UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Monitorización de los parámetros de conducción con el sistema OBD y Raspberry Pi para el análisis del comportamiento del conductor

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN

MENCIÓN EN TELEMÁTICA

Autor: Javier San José González

Tutor: David González Ortega

Valladolid, 13 de abril de 2018

TÍTULO:	**Monitorización de los parámetros de conducción con el sistema OBD y Raspberry Pi para el análisis del comportamiento del conductor**
AUTOR:	**Javier San José González**
TUTOR:	**D. David González Ortega**
DEPARTAMENTO:	**Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dña. Miriam Antón Rodríguez**
VOCAL:	**D. Mario Martínez Zarzuela**
SECRETARIO:	**D. David González Ortega**
SUPLENTE:	**D. Francisco Javier Díaz Pernas**
SUPLENTE:	**D. José Fernando Díez Higuera**

FECHA:	13/04/2018
CALIFICACIÓN:	

AGRADECIMIENTOS

Sería imposible comenzar estos agradecimientos sin nombrar a mis padres: Miguel y Carmen. Este documento finaliza una etapa, la cual ni siquiera habría existido sin vuestro apoyo incondicional e inagotable. Gracias, de todo corazón, por haber hecho y seguir haciendo posible este y otros muchos momentos.

A Guzmán, porque fuiste tú el primero que me animó a afrontar este gran paso, y porque durante todos estos años te has mantenido presente en cada caída y cada logro.

A todos los familiares y amigos, que por una razón u otra habéis aportado una gota de tinta con la cual he escrito este capítulo de mi vida. Tengo la gran suerte de que ellos mismos saben quién son.

Y a David, mi tutor, por haberme brindado la oportunidad de formarme con él a través de estos años.

Gracias.

TABLA DE CONTENIDOS

LISTA DE TABLAS

LISTA DE FIGURAS

RESUMEN

Con este Trabajo de Fin de Grado, el alumno intenta acercarse a las tecnologías de comunicación de datos utilizadas en los vehículos. Todas ellas, definidas en lo que se conoce como *OBD*, ayudan a estudiar la eficiencia en la conducción. Para ello, se desarrolla una aplicación en C, *Raspberry Pi Driving Data Acquisition*, la cual permite solicitar, almacenar y ofrecer los datos del vehículo de una forma adecuada para posibles estudios futuros.

Palabras clave: OBD, Raspberry Pi, C, eficiencia

ABSTRACT

With this Bachelor's Degree Thesis, the student tries to get close to the data communication technologies used on vehicles. All of that, defined under the *On-Board Diagnostics* standards, helps with the study of driving efficiency. In order to do this, an application called *Raspberry Pi Driving Data Acquisition* is developed. This application requests, stores and offers the data in a way that is possible for future studies to use it.

CAPÍTULO 1. INTRODUCCIÓN

Desde hace tiempo, los vehículos han sido parte de nuestras vidas. Desde los primeros vehículos con motor de combustión interna, hasta los últimos vehículos propulsados por electricidad Todos ellos han sido posibles gracias a los avances en ingeniería y tecnología. Y estos, a su vez, son posibles gracias a la obtención y análisis de datos.

En los comienzos, los vehículos no eran más que un conjunto de piezas, engranajes, etc., que, en conjunto, hacían que la fuerza producida en el motor se transmitiera a las ruedas, y que el conductor, a través del volante y los pedales, pudiera controlar la dirección y aceleración.

En la actualidad, es probable que el motor de un vehículo no arranque si el conductor no tiene puesto el cinturón de seguridad. El pedal de aceleración ya no está unido físicamente al motor, sustituyendo esta unión por la combinación de un potenciómetro y la transmisión de la posición de este a través de señales digitales por un bus de comunicación. A través del mismo, se transmiten además los datos de cientos de sensores que aportan información valiosa a la unidad de control del motor. Y esta, utilizando estos datos, es capaz de tomar decisiones en tiempo real respecto al par entregado por el vehículo, el par solicitado por el conductor y, en última instancia, tomar decisiones respecto a la trayectoria que debe seguir el vehículo si delante del mismo se encuentra un obstáculo.

Como se puede comprobar, un vehículo ya no es solo una combinación de partes mecánicas y engranajes que producen como resultado final un movimiento. Las tecnologías de la información han conquistado completamente este medio de transporte, que, gracias a estos avances, ha ganado en seguridad y eficiencia.

Pero como se ha mencionado anteriormente, esto no habría sido posible sin la obtención y análisis de los datos producidos por el vehículo. Por ello, tanto *ISO* (*International Organization for Standarization*) como *SAE* (*Society of Automotive Engineers*) se lanzaron a estandarizar lo que se conoce con las siglas de *OBD* (*On-Board Diagnostics*). Este conjunto de estándares y protocolos permite obtener los datos necesarios para diagnosticar un vehículo con el fin de detectar posibles averías que hagan que contamine más de lo estipulado.

El presente Trabajo de Fin de Grado pretende ser un ejemplo de utilización de *OBD* para la obtención en tiempo real de los datos de un vehículo. Para ello, se propone la creación de una aplicación software, desarrollada en C, que permita solicitar, almacenar y exponer estos datos para que puedan ser estudiados o "consumidos" por otras aplicaciones. La aplicación que se presenta en este trabajo, llamada *Raspberry Pi Driving Data Acquisition* (*RPIDDA*), no pretende ser un competidor directo de aplicaciones ya existentes en el

mercado. En vez de eso, se puede considerar como el objetivo final de un proceso de aprendizaje basado en el *DIY* (*Do It Yourself*).

La estructura de este documento se describe a continuación: seguido de este capítulo introductorio, en el capítulo 2, se dedicarán unas breves líneas a explicar la eficiencia en la conducción, y como pequeños hábitos pueden cambiar el consumo energético de un vehículo. En el capítulo 3 se introducirá el sistema OBD, mencionando algunas características técnicas de utilidad para este trabajo. En el capítulo 4 se listarán todos los elementos hardware utilizados en el prototipo final. Para cada elemento, se indicarán las características deseadas, y como el elemento escogido se adapta a las necesidades. La implementación del hardware se explicará en el capítulo 5. Los capítulos 6 y 7 se dedicarán plenamente al desarrollo software. En el sexto, se describe la configuración software necesaria alrededor de la aplicación final. En el séptimo, se describe detalladamente el funcionamiento de la aplicación *RPIDDA*. Se recomienda acompañar la lectura de este capítulo mediante el código fuente, el cual está debidamente comentado. Los resultados obtenidos mediante el uso de este prototipo, y su análisis, se encuentran en el capítulo 8. Como no podría ser de otra manera, el capítulo 9 está dedicado al presupuesto económico y tiempo dedicado. El capítulo final incluye unas conclusiones derivadas del desarrollo, las cuales serán muy útiles en futuras ampliaciones. Además, este capítulo dedica unas líneas a las posibles utilizaciones que este proyecto podría tener en conjunto con las tecnologías y paradigmas de lo que se conoce como *Internet of Things* (*IoT*).

CAPÍTULO 2. EFICIENCIA EN LA CONDUCCIÓN

Este capítulo no pretende ser una reescritura de los datos y estudios ya existentes respecto a la eficiencia en la conducción. Su existencia es una mera justificación y exposición de uno de los casos de uso para los que se podría utilizar este proyecto, y todo lo aquí mencionado se encuentra recogido en el *Manual de Conducción Eficiente para Conductores del Parque Móvil del Estado* [1].

Según indica en el documento mencionado anteriormente, la conducción eficiente tiene como objetivos:

- Bajo consumo de carburante.
- Reducción de la contaminación ambiental.
- Mayor confort de conducción
- Disminución de riesgos en la carretera.

Para conseguirlo, es posible seguir unas reglas sencillas y eficaces que aprovechen las posibilidades que ofrecen las tecnologías de los motores de los vehículos actuales.

Los aspectos relevantes para este proyecto son aquellos en los que interviene el conductor a través del control directo de la aceleración del vehículo, ya sea a través del pedal del acelerador, de la marcha engranada, o de diversos sistemas de control automático.

Así, por ejemplo, utilizando este proyecto, es posible obtener la marcha engranada en un vehículo de transmisión manual. Esto permite la realización de estudios respecto al régimen de revoluciones por minutos, marcha engranada, etc.

En el documento mencionado anteriormente aparece gran cantidad de información que conviene tener presente antes de realizar estudios.

CAPÍTULO 3. SISTEMA OBD (ON-BOARD DIAGNOSTICS)

3.1 ¿Qué es OBD?

OBD son las siglas de *On-Board Diagnostics*. Es un sistema originalmente diseñado para reducir las emisiones de sustancias contaminantes a través de la monitorización de ciertos parámetros que caracterizan el rendimiento de un motor.

El sistema OBD se compone de:

- ECU (*Engine Control Unit*): Es el sistema electrónico responsable del control del motor de un vehículo. Recibe información de sensores (pedal de aceleración, sensores de oxígeno, etc.) y controla actuadores (inyectores, etc.) para obtener el rendimiento solicitado (dentro de unos márgenes) por el conductor.

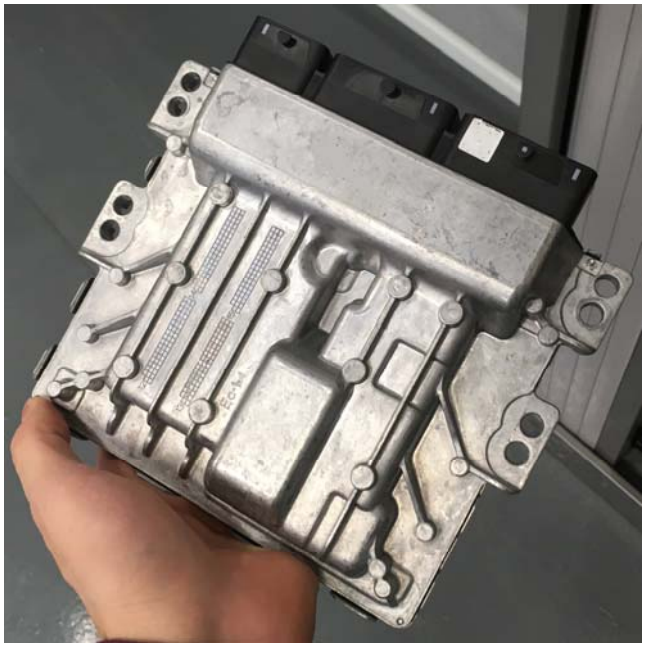


Figura 1: Exterior de una ECU

Figura 2: Conectores de una ECU

- DLC (*Diagnostic Link Connector*): Conector situado en las proximidades de la posición del conductor. A el se conecta la herramienta de escaneo o diagnóstico.

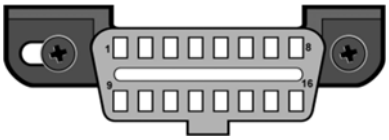


Figura 3: *Diagnostic Link Connector*

Existen dos estandarizaciones de OBD, llamadas *OBD-I* y *OBD-II*. Este proyecto se basa en la segunda estandarización.

En los comienzos de 1990, SAE (*Society of Automotive Engineers*) e ISO (*International Standarization Organization*) establecieron un conjunto de estándares que describen el intercambio digital de información entre ECU y la herramienta de diagnóstico. Previamente, cada fabricante de automóviles utilizaba sus propios sistemas, conectores y protocolos, lo cual complicaba la tarea de obtener información. Todos los vehículos con motor de combustión que cumplan el estándar *OBD-II* (obligatorio desde 1996) deben incluir un conector de diagnóstico estándar, siguiendo la normativa SAE J1962, y comunicarse a través de este utilizando uno de los protocolos de comunicación establecidos para *OBD-II*.

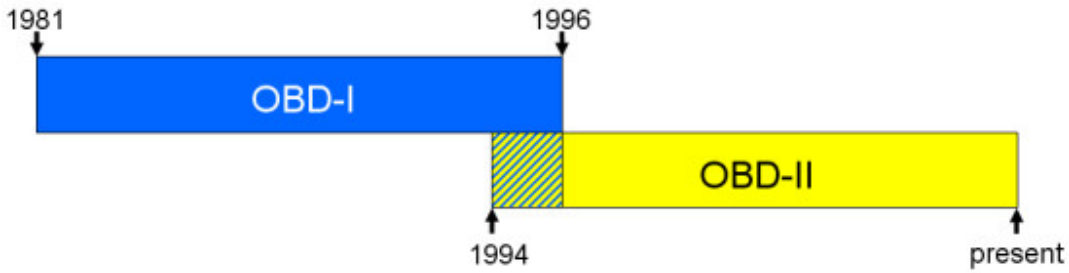


Figura 4: Línea temporal de estandarización OBD

Es importante indicar que no todos los vehículos deben cumplir el estándar. OBD fue diseñado para obtener información referente al sistema de anticontaminación de los vehículos de combustión interna. Por tanto, los vehículos eléctricos quedan excluidos de cumplir con este estándar, si bien pueden incluir el conector OBD.

3.2 Adaptador OBD

Un adaptador OBD es un dispositivo que permite el acceso a la red del vehículo a través de la interpretación de varios protocolos. En la Figura 5 se puede ver el esquema de conexión OBD:

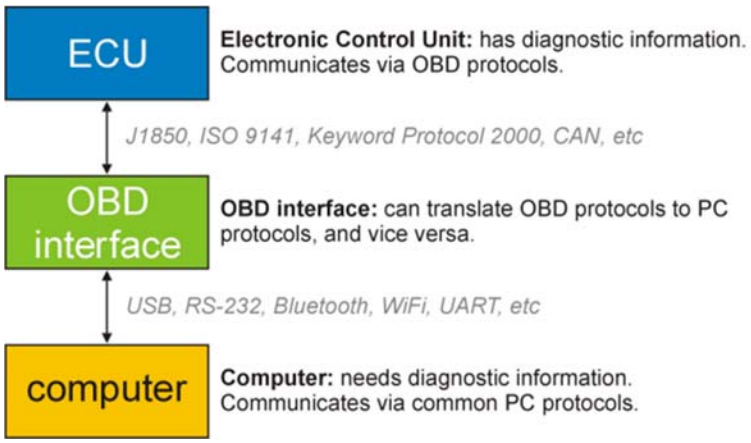


Figura 5: Esquema de conexión OBD

Se describe a continuación un caso básico de comunicación:

- En el caso de este proyecto, la aplicación a desarrollar (*Computer* en la figura) se conecta al interfaz OBD a través de Bluetooth.

- Una vez realizado el emparejamiento, la aplicación solicita información del vehículo al interfaz OBD a través de PIDs (*Parameter IDs*).
- El adaptador OBD recibe el PID y lo retransmite a la ECU del vehículo utilizando uno de los protocolos soportados por este último.
- La ECU responde con los datos solicitados (si están disponibles para ese vehículo), los cuales son retransmitidos a la aplicación por el adaptador OBD.
- La aplicación interpreta la respuesta.

3.3 PIDs y modos

Los PIDs (*Parameter IDs*) son códigos utilizados para solicitar información a la ECU de un vehículo a través de un adaptador OBD. Estos PIDs siguen el estándar SAE J1979 [2], aunque se permite a los fabricantes de vehículos definir sus propios PIDs.

