descripción	Valor mín.	Valor max.	unidad	Fórmula
00	4	PIDs implementados [01 - 20]				Cada bit indica si los siguientes 32 PID están implementados.
01	4	Estado de los monitores de diagnóstico desde que se borraron los códigos de fallas DTC;				Codificación en bits
02	2	Almacena los códigos de fallas de diagnóstico DTC de un evento				
03	2	Estado del sistema de combustible				Codificación en bits
04	1	Carga calculada del motor	0	100	%	$\frac{100}{255}A$
05	1	Temperatura del líquido de enfriamiento del motor	-40	215	°C	$A - 40$
06	1	Ajuste de combustible a corto plazo—Banco 1	-100 (Reducción de combustible: muy rico)	99.2 (Aumento de combustible: muy magro)	%	$\frac{100}{128}A - 100$
07	1	Ajuste de combustible a largo plazo—Banco 1	-100	99.2	%	$\frac{100}{128}A - 100$
08	1	Ajuste de combustible a corto plazo—Banco 2	-100	99.2	%	$\frac{100}{128}A - 100$
09	1	Ajuste de combustible a largo plazo—Banco 2	-100	99.2	%	$\frac{100}{128}A - 100$
0A	1	Presión del combustible	0	765	kPa	$3A$
0B		Presión absoluta del colector de admisión	0	255	kPa	A
0C	2	RPM del motor	0	16,383.75	rpm	$\frac{256A + B}{4}$

0D	1	Velocidad del vehículo	0	255	km/h	A
0E	1	Avance del tiempo	-64	63.5	° antes TDC	$\frac{A}{2} - 64$
0F	1	Temperatura del aire del colector de admisión Velocidad del flujo del aire MAF	-40	215	°C	$A - 40$
10	2	Velocidad del flujo del aire MAF	0	655.35	gr/sec	$\frac{256A + B}{100}$
11	1	Posición del acelerador	0	100	%	$\frac{100}{255}A$
12	1	Estado del aire secundario controlado				Codificación en bits
13	1	Presencia de sensores de oxígeno (en 2 bancos)				[A0..A3] == Banco 1, sensores 1-4. [A4..A7] == Banco 2...
14	2	Sensor de oxígeno 1 A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
15	2	Sensor de oxígeno 2 A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
16	2	Sensor de oxígeno 3 A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
17	2	Sensor de oxígeno A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
18	2	Sensor de oxígeno 5 A: Voltaje B: Ajuste de combustible a corto plaz	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
19	2	Sensor de oxígeno 6 A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
1A	2	Sensor de oxígeno 7 A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
1B	2	Sensor de oxígeno 8 A: Voltaje B: Ajuste de combustible a corto plazo	0 -100	1.275 99.2	voltios %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (si B==FF, entonces el sensor no se usa en el cálculo del ajuste)
1C	1	Estándar OBD implementado en este vehículo				Codificación en bits

1D	1	Sensores de oxígenos presentes en el banco 4				Similar a PID 13, pero [A0..A7] == [B1S1, B1S2, B2S1, B2S2, B3S1, B3S2, B4S1, B4S2]
1E	1	Estado de las entradas auxiliares				A0 == Estado de Power Take Off, PTO(1 == activo) [A1..A7] sin uso
1F	2	Tiempo desde que se puso en marcha el motor	0	65,535	sec	$256A + B$
20	4	PID implementados [21 - 40]				Cada bit indica si los siguientes 32 PID están implementados
21	2	Distancia recorrida con la luz indicadora de falla (Malfunction Indicator Lamp, MIL) encendida	0	65,535	km	$256A + B$
22	2	Presión del tren de combustible, relativa al colector de vacío	0	5177.265	kPa	$0.079\,(256A + B)$
23	2	Presión del medidor del tren de combustible (Diesel o inyección directa de gasolina)	0	655,350	kPa	$10\,(256A + B)$
24	4	Sensor de oxígeno 1 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D)$
25	4	Sensor de oxígeno 2 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$
26	4	Sensor de oxígeno 3 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$
27	4	Sensor de oxígeno 4 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$
28	4	Sensor de oxígeno 5 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$
29	4	Sensor de oxígeno 6 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$
2A	4	Sensor de oxígeno 7 AB: Relación equivalente de combustible - aire CD: Voltaje	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$
2B	4	Sensor de oxígeno 8 AB: Relación equivalente de	0 0	<2 <8	Prop. V	$\frac{2}{65536}(256A + B)$ $\frac{8}{65536}(256C + D$

