Integración y caracterización del sistema fotovoltaico en el vehículo UAL-eCARM

por

Aarón Raúl Poyatos Bakker

Trabajo Fin de Grado
Grado en Ingeniería Mecánica

Universidad de Almería - Escuela Superior de Ingeniería
Almería, Septiembre 2019

Integración y caracterización del sistema fotovoltaico en el vehículo UAL-eCARM

por

Aarón Raúl Poyatos Bakker

Para la obtención del
Título del Grado en Ingeniería Mecánica

Director	**Director**	**Autor**
Dr. Jose Luis Blanco Claraco	Dr. Jose Luis Torres Moreno	Aarón Raúl Poyatos Bakker

A mis padres Muriel y Paco
y a mi hermano Rubén

Agradecimientos

En este apartado quiero mostrar mis agradecimientos a todas las personas e instituciones que se han involucrado de alguna manera en el proyecto, ya que han ayudado a que se materialice en esta memoria y soy consciente de que solo, esto no hubiera sido posible.

Comenzando por mi director del TFG, Jose Luis Blanco, quién aceptó la propuesta de realizar el proyecto sin dudarlo y ha mostrado un apoyo incondicional durante todo el período. Su forma de guiar mis propuestas ha sido esencial para poder trazar el camino de este trabajo y sumergirme de lleno en la programación.

También quiero agradecer enormemente el gran aporte de mi codirector, José Luis Torres, por el interés y la atención dedicada para brindarme todo el material y la ayuda necesaria para llevar a cabo este trabajo de la mejor manera posible. Su disponibilidad y su actitud no tienen precio.

A la ayuda de Ismael y de Julián, dos compañeros con los que he estado gran parte del tiempo. Sin ellos no habría disfrutado tanto cada momento del trabajo y han sido un pilar fundamental a la hora de realizar los ensayos del vehículo.

Por supuesto, a toda mi familia, por animarme y por darme la inspiración y el apoyo necesario en todo momento.

Finalmente, también quiero agradecer al grupo de investigación de Automática, Robótica y Mecatrónica, por permitir el uso del vehículo eléctrico y por las recomendaciones dadas en cuanto a la redacción del proyecto.

Resumen

Hoy en día, la sociedad tiene cada vez más presente el impacto negativo que producen al medioambiente los vehículos que emplean fuentes no renovables. Con el desarrollo de la tecnología y la optimización de las fuentes renovables, se abren las posibilidades de implementar sistemas de carga en vehículos eléctricos.

El objetivo de este trabajo ha sido precisamente el estudio y la selección de los elementos necesarios para integrar un sistema fotovoltaico como fuente de energía aislada en un vehículo eléctrico; así como su lectura y transmisión de estados vía la red de comunicaciones que emplea el propio vehículo, a través del uso de un middleware robótico llamado ROS (Robot Operating System).
Para ello se ha desarrollado un paquete específico en el entorno ROS, dedicado exclusivamente al filtrado e interpretación de los valores del sistema fotovoltaico y transmitir de manera automática aquellos valores de interés (voltaje e intensidad del sistema de baterías y de los módulos fotovoltaicos).
Posteriormente, se ha realizado la instalación en el vehículo de todos los componentes que integran el sistema fotovoltaico y se han llevado a cabo ensayos experimentales para el análisis energético del sistema.

Este trabajo aporta una visión clara sobre el procedimiento de instalación de un sistema fotovoltaico en un vehículo eléctrico. Y describe la configuración necesaria dentro del entorno ROS, de cómo establecer una vía de comunicación con el regulador de carga: el dispositivo encargado de controlar y estabilizar el sistema fotovoltaico.

Palabras clave: Sistema Fotovoltaico, Vehículo Eléctrico, Control Autónomo, Entorno ROS

Abstract

Nowadays, our society is increasingly aware of how vehicles that use non-renewable sources have a negative impact on the environment. The development of technology and the optimization of renewable sources, opens new possibilities of implementing charging systems in electric vehicles.

The objective of this work has been precisely the study and selection of the necessary elements to integrate a photovoltaic system as an isolated energy source in an electric vehicle; as well as its reading and transmission of states via the communications network used by the vehicle itself, through the use of a robotic middleware called ROS (Robot Operating System).
To achieve this, a novel package has been developed in the ROS environment, dedicated exclusively to filtering and interpreting the values of the photovoltaic system and automatically transmitting those values of interest, voltage and the intensity of the battery system and photovoltaic modules.
Subsequently, the installation in the vehicle of all the components that make up the photovoltaic system has been carried out, as experimental tests have been accomplished for energy analysis of the system.

This work provides a clear vision about the installation procedure of a photovoltaic system in an electric vehicle, and describes the necessary configuration within the ROS environment to establish a communication link to the charge controller, the device in charge of controlling and stabilizing the photovoltaic system.

Keywords: Photovoltaic System, Electric Vehicle, Autonomous Control, ROS Environment

Índice general

Índice de figuras

Índice de tablas

Siglas y acrónimos

Abreviatura	Denominación original	Denominación en castellano
GEI	-	Gas de Efecto Invernadero
IPCC	Intergovernmental Panel on Climate Change	Grupo Intergubernamental de Expertos sobre el Cambio Climático
MPPT	Maximum Power Point Tracker	Regulador de carga con seguidor de punto de máxima potencia
PSC	Partial Shaded Condition	Condición de sombra parcial
PWM	Pulse-Width Modulation	Modulación por ancho de pulsos
ROS	Robot Operating System	Middleware Robótico
STC	Standart Test Conditions	Condiciones de ensayos estándar
VRLA	Valve Regulated Lead Acid Batteries	Batería de ácido-plomo regulada por válvula

Nomenclatura

Notación	Significado	Unidades
AM	Masa de aire	-
I	Corriente, intensidad	[A]
Ibat	Corriente del sistema de baterías	[A]
Im, Imp	Corriente en el punto de máxima potencia	[A]
Ipv	Corriente del sistema de paneles solares	[A]
Ipwm	Corriente en el regulador de carga PWM	[A]
Irradiancia	Potencia incidente por unidad de superficie	[W/m^2]
Isc	Corriente en cortocircuito	[A]
P, Pot	Potencia	[W]
Pm, Pmáx	Potencia máxima	[W]
Pbat	Potencia en el circuito del sistema de baterías	[W]
Ppv	Potencia generada por el sistema de paneles solares	[W]
V	Tensión, diferencia potencial	[V]
Vbat	Tensión del sistema de baterías	[V]
Vm, Vmp	Tensión en el punto de máxima potencia	[V]
Voc	Tensión en circuito abierto	[V]
Vpv	Tensión del sistema de paneles solares	[V]
Vpwm	Tensión en el regulador de carga PWM	[V]

Capítulo 1

Introducción

1.1. Motivación del trabajo

El cambio climático por el que atraviesa actualmente el planeta es motivo de alarma debido al crecimiento de las emisiones antropogénicas de gases de efecto invernadero (GEI). Según el quinto informe realizado por el Grupo Intergubernamental de Expertos sobre Cambio Climático (IPCC (2016)), entre el año 2000 y el 2010 las emisiones de GEI han crecido más que en tres décadas previas.

En la imagen mostrada a continuación se puede observar el impacto negativo que podrían producir los principales sectores (indicados en una tonalidad transparente) y su mitigación en caso de tomar las medidas necesarias (indicados en una tonalidad opaca). Se muestran las diferencias en 3 fechas (2030, 2050 y 2100).

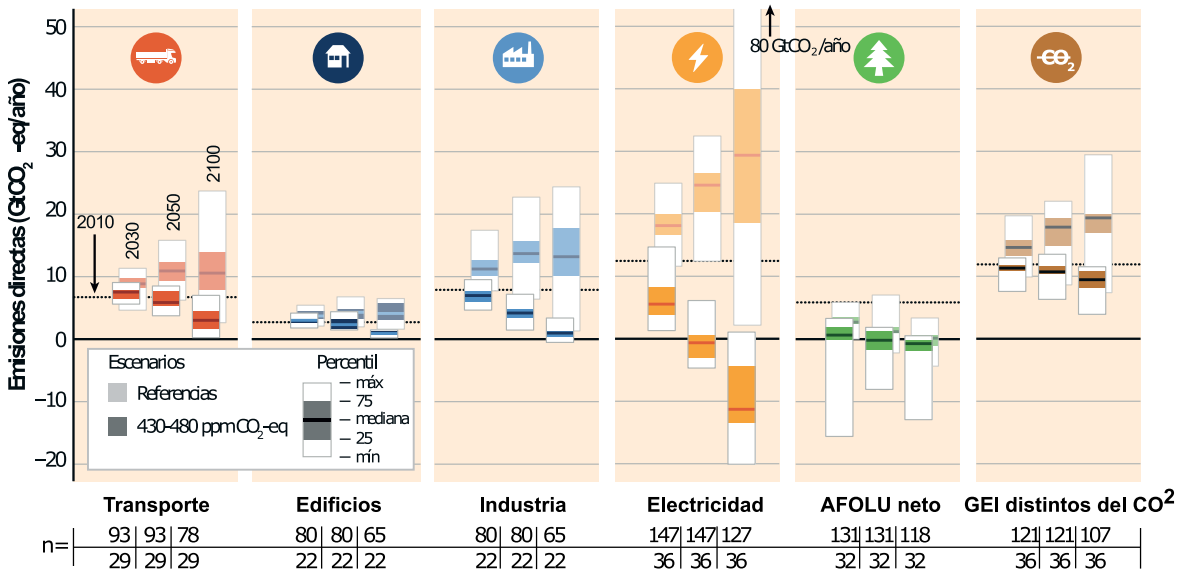


Figura 1.1: *Emisiones directas de CO_2 por sectores principales y emisiones de GEI distintos del CO_2 (IPCC (2016))*

Debido al riesgo que esto supone, un objetivo prioritario en la actualidad es mitigar esos gases proponiendo soluciones alternativas a la obtención y utilización de fuentes de energía, siendo de principal interés las fuentes de energía renovables.

En este trabajo se ha tenido especial atención en la manera de implementar fuentes renovables en los vehículos de transporte, ya que su mitigación, aparte de reducir las emisiones directas de CO_2 como se ha mostrado en la Figura 1.1, ayudaría enormemente a luchar contra la grave contaminación que emiten los vehículos con motor de combustible (sobre todo en las grandes ciudades, donde ya se han tenido que tomar medidas para reducir la circulación de dichos vehículos).
Esto se debe a que la implementación de un sistema fotovoltaico aislado fomenta el uso de vehículos totalmente eléctricos, aparte de ser otra autonomía adicional ya que su regulación se realiza internamente mediante los reguladores de carga.

Es por ello que, con este trabajo, se pretende definir e implementar un sistema independiente de carga en un vehículo eléctrico con energía solar, además de ser actualmente una de las alternativas más viables debido al gran desarrollo que han tenido los paneles solares en los últimos años, haciéndolos más eficientes y económicos.

1.2. Objetivos

Este trabajo tiene como objeto principal la integración de los módulos fotovoltaicos y del regulador de carga en el sistema eléctrico del vehículo UAL e-CARM, cumpliendo también con los siguientes subobjetivos:

- El diseño y la programación del software necesario para la adquisición de las señales requeridas para monitorizar el voltaje y la corriente generada en cada momento por los paneles fotovoltaicos.
- El análisis energético y el estudio del sistema fotovoltaico instalado en el vehículo.