Los PIDs están organizados en Modos, los cuales se listan a continuación:

Modo (hex)	Descripción
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information
0A	Permanent Diagnostic Trouble Codes

Tabla 1: Modos OBD

Los fabricantes no están obligados a soportar todos los modos incluidos en el estándar, e incluso pueden añadir nuevos modos para soportar otras funcionalidades, como, por ejemplo, monitorización del sistema eléctrico de vehículos eléctricos.

La aplicación desarrollada en este proyecto soporta el modo 01, *Show current data*, aunque ésta ha sido desarrollada para facilitar su ampliación al resto de modos soportados por el estándar.

A continuación, se listan los PIDs soportados por la aplicación. Puede encontrarse una lista más completa en el estándar o en Wikipedia [3]:

PID (hex)	Descripción
04	Calculated engine load
05	Engine coolant temperatura
0C	Engine RPM
0D	Vehicle Speed
10	Mass Air Flow rate (MAF)
11	Throttle position
47	Fuel Tank Level Input
5E	Engine fuel rate
61	Driver's demand engine - percent torque
62	Actual engine - percent torque

Tabla 2: PIDs soportados por la aplicación

Debido a ciertas limitaciones, generalmente impuestas por los fabricantes de vehículos, la aplicación desarrollada es capaz de registrar un máximo de 5 PIDs por segundo. Estos PIDs deben ser los mismos durante toda la sesión de captura.

CAPÍTULO 4. SELECCIÓN DE HARDWARE

Este capítulo pretende ser una descripción de los componentes hardware utilizados en el proyecto. Se tendrán en cuenta las siguientes secciones hardware:

- Sistema de procesamiento.
- Módulo GPS.
- Pantalla.
- Adaptador Bluetooth.
- Escáner OBD (adaptador OBD),
- Caja.

De cada uno de ellos se expondrán los requisitos para seleccionar un determinado producto, además de ventajas, inconvenientes, y las razones para seleccionar un componente frente a otro del mismo tipo.

4.1 Sistema de procesamiento: Raspberry Pi 2 Model B

Como sistema que ejecuta la aplicación, se ha tenido que escoger entre varias de las plataformas más utilizadas actualmente:

- Raspberry Pi
- Arduino
- Orange Pi
- BeagleBone
- Intel Edison, Joule…

Puesto que este proyecto no establece restricciones excesivas respecto a procesamiento en tiempo real, se ha decidido utilizar un sistema operativo como mediador entre el hardware y nuestra aplicación. Esto facilitará tareas como:

- Gestión de ficheros: bases de datos de usuarios, ficheros de configuración…

- Gestión de interrupciones hardware: utilizadas para los botones de control.
- Facilidad para la comunicación entre software y hardware: como por ejemplo, hardware Bluetooth.
- etc.

Este requisito elimina automáticamente de la lista a Arduino. Esta plataforma ofrece muchas posibilidades en cuanto a control físico del hardware, pero también conlleva muchas complicaciones, las cuales son evitables utilizando un sistema operativo.

Respecto a las plataformas Intel, es cierto que se apoyan en un sistema operativo GNU/Linux Yocto, pero la falta empuje por parte de Intel ha hecho que sean retiradas del mercado, por lo que no ofrecen demasiado soporte.

De entre las restantes, las dos que más soporte ofrecen por parte de la comunidad son BeagleBone y Raspberry Pi. Debido a la preferencia del autor por esta última, Raspberry Pi [4] se ha escogido como plataforma para este proyecto.

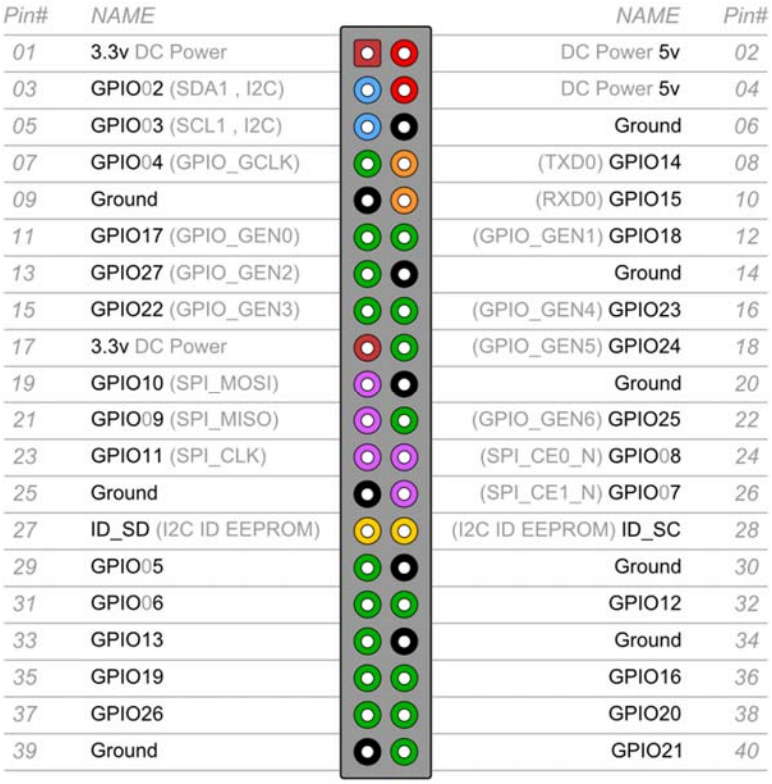
Lanzada en febrero de 2012, este dispositivo se ha hecho un hueco en el mercado de ordenadores de placa simple, *SBC* (*Single Board Computer*). Raspberry Pi (Figura 6) es un ordenador completo embebido en una sola placa y a precio reducido. Puede ser utilizado como ordenador personal, en la enseñanza, o como sistema de proceso en proyectos hardware/software. No incluye periféricos como ratón, teclado, aunque en este proyecto no son necesarios.

Figura 6: Raspberry Pi 2

Respecto al hardware ofrecido:

- SoC Broadcom BCM2836 32 bits:
    - Procesador ARM Cortex-A7 de 4 núcleos, 900 MHz.
    - Gráficos VideoCore IV 3D.
- 1 GB RAM.
- 40 pines GPIO.
- Ethernet 100BaseT.
- 4 puertos USB.
- Puerto HDMI.
- Jack para audio/video analógico.
- Interfaz para cámara (CSI).
- Interfaz para display (DSI).

Con estas características cubrimos completamente los posibles requisitos de rendimiento de la aplicación, tanto por potencia de procesamiento como por disponibilidad de memoria. Una de las más importantes es la cantidad de pines GPIO que ofrece (Figura 7). Esto permite la conectividad de numerosos dispositivos. En nuestro caso solo utilizaremos un GPS a través de los pines de la UART, cuatro botones (4 pines y tierra) y dos pines para el bus I2C que conecta la pantalla. Los restantes ofrecen grandes posibilidades de ampliación hardware de este proyecto.



Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figura 7: GPIO en Raspberry Pi

Respecto al software, Raspberry Pi nos permite la instalación de un sistema operativo completo. En el siguiente capítulo se expondrá brevemente el sistema operativo utilizado para este proyecto: GNU/Linux en su distribución Raspbian [5].

Una de las desventajas de Raspberry Pi es que no posee un reloj de tiempo real (*RTC, Real Time Clock*). Esto hace que cada vez que se apaga, la referencia horaria se pierde. Puesto que en este proyecto se hace uso de un GPS, la referencia horaria puede ser obtenida a través del sistema de posicionamiento. Por tanto, la falta de *RTC* no es un problema para este proyecto, pero ha de ser tenido en cuenta en otros proyectos que sí puedan necesitar este elemento.

4.2 Módulo GPS: Adafruit Ultimate GPS hat for Raspberry Pi

Como módulo GPS se ha decidido utilizar el producto Adafruit Ultimate GPS hat [6](Figura 8). Este *hat* dispone de un chip GPS con capacidad para 66 canales, una sensibilidad de -165 dBm, y una frecuencia de actualización máxima de 10 Hz. La antena integrada es adecuada para situaciones en exteriores, e incluye también un conector u.FL para conectar una antena externa. La placa dispone de un LED que indica el estado del GPS, y un *RTC* el cual mantiene los datos de fecha mediante una batería tipo CR1220. La conectividad se realiza mediante puerto serie, por lo que es necesario realizar una pequeña configuración en Raspberry Pi para la comunicación.

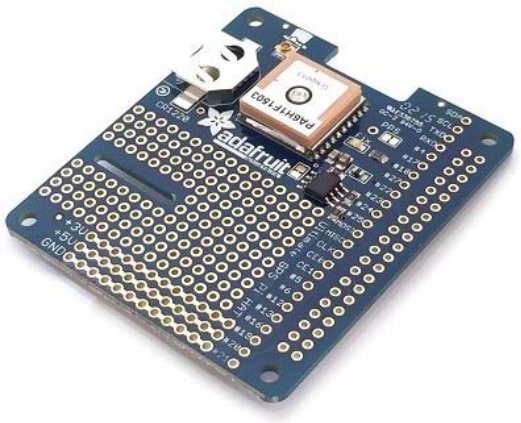


Figura 8: Adafruit Ultimate GPS hat

En este apartado no hay mucha variedad en el mercado disponible para pequeños proyectos. Por tanto, el factor decisivo a la hora de escoger este módulo ha sido el factor forma. Debido a que es diseño tipo *hat*, se puede conectar fácilmente en el puerto *GPIO* de Raspberry Pi sin necesidad de cableado. Además, incluye un área de prototipado donde poder soldar otros elementos hardware. Este es el motivo principal de su elección, pues ese espacio es adecuado para la pantalla.

La única desventaja de este modelo es que no se puede utilizar con Raspberry Pi 3, debido a que la conectividad se realiza por la misma UART que Raspberry Pi 3 tiene asignada a la conectividad Bluetooth. Por tanto, en caso de utilizar una Raspberry Pi 3, se recomienda adquirir otro modelo que no venga soldado en un *hat*.

4.3 Pantalla: Adafruit FeatherWing OLED 128x32

La selección de este hardware ha sido la más complicada debido a requerimientos de espacio y control. La pantalla debía adecuarse al espacio disponible en el *hat* GPS, además de incluir botones para el control de la aplicación. El único modelo encontrado que cumple con estas restricciones es la pantalla FeatherWing OLED 128x32 de Adafruit [7] (Figura 9).

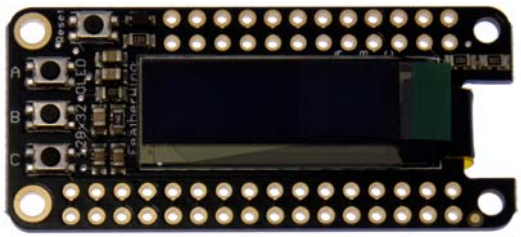


Figura 9: Adafruit FeatherWing OLED

El display está gobernado por un controlador SSD1306 (situado bajo la pegatina a la derecha de la pantalla) que permite gran versatilidad a la hora de mostrar contenido. La matriz de pixeles tiene una resolución de 128x32, con una diagonal de 2,54 cm. Puede parecer pequeño, pero esto añade portabilidad al proyecto. El módulo dispone además de 4 botones (A, B, C, RST), los cuales serán utilizados para el control de la aplicación. La conectividad se realiza mediante bus I2C, lo que permite añadir otros dispositivos que utilicen este bus en futuras ampliaciones.

4.4 Adaptador Bluetooth: Asus BT400

Este hardware permite la conectividad entre Raspberry Pi 2 y el adaptador OBD. El dispositivo escogido, Asus BT400 [8] (Figura 10), cumple perfectamente los requisitos, a saber:

- Dotar a Raspberry Pi 2 de la misma tecnología de comunicación que el adaptador OBD (Wifi o Bluetooth).
- Compatibilidad con el sistema operativo GNU/Linux Raspbian.

Figura 10: Asus BT400

Respecto al primero, el dispositivo Bluetooth Asus BT400 soporta la versión 2.1 del protocolo, la cual es utilizada por el adaptador ODB. A mayores, este dispositivo ofrece compatibilidad con Bluetooth LE (Bluetooth *Low Energy*), ofreciendo posibilidades de ampliación de este proyecto.

Respecto a la compatibilidad con el sistema operativo Raspbian, este dispositivo no necesita instalación de drivers, pues es reconocido de inmediato.

4.5 Escáner OBD: OBDLink LX Bluetooth

Este es el dispositivo encargado de hacer de interfaz entre Raspberry Pi 2 y el vehículo. En el mercado existen varios modelos, dependiendo de la conectividad utilizada entre el dispositivo y la aplicación, los protocolos soportados para la comunicación con el vehículo, restricciones de memoria, intérprete utilizado, etc. Dependiendo de la interfaz entre el adaptador y la aplicación, encontramos tres grupos:

- Wifi
- Bluetooth
- USB

Dependiendo del intérprete utilizado, otros tres grupos:

- ELM327
- STN11XX
- Copias

Dependiendo los protocolos soportados en la interfaz adaptador-vehículo, nos encontramos con:

- Soporte para protocolos estandarizados
- Soporte para protocolos propietarios

Según el modelo de vehículo, el conector OBD puede ser más o menos accesible. Para evitar problemas y molestias con la longitud del cable, se decidió buscar un dispositivo que utilizara transmisión inalámbrica para la interfaz con la aplicación. Por tanto, quedan descartados todos aquellos que no utilicen Bluetooth o Wifi.

Analizando el tipo de intérprete utilizado, tenemos dos grandes protagonistas en el mercado: el chip ELM327 y el chip STN11XX. ELM327 fue el primero en el mercado en ofrecer características completas para la interfaz intérprete-vehículo. Permite configuración utilizando comandos tipo AT, además de soporte para todos los protocolos estandarizados utilizados por los vehículos. En el mercado se encuentra con interfaz intérprete-aplicación tanto inalámbrica como alámbrica. La principal desventaja de este intérprete no se encuentra en el propio chip, si no en un descuido por parte de los diseñadores. En las primeras versiones liberaron un hardware sin protección, lo que hizo que aparecieran multitud de copias de baja calidad y con errores en el mercado. Como resultado, no es seguro adquirir un dispositivo que utilice este intérprete, pues es una "lotería" que sea una copia defectuosa o una copia utilizando la patente original. Esto nos lleva al segundo intérprete, el chip STN11XX [9]. Este es un intérprete OBD a UART que soporta todos los protocolos del estándar OBD-II en uso, además de varios protocolos propietarios. Entre sus principales características:

- Soporta el conjunto completo de comandos AT del intérprete ELM327: esto ofrece compatibilidad con todas las aplicaciones desarrolladas para el ELM.
- Soporta el conjunto extendido de comandos ST: puesto que el microcontrolador posee más características, este conjunto de comandos nos permite control total sobre el intérprete.
- Tasas de transferencia intérprete-aplicación desde 38 bps hasta 10 Mbps.
- Soporte para protocolos OBD-II estandarizados:

- ISO 15765-4 (CAN)
- ISO 14230-4 (KWP2000)
- ISO 9141-2 (vehículos Chrysler)
- J1820 VPW (vehículos GM)
- J1850 PWM (vehículos Ford)

Este intérprete es fabricado por una sola empresa. Esto evita los problemas de copias mencionados anteriormente. Además, la empresa tiene modelos tanto Wifi como Bluetooth.

El modelo escogido, OBDLink LX Bluetooth [10] (Figura 11), utiliza Bluetooth para la interfaz escáner-aplicación.