		combustible - aire CD: Voltaje				
2C	1	EGR comandado	0	100	%	$\frac{100}{255}A$
2D	1	falla EGR	-100	99.2	%	$\frac{100}{255}A - 100$
2E	1	Purga evaporativa comandada	0	100	%	$\frac{100}{255}A$
2F	1	Nivel de entrada del tanque de combustible	0	100	%	$\frac{100}{255}A$
30	1	Cantidad de calentamientos desde que se borraron los fallas	0	255	cuenta	A
31	2	Distancia recorrida desde que se borraron los fallas	0	65,535	km	$256A + B$
32	2	Presión de vapor del sistema evaporativo	-8,192	8191.75	Pa	$\frac{256A + B}{4}$ (AB es número binario en complemento de dos con signo)
33	1	Presión barométrica absoluta	0	255	kPa	A
34	4	Sensor de oxígeno 1 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
35	4	Sensor de oxígeno 2 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
36	4	Sensor de oxígeno 3 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
37	4	Sensor de oxígeno 4 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
38	4	Sensor de oxígeno 5 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
39	4	Sensor de oxígeno 6 AB: Relación equivalente de combustible - aire CD: Actua	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
3A	4	Sensor de oxígeno 7 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
3B	4	Sensor de oxígeno 8 AB: Relación equivalente de combustible - aire CD: Actual	0 -128	<2 <128	Prop. mA	$\frac{2}{65536}(256A + B)$ $\frac{256C + D}{256} - 128$
3C	2	Temperatura del catalizador: Banco 1, Sensor 1	-40	6,513.5	°C	$\frac{256A + B}{10} - 40$

3D	2	Temperatura del catalizador: Banco 1, Sensor 1	-40	6,513.5	°C	$\frac{256A+B}{10}-40$
3E	2	Temperatura del catalizador: Banco 1, Sensor 2	-40	6,513.5	°C	$\frac{256A+B}{10}-40$
3F	2	Temperatura del catalizador: Banco 2, Sensor 2	-40	6,513.5	°C	$\frac{256A+B}{10}-40$
40	4	PID implementados [41 - 60]				Cada bit indica si los siguientes 32 PID están implementados
41	4	Estado de los monitores en este ciclo de manejo				Codificación en bits
42	2	Voltaje del módulo de control	0	65.535	V	$\frac{256A+B}{1000}$
43	2	Valor absoluto de carga	0	25,700	%	$\frac{100}{255}(256A+B)$
44	2	Relación equivaliente comandada de combustible - aire	0	< 2	prop.	$\frac{2}{65536}(256A+B)$
45	1	Posición relativa del acelerador	0	100	%	$\frac{100}{255}A$
46	1	Temperatura del aire ambiental	-40	215	°C	$A-40$
47	1	Posición absoluta del acelerador B	0	100	%	$\frac{100}{255}A$
48	1	Posición absoluta del acelerador C	0	100	%	$\frac{100}{255}A$
49	1	Posición del pedal acelerador D	0	100	%	$\frac{100}{255}A$
4A	1	Posición del pedal acelerador E	0	100	%	$\frac{100}{255}A$
4B	1	Posición del pedal acelerador F	0	100	%	$\frac{100}{255}A$
4C	1	Actuador comandando del acelerador	0	100	%	$\frac{100}{255}A$
4D	2	Tiempo transcurrido con MIL encendido	0	65,535	min	$(256A+B)$
4E	2	Tiempo transcurrido desde que se borraron los códigops de fallas	0	65,535	min	$(256A+B)$
4F	4	Valor máximo de la relación de equivalencia de combustible - aire, voltaje del sensor de oxígenos, corriente del sensor de oxígenos y presión absoluta del colector de entrada	0, 0, 0, 0	255, 255, 255, 2550	prop., V, mA, kPa	A, B, C, D*10
50	4	Valor máximo de la velocidad de flujo de aire del sensor de flujo de aire masivo	0	2550	g/s	A*10, B, C, y D están reservados para uso futuro
51	1	Tipo de combustible				
52	1	Porcentaje de combustible Ethanol	0	100	%	$\frac{100}{255}A$
53	2	Presión absoluta del vapor del sistema de evaporación	0	327.675	kPa	$\frac{256A+B}{200}$