La instalación del sistema fotovoltaico en el vehículo requiere cierto conocimiento sobre los diversos elementos que componen la instalación y la forma de implementarlos con el sistema de baterías que ya posee el vehículo eléctrico.

Además, para la adquisición de las señales del sistema fotovoltaico, es necesario establecer una vía de comunicación en el entorno ROS (middleware robótico, por sus siglas en inglés: Robot Operating System) que ya emplea el vehículo, implicando la programación de la obtención y el envío de los datos de mayor interés.
Para ello es indispensable tener el conocimiento de las diversas comunicaciones que se deben realizar entre el regulador de carga que controla el sistema fotovoltaico y el sistema ROS del vehículo; así como la interpretación de los datos que se obtienen a través de dicha vía de comunicación y su procesado, interpretación y envío posterior.

1.3. Contexto

Esta memoria supone el punto y final de los estudios de grado en Ingeniería Mecánica del autor en la Universidad de Almería, demostrando la aplicación de los conocimientos adquiridos durante el período de estudio en un tema de investigación bastante demandado hoy en día debido a lo mencionado en el apartado 1.1 y ha sido propuesto por el Grupo de Investigación TEP-197 Automática, Robótica y Mecatrónica (Grupo ARM) de la propia universidad.

La Universidad de Almería dispone de un vehículo eléctrico urbano Greenland modelo LITA GLe2-2S bajo el nombre de UAL-eCARM empleado en varias líneas de investigación y en la que tienen por objeto el estudio de la eficiencia energética en los vehículos eléctricos.

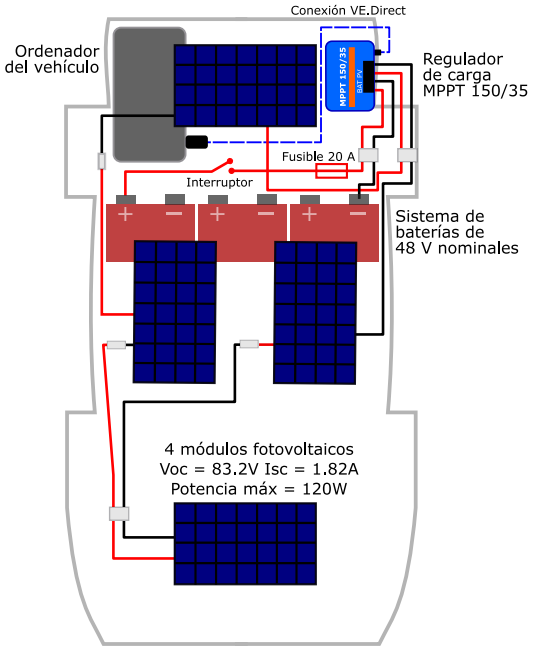
Figura 1.2: *Vehículo eléctrico UAL-eCARM (Greenland modelo LITA GLe2-2S) de la Universidad de Almería*

En una de dichas líneas de investigación tuvo lugar precisamente un trabajo fin de grado realizado por Javier Guerrero acerca del análisis y la instalación de un sistema fotovoltaico en el vehículo UAL-eCARM (Guerrero (2016)). En dicho trabajo se realizaron los estudios necesarios para caracterizar un sistema fotovoltaico aislado como fuente alternativa o complementaria a la red eléctrica del vehículo, diseñando un equipo de medida propio y facilitando de dicha manera la identificación de los factores en la eficiencia energética de un sistema fotovoltaico.

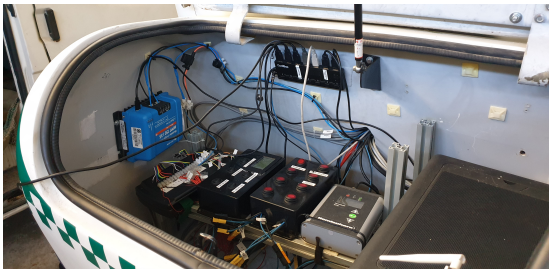
Con el trabajo previo de Javier Guerrero y el redactado en el presente documento, se ha definido finalmente la forma de implementar físicamente y a nivel de software el sistema fotovoltaico en el vehículo eléctrico.

1.4. Resumen de resultados

La integración del sistema fotovoltaico en el vehículo eléctrico ha quedado tal y como se representa en la Figura 1.3, cumpliendo el objetivo principal del trabajo.



(a) *Esquema gráfico de las conexiones en planta*



(b) *Ubicación del regulador de carga*

(c) *Vehículo con los paneles solares instalados*

Figura 1.3: *Resultados de la integración del sistema fotovoltaico en el vehículo eléctrico*

También se ha desarrollado un software para adquirir las señales de voltaje y corriente tanto del sistema de baterías como de los módulos solares, implementado en el sistema ROS del vehículo (Figura 1.4) y capaz de filtrar todas las tramas que emite el regulador de carga.

Por último, se han realizado varios ensayos con el vehículo eléctrico, permitiendo el análisis y el estudio del sistema fotovoltaico, gracias a las gráficas generadas por el software desarrollado. Una de dichas gráficas es como la mostrada en la Figura 1.5 que, junto con otros ensayos realizados, señalan dos ventajas de la integración del sistema fotovoltaico en el vehículo. Una de ellas es que se ha observado una mayor estabilidad en la carga y la descarga del vehículo (no hay caídas de tensión importantes al contrario que en los ensayos sin módulos fotovoltaicos), y la otra ventaja es que hay una clara mitigación en la descarga del sistema baterías, ya que la diferencia de voltaje al inicio y al final de los ensayos se ha visto reducida. Además, representando posteriormente los puntos de la potencia generada en cada tramo V-I registrado (véase la Figura 1.6) se ha podido comprobar que el regulador de carga estima de manera correcta el punto de máxima potencia (en torno a los 60 V, como bien ha definido Javier Guerrero en la Curva Voltaje-Potencia (V-P) que ha obtenido de los módulos fotovoltaicos).

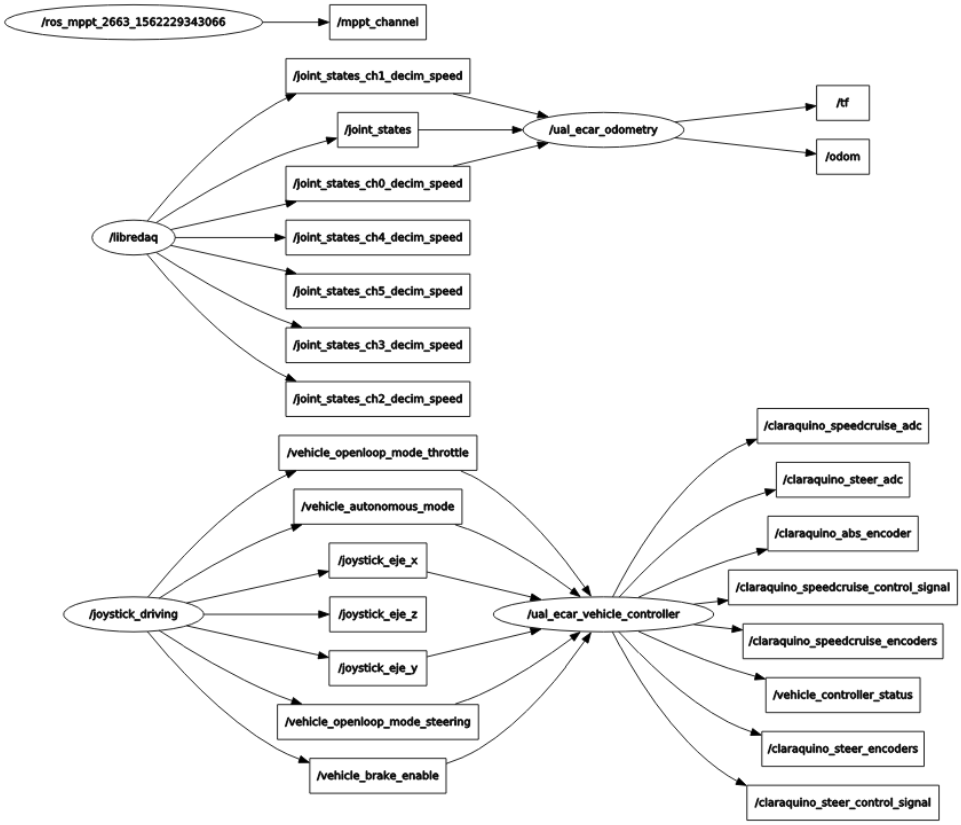


Figura 1.4: *RosGraph del vehículo eléctrico, con el nodo* `ros_mppt` *desarrollado en este trabajo (ubicado en la parte superior de la figura)*

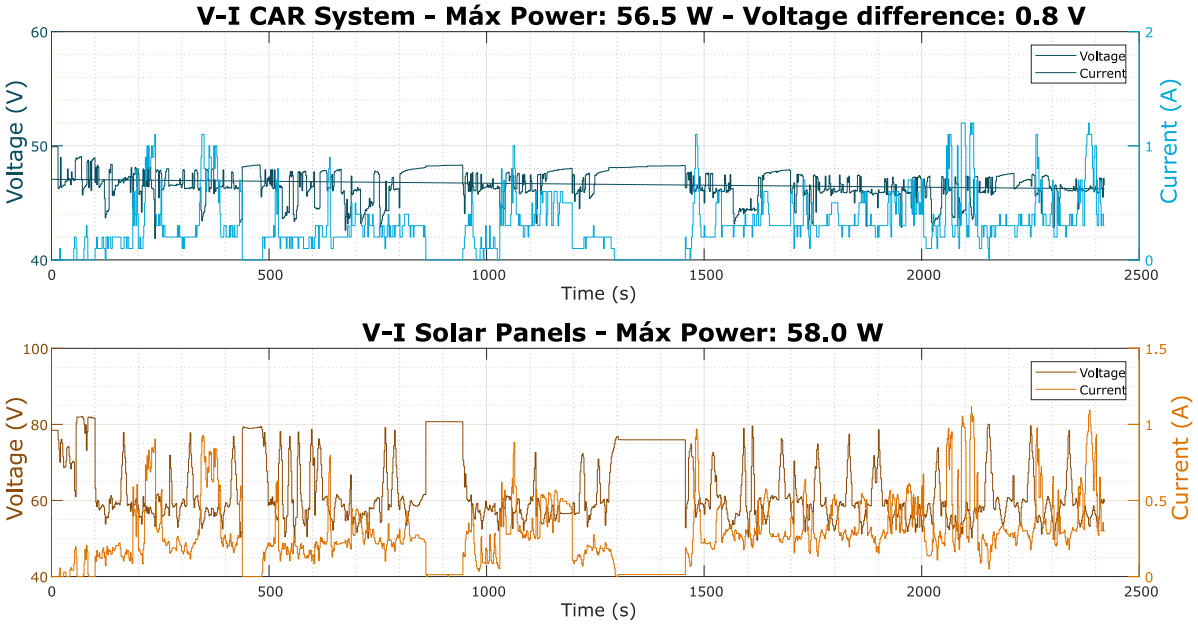


Figura 1.5: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos de un ensayo de larga duración*