Figura 11: OBDLink LX Bluetooth

Además de todo lo mencionado, dispone de un pequeño botón que nos permite activar y desactivar la conectividad Bluetooth evitando así que dispositivos no deseados se conecten a él.

CAPÍTULO 5. IMPLEMENTACIÓN DE HARDWARE

En el capítulo anterior hemos podido ver un listado de componentes hardware que van a ser utilizados en este proyecto. En el presente capítulo veremos los trabajos necesarios para posibilitar su uso. Estos trabajos se dividen en dos partes:

- Soldadura.
- Impresión 3D de la caja.

En el primer apartado se describirá el proceso de soldadura de la pantalla en el espacio de prototipado del *hat* GPS, además de los materiales necesarios para ello, problemas encontrados durante el proceso, y posibles mejoras. Respecto a la impresión 3D de la caja, no será posible mostrar todo el proceso pues el autor no ha sido responsable. Aun así se dará una breve explicación del trabajo realizado.

5.1 Soldadura

Como se mencionó en el capítulo anterior, el módulo GPS fue escogido debido al espacio disponible que presenta para prototipado. Este espacio ha sido utilizado para soldar la pantalla. Debido a la posición final de la pantalla en el espacio de prototipado, los pines utilizados para el bus de comunicación I2C quedan sin posibilidad de soldadura. Ha sido necesario trasladar la funcionalidad de esos pines a otros que puedan ser soldados. Para ello, se han soldado dos cables en los respectivos pines, como muestra la figura:

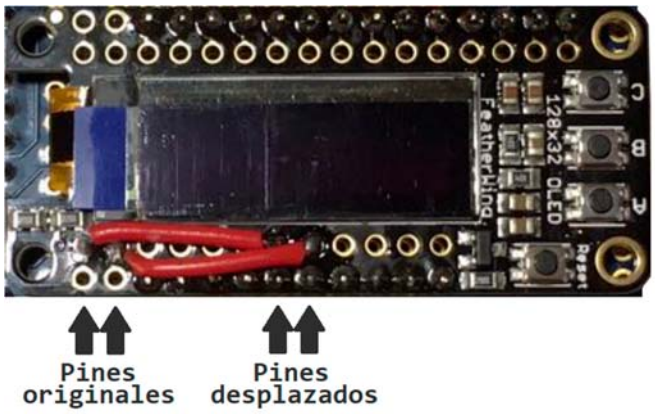


Figura 12: Desplazamiento de pines I2C

Numerando las columnas de izquierda a derecha, la primera columna (señal SDA) ha sido desplazada a la columna 7, mientras que la segunda columna (señal SDL) ha sido desplazada a la columna 8. La razón se debe a que los pines correspondientes a los botones A, B y C son las columnas 3, 4 y 5. Se ha dejado la columna 6 como separación entre sección de botones y sección de bus I2C. Una vez realizado este desplazamiento, la pantalla ha sido soldada en el espacio de prototipado de la placa GPS. El resultado se puede observar en la siguiente figura:

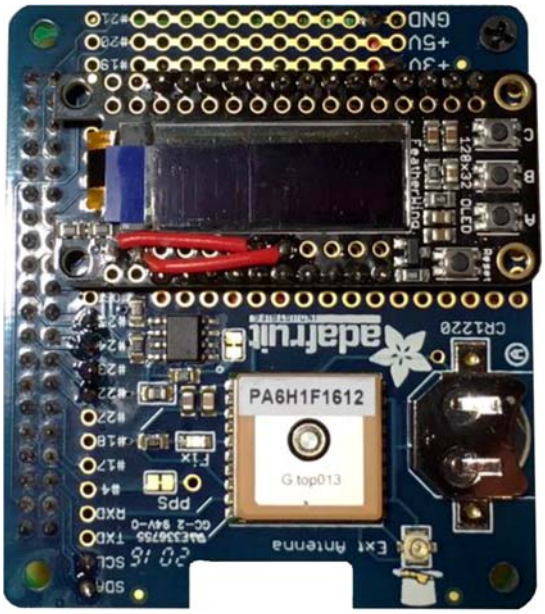


Figura 13: Pantalla soldada al módulo GPS

A continuación, es necesario soldar los pines de la pantalla a sus correspondientes conexiones en la placa. El *hat* GPS dispone de puntos de soldadura conectados al socket GPIO incluido. La correspondencia es la siguiente (la posición y numeración de columnas se hace respecto a la figura anterior):

- Alimentación: Conjunto superior de pines, columna número 15, soldado a raíl 3V+.
- Tierra: Conjunto superior de pines, columna número 13, soldado al raíl GND.
- Botones A, B y C: Conjunto inferior de pines, columnas número 3, 4 y 5 respectivamente, soldados a los pines #22, #23 y #24.
- Botón RST: Conjunto superior de pines, columna número 16, soldado al pin #25.
- Señal SDA: Conjunto inferior de pines, columna número 7, soldado al pin SDA.
- Señal SCL: Conjunto inferior de pines, columna número 8, soldado al pin SCL.

Las soldaduras se realizan en la parte posterior de la placa GPS, como muestra la siguiente figura:

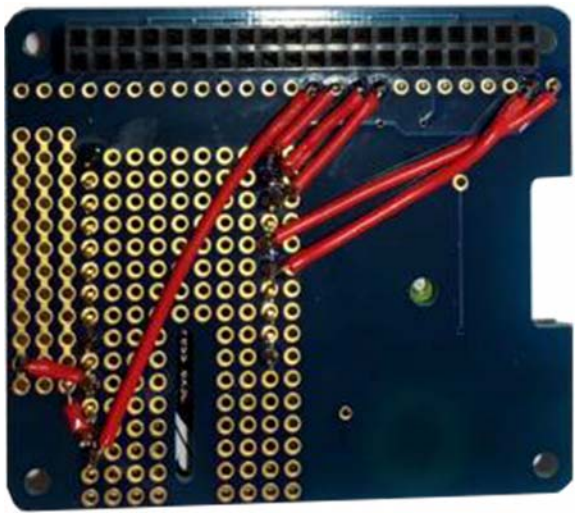


Figura 14: Cableado de la pantalla

Una vez soldadas las conexiones, podemos colocar el *hat* GPS en el conector GPIO de Raspberry Pi. Utilizamos dos tetones de 5 mm en una de las esquinas opuestas al conector

para mantener la parte alejada del conector lo más estable posible. La siguiente figura muestra el resultado final (se pueden ver los tetones en la esquina inferior derecha):

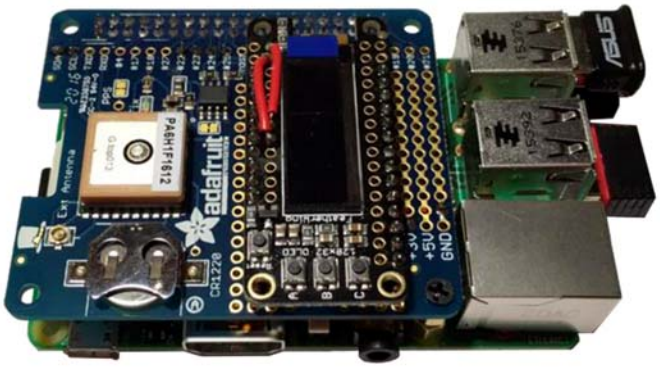


Figura 15: Unión del módulo GPS con Raspberry Pi

Como último paso, colocamos la batería en el soporte para la misma. Esta batería es responsable de mantener el almanaque GPS, el cual puede tardar hasta 12,5 minutos en ser recibido por completo si no disponemos de batería.

5.2 Impresión 3D de la caja

Este proceso no ha podido ser documentado exhaustivamente pues el autor no ha sido responsable de la impresión 3D.

Debido a la fragilidad de la pantalla, y con objetivo de proteger a ésta y al resto de componentes del *hat* GPS, se ha diseñado una caja mediante impresión en 3D. Como modelo inicial, se utilizó un diseño genérico descargado desde Thingverse [11]

Puesto que Raspberry Pi incluye el *hat* GPS y la pantalla, fue necesario realizar algunas modificaciones al modelo original, listadas a continuación:

- Agujero para la antena GPS: de 1 cm de lado, este agujero permite la correcta recepción de la señal GPS por parte de la antena.

- Agujero para la pantalla y botonera: con unas medidas de 2,75 cm de ancho y 5,2 cm de largo, este agujero permite el acceso a los botones y la visualización de la pantalla.
- Altura de la caja: fue necesario aumentar la altura 0,5 cm para albergar todo el conjunto.

El resultado final puede verse en la siguiente figura:

Figura 16: Resultado final hardware

Como se puede observar, aún es posible ajustar la ventana para la pantalla, haciendo que el controlador de la pantalla (el circuito integrado SSD1306, situado a la derecha) quede tapado, evitando así posibles roturas de esta parte de la pantalla (lo cual le sucedió al autor).

CAPÍTULO 6. CONFIGURACIÓN DE SOFTWARE

Este capítulo pretende ser una descripción de los pasos realizados para configurar el entorno software.

- Configuración inicial de Raspberry Pi
- Configuración del puerto serie para el módulo GPS
- Configuración de las herramientas de *cross-compiling*

Si hemos conectado la placa de GPS, es necesario desconectarlo en el momento del primer arranque, antes de la configuración.

6.1 Configuración de Raspberry Pi

El sistema operativo escogido para Raspberry Pi es *Raspbian*, en su versión *Stretch Lite*. Su descarga e instalación no son objetivo de este proyecto, pudiendo encontrar una guía en su página oficial [12]. Es importante la selección de la versión *Lite,* pues al tratarse de una aplicación *headless* no es necesario que el sistema operativo incluya un sistema de escritorio, además de diversas aplicaciones preinstaladas.

6.1.1 Dirección IP y habilitación del servicio ssh

Desde noviembre de 2016, el servicio *ssh* está deshabilitado de forma predeterminada en las nuevas instalaciones de Raspbian. Para activarlo, es necesario crear un fichero (sin contenido ni extensión) llamado *ssh*, en la raíz de la partición *BOOT* de la instalación. Esto será suficiente para habilitar el servicio.

En este proyecto se asigna una dirección IP fija a la interfaz Ethernet. De esta forma será posible acceder a Raspberry Pi a través de esta interfaz sin necesidad de disponer de un

servidor DHCP en la red (pudiendo crear redes punto a punto). La configuración de esta dirección se realiza en dos etapas: antes del primer arranque, y después del primer arranque.

Antes del primer arranque se realiza una configuración estática mediante el fichero *cmdline.txt*, añadiendo el parámetro *ip=192.68.1.30*. De esta forma, en el primer arranque ya dispondremos de interfaz de red configurada.

Una vez el sistema arranca, borramos el parámetro mencionado anteriormente y configuramos la dirección IP utilizando los ficheros de configuración correspondientes:

- En el fichero */etc/network/interfaces*, añadimos las siguientes líneas:

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface eth0 inet manual
```

- En el fichero */etc/dhcpcd.conf*, añadimos las siguientes líneas:

```
nodhcp

interface eth0
static ip_address=192.168.1.30/24
static routers=192.168.1.1/24
static domain_name_servers=8.8.8.8
```

Esta configuración en dos pasos se debe a que el fichero *cmdline.txt* configura la dirección IP pasándosela al *kernel* directamente en el momento del arranque. Esto hace que el *kernel* no termine de arrancar hasta que no se asigna una dirección IP utilizando los servicios dedicados a ello, o hasta que termina un temporizador, retrasando en hasta 120 segundos el arranque.

6.1.2 Configuración general de Raspberry Pi

Una vez Raspberry Pi ha iniciado y disponemos de conexión a Internet, lo primero es actualizar el sistema operativo. Para ello, utilizamos el comando *sudo apt-get upgrade*, el cual realizará todo el proceso automáticamente.

Finalizada la actualización de paquetes, podremos configurar Raspberry Pi mediante la utilidad *sudo raspi-config* (Figura 17). En ella realizaremos los siguientes cambios:

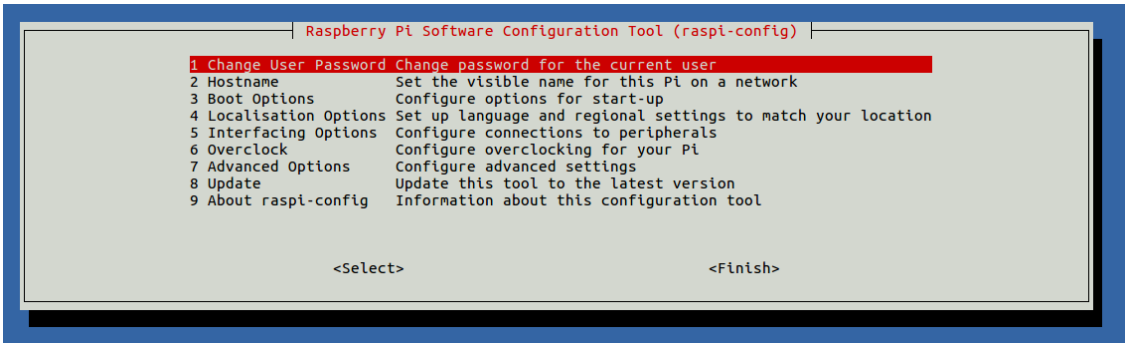


Figura 17: Utilidad de configuración *raspi-config*

- *Localisation Options -> Change timezone*: Seleccionamos *Europe, Madrid*.
- *Interfacing Options:*
  - *Camera: disable.*
  - *ssh: enable.*
  - *VNC: disable.*
  - *SPI: disable.*
  - *I2C: enable.*
  - *Serial: enable.*
  - *1-wire: disable.*
  - *Remote GPIO: disable.*
- *Overclock: disable.*
- *Advanced options:*
  - *Expand filesystem:* el sistema se ha expandido automáticamente en el momento de la instalación gracias a la ejecución del *script* llamado *expander*, incluído como atributo en el fichero de configuración

cmdline.txt. De esta forma, disponemos del espacio completo de la tarjeta SD.

- *GL driver*: *disabled.*

6.1.3 Configuración del puerto serie para el módulo GPS

Inicialmente, Raspberry Pi utiliza los pines de comunicación serie para la conexión con un terminal. El módulo GPS utiliza los mismos pines para la transmisión de los datos recibidos, por lo que se hace necesario deshabilitar el terminal en ellos. Para ello, abrimos la utilidad *raspi-config* y modificamos la siguiente opción:

- *Interfacing options*
  - *Serial*
    - *Login shell over serial*: No
    - *Serial port hardware enabled*: Yes

Apagamos Raspberry Pi y conectamos el módulo GPS al GPIO. Al encender el sistema, veremos que el led *fix* (en el módulo GPS, en rojo) parpadea cada segundo. Esto se debe a que el GPS aún no ha obtenido satélites para determinar la posición. Al ser la primera vez, puede tardar hasta 12.5 minutos en obtener todos los datos necesarios, aunque una vez obtenidos estos serán mantenidos en memoria gracias a la batería externa.

En el momento en el que el módulo GPS obtenga conexión con los satélites necesarios, entonces el led *fix* parpadeará cada 10 segundos.

6.1.4 Inicio de la aplicación RPIDDA como servicio del sistema

Una vez termina de arrancar todo el sistema operativo, *systemd* es el encargado de arrancar la aplicación como un servicio más. Este servicio se ha configurado teniendo en cuenta dos características:

- El servicio es un servicio de usuario (y no de sistema).
- El servicio es reiniciado automáticamente por *systemd*.

La primera característica significa que *systemd* esperará a que todos los servicios del sistema hayan arrancado, y solo entonces intentará ejecutar la aplicación.

La segunda característica implica que *systemd* actuará como *watchdog* de nuestra aplicación. Si detecta que esta ha terminado con algún código de error, entonces *systemd* reiniciará el servicio automáticamente después de cinco segundos. Esto se utiliza en caso de que la aplicación sufra un error del que no se puede recuperar. Puede verse la utilización de esta característica en el diagrama de flujo, marcada con los caminos de *Error*.

Para configurar el servicio, se crea el siguiente fichero:

```
sudo touch /etc/systemd/system/rpidda.service
```

El contenido del fichero ha de ser el siguiente:

```
[Unit]
Description=rpidda service
After=multi-user.target

[Service]
Type=simple
ExecStart=/home/pi/rpidda/rpidda
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Una vez creado el fichero que describe el servicio, activamos el servicio con el comando:

```
sudo systemctl enable rpidda.service
```

6.2 Instalación en Raspberry Pi de librerías necesarias

6.2.1 wiringPi

wiringPi es una librería escrita en C para Raspberry Pi, la cual permite controlar los pines GPIO de una forma sencilla. Para instalar *wiringPi*, podemos seguir estos pasos:

- Clonar el proyecto desde Github:

```
git clone git://git.drogon.net/wiringPi
```

- Acceder al directorio del Proyecto y ejecutar el commando de instalación

```
cd wiringpi
./build
```

Es necesario enlazar la librería con la opción *-lwiringpi*

6.2.2 Librería de desarrollo de Bluetooth

Esta librería es necesaria para poder controlar las funciones Bluetooth como conexión a dispositivos o escaneo de los mismos. Para instalar las librerías de desarrollo de Bluetooth:

```
sudo apt-get install libbluetooth-dev
```

Es necesario enlazar la librería con la opción *-lbluetooth*

6.3 Elección del sistema operativo para el entorno de desarrollo

Para el proceso de selección del sistema operativo que utilizará el sistema de desarrollo, se han tenido en cuenta dos condiciones:

- Este proyecto se basa principalmente en hardware y software *open-source*. Por tanto, el sistema operativo deberá pertenecer a aquellos en los que no se necesita el pago de una licencia para su uso.

- Debido a la necesidad de utilizar herramientas de *cross-compiling*, se utilizará un sistema operativo que facilite la tarea de instalación y configuración de estas herramientas.

Como resultado de estas condiciones, el sistema operativo escogido para el entorno de desarrollo será una distribución GNU/Linux, no habiendo limitación en cuanto al tipo de distribución. En este proyecto se utiliza la distribución Ubuntu en su versión 16.04 LTS. La elección de un sistema operativo GNU/Linux hace que los costes asociados al desarrollo se minimicen, ya que no hay necesidad de adquirir un sistema operativo propietario, como pueda ser Microsoft Windows o MacOS. La instalación de esta distribución no se explicará en este documento, pero puede encontrarse una guía en la página web oficial de Ubuntu [13].

6.4 Instalación de las herramientas de *cross-compiling*

Aunque Raspberry Pi pueda utilizarse como un completo y funcional entorno de desarrollo, esto no significa que haya que utilizar la propia plataforma de destino del software para desarrollar ese software. Por diversas razones técnicas y de comodidad, en este proyecto se ha decidido hacer una separación entre el sistema objetivo para el que se desarrolla el software y el sistema utilizado para el desarrollo del software. Entre las razones para escoger este esquema, la principal es la independencia física para desarrollar el software. Siguiendo este planteamiento, es posible compilar el código en el sistema de desarrollo, sin necesidad de tener una Raspberry Pi para ello.

Una limitación que surge de este contexto es la arquitectura hardware. En este caso, Raspberry Pi 2 utiliza una arquitectura ARMv7, mientras que el sistema de desarrollo utiliza una arquitectura amd64 (Intel x86-64). Por tanto, debido a esta diferencia, hay que utilizar herramientas de compilación cruzada o *cross-compiling*.

Para instalar dichas herramientas, ejecutamos el siguiente comando:

```
apt-get install crossbuild-essential-armhf.
```

Esto permitirá utilizar el compilador *gcc* (*arm-linux-gnueabihf-gcc)* para generar binarios de arquitectura ARM.

6.5 Copia del *sysroot* de Raspberry al sistema de desarrollo

Para realizar *cross-compiling* desde *x86* a ARM es necesario disponer de las librerías correspondientes a la arquitectura hardware para la que se desarrolla el software. Puesto que se han descargado en Raspberry Pi los paquetes necesarios, simplemente hay que copiar las carpetas que contienen las librerías (o *sysroot*) desde Raspberry Pi al entorno de desarrollo. Para ello utilizamos los siguientes comandos en el sistema de desarrollo:

```
mkdir ~/rpisysroot
cd ~/rpisysroot
mkdir usr lib
scp -r rpi:/usr ./usr
scp -r rpi:/lib ./lib
```

En este caso, el comando *scp* hace referencia a Raspberry pi a través de un alias, *rpi*.

Para utilizar este nuevo *sysroot*, añadimos en las opciones del *linker* el siguiente parámetro:

```
--sysroot=~/rpisysroot/
```

CAPÍTULO 7. APLICACIÓN: RASPBERRY PI DRIVING DATA ACQUISITION

Como actividad principal de este trabajo se ha desarrollado una nueva aplicación, *Raspberry Pi Driving Data Acquisition* (RPIDDA). La estrategia para describir el funcionamiento será presentar diagramas de flujo, a partir de los cuales se irá desglosando cada una de las acciones. Primeramente, se expondrá una breve justificación para la creación de esta aplicación. Como segundo apartado, se explicará el conjunto de ficheros de configuración de los que se compone la aplicación, con sus tipos de datos y formato permitido. A continuación, se presentarán los diagramas de flujo de la aplicación, y a partir de estos se irán desglosando las partes de forma independiente.

7.1 Justificación

Existen multitud de aplicaciones dedicadas a la obtención de datos del vehículo utilizando el sistema OBD, tanto para sistemas operativos Microsoft Windows, Android, iOS, y Linux. Varios requisitos limitan la elección:

- La aplicación ha de poder ser ejecutada en la plataforma escogida como sistema de proceso, Raspberry Pi, por lo que se necesita software compilado para arquitectura ARM.
- Ha de ser ligera y no necesitar de interfaz gráfica de usuario.
- Posibilidad de utilizar software open source.

Lo anterior hace que la lista de opciones disminuya drásticamente. Una de las pocas alternativas es PyOBD [14]

Si bien esta aplicación cumple todos los requisitos mencionados anteriormente, no cumple uno de preferencia del autor: estar desarrollada en C. Por ello, como objetivo de este proyecto se decidió desarrollar una aplicación en C que permitiera la obtención de datos

utilizando el sistema OBD del vehículo. Como características de esta aplicación, las siguientes:

- Lenguaje de desarrollo: C
- Permite configurar una lista de usuarios, vehículos y PIDs deseados, que serán añadidos a la aplicación en el momento de su ejecución.
- Mínima configuración.
- Fácil utilización: una vez conectada a la alimentación del vehículo, la aplicación se ejecuta automáticamente. Solo es necesario seleccionar vehículo y usuario, y la aplicación comenzará a obtener datos.
- Posibilidad de solicitar al vehículo un listado de PIDs disponibles. De esta forma, antes de iniciar la sesión de captura podemos configurar los PIDs correctamente.
- Almacenamiento de los datos en una base de datos SQLite, que permita facilidad de acceso en los estudios posteriores.

7.2 Ficheros de configuración

La aplicación permite su configuración a través de ficheros de texto. La ruta donde se almacenan estos ficheros de configuración es */home/pi/rpidda*.

El conjunto de ficheros de configuración se compone de:

- rpidda_config_user_list.cfg
- rpidda_config_vehicle_list.cfg
- rpidda_config_pid_list.cfg
- rpidda_config_bt_remote_device.cfg

7.2.1 rpidda_config_user_list.cfg

Este fichero contiene una lista de usuarios bajo estudio. Para cada usuario, se almacena la siguiente información:

- id: número entero, comenzando en 0, que identifica a cada usuario.
- Nombre: nombre y apellidos del usuario. Admite espacios.
- Fecha de nacimiento: en formato DDMMYYYY.
- Año de obtención del carné de conducir: en formato YYYY.
- Distancia media (en kilómetros) recorrida anualmente: número entero.
- Número de siguiente sesión: comenzando en 1, indica el identificador para la siguiente sesión de este usuario. Se autoincrementa en el momento de comenzar la grabación.

La aplicación no comprueba la validez de los datos, tanto para el contenido como para el formato, por lo que es responsabilidad del usuario introducir unos datos correctos.

En este fichero, todas las líneas que comienzan por el carácter '#' se consideran comentarios, y no se tienen en cuenta a la hora de interpretar las lecturas. Ocurre lo mismo con las líneas que solo incluyen un salto de línea.

Esta información es utilizada para identificar los ficheros de base de datos resultantes de las sesiones de grabación.

7.2.2 rpidda_config_vehicle_list.cfg

Este fichero contiene una lista de vehículos que pueden ser usados en las sesiones. Para cada vehículo, se almacena la siguiente información:

- id: número entero, comenzando en 0, que identifica a cada vehículo.
- Fabricante: nombre del fabricante.
- Modelo: modelo de vehículo.
- Caja de cambios: manual o automático.
- Número de marchas.

Igual que ocurre con el fichero de usuarios, la aplicación no comprueba la validez de los datos introducidos, tanto para el contenido como para el formato. Es responsabilidad de usuario la introducción de unos datos correctos.

En este fichero, todas las líneas que comienzan por el carácter '#' se consideran comentarios, y no se tienen en cuenta a la hora de interpretar las lecturas. Ocurre lo mismo con las líneas que solo incluyen un salto de línea.

Esta información es utilizada para identificar los ficheros de base de datos resultantes de las sesiones de grabación.

7.2.3 rpidda_config_pid_list.cfg

Este fichero almacena el listado de PIDs que se desea solicitar al vehículo. El formato es el siguiente:

- Nombre 1: nombre del atributo que queremos solicitar, tal y como aparecerá en la base de datos.
- PID 1: identificador hexadecimal correspondiente al atributo OBD.
- Nombre 2
- PID 2
- …

Este fichero no permite líneas con comentarios o saltos de línea, por lo que la información va seguida.

Todos los PIDs indicados en este fichero serán solicitados al vehículo. Aquellos que no sean reconocidos, mostrarán el valor '0' en su registro. La aplicación deja al usuario la responsabilidad de interpretar la validez de estos datos.

7.2.4 rpidda_config_bt_remote_device.cfg

Este fichero contiene la información del dispositivo Bluetooth remoto. En este caso, se considera dispositivo remoto al escáner OBD. La información almacenada es la siguiente:

- Nombre del dispositivo OBD
- (OPCIONAL) Dirección Bluetooth del dispositivo.

En el caso de que ambos campos hayan sido indicados, la aplicación intentará conectarse directamente a la dirección Bluetooth. En caso de que solo el nombre del dispositivo haya sido indicado, la aplicación realizará un escaneo Bluetooth para obtener la dirección Bluetooth del dispositivo deseado. Esto puede ser un inconveniente si en el momento del escaneo Bluetooth la aplicación detecta varios dispositivos con el mismo nombre. La dirección obtenida en este caso será la del primer dispositivo Bluetooth detectado, y puede no coincidir con el deseado. Se recomienda utilizar indicar la dirección Bluetooth.

7.3 Ejecución de la aplicación

A continuación, se describe la ejecución de la aplicación. Esta se compone de las siguientes etapas, las cuales serán descritas en detalle en las secciones siguientes:

- Inicialización de *wiringPi*.
- Configuración de los botones.
- Inicialización del display.
- Configuración de usuarios.
- Configuración de vehículos.
- Configuración de PIDs.
- Configuración de la información del dispositivo Bluetooth remoto (escáner OBD).
- Bucle del menú de conexión.
- Bucle del menú principal de la aplicación.
- Escaneo de PIDs soportados por el vehículo.
- Configuración de la sesión.
- Inicio de sesión de captura.

En la siguiente figura puede verse el diagrama de flujo principal:

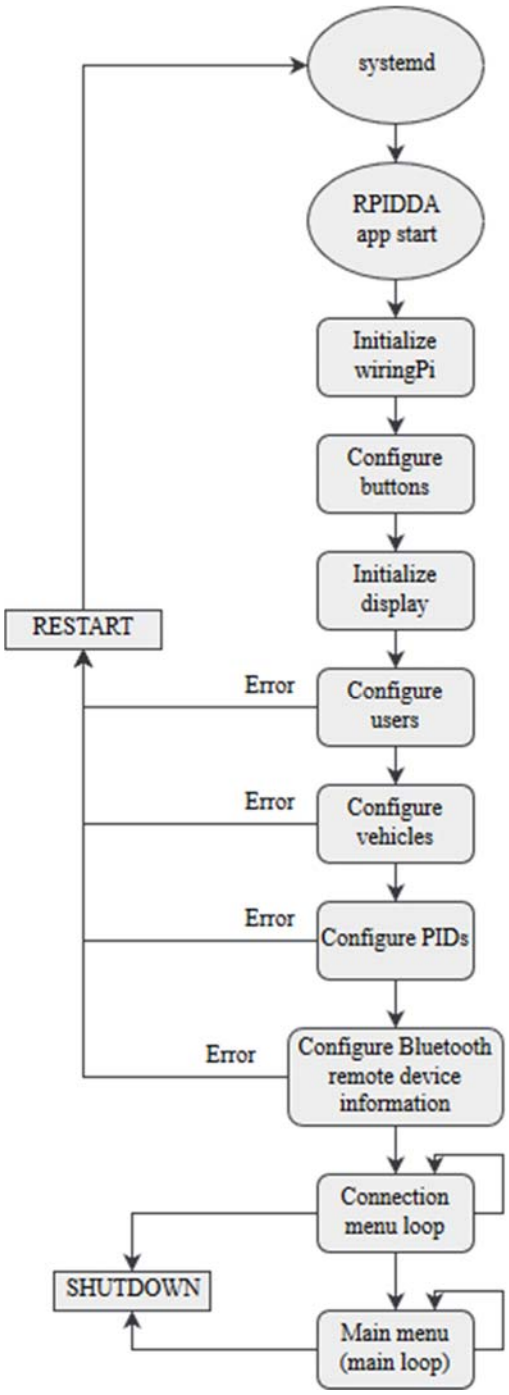


Figura 18: Diagrama de flujo principal

7.3.1 Inicialización de wiringPi

Una vez arranca la aplicación, el primer paso es inicializar *wiringPi*. *wiringPi* [15] es una librería escrita en C, que nos permite utilizar los pines de GPIO de una forma fácil. Esto incluye configurar resistencias internas, utilizar los protocolos de comunicación ofrecidos por los pines, o detectar eventos como flancos de subida o bajada de una señal, etc.

Para realizar la configuración de *wiringPi*, simplemente hacemos una llamada a la función

```
wiringPiSetup();
```

De las funciones de configuración disponibles, esta indica a la librería que utilice el sistema de numeración de pines de *wiringPi*. Respecto a otros sistemas de numeración, como por ejemplo el sistema de *Broadcom*, no existe diferencia más allá de la nomenclatura de los pines.