54	2	Presión del vapor del sistema de evaporación	-32,767	32,768	Pa	$256A + B - 32767$
55	2	Ajuste del sensor de oxígeno secundario de plazo corto. A: banco 1. B: banco 3	-100	99.2	%	$\frac{100}{128}A - 100$ $\frac{100}{128}B - 100$
56	2	Ajuste del sensor de oxígeno secundario de plazo largo. A: banco 1. B: banco 3	-100	99.2	%	$\frac{100}{128}A - 100$ $\frac{100}{128}B - 100$
57	2	Ajuste del sensor de oxígeno secundario de plazo corto. A: banco 2. B: banco 4	-100	99.2	%	$\frac{100}{128}A - 100$ $\frac{100}{128}B - 100$
58	2	Ajuste del sensor de oxígeno secundario de plazo largo. A: banco 2. B: banco 4	-100	99.2	%	$\frac{100}{128}A - 100$ $\frac{100}{128}B - 100$
59	2	Presión absoluta del tren de combustible	0	655,350	kPa	$10(256A + B)$
5A	1	Posición relativa del pedal del acelerador	0	100	%	$\frac{100}{255}A$
5B	1	Tiempo de vida del banco de baterías híbridas	0	100	%	$\frac{100}{255}A$
5C	1	Temperatura del aceite del motor	-40	210	°C	$A - 40$
5D	2	Sincronización de la inyección de combustible	-210.00	301.992	°	$\frac{256A + B}{128} - 210$
5E	2	Velocidad del combustible del motor	0	3276.75	L/h	$\frac{256A + B}{20}$
5F	1	Requisitos de emisiones para los que el vehículo fue diseñado				Codificación en bits
60	4	PID implementados [61 - 80]				Cada bit indica si los siguientes 32 PID están implementados
61	1	Porcentaje de torque solicitado por el conductor	-125	125	%	$A - 125$
62	1	Porcentaje de torque actual del motor	-125	125	%	$A - 125$
63	2	Torque de referencia del motor	0	65,535	Nm	$256A + B$
64	5	Datos del porcentaje de torque del motor	-125	125	%	A-125 Ocioso B-125 Motor punto 1 C-125 Motor punto 2 D-125 Motor punto 3 E-125 Motor punto 4
65	2	Entrada / salida auxiliar implementada				Codificación en bits
66	5	Sensor de flujo de aire masivo				
67	3	Temperatura del enfriador del motor				

68	7	Sensor de temperatura de aire de entrada				
69	7	EGR comandado y falla de EGR				
6A	5	Control comandado del flujo de aire de entrada de Diesel y posición relativa de la entrada del flujo de aire				
6B	5	Temperatura de recirculación del gas del escape				
6C	5	Control comandado del actuador del acelerador y posición relativa del acelerador				
6D	6	Sistema de control de presión del combustible				
6E	5	Sistema de control de presión de inyección				
6F	3	Presión de entrada del compresor del turbocargador				
70	9	Control de presión de aumento				
71	5	Control del turbo de geometría variable (Variable Geometry Turbo, VGT)				
72	5	Control de la compuerta de desperdicio				
73	5	Presión del escape				
74	5	RPM del turbocargador				
75	7	Temperatura del turbocargador				
76	7	Temperatura del turbocargador				
77	5	Temperatura del enfriador del aire de carga (Charge Air Cooler Temperature, CACT)				
78	9	Temperatura del gas del escape (Exhaust Gas Temperature, EGT) Banco 1				PID especial
79	9	Temperatura del gas del escape (Exhaust Gas Temperature, EGT) Banco 2				PID especial
7A	7	Filtro de partículas Diesel (Diesel Particulate Filter, DPF)				
7B	7	Filtro de partículas Diesel (Diesel Particulate Filter, DPF)				
7C	9	Temperatura del filtro de partículas Diesel				

		(Diesel Particulate Filter, DP F)				
7D	1	Estado del área de control NOx NTE				
7E	1	Estado del área de control PM NTE				
7F	13	Tiempo que el motor ha estado en marcha				
80	4	PID implementados [81 - A0]				Cada bit indica si los siguientes 32 PID están implementados
81	21	Tiempo de marcha del motor para el dispositivo auxiliar de control de emisiones (Auxiliary Emissions Control Device, AECD)				
82	21	Tiempo de marcha del motor para el dispositivo auxiliar de control de emisiones (Auxiliary Emissions Control Device, AECD)				
83	5	Sensor de NOx				
84		Temperatua de superficie del colector				
85		Sistema reactivo NOx				
86		Sensor de partículos (Particular Matter, PM)				
87		Presión absoluta del colector de admisión				
A0	4	PID implementados [A1 - C0]				Cada bit indica si los siguientes 32 PID están implementados
C0	4	PID implemantados [C1 - E0]				Cada bit indica si los siguientes 32 PID están implementados
C3						Muestra datos, incluye ID de la condición del motor y su velocidad
C4						B5: Solicitud de poner el motor en estado ocioso B6: Solicitud de detener el motor

Tabla 10. Códigos PID para el modo de operación 1 bajo el estándar J1939 [8].