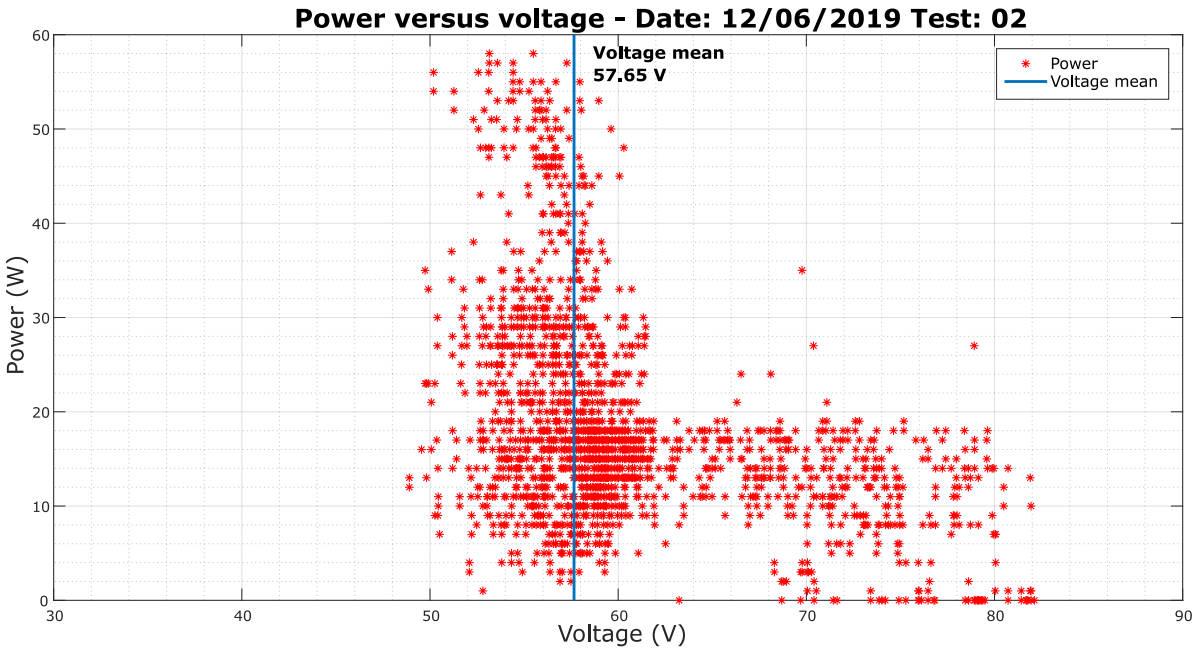


Figura 1.6: *Gráfica V-P de los módulos fotovoltaicos, con la tendencia de carga en la línea marcada en azul, del mismo ensayo de larga duración*

1.5. Planificación del trabajo

El presente trabajo se ha organizado en 7 tareas diferentes, algunas dependientes entre sí. La descripción de cada tarea se explica a continuación y se ha incluido la Tabla 1.1, donde se presenta las fechas de inicio y fin de cada tarea, así como los días que corresponden a ese intervalo de fechas, el tiempo dedicado a dicha tarea y las dependencias de tareas anteriores.

Tarea 1. Recopilación de información y revisión bibliográfica: esta tarea ha sido constante durante todo el período del trabajo ya que de ello se obtiene la información y la documentación necesaria para llevar a cabo todas las partes del proyecto.

Tarea 2. Montaje del banco de ensayos del sistema: en esta tarea se define el sistema fotovoltaico con los elementos seleccionados a partir de la búsqueda de información y los trabajos previos. Con ello se define un prototipo del sistema, para poder observar el comportamiento del sistema.

Tarea 3. Lectura de datos del regulador de carga: una vez realizado el banco de ensayos se realizan los programas de lecturas para interpretar las señales que envía el regulador de carga. Con dicha tarea se define el filtro necesario en el paquete ROS, que será el programa que finalmente se encargue de enviar los valores al sistema ROS del vehículo.

Tarea 4. Programa de lectura en ROS: definido el filtro de las tramas que envía el regulador de carga y la interpretación de los mismos, se procede a realizar el paquete en ROS.

Tarea 5. Montaje del sistema en el vehículo eléctrico: para esta tarea ya se tienen todos los elementos y materiales necesarios para componer el sistema e instalarlo en el vehículo eléctrico.

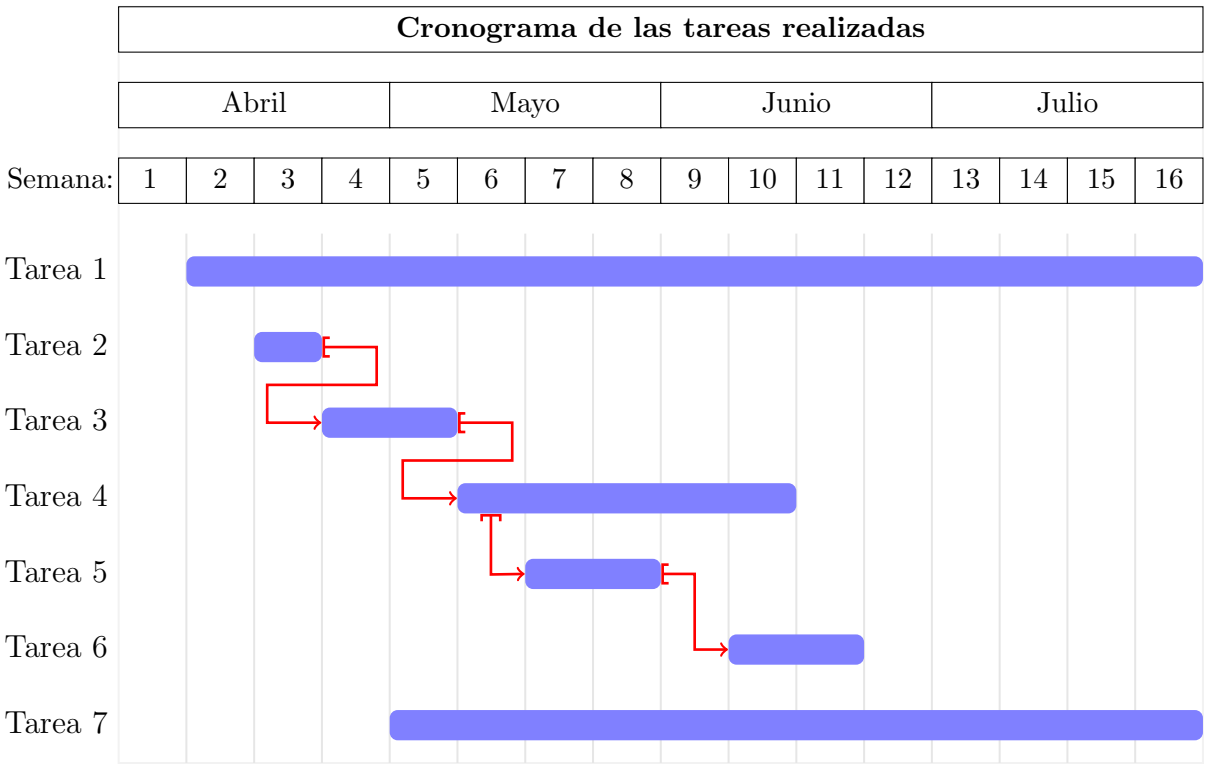
Tarea 6. Ensayos realizados con el vehículo: en esta tarea se realizan los ensayos experimentales para observar que todos los elementos que integran el sistema funcionan de manera adecuada, aparte de determinar cómo influye sobre el sistema de baterías del vehículo.

Tarea 7. Redacción de la memoria y conclusiones: esta tarea también ha sido constante al comenzar con el trabajo, amplificándose sobre todo una vez realizado el software de lectura de datos y la instalación del sistema en el vehículo. Se trata de la redacción y elaboración del presente documento y las valoraciones finales acerca de los resultados obtenidos.

Tabla 1.1: Planificación del TFG y cronograma de las tareas

		Fechas		**Duración**	
Tarea	**Dependencia**	**Inicio**	**Fin**	**Intervalo** (días)	**Tiempo** (horas)
1	-	08/04/2019	26/07/2019	140	45
2	-	15/04/2019	25/04/2019	10	23
3	Tarea 2	26/04/2019	06/05/2019	10	38
4	Tarea 3	08/05/2019	17/06/2019	40	34
5	Tarea 4	20/05/2019	27/05/2019	7	14
6	Tarea 5	11/06/2019	19/06/2019	8	27
7	Todas	04/05/2019	26/07/2019	113	136

El trabajo ha supuesto una duración de 140 días, de los cuales se han empleado 317 horas en total para la planificación, realización y redacción del mismo.



1.6. Competencias empleadas

1.6.1. Competencias generales de la UAL

UAL 1. Conocimiento, habilidades y actitudes que posibilitan la comprensión de nuevas teorías, interpretaciones, métodos y técnicas dentro de los diferentes campos disciplinares, conducentes a satisfacer de manera óptima las exigencias profesionales.

UAL 2. Utilizar las Técnicas de Información y Comunicación (TICs) como una herramienta para la expresión y la comunicación, para el acceso a fuentes de información, como medio de archivo de datos y documentos, para tareas de presentación, para el aprendizaje, la investigación y el trabajo cooperativo.

UAL 3. Capacidad para identificar, analizar, y definir los elementos significativos que constituyen un problema para resolverlo con rigor.

UAL 5. Es el comportamiento mental que cuestiona las cosas y se interesa por los fundamentos en los que se asientan las ideas, acciones y juicios, tanto propios como ajenos.

UAL 6. Integrarse y colaborar de forma activa en la consecución de objetivos comunes con otras personas, áreas y organizaciones, en contextos tanto nacionales como internacionales.

UAL 8. Capacidad para pensar y actuar según principios de carácter universal que se basan en el valor de la persona y se dirigen a su pleno desarrollo.

UAL 9. Capacidad para diseñar, gestionar y ejecutar una tarea de forma personal.

1.6.2. Competencias específicas del grado

Generales en todo el proyecto

CT1. Capacidad para la redacción, firma y desarrollo de proyectos en el ámbito de la ingeniería industrial que tengan por objeto la construcción, reforma, reparación, conservación, demolición, fabricación, instalación, montaje o explotación de: estructuras, equipos mecánicos, instalaciones energéticas, instalaciones eléctricas y electrónicas, instalaciones y plantas industriales y procesos de fabricación y automatización.

CT4. Capacidad de resolver problemas con iniciativa, toma de decisiones, creatividad, razonamiento crítico y de comunicar y transmitir conocimientos, habilidades y destrezas en el campo de la Ingeniería Industrial.

CT5. Conocimientos para la realización de mediciones, cálculos, valoraciones, tasaciones, peritaciones, estudios, informes, planes de labores y otros trabajos análogos.

CT6. Capacidad para el manejo de especificaciones, reglamentos y normas de obligado cumplimiento.

CT9. Capacidad de organización y planificación en el ámbito de la empresa, y otras instituciones y organizaciones.

CT10. Capacidad de trabajar en un entorno multilingüe y multidisciplinar.

CTEQ2. Capacidad para el análisis, diseño, simulación y optimización de procesos y productos.

Montaje, modificación y análisis de los diversos circuitos eléctricos del sistema fotovoltaico integrado en el vehículo

CB2. Comprensión y dominio de los conceptos básicos sobre las leyes generales de la mecánica, termodinámica, campos y ondas y electromagnetismo y su aplicación para la resolución de problemas propios de la ingeniería.