7.3.2 Configuración de los botones

Para realizar la configuración de los botones, es necesario tener en cuenta la siguiente información para los pines correspondientes:

- Número de pin.
- Dirección del pin.
- Tipo de resistencia asociada al pin.
- Función *callback* para la interrupción.

El esquema de numeración escogido para los pines es el de *wiringPi*. En nuestro caso, los pines corresponden con:

- Botón A: pin 22, wiringPi 3
- Botón B: pin 23, wiringPi 4
- Botón C: pin 24, wiringPi 5
- Botón RST: pin 25, wiringPi 6

Las constantes correspondientes a estos valores se encuentran definidas en el fichero de cabecera *buttons_tfg.h*.

Para todos los pines asociados a los botones, la dirección será *INPUT*, pues vamos a recibir una señal por ellos. Para configurar la dirección del pin, utilizamos la siguiente función (por cada pin):

```
pinMode(BUTTON_A, INPUT);
```

Respecto al tipo de resistencia asociada a los pines, los botones incluidos en el módulo de la pantalla OLED se conectan a tierra cada vez que son pulsados, por lo que es necesario configurar una resistencia de *pull-up*. La función utilizada para ello es la siguiente (por cada pin):

```
pullUpDnControll(BUTTON_A, PUD_UP);
```

El siguiente paso es configurar las interrupciones asociadas a cada pin, y la función *callback* que será ejecutada cuando aparezca la interrupción. Para registrar las interrupciones, utilizamos la siguiente función de la librería *wiringPi* (por cada pin):

```
wiringPiISR(BUTTON_A, INT_EDGE_FALLING, &on_button_a);
```

En esta función, el primer parámetro es el número del pin para el que queremos configurar la interrupción. El segundo parámetro es el flanco que queremos detectar en la señal. En este caso, solamente los flancos de bajada. El tercer parámetro es la función *callback* que será ejecutada en el momento de detectar la interrupción.

7.3.3 Inicialización del display OLED

Para el correcto funcionamiento del display es necesario inicializar el controlador SSD1306, siguiendo una secuencia de comandos bien definida. Esta secuencia está explicada en la documentación del controlador SSD1306. Se ha creado una función específica para esta inicialización:

```
ssd1306Init();
```

Una llamada a esta función es suficiente para tener el display totalmente configurado. Esto permite el uso de funciones de escritura de texto creadas específicamente para este display.

Aunque aún no se hayan utilizado en el programa, se va a explicar el proceso de escritura en el display en esta sección.

Para la escritura en pantalla disponemos de un buffer que mapea cada bit a cada pixel. Este buffer puede ser modificado a través de las funciones de escritura en pantalla, o limpiado completamente a través de la función de limpieza de buffer. La limpieza no es obligatoria, pero de no realizarla habrá que tener en cuenta los contenidos presentes en el buffer.

Para esta explicación, supondremos que tenemos datos en el buffer que queremos eliminar. La escritura de texto en pantalla comienza con la limpieza del buffer de pantalla utilizando una función específica para ello:

```
ssd1306ClearBuffer(char *buffer);
```

Esta función escribe ceros en el buffer indicado como parámetro. Esto nos permite tener varios buffers, cada uno de ellos dedicado a mostrar datos distintos, y controlarlos de forma independiente.

Una vez el buffer está limpio, utilizamos la siguiente función de escritura en pantalla:

```
ssd1306DrawString(uint16_t x, uint16_t y, char* string, charset font, char *buffer);
```

El primer y segundo parámetro indican, respectivamente, la coordenada X e Y donde comenzará la cadena, indicada en el parámetro *string*.

Mediante el parámetro *font* se indica el tipo de fuente a utilizar. Las siguientes fuentes están disponibles en este programa:

- font3x6 (solo letras mayúsculas).
- font5x8.
- font7x8.
- font8x8.

- font8x8Thin.

Cuando el buffer contenga la información que se desea mostrar en pantalla, se hace una llamada a la función

```
ssd1306RefreshScreen(char *buffer);
```

la cual envía el buffer a la pantalla utilizando comunicación I2C.

Internamente, la función *ssd1306DrawString* utiliza la siguiente función para cada carácter de la cadena:

```
ssd1306DrawChar(uint16_t x, uint16_t y, uint8_t c, charset font, char *buffer);
```

En la llamada a la función *ssd1306DrawChar,* la función *ssd1306DrawString* suma una unidad a la anchura de cada carácter para añadir un espacio entre caracteres. La coordenada X final para cada carácter viene dada por la siguiente fórmula:

$$x + char\_number(char\_width + 1)$$

El grafo de un carácter está formado por *n* bytes, donde *n* es la anchura del grafo. Cada byte es una columna, numerados de izquierda a derecha de 0 a n-*1*, según la siguiente matriz:

$$\begin{bmatrix} bit\ 0 & \cdots & bit\ 0 \\ \vdots & \ddots & \vdots \\ bit\ 7 & \cdots & bit\ 7 \end{bmatrix}$$

Byte 0 … Byte *n-1*

Cada bit de cada byte se corresponde con un pixel del grafo. Dentro de la función *ssd1306DrawChar*, la variable *columnIndex* identifica cada byte del grafo. Si el carácter no está disponible, se dibuja un rectángulo blanco como sustituto. Para ello, se escribe en cada byte el valor *0xFF*, que corresponde a todos los bits con valor 1. Si el carácter está disponible, entonces recorre cada byte del grafo, obteniendo su valor.

Cada fuente es un array de tipo *uint8_t*. Cada elemento de ese array es un byte de uno de los grafos. Para obtener los bytes correspondientes, se usa la siguiente formula:

$$numGraph * characterWidth + columnIndex$$

Para obtener *numGraph*, el número de grafo, hay que tener en cuenta que en los caracteres ASCII, el primer carácter que tiene un grafo como tal es el espacio, identificado con el número decimal 32. Las tablas comienzan con el primer carácter siendo el espacio. Por tanto, el carácter ASCII 32 decimal se corresponde con el grafo número 0, etc. La fórmula superior, entonces, quedaría de la siguiente forma:

$$(ascii_{decimal} - 32) * characterWidth + columnIndex$$

Una vez tenemos los bytes que componen el grafo, renderizamos cada pixel. Esto es, recorremos cada byte identificando el estado de sus bits. Indexamos los bytes con *columnIndex* y los bits con *rowIndex*. Para saber el valor de cada bit, se hace una operación AND del byte con una máscara todo cero, exceptuando el bit de interés. Después, se desplaza bit a bit el resultado *n* posiciones hacia la derecha, siendo *n* la posición del bit de interés. Si el bit es 0, se llama a la función *clearPixel*. Si es 1, se llama a la función *drawPixel*.

Las funciones *drawPixel* y *clearPixel* manipulan el buffer cambiando el valor de cada bit. Para ello, utilizan el método *Row-major order*, que permite almacenar valores de un array multidimensional de forma lineal [16] utilizando la siguiente formula:

$$vectorElement = y * matrixWidth + x$$

En nuestro caso, cada 8 pixeles de altura tenemos un nuevo byte. Por tanto, hay que hacer una división entera *y/8* y después multiplicar por el número de pixeles de ancho de la pantalla (número de bytes de ancho). Una vez conocido el byte, hay que averiguar el bit a cambiar. Se puede ver que la división entera indica un número de bytes, y el resto de esa división (operador módulo) indica el bit dentro del byte.

Como último paso se da un valor al bit, utilizando las funciones *clearPixel* y *setPixel*. *clearPixel* hace una operación AND con una máscara todo 1 y un cero en la posición del bit de interés. *setPixel* hace una operación OR de una máscara con todos los bits a 0 y un 1 desplazado al bit de interés.

7.3.4 Configuración de usuarios y vehículos

Los dos siguientes pasos en la inicialización de la aplicación, configuración de usuarios y configuración de vehículos, serán explicados en esta misma sección debido a la similitud del código. Se utilizará la configuración de usuarios para la explicación.

Para configurar los usuarios, la aplicación lee el fichero de configuración llamado *rpidda_config_user_list.cfg*. El contenido de este fichero ha sido explicado anteriormente. La función que realiza la lectura de este fichero es la siguiente:

```
rpidda_config_read_users_file(char *users_file, user_data *user_list);
```

El parámetro *users_file* es la ruta al fichero de configuración. El parámetro *user_list* es un array de estructuras de tipo *user*_data, definida en el fichero *rpidda.h*. Esta estructura contiene la información que identifica a cada usuario. El array que contiene la lista de usuarios tiene un tamaño determinado, para 255 usuarios.

Internamente, la función crea un puntero al fichero indicado como parámetro, y lee secuencialmente cada línea contenida. Utilizando la variable *line*, la función lleva un registro del número de línea leído para cada usuario. Si la información contenida en el fichero sigue el formato indicado, entonces cada línea se corresponderá con un atributo de usuario, el cual será almacenado en la estructura de usuario.

7.3.5 Configuración de PIDs

A través de este paso, la aplicación configura los PIDs que deseamos que solicite al vehículo. La función correspondiente para ello es la siguiente:

```
rpidda_config_read_pid_file();
```

La funcionalidad interna es prácticamente la misma que lo explicado en el apartado anterior. La función crea un puntero al fichero indicado como parámetro, y lee secuencialmente el contenido de este. La información se almacena en un array de estructuras de tipo *obd_datum*, la cual está definida en el fichero de cabecera *obd_tfg.h.*

Además de almacenar en la estructura el modo correspondiente al PID y el propio PID, tanto en hexadecimal como en decimal, también se asigna un puntero a una función de conversión de datos, dependiendo del PID indicado. Esta función es utilizada en el momento de la recepción de los bytes de información OBD. La función que registra estas funciones de conversión, *obd_register_convert_funcion()*, utiliza el PID en decimal para asignar el puntero a la función correspondiente, según una estructura *switch*.

Para añadir soporte para nuevos PIDs, dos pasos son suficientes:

- Crear una nueva función de conversión: estas funciones están situadas en el fichero *obd_tfg.c* y declaradas en el fichero *obd_tfg.h*.
- Añadir un nuevo *case* dentro del *switch* de asignación de funciones de conversión.
-

7.3.6 Configuración de la información del dispositivo Bluetooth remoto

Puesto que el escáner OBD puede cambiar, es necesario permitir que la información de conexión Bluetooth también pueda hacerlo. Para introducir esta información en la aplicación, se utiliza el fichero *rpidda_config_bt_remote_device.cfg*. La función que lee este fichero es:

```
rpidda_config_read_remote_device_file();
```

El funcionamiento interno de esta función es similar a lo visto anteriormente. En este caso, la información es almacenada en una estructura de tipo *bluetooth_device*, definida en el fichero *bluetooth_tfg.h*.

7.3.7 Bucle del menu de conexión

Este bucle se utiliza para conectar la aplicación al escáner OBD, o para apagar Raspberry Pi Existen dos modos de comunicación:

- Comunicación serie, utilizando la función

  `rpidda_connect_obd_serial();`

- Comunicación Bluetooth a través de un socket Bluetooth, utilizando la función

  `rpidda_connect_obd_bluetooth();`

Estos modos son intercambiables realizando la llamada a la función correspondiente dentro del bucle del menú de conexión. Por tanto, es necesario modificar y recompilar el código para utilizar uno u otro modo. A continuación, se explica el funcionamiento del modo de conexión por Bluetooth, el cual es de mayor interés.

El funcionamiento interno de esta función depende de la etapa de configuración del dispositivo Bluetooth remoto. Si no se ha configurado una dirección Bluetooth, entonces esta función realizará un escaneo para intentar detectar el dispositivo a través del nombre. Si lo encuentra, entonces intentará obtener la dirección Bluetooth asociada al dispositivo. En caso fallido, entonces la aplicación mostrará de nuevo el menú de conexión con las opciones de *Conexión* o *Apagado*. En el caso de que se haya configurado una dirección Bluetooth para el adaptador OBD, entonces la función intentará conectarse directamente sin realizar el escaneo previo. Las ventajas de configurar una dirección Bluetooth son el menor tiempo de establecimiento de la conexión, además de evitar posibles problemas debido a la existencia de dos dispositivos con el mismo nombre.

Una vez se dispone de la dirección Bluetooth, la aplicación se conecta al dispositivo mediante la creación de un socket Bluetooth. Aunque esta parte del proyecto ha sido algo complicada debido a la falta de documentación disponible, la mayoría de conexiones Bluetooth se crean de la misma forma, por lo que disponer del código facilita su aplicación en futuros proyectos.

Si la comunicación es satisfactoria, entonces se procede a la configuración del escáner OBD mediante la función

```
obd_device_init();
```

Esta función envía los comandos necesarios para configurar el escáner OBD, según la siguiente lista:

- Reinicio del escáner OBD: Se realiza un reinicio para comenzar con un dispositivo limpio.
- *Echo off*: Se deshabilita la función de eco. Esto hace que el PID o comando enviado no sea repetido mediante eco desde el dispositivo OBD, facilitando la lectura de las respuestas.
- *Spaces off*: Se deshabilitan los espacios en blanco entre bytes de las respuestas. De esta forma, se facilita la lectura de las mismas.
- *Line Feed extra off*: Se deshabilitan los saltos de línea extra añadidos por el dispositivo OBD, por la misma razón que lo anterior.
- Configuración automática de protocolo: este comando hace que el dispositivo OBD detecte automáticamente el protocolo utilizado por el vehículo, sin necesidad de conocer este de antemano.

Todos estos comandos están definidos en el fichero de cabecera *obd_tfg.h*. Existen otros muchos comandos utilizados para poder configurar el escáner OBD de una forma muy precisa, pero en esta aplicación no es necesario configurar ningún aspecto más. Para una referencia más completa, ver manual del dispositivo OBD.

Si la configuración se ha realizado correctamente, entonces el escáner OBD estará listo para su utilización, y la aplicación mostrará el menú principal.

7.3.8 Bucle del menú principal de la aplicación

El menú principal muestra todas las opciones implementadas respecto a la funcionalidad de la aplicación, siguiendo el flujo mostrado en la Figura 19. Esas opciones son:

- Botón A: nueva sesión de captura.
- Botón B: configurar sesión de captura.
- Botón C: escaneo de PIDs soportados.
- Botón RST: apagado del sistema.

Una vez mostrado el menú, la aplicación quedará a la espera de que el usuario pulse un botón. El botón RST indica a la aplicación que el usuario desea apagar el sistema. Esto hará que pasados aproximadamente 10 segundos podamos desconectar de forma segura Raspberry Pi de la alimentación. El resto de botones ejecutan las diversas funcionalidades que serán descritas a continuación.

Cabe destacar que no es posible el inicio de una sesión de captura sin que la aplicación conozca los detalles de dicha sesión. Por tanto, será necesario acceder al menú de configuración de sesión antes de la primera captura de datos para un vehículo y usuario determinados.