CRI4. Conocimiento y utilización de los principios de teoría de circuitos y máquinas eléctricas.

CTEE5. Conocimiento aplicado de instrumentación electrónica.

CTEE10. Conocimiento aplicado de informática industrial y comunicaciones.

CTEM1. Conocimientos y capacidades para aplicar las técnicas de ingeniería gráfica.

Creación de programas de lectura de datos en Matlab, Python y nodos ROS

CB3. Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.

CTEE10. Conocimiento aplicado de informática industrial y comunicaciones.

1.7. Estructura de la memoria

Esta memoria se ha estructurado en 6 capítulos, resumidos a continuación.
En los capítulos 2 y 3 se introducen los conceptos empleados para formar el sistema fotovoltaico del vehículo, el sistema ROS y el lenguaje de programación Python, y la selección de los elementos que se van a utilizar.
El capítulo 4 de la memoria agrupa el trabajo principal del proyecto, en la que se detalla tanto la creación de los diversos programas de lectura de datos del regulador de carga como la implementación del sistema en el vehículo.
En el capítulo 5 se detallan las pruebas y los ensayos realizados, tanto de los programas a nivel de software como del propio vehículo con el sistema integrado.
Por último, en el capítulo 6 se justifican los resultados obtenidos y se exponen las conclusiones del trabajo.

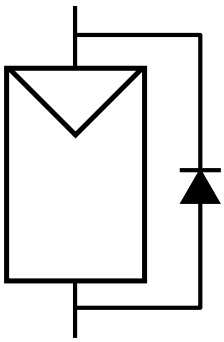
Capítulo 2

Revisión bibliográfica

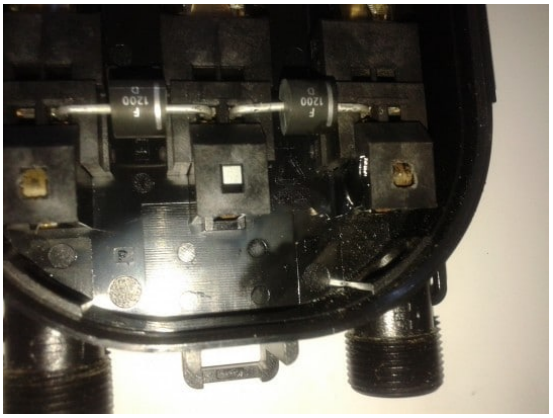
2.1. Reguladores de carga y curvas características

Los módulos fotovoltaicos se colocan en serie y/o en paralelo para proporcionar la potencia demandada por el sistema de cargas. Para recibir una adecuada irradiación del sol, los módulos se colocan en los tejados de los edificios así como en los techos de los vehículos o de cualquier otro dispositivo. Aún así, esa disposición no evita que pasen elementos que generan zonas sombreadas en los paneles (como nubes, pájaros o polvo), situación que se denomina condición de sombra parcial (PSC, por sus siglas en inglés Partial Shaded Condition). Esta condición no sólo afecta a la potencia que pueda aportar el sistema fotovoltaico sino que, además, puede ocasionar daños físicos en los módulos ya que los puntos de sombra generan una temperatura elevada (Nayak Bhukya und Reddy Kota (2019)). Por ello, para disminuir el riesgo de sobrecalentamientos en los paneles, disponen de un diodo de paso (también llamados de bypass) (véase la Figura 2.1) ubicados en la caja de conexiones en la parte trasera del panel solar.
Debido a estas condiciones, es de gran interés estudiar la optimización de los sistemas fotovoltaicos, gracias a los tipos de reguladores de carga que existen. El objetivo del dispositivo es aprovechar al máximo la energía suministrada por los módulos, a la vez que garantiza la protección adecuada y el buen servicio de las baterías y/o demandas, evitando la sobrecarga y la sobredescarga (en el caso de baterías) (Guerrero (2016)), incluso en PSC.
No obstante, no existe un único tipo de regulador de carga ya que, según el tipo de sistema fotovoltaico, interesa una regulación determinada. Existen varios tipos de reguladores de carga según las funciones que realizan, como ha descrito Javier Guerrero: reguladores PWM (por sus siglas en inglés (pulse-width modulation), modulación por ancho de pulsos), convertidores DC/DC, MPPT, etc...

Los más utilizados actualmente son los reguladores PWM y los reguladores MPPT, siendo las diferencias entre ambas las siguientes (VictronEnergy (2014)):
Un regulador PWM es, en esencia, un interruptor que conecta los módulos fotovoltaicos a la batería. Lo que realiza esta función es reducir la tensión de los módulos hasta la tensión de la batería, en lugar de tenerlos en circuito abierto. De esta manera, se produce una carga pulsada PWM de la corriente que carga de una forma más óptima la batería (Guerrero (2016)).



(a) *Esquema del módulo fotovoltaico con el diodo de paso*



(b) *2 diodos de paso colocadas en una caja de conexiones típica en paneles solares (SunFields (2018))*

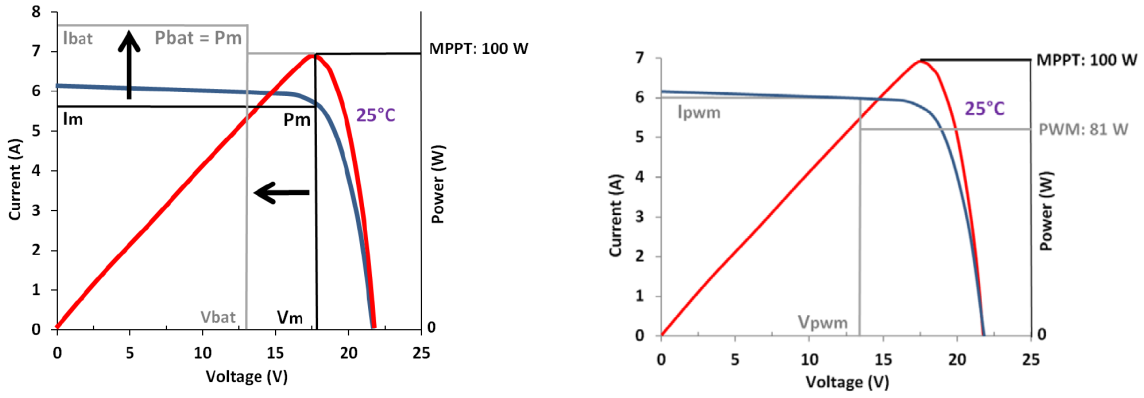
Figura 2.1: *Esquema y disposición de los diodos de paso en los módulos fotovoltaicos*

Por otro lado, un regulador MPPT es más sofisticado y por ello mucho más caro. El dispositivo toma el voltaje de entrada de los módulos y lo modifica de tal manera que pueda aportar la máxima potencia de los paneles, cumpliendo con el voltaje de carga que requieran las baterías (que es más elevada que la nominal). De esta forma, también pueden actuar como convertidores DC/DC, debido a que pueden acoplarse por ejemplo un sistema fotovoltaico cuyos módulos generen una diferencia potencial de 60 V a un sistema de baterías de 48 V nominales.
Generalmente, los reguladores MPPT son más eficientes que los reguladores PWM en climas fríos y templados, mientras que ambos reguladores muestran el mismo rendimiento en climas tropicales y subtropicales, como es el caso de Almería.
Por dichas diferencias, se puede decir que un regulador MPPT optimiza mejor la energía proporcionada por los módulos incluso en PSC, mientras que el regulador PWM se limita a reducir el voltaje para realizar una mejor carga (VictronEnergy (2014)).

A continuación se muestran las curvas V-I y V-P (en la Figura 2.2) para comparar las acciones que realiza el regulador MPPT y el PWM, con un módulo solar monocristalino de 100 W y 36 células (valores definidos en la Tabla 2.1), de la empresa Victron. Nótese que dichos módulos no son los utilizados para el sistema fotovoltaico del vehículo.

Tabla 2.1: Características del módulo solar monocristalino de 100 W y 36 células (VictronEnergy (2014))

Característica	**Valor**
Potencia máxima (Pm)	100 W
Voltaje en máxima potencia (Vm)	18 V
Intensidad en máxima potencia (Im)	5.56 A
Voltaje en circuito abierto (Voc)	21.6 V
Intensidad en cortocircuito (Isc)	6.12 A



(a) *Representación gráfica de la transformación DC/DC del regulador MPPT*

(b) *Curvas del regulador PWM*

Figura 2.2: *Curvas V-I y V-P obtenidas con el regulador MPPT y el regulador PWM (VictronEnergy (2014))*

Como resultados de la situación de ejemplo, en el que se carga una batería con una tensión de carga de 13 V, el regulador PWM al no ser un conversor DC/DC, interrumpirá el aporte cuando la tensión de absorción supera la tensión de carga (sumándole también pérdidas de tensión en el cable empleado). Por ello realiza una carga por pulsos como se ha explicado anteriormente. Para el caso del regulador MPPT, el microprocesador del dispositivo detecta inicialmente que la potencia máxima aportada por el módulo es de 100 W, por lo que cuadra el voltaje de entrada de tal manera que siempre intenta mantener dicha potencia constante, dependiendo de la corriente de entrada de los módulos.
Aún así, cabe destacar que estos ensayos de ejemplo se han realizado en condiciones estándar (STC, por sus siglas en inglés Standard Test Conditions: temperatura de las células 25 °C, una irradiación de 1000 W/m^2 y una masa de aire (AM, por sus siglas en inglés Air Mass) de 1.5).
Se puede concluir que las ventajas de cada tipo de regulador varían de forma significativa dependiendo de la temperatura (en el manual de Victron se pueden observar más ensayos relacionados con la temperatura que demuestran la importancia de la misma para los reguladores de carga (VictronEnergy (2014))). Los reguladores PWM son más óptimos para sistemas fotovoltaicos simples y cuando las células de los módulos se encuentran entre 45 y 75 °C (un rango de temperatura bastante elevado). En cambio, los reguladores MPPT deben evitar temperaturas altas, pero ofrecen una mayor flexibilidad en cuanto a las formas de acoplar los módulos solares (permitiendo realizar sistemas muy complejos) y ofrecen un mayor rendimiento del sistema fotovoltaico.

Hoy en día, se siguen desarrollando nuevos algoritmos que permiten mejorar la función del regulador MPPT, debido a que una de sus desventajas es que realiza una medición del punto de máxima potencia de los módulos y lo considera directamente como el punto óptimo del módulo, mientras que generalmente no es así. Un módulo puede generar varios picos de máxima potencia, siendo una de ellos el punto de máxima potencia global (GMPP), mientras que los demás picos son tratados como puntos de máxima potencia locales (Nayak Bhukya und Reddy Kota (2019)). El objetivo de algunas investigaciones es precisamente establecer un algoritmo que sea capaz de encontrar el GMPP, optimizando aún más el funcionamiento del sistema incluso bajo PSC.

2.2. Normativa aplicada a los sistemas fotovoltaicos

2.2.1. Marco europeo y estatal

La Unión Europea a través de varios documentos ha establecido 3 objetivos con respecto a la energía y el cambio climático para el año 2020 (más conocidos como 20-20-20 para 2020):

- Incremento de un 20 % de la eficiencia energética.
- Reducción de las emisiones de GEI en un 20 %.
- Aumento hasta el 20 % de la contribución de las energías renovables en el consumo energético.

A nivel nacional, en el caso concreto de la energía solar, el Plan de Energías Renovables (PER) 2005-2010 ha establecido en dicho periodo un aumento en la superficie instalada de energía solar térmico de 4 200 000 m^2 y, en cuanto a energía solar fotovoltaica, un incremento de potencia instalada de 363 MW. Para ello ha sido vital importancia la modificación del Código Técnico de la Edificación (CTE) en 2006, donde se establece que "en los edificios incluidos en el ámbito de aplicación del CTE se incorporarán sistemas de captación y transformación de la energía solar en energía eléctrica por procedimientos fotovoltaicos para uso propio o suministro a la red" (CTE-Sección HE 5: Contribución fotovoltaica mínima de energía eléctrica).
El PER además, ha alcanzado en el año 2010 una contribución de las fuentes de energía renovables superior al 12,1 % de la demanda total de energía primaria.

Como explica Javier Guerrero en su trabajo, el crecimiento de la potencia instalada por la tecnología solar fotovoltaica ha sido muy superior al esperado. Dicha evolución en el ámbito de la energía solar fotovoltaica ha producido multitud de inversiones, situación que ha impulsado al Gobierno a aprobar el Real Decreto 900/2015 de 9 de octubre, que regula las condiciones administrativas, técnicas y económicas de las modalidades de suministro de energía eléctrica con autoconsumo y de producción con autoconsumo.

2.2.2. Real Decreto 244/2019 Autoconsumo fotovoltaico

Real Decreto (RD) 244/2019, de 5 de abril, por el que se regulan las condiciones administrativas, técnicas y económicas del autoconsumo de energía eléctrica.

Debido al gran desarrollo de las instalaciones fotovoltaicas, la inversión por parte de las empresas distribuidoras y el comercio generado de las mismas a particulares, el Gobierno da un paso realizando una propuesta de un balance neto a través de este RD. En el documento se detalla que, en lugar de compensar kilovatios, se va a recibir un saldo por cada kilovatio vertido a la red y se descontará de la factura eléctrica.
Sin embargo, este RD sólo afecta a instalaciones realizadas en hogares e industrias con conexión a la red eléctrica. No se contemplan los casos de sistemas fotovoltaicos aislados o con objeto a la venta de energía eléctrica.

Dentro del RD se definen dos modalidades de autoconsumo: con o sin excedentes. En el caso de un autoconsumo sin excedentes, se debe implementar un equipo de antivertido para asegurar que no se vierte ningún excedente a la red eléctrica. Para el caso de un autoconsumo con excedentes, el RD distingue dos bloques: acogidos a compensación (para potencias no superiores a 100 kW que vertirán a red en compensación de la factura eléctrica) o no acogida a compensación simplificada (para potencias superiores a 100 kW, que vierten a red pero sólo a régimen de venta, no de compensación).

2.2.3. UNE-EN IEC 61730-1:2019

Cualificación de la seguridad de los módulos fotovoltaicos (FV). Parte 1: Requisitos de construcción.
Los módulos fotovoltaicos deben ser aptos para operación en ubicaciones exteriores no protegidas por el clima, expuestas a radiación solar completa o parcial (albedo), en un rango de temperaturas entre al menos -40 °C y 40 °C y hasta el 100 % de humedad relativa así como lluvia. Los módulos deben cumplir con los ensayos específicos descritos en la norma IEC 61730-2.

Además, los módulos deben llevar los siguientes componentes electrónicos:

- **Cableado interno** (por ejemplo, las células solares e interconexiones entre células): debe tener la capacidad suficiente para soportar la corriente de aplicación correspondiente. Se comprueba mediante el ensayo de sobrecarga por corriente inversa (MST 26).
- **Cableado externo y cables de salida**: deben cumplir los requisitos de la Norma EN 50618.
- **Conectores**: los conectores externos DC deben cumplir los requisitos de la Norma IEC 62852.
- **Cajas de conexión para módulos fotovoltaicos**: deben cumplir con los requisitos de la Norma IEC 62790.
- **Superestrato y sustrato**: capas de materiales en forma de láminas o películas que proporcionan aislamiento eléctrico, mecánico o térmico suficiente para soportar dichos esfuerzos. Deben cumplir con el ensayo marcado en la Norma IEC 61730-2.
- **Barras aislantes**: una barrera aislante polimérica debe soportar todos los esfuerzos mecánicos, eléctricos, térmicos y ambientales. Deben cumplir con el ensayo marcado en la Norma IEC 61730-2.
- **Conexiones eléctricas**: deben estar diseñadas de tal manera que la presión de contacto no sea transmitida a través de un material aislante distinto a la cerámica, mica pura u otro material con las características apropiadas. La conformidad se verifica mediante inspección visual (MST 01), ensayo de conexiones equipotenciales (MST 13) y ensayo de conexiones atornilladas (MST 33).
- **Encapsulante**: las propiedades técnicas deben ser apropiadas para la aplicación a la que está destinado. En particular el rango de la temperatura nominal de operación y el grupo de material, la resistencia de aislamiento y resistencia dieléctrica.

- **Diodos de paso**: deben ser seleccionados para soportar la intensidad y tensión para su uso previsto. La conformidad se verifica mediante el ensayo térmicos de diodo de paso (MST 25), ensayo de puntos calientes (MST 22), ensayo de funcionalidad del diodo de paso (MST 07) y ensayo de inspección visual (MST 01).

2.2.4. UNE-EN IEC 61730-2:2019

Cualificación de la seguridad de los módulos fotovoltaicos (FV). Parte 2: Requisitos para ensayos.
Para garantizar el correcto funcionamiento de los elementos que componen un módulo fotovoltaico, se debe demostrar que ha superado una serie de ensayos cuya secuencia permite verificar su seguridad. Este procedimiento se realiza una vez evaluada la construcción del módulo por la Norma IEC 61730-1. Los tipos de ensayos quedan definidos en la tabla mostrada a continuación. Los ensayos MQT vienen bien definidos en el trabajo de Javier Guerrero (Guerrero (2016)).

Tabla 2.2: Ensayos de seguridad para el correcto funcionamiento de los módulos fotovoltaicos según la Norma IEC 61730-2

MST	Título del ensayo	Descripción o referencia
01	Inspección visual	Detectar y documentar cualquier defecto visual y cambios en el módulo.
02	Ensayo de funcionamiento en STC	Verificar los valores de corriente de cortocircuito y tensión en circuito abierto en condiciones estándar (temperatura de las células 25 °C, una irradiación de 1000 W/m^2 y una masa de aire de 1.5)
03	Determinación de la potencia máxima	Verificación que el módulo muestra las características eléctricas de un dispositivo plenamente útil.
04	Ensayo de espesor de materiales	Cumplimiento del espesor mínimo de aislamiento para capas delgadas.
05	Durabilidad del marcado	Comprobación del marcado, que debe ser duradero y legible.
06	Ensayo de bordes afilados	Comprobación de que los bordes no sean afilados ni tengan rebabas para evitar daños con otros aislamientos conductores o daños mayores.
07	Ensayo de funcionalidad de los diodos de paso	Procedimiento equivalente a MQT 18.2 de la Norma IEC 61215-2.
11	Ensayo de accesibilidad	Comprobar si proporcionan una protección adecuada contra la accesibilidad a partes activas peligrosas ($>$ 35 V).

MST	Título del ensayo	Descripción o referencia
12	Ensayo de susceptibilidad al cortado	Comprobar que las partes superficiales son capaces de soportar el manejo rutinario de instalación sin provocar choques eléctricos a las personas.
13	Ensayo de continuidad de la conexión equipotencial	Verificar la ruta continua entre las partes conductoras accesibles que están en contacto directo.
14	Ensayo de impulsos de tensión	Verificar la capacidad de aislamiento del módulo para soportar sobretensiones de origen atmosférico.
16	Ensayo de aislamiento	Determinar si el módulo está suficientemente aislado entre las partes portadoras de corriente y los componentes accesibles.
17	Ensayo de corriente en fugas en mojado	Procedimiento equivalente a MQT 15 de la Norma IEC 61215-2.
21	Ensayo de temperatura	Determinar temperaturas de referencia máxima para varios componentes y materiales del módulo.
22	Ensayo de puntos calientes	Procedimiento equivalente a MQT 09 de la Norma IEC 61215-2.
23	Ensayo de fuego	Comprobar características de resistencia al fuego para el caso de una exposición directa a ella.
24	Ensayo de inflamabilidad	Determinar la inflamabilidad del módulo aplicando una llama bajo una condición de irradiancia creo. Método de ensayo basado en la Norma ISO 11925-2.
25	Ensayo térmico del diodo de paso	Procedimiento equivalente a MQT 18 de la Norma IEC 61215-2.
26	Ensayo de sobrecarga de corriente inversa	Bajo unas condiciones de fallo donde las células fotovoltaicas tienden a disipar la energía, se comprueban el riesgo de ignición o de fuego.
32	Ensayo de rotura del módulo	Comprobación de que los daños por corte o perforación puedan ser minimizados en caso de rotura del módulo.
33	Ensayo de conexiones atornilladas	Se comprueban que los tornillos cumplan con el par de apriete óptimo tabulado según el ensayo.

MST	Título del ensayo	Descripción o referencia
34	Ensayo de carga mecánica estática	Procedimiento equivalente a MQT 16 de la Norma IEC 61215-2.
35	Ensayo de pelado	Cualificar el aislamiento como unión cementada.
36	Ensayo de cizalladura	Cualificar el aislamiento como unión cementada.
37	Ensayo de deslizamiento de materiales	Validar que los materiales usados en los módulos no pierdan adhesión cuanto funcionen a altas temperaturas.
42	Ensayo de robustez de terminales	Procedimiento equivalente a MQT 14 de la Norma IEC 61215-2.
51	Ensayo de ciclos térmicos	Procedimiento equivalente a MQT 11 de la Norma IEC 61215-2.
52	Ensayo de humedad congelación	Procedimiento equivalente a MQT 12 de la Norma IEC 61215-2.
53	Ensayo de calor húmedo	Procedimiento equivalente a MQT 13 de la Norma IEC 61215-2, pudiendo omitir MQT 15.
54	Ensayo UV	Procedimiento equivalente a MQT 10 de la Norma IEC 61215-2.
55	Ensayo de acondicionamiento de frío	Aplicabilidad al grado de contaminación PD = 1. Este ensayo está en concordancia con la Norma IEC 61010-1 y adaptado a módulos fotovoltaicos.
56	Ensayo de acondicionamiento de calor seco	Aplicabilidad al grado de contaminación PD = 1. Este ensayo está en concordancia con la Norma IEC 61010-1 y adaptado a módulos fotovoltaicos.

Capítulo 3

Materiales y métodos

3.1. Introducción

En este capítulo se exponen todos los elementos empleados para realizar el sistema fotovoltaico así como la descripción de los métodos de comunicación del entorno ROS y la justificación del uso del lenguaje de programación Python.
Este capítulo está organizado en dos apartados. El primer apartado consiste en una introducción tanto de los elementos físicos que componen el sistema fotovoltaico como del entorno ROS y el lenguaje de programación que se emplean para el software de lectura de datos del regulador de carga. El segundo apartado desarrolla la descripción de los elementos físicos del sistema fotovoltaico, detallando sobre todo los aspectos fundamentales del regulador de carga que se va a emplear, debido a que el sistema de baterías ya está definido en el vehículo y los módulos solares se han empleado en proyectos anteriores, como el de Javier Guerrero (Guerrero (2016)).