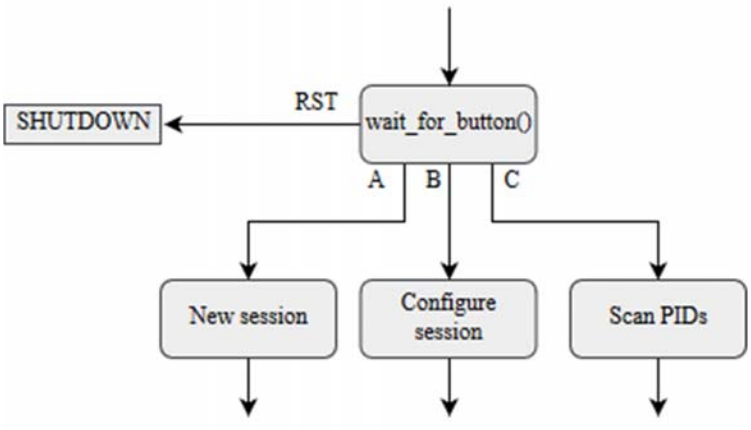


Figura 19: Diagrama de flujo del menú principal

7.3.9 Escaneo de PIDs soportados por el vehículo

Mediante esta función, es posible obtener los PIDs soportados por el vehículo, los cuales serán almacenados en un fichero llamado *rpidda_discovered_pids.txt*. Varias razones se han tenido en cuenta a la hora de decidir almacenar el resultado en un fichero de texto en vez de mostrarlo por pantalla:

- Dependiendo del modelo de vehículo, la lista de PIDs soportados puede variar considerablemente. Considerando que el tamaño de la pantalla no es adecuado para mostrar gran cantidad de texto, esta opción quedaba descartada.
- Es necesario utilizar un sistema externo (otro ordenador) para poder realizar la configuración de la aplicación (PIDs, usuarios…). Por tanto, se puede aprovechar este momento para obtener la información del fichero de PIDs soportados.

Al pulsar el botón C, la aplicación llama a la función

```
rpidda_menu_scan_pids();
```

Esta función es idéntica en comportamiento a la función que configura el dispositivo OBD mediante comandos AT. En vez de comandos, *rpidda_menu_scan_pids()* envía los PIDs 0100, 0120, 0140 y 0160. Estos PIDs hacen que el vehículo responda con los correspondientes siguientes 32 PIDs soportados (recordemos que los PIDs son valores hexadecimales). Por tanto:

- PID 0100: PIDs soportados del 0101 al 011F
- PID 0120: PIDs soportados del 0121 al 013F
- PID 0140: PIDs soportados del 0141 al 015F
- PID 0160: PIDs soportados del 0161 al 017F

Cuando una respuesta correspondiente a estos PIDs es detectada, la función *obd_send_pid()* obtiene el resultado y lo almacena en el fichero mencionado anteriormente.

El proceso de detección de las respuestas es sencillo, pues solo es necesario obtener el tercer y cuarto byte de cada respuesta para conocer el PID que la originó. Una vez se tiene almacenado el valor, se hace una llamada a la función

```
obd_print_supported_pids();
```

Los PIDs soportados son devueltos codificados de forma binaria. Así, cada bit de la respuesta indica si el correspondiente PID es soportado o no, siguiendo el siguiente esquema:

Bit	msb	msb - 1	…	lsb + 1	lsb
PID	01	02	…	1F	20

Tabla 3: Esquema de codificación de PIDs

Para resolver el problema que se nos plantea, se han barajado dos soluciones:

- Utilizar operaciones bit a bit:
- Utilizar una tabla de valores, o *lookup table* de valores binarios equivalentes.

Puesto que el driver SSD1306 utiliza operaciones bit a bit y máscaras binarias, se ha decidido variar un poco la implementación y utilizar una tabla de valores.

Debido a que los bytes que recibimos son caracteres ASCII representando valores hexadecimales, no podemos utilizar el valor binario para obtener los PIDs. Esta función procesa el resultado byte a byte, convirtiendo cada carácter recibido a una secuencia de caracteres igual a su secuencia binaria del valor hexadecimal. Para realizar la conversión, se siguen estos pasos:

- El carácter ASCII (que representa un valor hexadecimal pero no tiene dicho valor), es convertido a su valor hexadecimal representado utilizando la función *strtol()*, indicando una base 16.
- Dicha función devolverá un valor decimal de 0 a 15. Este valor es utilizado como índice en un array de cadenas de caracteres, donde cada elemento representa el valor binario del índice.
- Una vez obtenida la cadena de caracteres correspondiente, se recorre cada elemento de esta cadena (carácter), el cual representa un bit, y se interpreta.

- El resultado se almacena en el fichero de texto.

7.3.10 Selección de información de la sesión

Antes de comenzar una sesión de captura, es necesario indicar el usuario y vehículo a los que corresponderán los datos. Para ello, la aplicación dispone de las funciones, las cuales son llamadas una detrás de otra (Figura 20):

```
rpidda_menu_select_user();
rpidda_menu_select_vehicle();
```

El funcionamiento interno de estas funciones es idéntico y muy sencillo, por lo que solo una de ellas (usuarios) será explicada.

Estas funciones muestran un menú circular con tres entradas, denominadas *previous_user*, *show_user* y *next_user*. La lista de usuarios ha sido obtenida previamente en los pasos de configuración de la aplicación. El usuario, entonces, selecciona la entrada correspondiente utilizando los botones A para subir, C para bajar y B para seleccionar la entrada. La entrada seleccionada es almacenada en la información de la sesión como el usuario correspondiente a la captura de datos.

Una vez seleccionado usuario, se muestra un menú siguiendo el mismo estilo, para la selección de vehículo.

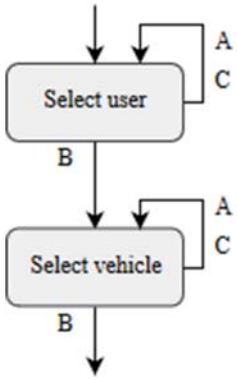


Figura 20: Diagrama de flujo de la configuración de sesión

7.3.11 Inicio de la session de captura

Esta sección pretende explicar la funcionalidad principal de la aplicación: la captura de datos. La captura de datos se compone de tres partes:

- Menú de captura.
- Configuración de captura.
- Bucle de captura.

El diagrama correspondiente al menú de captura se muestra en la Figura 21.

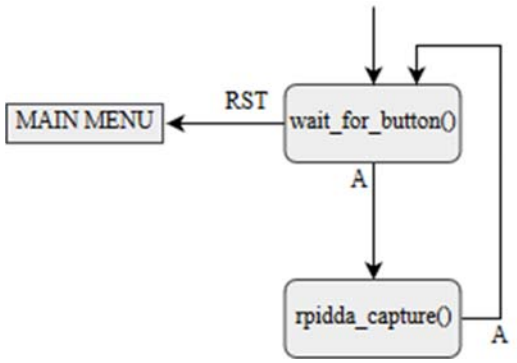


Figura 21: Diagrama de flujo del menú de captura

Este menú ofrece la capacidad de iniciar la captura o volver al menú principal. Además, ofrece capacidad de realizar capturas múltiples.

Si se pulsa el botón RST, la aplicación volverá a mostrar el menú principal, aunque manteniendo la configuración de usuario y vehículo.

Si se pulsa el botón A, la captura de datos dará comienzo. Si durante la captura de datos se vuelve a pulsar el botón A, esa captura será finalizada, pero se dará oportunidad al usuario de comenzar otra en vez de salir al menú principal. Pulsando el botón A, la aplicación llama a la función

```
rpidda_capture();
```

Esta función sigue los diagramas de flujo mostrados en las siguientes figuras (Figura 22 y Figura 23).

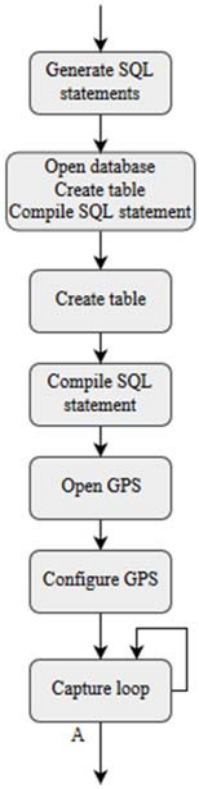


Figura 22: Diagrama de flujo de la captura de datos

La función genera primero las sentencias SQL necesarias para configurar la base de datos para esa captura, además de generar el nombre de la misma. Para ello utiliza las funciones

```
db_generate_statement_create_table();
db_generate_statement_insert();
db_generate_db_name();
```

La primera de ellas genera la sentencia (cadena de caracteres) necesaria para crear la tabla. Esta tabla contendrá las siguientes columnas:

- SAMPLE: integer, primary key, autoincrement.
- HOURS: integer, not null.
- MINUTES: integer, not null.

- SECONDS: integer, not null.
- LATITUDE: real, not null.
- LONGITUDE: real, not null.
- GPS SPEED: real, not null.
- *n* campos: real, not null.

Exceptuando el primer campo, que es automático, los seis siguientes obtienen los datos del GPS. A partir de ahí, los *n* siguientes campos son creados dinámicamente para cada uno de los *n* PIDs configurados para la captura, utilizando los nombres indicados en el fichero de configuración. Todos estos campos son concatenados uno a uno formando una cadena de caracteres.

La segunda función, *db_generate_statement_insert()*, utiliza el mismo método que la primera para formar una cadena de caracteres que será utilizada como sentencia para insertar datos en la base de datos. En este caso, no es necesario indicar un nombre para las columnas, pues en el momento de inserción se indicará el número de columna.

Por último, la función *db_generate_db_name()* genera un nombre para el fichero de base de datos. Este nombre se obtiene a través de la concatenación de la información de la sesión, a saber:

```
<nombre-usuario>_<marca-vehículo>_<modelo-vehículo>_<id-secuencial>
```

Los espacios en el nombre de usuario, marca y modelo de vehículo son sustituidos por el carácter barra baja. El campo *id-secuencial* es un identificador secuencial, distinto para cada sesión perteneciente a un usuario, y el cual es incrementado en el momento de la creación del nombre de la base de datos. No es posible, en este caso, utilizar fecha y hora de creación como nombre del fichero de base de datos. Esto se debe a que Raspberry Pi no posee reloj en tiempo real, por lo que la hora no sería correcta. Tampoco es buena idea utilizar la hora ofrecida por el GPS, pues puede no haber obtenido señal de satélites y por ello disponer de una hora errónea.

Una vez obtenida la información mencionada anteriormente, la aplicación crea y configura la base de datos utilizando las funciones de la API *Sqlite3*.

Como paso final antes del bucle de captura, se abre y configura una conexión serie con el GPS, utilizando las funciones

```
gps_serial_open();
gps_serial_config();
```

La conexión serie utiliza una velocidad de B9600 bits por segundo, con 8 bits de datos y sin bit de paridad. Dicha velocidad de conexión es más que suficiente para recibir una sentencia NMEA cada segundo.

Tras la configuración de la captura, comienza el bucle de captura, el cual se muestra en la siguiente figura:

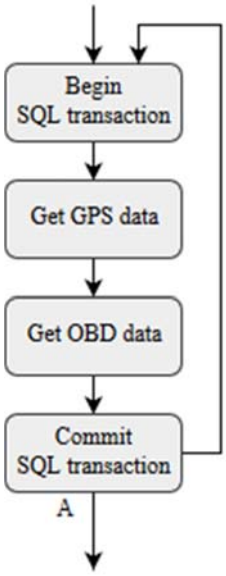


Figura 23: Diagrama de flujo del bucle de captura

El proceso comienza con la creación de una transacción SQL. Cada transacción contendrá por defecto cinco iteraciones del bucle, aunque este número es configurable. Esto significa que los datos no serán almacenados en el fichero de base de datos hasta que no realice ese número determinado de iteraciones. Esto minimiza el riesgo de corrupción y perdida de datos en caso de perder la alimentación durante la escritura de la base de datos.

Los primeros datos que se obtienen son los del GPS, mediante la función

```
gps_get_location();
```

Esta función obtiene las sentencias NMEA de tipo GPRMC, las cuales son las mínimas recomendadas para los datos de tránsito (los únicos interesantes en este proyecto). Una vez almacenados esos datos en su estructura correspondiente, estos son compilados en la sentencia SQL utilizada para insertar datos en la base de datos.

La lectura de la conexión con el GPS es bloqueante, por lo que el proceso no continuará hasta que se reciba una sentencia NMEA completa. Esto dota automáticamente al programa de una sincronización precisa. Puesto que recibimos una sentencia NMEA GPRMC cada segundo, disponemos del tiempo entre sentencias NMEA para solicitar PIDs al vehículo. En la práctica, se ha comprobado que este es tiempo suficiente para realizar entre cinco y seis peticiones.

Las peticiones a través de OBD las realiza la función

```
obd_send_pid()
```

Esta función es llamada una vez por cada PID configurado. Internamente, envía un PID al dispositivo OBD, bloqueándose hasta recibir una respuesta (por lo que no es necesario comprobar la correspondencia entre petición y respuesta). Cuando lee la respuesta a través del socket Bluetooth, obtiene de ella los bytes correspondientes al valor del parámetro solicitado. Este valor es una representación mediante caracteres ASCII del valor hexadecimal del parámetro. El primer paso es hacer una conversión desde esa representación ASCII a decimal, utilizando la función *strtol()* con base 16. El valor resultante es entonces convertido al valor correcto mediante la función de conversión *convert_function()* asignada a cada PID. Estas funciones de conversión siguen las fórmulas indicadas por el estándar para obtener los valores reales de los parámetros.

Al finalizar todo ese proceso, el resultado es compilado en la sentencia SQL de inserción de datos en el fichero de base de datos. Como paso final, se cierra la transacción y los datos son transferidos a la base de datos.

Cada vez que finaliza una transacción, la aplicación hace una lectura rápida del estado de los botones. Si detecta que el botón A ha sido pulsado durante la transacción, entonces la

captura finaliza, cerrando la base de datos y la conexión serie con el GPS. Es entonces cuando se muestra de nuevo el menú de captura, permitiendo al usuario iniciar inmediatamente otra captura de datos.

CAPÍTULO 8. OBTENCIÓN Y ANÁLISIS DE RESULTADOS

En este capítulo se van a exponer los pasos necesarios para extraer la información de los ficheros de base de datos. Además, se mostrará un breve ejemplo utilizando el software Microsoft Excel.

8.1 Obtención de los ficheros de datos

Los ficheros de base de datos creados por la aplicación se almacenan automáticamente en la raíz del sistema, como muestra la Figura 24:

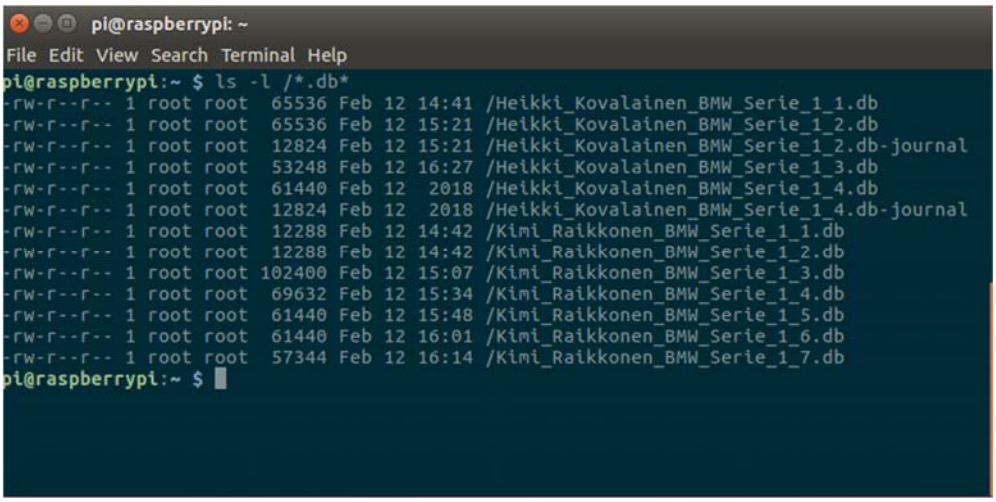


Figura 24: Listado de ficheros de datos

La lista puede contener dos tipos de ficheros:

- Ficheros *.db*: Contienen la información de interés, y los cuales serán abiertos por la herramienta de análisis.
- Ficheros *.db-journal*: Estos ficheros contienen información binaria sobre una transacción de base de datos. Se crean automáticamente durante una transacción, y se eliminan automáticamente cuando la transacción finaliza. Si el sistema se apaga durante una transacción, el fichero queda en el sistema. No son de utilidad.