3.1.1. Sistemas fotovoltaicos

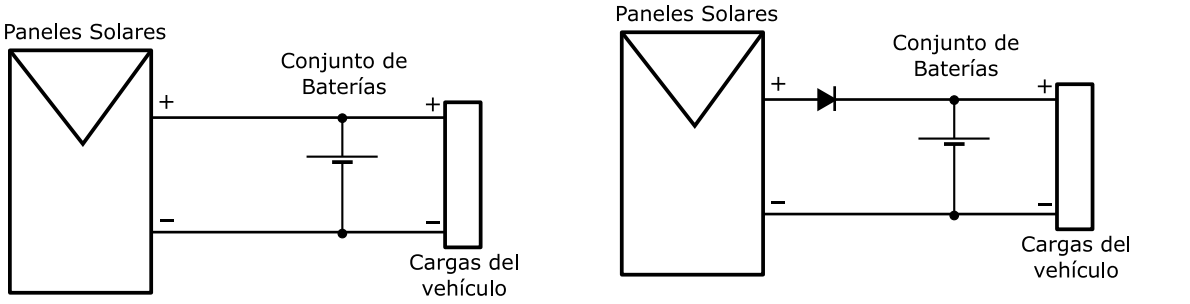
Un sistema fotovoltaico es un sistema compuesto que transforma la energía solar en una diferencia potencial y la emplea para atender ciertas demandas como, en el caso de este trabajo, la carga de un sistema de baterías.
En la actualidad se encuentran disponibles varios tipos de configuraciones de sistemas fotovoltaicos, siendo más eficientes y precisos conforme se incorporan más elementos de control y la calidad de los elementos que los componen se desarrolla y mejora.

A simple vista, el sistema parece poder limitarse a unos paneles solares colocados directamente a las baterías (véase la Figura 3.1a) consiguiendo que el coste de esta configuración sea muy reducido. Sin embargo, esta configuración no es correcta ni aconsejable debido a que puede generar varios problemas. Uno de los problemas más graves ocurre cuando las baterías poseen mayor voltaje que los paneles solares, invirtiéndose la corriente del sistema y generando daños en los paneles.
Este inconveniente puede resolverse agregando un diodo (Figura 3.1b) que controle la dirección de la corriente del circuito. Aún así, hay otros problemas que no logran resolverse con esta configuración (Liu und Makaran (2009)):

1. Los paneles solo pueden proporcionar una corriente de carga cuando su voltaje supera al de las baterías.

2. La corriente de carga se ve muy influenciada según la diferencia de voltaje entre el sistema de paneles y el sistema de baterías, ya que al estar cargando, el voltaje de los paneles se ve disminuido y el sistema de baterías aumenta su voltaje.

3. El sistema en general no siempre trabajará de la forma más óptima posible para transformar toda la potencia posible en electricidad.



(a) *Configuración de conexión directa*

(b) *Configuración de conexión controlada con diodo*

Figura 3.1: *Configuraciones ineficientes y problemáticas de sistemas fotovoltaicos*

Además de lo enumerado, este segundo tipo sistema sigue siendo bastante vulnerable a cambios bruscos del voltaje de los paneles solares. Dicha inestabilidad no es aconsejable en un sistema de carga ya que puede provocar daños irreversibles en el grupo de baterías y, lo más crítico, poner en grave riesgo a las personas que monten el vehículo.

Por ello, un sistema fotovoltaico seguro es aquel definido y controlado por un regulador de carga continua (también conocidos como reguladores DC/DC) ya que, en nuestro caso, el regulador alimenta directamente el sistema de baterías (de corriente continua) del vehículo eléctrico, mediante un algoritmo de control.

3.1.2. Middleware robótico ROS

Para la introducción al entorno ROS, expuesto en este apartado, se ha tenido como referencia el libro "Programming Robots with ROS"(Morgan Quigley (2015)) de introducción a la programación con ROS.

ROS es un metasistema operativo que agrupa una serie de herramientas, librerías y convenciones con el fin de escribir un código que sea fácil de manejar e interpretar por una amplia serie de robots y tratar de simplificar la comunicación entre ellos.

Esto se debe a que crear un código robusto que comunique todas las órdenes de la manera que se desea es difícil y complejo. Desde la perspectiva de un robot o vehículo autónomo, pueden surgir varios problemas que parecen triviales para el ser humano, pudiendo derivar en actuaciones que distan de las órdenes configuradas.

A continuación se van a introducir algunas de las funciones primarias del entorno ROS.

Ros Graph (Organización dentro del sistema ROS)

Uno de los desafíos que incentivó al diseño del entorno fue el problema de "búsqueda de cierto elemento". Este problema consiste en pedir a un robot complejo a que navegue por el entorno donde se encuentra alojado, encuentre cierto objeto y lo envíe a una dirección solicitada.
Esta misión ha dado pie a los objetivos principales del diseño de ROS:

- La tarea principal puede descomponerse en varios sub-sistemas, como pueden ser la navegación, la visión computacional, etcétera.
- Estos sub-sistemas pueden ser aprovechados por otras tareas. Pueden tratarse de envíos de mensajes al sub-sistema o lectura de estados del mismo, entre otros.
- Con el conocimiento y el hardware necesario, se pueden recrear la mayoría de las aplicaciones en cualquier tipo de robot.

Todos estos objetivos se pueden ilustrar con una de las funciones de renderizado que incluye el entorno ROS para cualquier sistema y es su esquema gráfico; como el representado en la Figura 3.2. Los nodos representan los programas individuales del sistema, los rectángulos señalan los tópicos (canales de mensajes definidos de cada programa individual, que pueden registrarse por el sistema ya sea para guardar datos de interés como para observar ciertos estados), y las comunicaciones que establecen entre nodos vienen representados por las flechas.

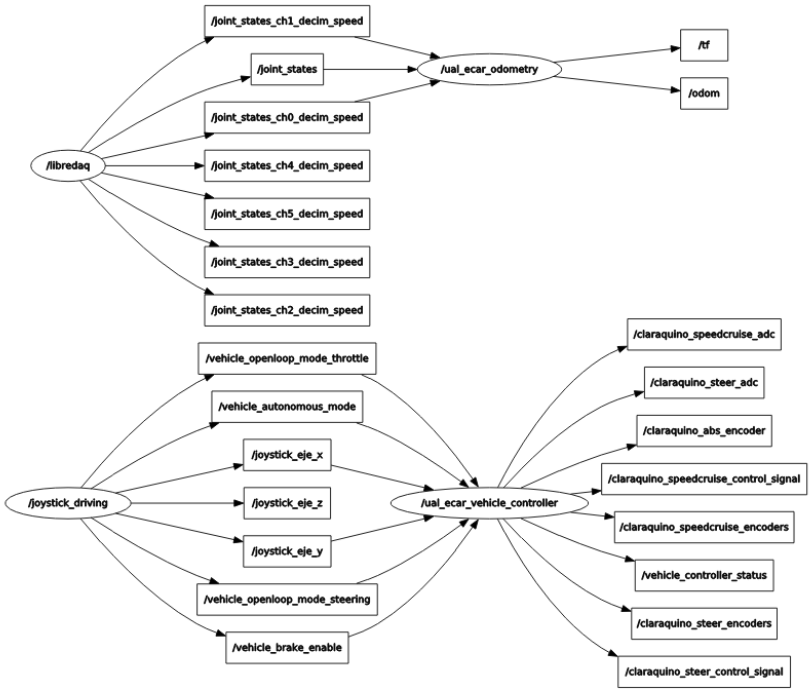


Figura 3.2: *RosGraph del sistema ROS del vehículo UAL-eCARM*

Estos esquemas son muy útiles, sobre todo para situaciones en las que interactúan varios robots y se tenga una red de comunicaciones muy extensa; resultando complicado tratar de visualizar dónde se puede encontrar algún problema de comunicación o dónde poder obtener datos de interés, por ejemplo.

Roscore (Comunicaciones dentro del sistema ROS)

Roscore es un servicio que permite que los nodos de un sistema ROS se puedan encontrar. Cada nodo se conecta con roscore y éste empieza a registrar los detalles de los mensajes que se publican y entre qué nodos se debe establecer una comunicación.

Todo sistema ROS necesita un servicio roscore ejecutado para que los nodos puedan encontrar a los demás nodos del sistema e intercambiar la información necesaria.
Un aspecto fundamental del sistema roscore es que establece sus comunicaciones en una red entre pares, más conocido como P2P (peer-to-peer). Esto quiere decir que la red del sistema ROS establece una comunicación directa entre los nodos, sin necesidad de un servicio intermedio.
Además, los nodos sólo emplean el servicio roscore para saber a qué "par"deben comunicarse, como se muestra en la Figura 3.3.

En definitiva, la arquitectura de ROS es un híbrido entre un sistema clásico cliente/servidor y otro sistema totalmente distribuido.

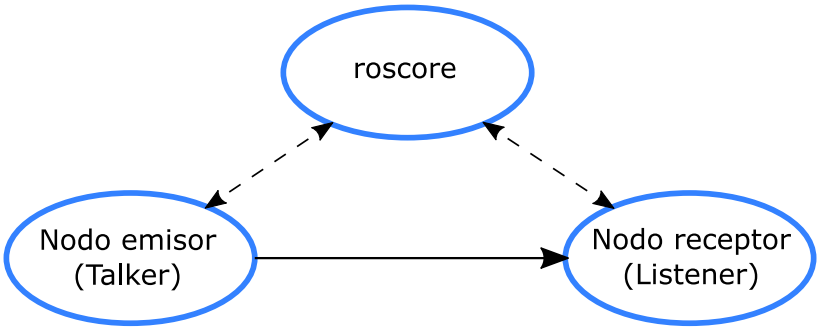


Figura 3.3: *Esquema básico de comunicaciones entre dos nodos, con el servicio roscore*

Catkin, Espacios de Trabajo y Paquetes ROS

Catkin es el sistema de compilación de ROS, es decir, el sistema que contiene todas las herramientas necesarias para generar los programas ejecutables, las librerías a emplear y crear las interfaces que otros códigos o programas puedan utilizar.

Catkin Catkin está compuesto por un conjunto de macros definidas en CMake y los propios scripts elaborados en Python para proporcionar funcionalidades extras y personalizadas al funcionamiento normal de CMake. CMake es uno de los compiladores multiplataforma más empleados en la programación.
Mediante Catkin se asignan las herramientas necesarias para realizar el sistema ROS.

Espacios de Trabajo (Workspace) Antes de comenzar a preparar cualquier código se requiere establecer un espacio de trabajo, más conocido como su denominación en inglés: workspace. Un workspace es un conjunto de directorios donde se crean y definen todos los elementos del sistema ROS, entre ellos, los diferentes scripts de cada nodo.

Paquetes ROS Un sistema ROS se organiza internamente en paquetes. Cada paquete contiene su código, sus valores y su documentación. Los paquetes se encuentran alojados en el directorio src (source) del workspace. Cada uno de estos paquetes debe contener dos archivos base, que describen el contenido de dicho paquete: *CMakeLists.txt* y *package.xml*.

Rosbag (Análisis de registros y guardado de datos)

Se trata de la herramienta que permite guardar los mensajes de los tópicos señalados para registrar y depurar las acciones del código.

Con Rosbag se evita precisamente tener que hacer un doble trabajo en la creación de nodos, ya que con realizar el emisor que envía los mensajes vía cierto tópico o varios, el servicio actuará como receptor de dichos valores para su posterior registro y guardado. Rosbag permite seleccionar los tópicos que se desean guardar y, una vez terminado el registro, genera un archivo .bag que posteriormente se puede usar para extraer los valores registrados de cada tópico.

Los tópicos (Comunicación entre nodos)

Los nodos en sí no son de mucha utilidad en la mayoría de los casos. Lo realmente interesante son las comunicaciones que realizan entre sí. La manera más frecuente de establecer estas vías de comunicación es mediante los tópicos.

Los tópicos implementan un sistema de comunicación "publish/subscribe", uno de los métodos más comunes para intercambiar datos en un sistema distribuido. Previo al envío de los propios datos, el nodo comunica al sistema ROS qué tipos de datos va a tratar de enviar, junto con el nombre del tópico. Los nodos que quieran subscribirse a dicho tópico envían una petición al servicio roscore. Una vez suscritos, todos los mensajes enviados del nodo se envían al nodo receptor.

La gran ventaja de los tópicos en ROS es que el propio sistema se encarga de toda la comunicación interna del sistema, ahorrando dicho trabajo al desarrollador del sistema.

3.1.3. Lenguaje de programación Python

Para explicar el origen y las ventajas del lenguaje, se ha realizado la siguiente introducción teniendo como referencia el libro de Toby Donaldson "Python" (Donaldson (2008, c2009)) que introduce y desarrolla las bases del lenguaje Python.

Python es un lenguaje de programación compuesto por un conjunto de librerías y herramientas de software. Se desarrolló a principios de los 90 por Guido van Rossum, quien le dió ese nombre inspirado por el grupo de comedia "Monty Python", o también conocidos como "Los Python"; y actualmente se mantiene por miles de desarrolladores y programadores, incluido van Rossum.

El lenguaje fue diseñado con la idea de ser sencillo de leer y de aprender. Comparado con otros lenguajes de programación como C++ o Java, Python tiene una estructura más ordenada y clara. Además, posee una simbología mucha más acotada, eliminando varios que considera innecesarios y posee una sintaxis más directa y clara con palabras anglosajonas, en vez de emplear una estructura más complicada como Ruby o Perl.

Python es un lenguaje bastante productivo porque permite también una programación orientado a objetos, sin llegar a forzar al usuario a utilizarlo. Otra de sus ventajas es su mantenimiento y es que, debido a toda la comunidad de programadores que tiene detrás, Python es un lenguaje que está constantemente actualizado debido a su facilidad de revisión.

Aplicaciones del lenguaje

Python es un lenguaje que se puede usar para cualquier tipo de programa que uno desea realizar. Comúnmente tiene las siguientes aplicaciones:

- **Scripts**: programas cortos y automáticos que se suelen emplear para múltiples tareas como cargas y descargas a cierta página web sin necesidad de utilizar un navegador, reordenación de archivos en una cierta ruta de tu ordenador y un largo etcétera. Debido a su simpleza, suele ser una opción utilizada en muchos productos.

- **Desarrollo web**: Varios proyectos web realizados con Python son bastante populares por su gran desarrollo y opciones en cuanto a herramientas, como por ejemplo, la página de www.reddit.com.

- **Procesado de texto**: Python tiene un soporte excelente para manejar e interpretar cadenas de texto, incluyendo expresiones generales y caracteres Unicode.

- **Programación científica**: Python ofrece varias librerías dedicadas para la investigación, como pueden ser librerías de estadística, tratamiento de datos y representación gráfica.

- **Educación**: Debido a su gran utilidad y sencillez, es cada vez más frecuente su implementación en los planes de educación.

3.2. Elementos del sistema fotovoltaico

El sistema fotovoltaico diseñado está compuesto por 4 paneles fotovoltaicos, conectados a través de un regulador de carga con seguidor de punto de máxima potencia (MPPT) al sistema de baterías del vehículo eléctrico.
El esquema eléctrico planteado es como sigue en esta figura:

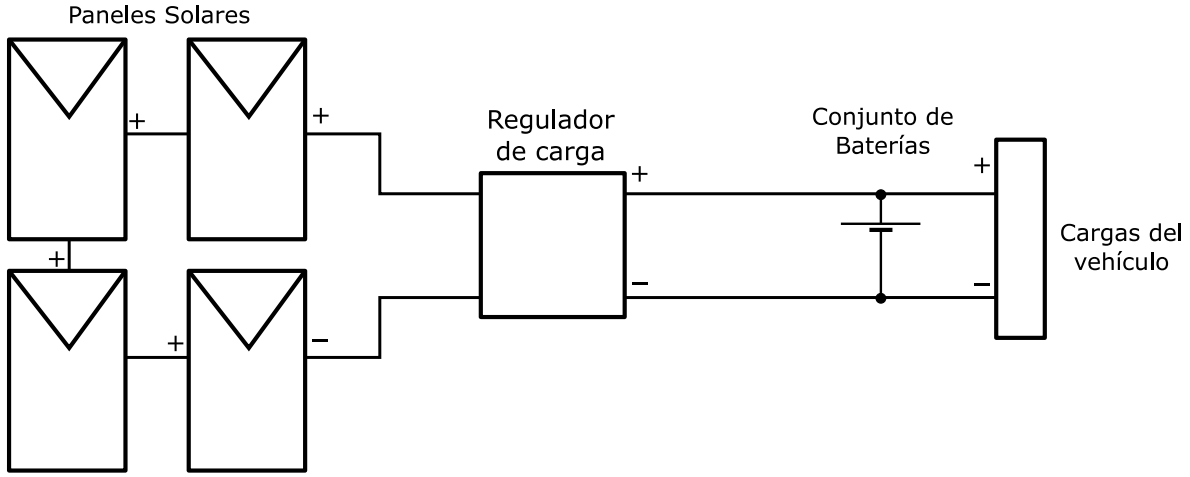


Figura 3.4: *Esquema eléctrico del sistema fotovoltaico planteado para el vehículo eléctrico*

Dicho esquema es independiente de cualquier otro sistema del vehículo y enviará los datos registrados mediante un nodo implementado en el sistema ROS del vehículo.
La comunicación del nodo para la lectura de datos se realizará mediante un cable que transmite los registros.

A continuación, se explica con mayor profundidad las características de cada elemento que compone el sistema.

3.2.1. Sistema de baterías

El sistema está compuesto por 6 baterías de gel electrolítico de ácido-plomo reguladas por válvula (VRLA) marca Trojan (Figura 3.5) de 6 V cada una, dispuestas en serie para formar un sistema de 48 V nominales (disposición contemplada en la hoja de datos de la batería).

Figura 3.5: *Batería gel VRLA de 6 V marca Trojan*

Con esta disposición (véase la Figura 3.6) se garantizan menores demandas de corriente que con sistemas más pequeños (de 24 V ó 12 V) para la misma potencia solicitada por las diversas cargas del vehículo eléctrico. Esto quiere decir que se evitan sobrecalentamientos y problemas de dimensionamiento en el sistema eléctrico del vehículo.

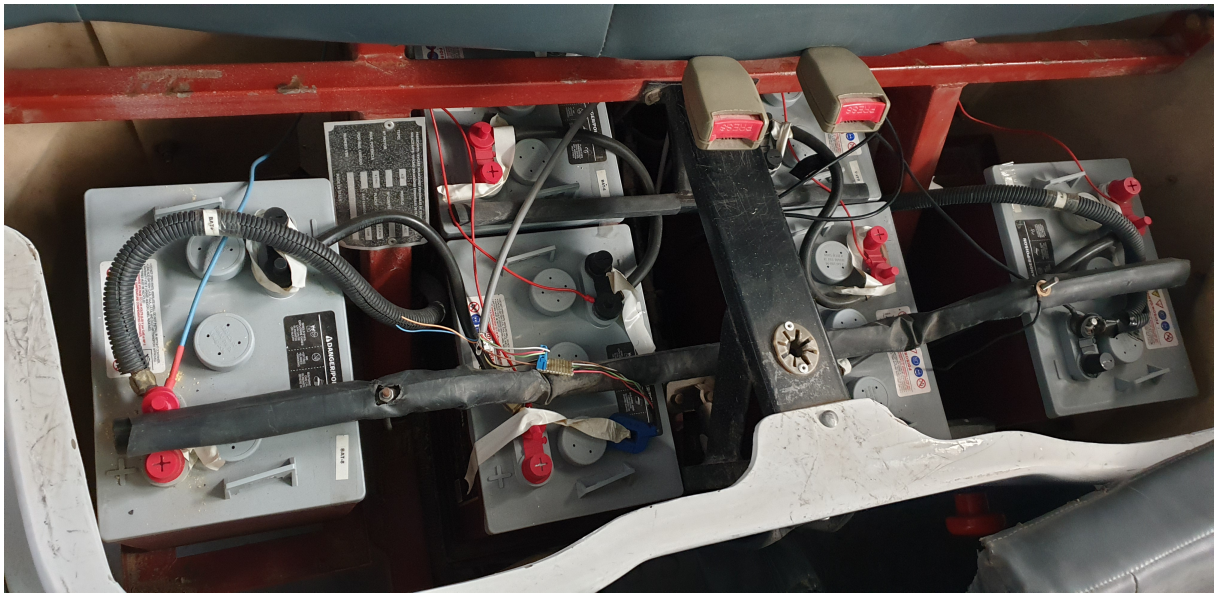
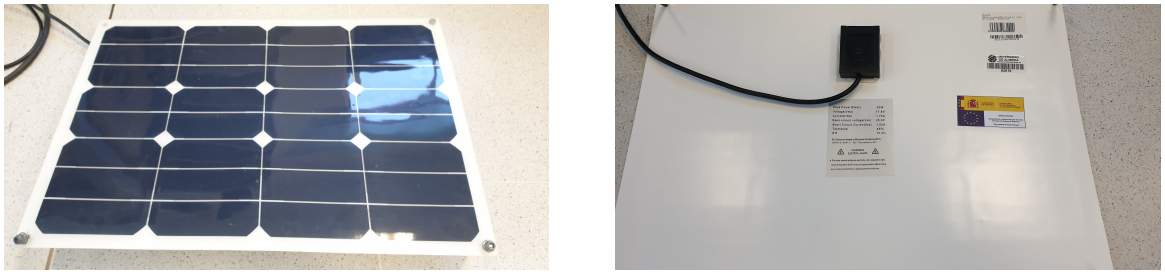


Figura 3.6: *Disposición de las baterías en el vehículo UAL e-CARM*

3.2.2. Paneles solares

El sistema de paneles solares consta de 4 módulos curvables de la marca Techo Sun modelo 30W12V (mostrada en la Figura 3.7) compuesto de 32 celdas de silicio monocristalino que tienen una eficiencia en torno al 19,6 %.



(a) *Parte de las celdas fotovoltaicas* (b) *Parte trasera del módulo fotovoltaico*

Figura 3.7: *Módulo curvable Flex 30W12V de Techno Sun*

Los paneles solares tienen un voltaje en circuito abierto de 20.7 V (las características de los módulos fotovoltaicos se muestran en la tabla 3.1), por lo que para conseguir un voltaje que supere los 48 V nominales del sistema de baterías deben estar dispuestos en serie (mostrado en la tabla 3.2); tal y como se ha esquematizado en la Figura 3.4.