Como se ha indicado anteriormente, estos ficheros se encuentran en la raíz de Raspberry Pi, por lo que habrá que utilizar otro sistema externo para descargarlos. En el capítulo 6 se muestra como conectarse a Raspberry Pi por red a través de una sesión SSH.

8.2 Configuración de la herramienta de análisis de datos

Como herramienta de análisis de datos se puede utilizar cualquiera de las más conocidas:

- Microsoft Access
- Microsoft Excel
- Suite OpenOffice
- Suite LibreOffice
- IBM SPSS Statistics
- …

En este ejemplo se utiliza la herramienta Microsoft Excel, en su versión Office 2016.

8.2.1 Driver ODBC SQLite3

Por defecto, Microsoft Excel no permite la obtención de datos a partir de ficheros *SQLite3*, por lo que es necesario utilizar un driver ODBC para SQLite3. En la web personal de Christian Werner [17] puede obtenerse dicho driver. Su instalación es sencilla y no requiere explicación alguna.

8.2.2 Extracción de datos

Una vez instalado el driver ODBC, es posible extraer los datos. Para ello, abrimos una nueva hoja de cálculo de Excel y nos dirigimos al menú *Datos->Obtener datos externos->Desde otras fuentes->Desde Microsoft Query* (Figura 25)

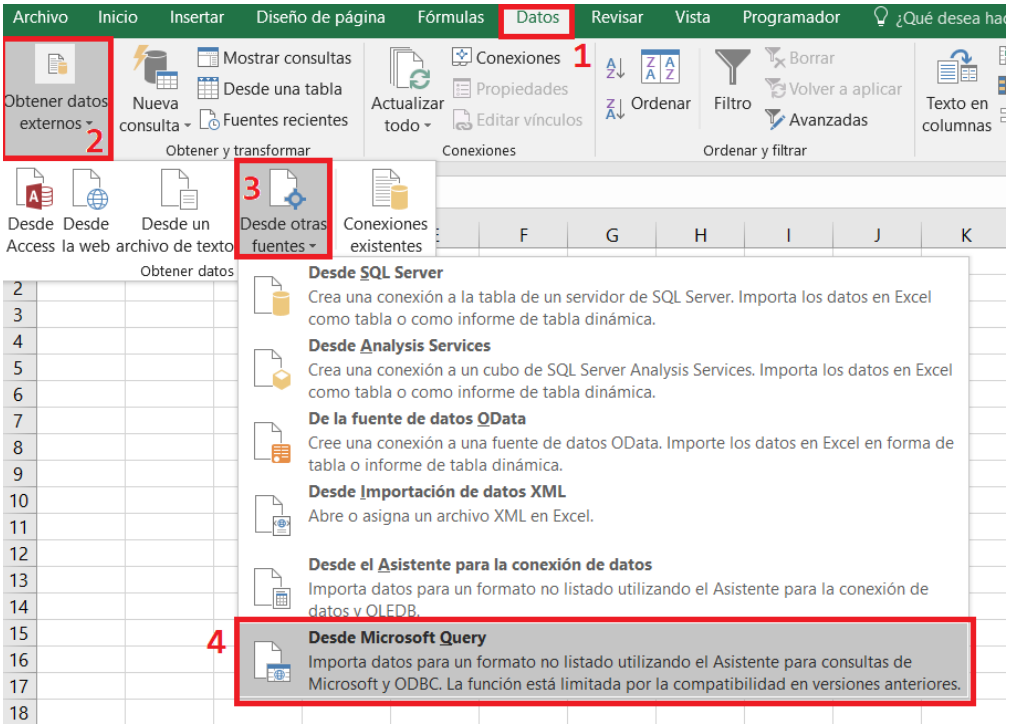


Figura 25: Configuración de Microsoft Excel

Se abrirá una nueva ventana titulada *Elegir origin de datos*. En ella, seleccionamos la opción *SQLite3 Datasource* (Figura 26).

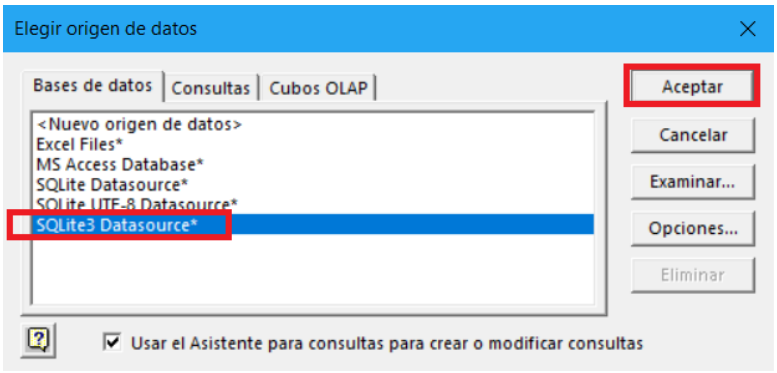


Figura 26: Ventana de seleccion de origen de datos

A continuación, seleccionamos el fichero de base de datos del que deseamos extraer la información ()Figura 27:

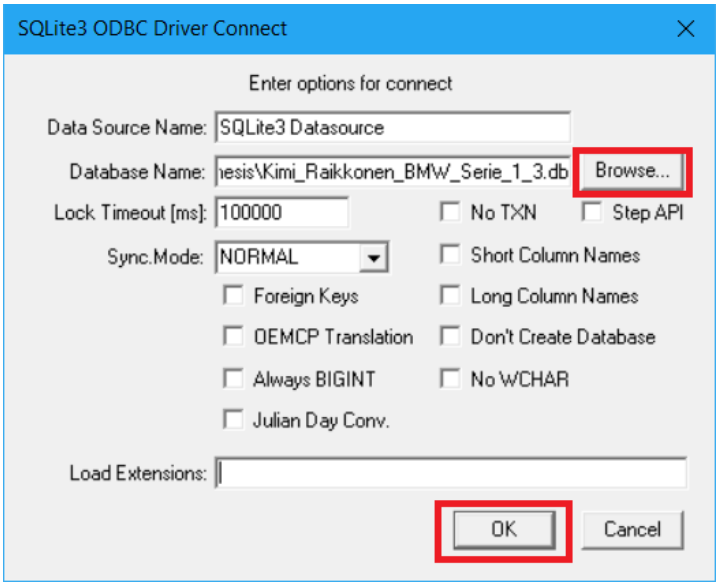


Figura 27: Configuración de conexión del driver ODBC para SQLite3

En este punto se mostrará un mensaje de error que indica "El origen de datos no contiene tablas visibles". Esto se debe a un problema con el driver ODBC. Como resultado, en la siguiente ventana titulada *Asistente para consultas - Elegir columna*, no es posible ver las columnas disponibles. Para solucionarlo, realizar lo siguiente:

- Acceder al menú de opciones.
- Desmarcar una opción.
- Aceptar
- Acceder al menú de opciones.
- Marcar la opción desmarcada anteriormente.
- Aceptar.
-

Ahora las columnas se mostrarán correctamente, como se puede ver en la Figura 28

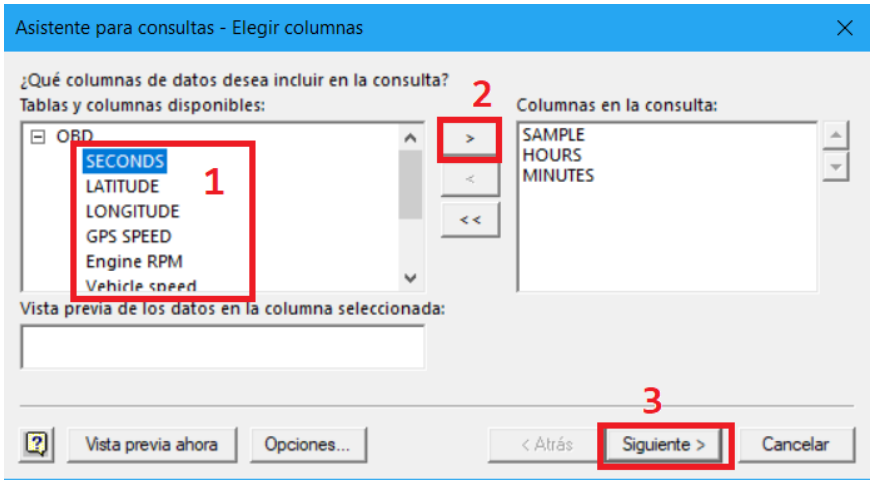


Figura 28: Asistente para la selección de columnas

Una vez seleccionadas las columnas deseadas, pulsamos *Siguiente*. Para las dos siguientes ventanas, *Filtrar datos* y *Criterio de ordenación*, no modificaremos ninguna opción.

Por último, seleccionamos *Devolver datos a Microsoft Excel* (Figura 29), y seleccionamos la columna donde importarlos (Figura 30).

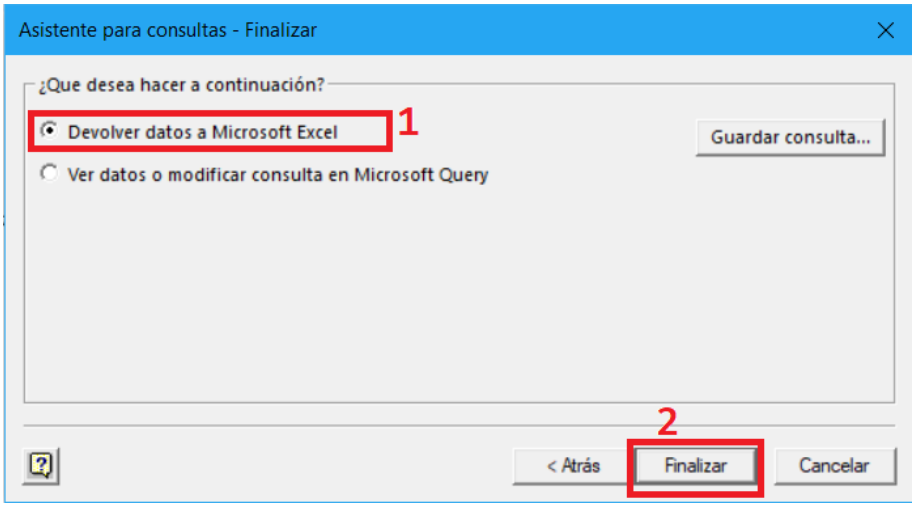


Figura 29: Finalización de importación de datos

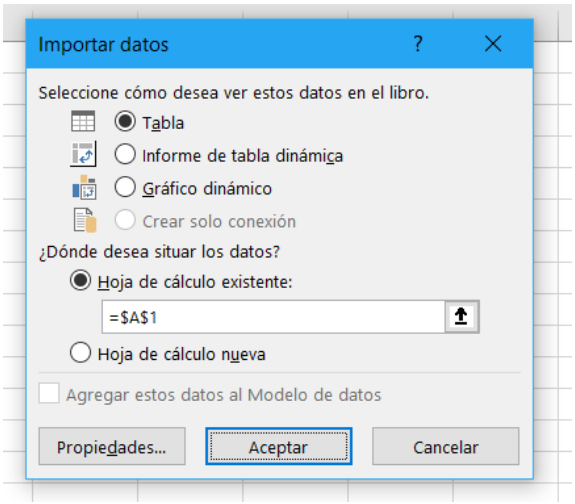


Figura 30: Selección de posición para los datos

8.3 Ejemplo de análisis: control de velocidad de crucero

Como ejemplo de uso, se presenta un breve estudio de cómo afecta el uso del control de velocidad de crucero de un vehículo frente al control manual de la velocidad. Se presentarán gráficas de las velocidades y revoluciones por minuto correspondientes a dos conductores distintos, y se mostrarán algunos resultados determinantes.

8.3.1 Vehículo y usuarios para la prueba

Para este ejemplo se han seleccionado dos conductores distintos. Uno de ellos utiliza el control de velocidad, y el otro no. Respecto a los datos de los conductores:

- Usuario 1:
  - Edad: 29 años
  - Año de obtención de permiso: 2007
  - Distancia media anual: 4500 km
- Usuario 2:
  - Edad: 28 años

- Año de obtención de permiso: 2008
- Distancia media anual: 6000 km

El vehículo utilizado para la prueba es un BMW modelo Serie 1, con transmisión automática.

8.3.2 Datos del trayecto

Por privacidad, la localización geográfica de la ruta no será mostrada. El trayecto consta de tres secciones: dos de carácter urbano y una interurbano. A su vez, la sección interurbana dispone de dos limitaciones de velocidad distintas. La siguiente tabla recoge los datos de distancia y límites de velocidad:

Tipo de tramo	Distancia	Límite de velocidad
Urbano	1.17 km	40 km/h
Interurbano 100	7 km	100 km/h
Interurbano 80	6.42 km	80 km/h

Tabla 4: Datos del trayecto

Puesto que este estudio versa sobre el uso del control de velocidad de crucero, no se tendrán en cuenta las partes del recorrido entre los tramos indicados en la Tabla 4. Además, se indicarán valores estadísticos para los tramos urbanos, aunque no se utilice el control de velocidad en ellos.

8.3.3 Análisis de los resultados

A continuación, se muestran las gráficas correspondientes a las revoluciones por minuto del motor y velocidad del vehículo, tanto para el usuario 1 (Figura 31) como para el usuario 2 (Figura 32)

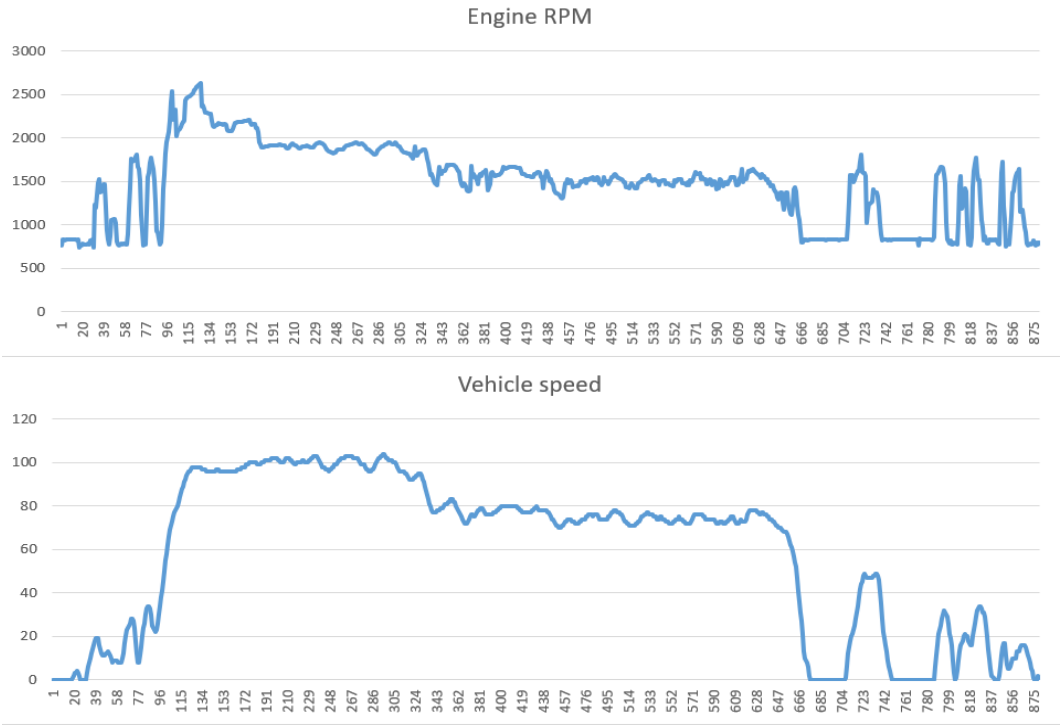


Figura 31: Datos del usuario 1 para el trayecto

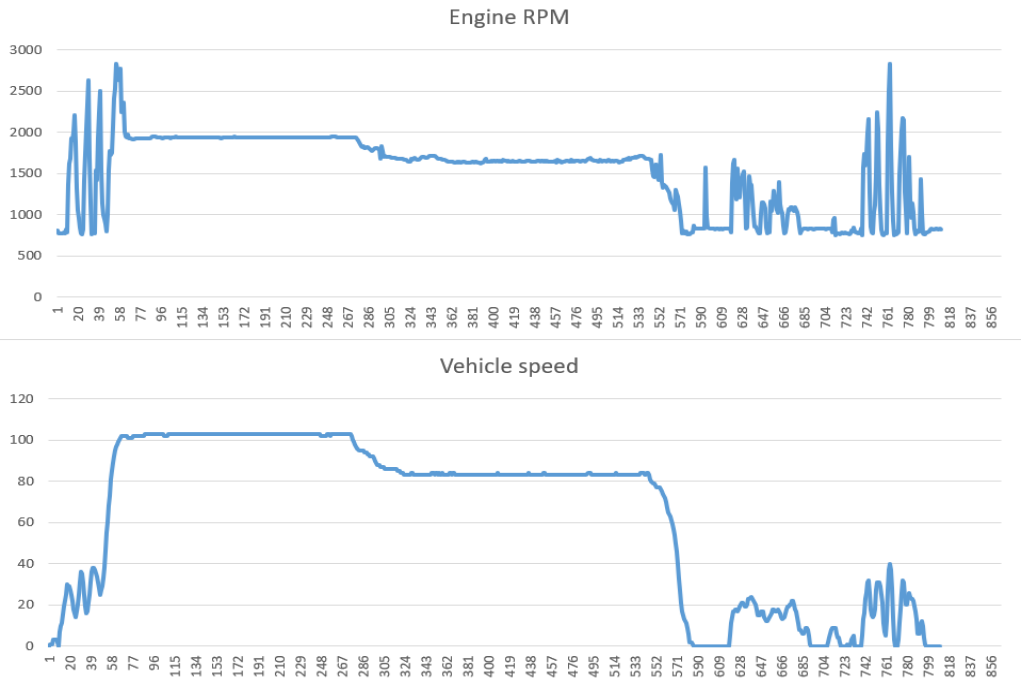


Figura 32: Datos del usuario 2 para el trayecto

A la vista de los resultados, rápidamente se observa una característica diferenciadora entre los dos conductores: la estabilidad de las curvas. En este caso, el usuario 2 lleva activo el control de velocidad de crucero para los tramos interurbanos, por lo que las variaciones de velocidad entorno a la velocidad objetivo (el límite de la vía) son menores. Se pueden caracterizar estadísticamente los datos correspondientes a la velocidad, calculando la media y la desviación típica de esta, como se muestra en las siguientes tablas:

	Urbano 1	Inter 100	Inter 80	Urbano 2
Media	11.44 km/h	99.27 km/h	75.56 km/h	13.22 km/h
Desviación típica	10.14 km/h	2.33 km/h	2.78 km/h	15.1 km/h

Tabla 5: Media y desviación típica de la velocidad para el usuario 1

	Urbano 1	Inter 100	Inter 80	Urbano 2
Media	19.93 km/h	102.79 km/h	83.09 km/h	13.01 km/h
Desviación típica	12.06 km/h	0.51 km/h	0.41 km/h	9.65 km/h

Tabla 6: Media y desviación típica de la velocidad para el usuario 2

Para los tramos interurbanos, la media se acerca, en ambos casos, a la velocidad límite de la vía. La diferencia se observa en la desviación típica. Con el control de velocidad de crucero activado (Tabla 6), la desviación típica es aproximadamente 0.51 km/h para el primer tramo interurbano, y de 0.41 km/h para el segundo tramo interurbano, mientras que sin el control de velocidad activado (Tabla 5) nos encontramos con valores de entre 4.5 y 7 veces mayores.

Si realizamos los cálculos para las revoluciones, obtenemos los siguientes datos:

	Urbano 1	Inter 100	Inter 80	Urbano 2
Media	1067.8 rpm	1992.51 rpm	1530.63 rpm	1043.23 rpm
Deviación típica	351.28 rpm	167.5 rpm	75.92 rpm	315.95 rpm

Tabla 7: Media y desviación típica de RPM para el usuario 1

	Urbano 1	Inter 100	Inter 80	Urbano 2
Media	1304.02 rpm	1937.39 rpm	1657.97 rpm	1063.9 rpm
Desviación típica	563.24 rpm	7.27 rpm	29.19 rpm	386.69 rpm

Tabla 8: Media y desviación típica de RPM para el usuario 2

Para las revoluciones por minuto se observan resultados con las mismas características (debido a que ambos tipos de datos están relacionados físicamente mediante la transmisión del vehículo). Mientras que la desviación típica para la utilización del control de velocidad (Tabla 8) es de solo 7.27 y 29.19 rpm respectivamente para los tramos Inter 100 e Inter 80, sin el control de velocidad (Tabla 7) obtenemos valores de hasta 23 veces mayores para el tramo Inter 100.

Cabe mencionar que la razón de que los datos de velocidad de vehículo y revoluciones por minuto del motor sean sensiblemente mayores durante la utilización del control de velocidad se debe simplemente a una velocidad objetivo ligeramente mayor que el límite de la vía. Aún así, esto no afecta en gran medida al consumo de combustible de un caso respecto al otro, como sí que afecta la variación en velocidad y revoluciones por minuto, pues en un caso estamos acelerando solo para mantener la velocidad, mientras que en el otro hay que estar recuperando la velocidad perdida, lo que se traduce en un mayor gasto de combustible, decrementando la eficiencia.

CAPÍTULO 9. PRESUPUESTO ECONÓMICO

Una de las partes más importantes de todo proyecto es la viabilidad económica del mismo. A continuación, se listan los gastos necesarios para adquirir el hardware:

Ítem	Precio
Raspberry Pi 2	49 euros
Tarjeta de memoria 16 GB	Incluida en Raspberry Pi.
OBDlink LX Scan Tool	66.82 euros
GPS Hat for Raspberry Pi	42.90 euros
FeatherWing OLED	24.22 euros x2
Asus USB-BT400	15.90 euros
Caja impresa en 3D	Gratis
TOTAL	**223.06 euros**

Tabla 9: Gastos de hardware

Se han utilizado más herramientas de las listadas en la tabla anterior, pero no se tienen en cuenta pues no fueron adquiridas con el objetivo de este proyecto. Son las siguientes:

- Soldador.
- Materiales de soldadura.
- Cableado.
- Herramientas variadas (tijeras, pinzas, etc.).
- Cable de alimentación USB para Raspberry Pi.
- Cable de red Ethernet.

En la siguiente table se listan los gastos de software:

Software	Precio
Sistema operativo Raspberry Pi	Gratis
Sistema operativo de desarrollo	Gratis
Entorno de desarrollo software	Gratis

Tabla 10: Gastos de software

Debido a la utilización de software open source, es posible reducir el coste de realización del proyecto. Puesto que Raspberry Pi utiliza distribuciones GNU/Linux, la verdadera reducción se da en el sistema operativo de desarrollo y en el entorno de desarrollo.

En la siguiente tabla se muestran las horas dedicadas:

Ítem	Cantidad de horas
Desarrollo de la aplicación RPIDDA	270 horas
Adaptación del hardware	3.5 horas
Configuración software y hardware	3 horas
Escritura de la memoria del TFG	30 horas
TOTAL	**306.5 horas**

Tabla 11: Dedicación en horas

Se puede observar que el grueso de dedicación se encuentra en el objetivo principal de este proyecto, el desarrollo de la aplicación para la obtención de datos.

CAPÍTULO 10. CONCLUSIONES Y LÍNEAS FUTURAS

Gracias a este trabajo ha sido posible descubrir en mayor profundidad las tecnologías y protocolos utilizadas en OBD. Se han afrontado momentos de todo tipo: falta de documentación, rotura de partes hardware, modificación de especificaciones… Y todos ellos han salido adelante gracias a "darle una vuelta más".

El desarrollo Bluetooth utilizando el lenguaje de programación C ha resultado no ser nada trivial. La falta de documentación, mencionada anteriormente, ha hecho que esta haya sido una de las partes más complicadas del desarrollo de la aplicación. Una vez comprendida, es inevitable ver la similitud con el desarrollo de *sockets*, por lo que facilita la búsqueda de información.

Se ha puesto en práctica, además, el manejo de tiempos de entrega, o *deadlines*. Esto ha hecho que muchas de las características planificadas hayan sido relegadas a futuras ampliaciones. Pero no todo tiene por qué ser negativo. La existencia de *deadlines* ha hecho que se hayan tenido que sopesar de manera más profunda las ventajas e inconvenientes de unos métodos frente a otros, dando la oportunidad de aprender un poco de aquellos no utilizados en el resultado final.

Como se mencionó en el capítulo introductorio, esta aplicación no pretendía ser una referencia. Simplemente ayudar al autor a aprender, y a unir dos mundos que no tienen por qué estar tan separados: hardware y software.

El resultado final ha sido una prueba de concepto, pero que cumple con su cometido: un hardware pequeño, asequible y montado a medida del caso de uso, y una aplicación software, *RPIDDA*, que permite la obtención de los datos de un vehículo de una forma sencilla. Esto da por cumplido el objetivo de este trabajo.

Tras el desarrollo de este trabajo, no se puede evitar pensar en nuevas ideas y mejoras, tanto software como hardware.

Comenzando por la aplicación *Raspberry Pi Driving Data Acquisition*, es posible ampliarla con el soporte para todos los PIDs disponibles en el estándar de OBD2. Probablemente la mayoría de vehículos no puedan dar respuesta a todos los PIDs, pero siempre es bueno disponer de soporte para ellos.

Si pensamos en la arquitectura de software, sería también una buena modificación el agregar mayor control de la temporización y sincronización interna, mediante *threads* o *signal handling*. En el lenguaje de programación escogido para la aplicación, C, éstas son dos de las características más complicadas de implementar de una forma segura y eficiente, pero sin duda es una ampliación interesante. Además de otros muchos detalles que pueden derivarse de la lectura e interpretación del código (como mejora de *logs*), los cuales se dejan para futuros lectores.

La utilización de tests para probar el código es una de las necesidades básicas si el proyecto sigue creciendo (incluso para el tamaño actual). Todo ello, incluyendo una planificación de *Continuous Integration*, permitiría el desarrollo software de una forma mucho más rápida y sin problemas de inclusión de *bugs* con cada mejora de características.

Respecto al hardware, la utilización de una pantalla de mayor diagonal permitiría mostrar la información en tiempo real, incluso sin necesidad del diseño de una interfaz de usuario complicada (mostrando solo un terminal). Esto requeriría un rediseño de la caja, que se ve facilitado gracias a la impresión 3D.

Y, por supuesto, es posible mirar al futuro. Dotar de una plataforma *IoT* sería la orientación más adecuada, permitiendo a Raspberry Pi conectarse mediante redes LTE o 5G a un servidor remoto en el que guardar los datos obtenidos. Utilizar los nuevos estándares de comunicación entre vehículos, en los que cada vehículo hace uso de la información de los otros vehículos que le rodean para analizar situaciones de tráfico, reducir consumos, evitar accidentes… Las posibilidades de ampliación son “infinitas”.

REFERENCIAS

[1] M. d. Hacienda y G. d. España, «Eficiencia en el transporte,» 2002. [En línea]. Available: http://www.idae.es/uploads/documentos/documentos_manualPME_6bc54e20.pdf.

[2] SAE, «SAE J1979: E/E Diagnostic Test Modes,» [En línea]. Available: https://www.sae.org/standards/content/j1979_201202/.

[3] Wikipedia, «OBD-II PIDs,» [En línea]. Available: https://en.wikipedia.org/wiki/OBD-II_PIDs.

[4] R. P. Foundation, «Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.org/.

[5] https://www.raspbian.org/, «Raspbian OS,» [En línea]. Available: https://www.raspbian.org/.

[6] Adafruit, «Adafruit Ultimate GPS HAT fro Raspberry Pi,» [En línea]. Available: https://learn.adafruit.com/adafruit-ultimate-gps-hat-for-raspberry-pi/overview.

[7] Adafruit, «Adafruit FeatherWing OLED,» [En línea]. Available: https://learn.adafruit.com/adafruit-ultimate-gps-hat-for-raspberry-pi/overview.

[8] Asus, «Asus BT400,» [En línea]. Available: https://www.asus.com/us/Networking/USBBT400/.

[9] O. Solutions, «STN1110: Multiprotocol OBD to UART Interpreter IC,» [En línea]. Available: http://www.obdsol.com/solutions/chips/stn1110/.

[10] OBDLink, «OBDLink LX Bluetooth,» [En línea]. Available: http://www.obdlink.com/lxbt/.

[11] Thingverse, «3D case for Raspberry Pi,» [En línea]. Available: https://www.thingiverse.com/thing:1015706.

[12] Raspberry, «Rasbian OS Download,» [En línea]. Available: https://www.raspberrypi.org/downloads/raspbian/.

[13] Ubuntu, «Ubuntu Installation Guide,» [En línea]. Available: https://help.ubuntu.com/lts/installation-guide/amd64/index.html.

[14] PyOBD, «PyOBD,» [En línea]. Available: https://github.com/peterh/pyobd.

[15] Drogon, «wiringPi,» [En línea]. Available: http://wiringpi.com/.

[16] Wikipedia, «Row- and column-major order,» [En línea]. Available: https://en.wikipedia.org/wiki/Row-_and_column-major_order.

[17] C. Werner, «SQLite ODBC Driver,» [En línea]. Available: http://www.ch-werner.de/sqliteodbc/.

[18] Eclipse, «Eclipse Project,» [En línea]. Available: https://www.eclipse.org/.