Tabla 3.1: Aspectos del módulo solar Techo Sun modelo 30W12V

Característica	**Valor**
Potencia máxima (Pmáx)	30 W
Voltaje en máxima potencia (Vmp)	17.6 V
Intensidad en máxima potencia (Imp)	1.70 A
Voltaje en circuito abierto (Voc)	20.8 V
Intensidad en cortocircuito (Isc)	1.82 A
Tolerancia	± 5 %
Eficiencia (Eff)	19.2 %

Tabla 3.2: Valores de voltaje e intensidad en circuito abierto según el tipo de montaje de los módulos

Configuración	**Característica**	**Valor**
2 líneas en paralelo, cada línea compuesta de 2 paneles en serie	Voltaje en circuito abierto (Voc)	41.6 V
	Intensidad en corto-circuito (Isc)	3.64 A
4 paneles en serie	Voltaje en circuito abierto (Voc)	83.2 V
	Intensidad en corto-circuito (Isc)	1.82 A

3.2.3. Regulador de carga solar

El regulador seleccionado es un regulador de carga con punto de seguimiento de máxima potencia (MPPT), mostrado en la Figura 3.8, que permite una tensión máxima de circuito abierto de hasta 150 V del sistema de paneles y una corriente máxima de la batería de 35 A (con una corriente máxima de cortocircuito de 40 A). Por lo tanto, es un regulador apto para nuestro sistema fotovoltaico de 82.8 V en circuito abierto y 1.82 A en cortocircuito.

Figura 3.8: *Regulador de carga MPPT de Victron Energy, modelo de 150 V y 35 A*

Según la documentación del regulador de carga proporcionado por la empresa Victron Energy (VictronEnergy (2019a)), este regulador de carga acepta sistemas de baterías de 12, 24 y 48 V nominales y el controlador del mismo ajustará la tensión nominal dependiendo de cuál es el sistema al que se acopla.

El modo de seguimiento ultrarrápido del punto de máxima potencia realiza una mejora de hasta el 30 % en cuanto a la recogida de la energía solar, sobre todo en cielos nublados donde la intensidad de la luz cambia continuamente, en comparación con otros tipos de controladores de carga. Además, en los casos donde se puedan dar dos o más puntos de máxima potencia en la curva de tensión de carga, el regulador trata de bloquearse en el punto de máxima potencia más óptimo.

En cuanto a medidas de protección, el regulador permite una potencia nominal completa con una resistencia térmica de hasta 40 °C, siendo la eficiencia máxima superior al 98 %. Incluye protección frente a:

- Sobrecalentamientos, reduciendo la potencia cuando se presentan temperaturas muy elevadas.
- Cortocircuito y polaridad inversa en los paneles solares.
- Corriente inversa de los paneles.

El regulador de carga contiene un sensor de temperatura interno para realizar las compensaciones de las tensiones de carga en absorción y flotación, en función de la temperatura.

Carga adaptativa en tres fases

El regulador de carga está configurado para llevar a cabo procesos de carga en tres tipos de fases: inicial, absorción y flotación.

- **Inicial**: Durante esta fase, el controlador suministra tanta corriente de carga como le sea posible, para recargar las baterías rápidamente.
- **Absorción**: Cuando la tensión de la batería alcanza la tensión de absorción predeterminada, el controlador cambia a modo de tensión constante. Cuando la descarga es escasa, la fase de absorción se acorta para evitar una sobrecarga de la batería. Después de una descarga elevada, el tiempo de carga de absorción aumenta automáticamente para garantizar que la batería se recargue completamente. Además, el periodo de absorción también se detiene cuando la corriente de carga disminuye a menos de 2 A.
- **Flotación**: Durante esta fase se aplica la tensión de flotación a la batería para mantenerla completamente cargada. Cuando la tensión de la batería cae por debajo de la tensión de flotación durante al menos 1 minuto, se iniciará un nuevo ciclo de carga.

Algoritmos preprogramados

El MPPT incorpora 8 algoritmos de control preprogramados, detallados en la documentación del mismo (VictronEnergy (2019a)). Los valores del algoritmo predeterminado se muestran a continuación en la Tabla 3.3. El algoritmo deseado se puede seleccionar manualmente mediante un interruptor giratorio ubicado en el propio dispositivo.

Tabla 3.3: Valores del algoritmo predeterminado en el MPPT 150/35 (VictronEnergy (2019a))

Característica	**Valor**
Tensión en Absorción	57.6 V
Tensión en Flotación	55.2 V
Ecualización a $\%I_{nom}$	64.8 al 8 %
Compensación Temp. (dV/dT ó mV/°C)	-32

Según lo indicado en la documentación, la ecualización automática está apagada por defecto. Mediante la aplicación de VictronConnect se puede configurar para que realice una ecualización desde cada día hasta cada 250 días.
Cuando la ecualización automática está activada, la carga de absorción irá seguida de un periodo de corriente constante con tensión limitada. La corriente está limitada al 8 % de la corriente inicial para el tipo de batería ajustado de fábrica.
La ecualización automática termina cuando se alcanza el límite de tensión 16.2 V o tras t = (tiempo de absorción)/8, lo que ocurra primero.
Para el caso de la instalación específica en este trabajo, al tener un sistema de baterías Gel VRLA, el manual indica que no se deben ecualizar dicho tipo de baterías. Por este motivo, se ha dejado la opción desactivada.

Carga de las baterías

El regulador de carga inicia un nuevo ciclo de carga cada día. El tiempo de absorción máximo viene determinado precisamente por la lectura inicial que realiza el regulador de carga al comienzo del día del voltaje de las baterías, que vienen definidas en la Tabla 3.4.

Si el periodo de absorción se interrumpiera debido a la nubosidad o a una carga muy elevada, el proceso de absorción se reanudaría al alcanzarse la tensión de absorción en otro momento del día, hasta haberse completado el periodo de absorción.
El periodo de absorción también se interrumpe cuando la corriente de salida del cargador solar cae por debajo de 2 A, que no se debe a que la salida de los paneles solares sea baja, sino a que la batería está completamente cargada (corte de la corriente de cola).

Tabla 3.4: Valores predeterminados de la duración máxima del MPPT 150/35 (VictronEnergy (2019a))

Tensión de la batería Vb al inicio	**Tiempo máximo de absorción**
Vb < 47.6 V	6 h
47.6 V < Vb < 48.8 V	4 h
48.8 V < Vb < 50.4 V	2 h
Vb > 50.4 V	1 h

Capítulo 4

Integración del sistema fotovoltaico en el vehículo eléctrico

4.1. Procedimientos seguidos para la implementación del sistema

Uno de los principales objetivos del proyecto es que el sistema del vehículo eléctrico pueda recibir los datos del regulador de carga, por lo que, previo a la instalación de los elementos del sistema en el vehículo eléctrico, se procede a crear un programa que realice esa función.

La creación del programa se ha dividido en dos versiones diferentes:

En la primera, el programa debe ser capaz de leer la información que emite el regulador de carga mediante el protocolo VE.Direct (VictronEnergy (2019d)) y guardar los registros en un fichero.

En la segunda versión del programa, se trabaja con el entorno ROS debido a que el vehículo presenta ese sistema de comunicaciones de todos los elementos que ya contiene (sensores, vatímetros, actuadores, etc). En esta versión, el programa debe ser capaz de leer la información indicada del MPPT (también mediante el protocolo VE.Direct) y enviar los registros de forma periódica a un tópico[1] definido hacia el sistema ROS del vehículo.

El motivo por el que se realiza una primera versión del programa de lecturas antes de trabajar con paquetes en entorno ROS es para realizar, de una forma óptima y sólida, el filtrado del mapeado de bytes. Esto es debido a que se caracteriza por ser un mapeado variable cada vez que se solicita un registro, ocasionando inestabilidad en el registro de los datos.

[1]Buses en los cuales los nodos que componen el sistema ROS envían mensajes de ciertos parámetros

4.2. Proceso de lectura del regulador de carga con seguidor de punto de máxima potencia

4.2.1. Comunicación con el protocolo VE.Direct

El protocolo VE.Direct realiza una conexión punto a punto que tiene un mapeado de bytes específico de la compañía Victron Energy.

La conexión se realiza a través de un cable USB a VE.Direct, diseñado por la propia empresa.

Figura 4.1: *Cable Interfaz VE.Direct a USB*

La conexión VE.Direct consiste en un puerto de 4 pines, que son los mostrados en la siguiente tabla:

Tabla 4.1: Pinout del puerto de conexión VE.Direct

Pin	**Producer**	**Consumer**
1	GND	GND
2	VE.Direct-RX	VE.Direct-TX
3	VE.Direct-TX	VE.Direct-RX
4	Power +	Power +

El cable establece una comunicación directa de VE.Direct a una interfaz USB, que también sería posible realizar mediante un conversor del protocolo RS232 a USB conectándolo a través de un conector DB9 (que suelen ser una de las entradas físicas más frecuentes en estos tipos de conversores), siguiendo el esquema de conexiones mostrado en la siguiente figura.

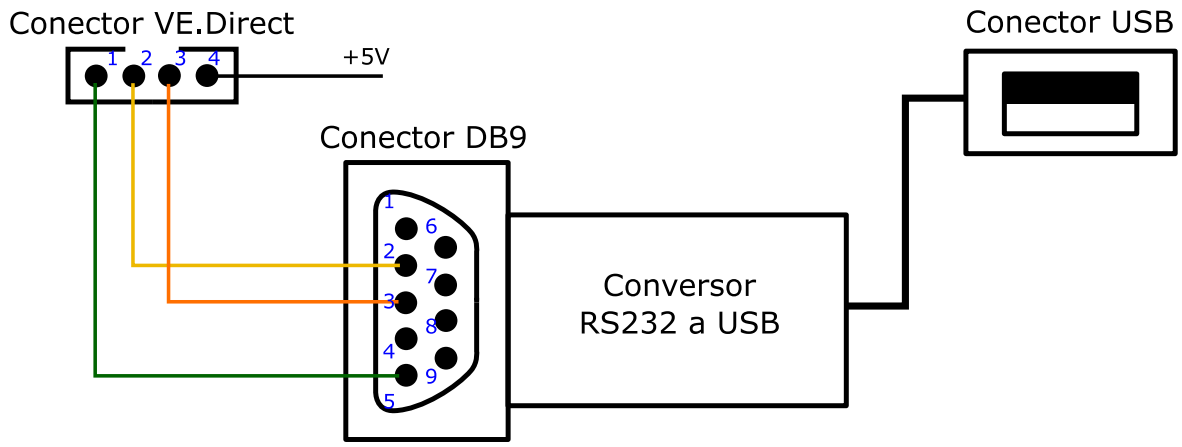


Figura 4.2: *Esquema de conexiones de VE.Direct vía un conversor del protocolo RS232 a USB*

Tabla 4.2: Pinout de los conectores que componen el esquema de conexiones

Conector VE.Direct		**Conversor DB9 a USB**	
Pin	**Señal**	**Pin**	**Señal**
1	GND	5	GND
2	VE.Direct-RX	2	RxD (Received Data)
3	VE.Direct-TX	3	TxD (Transmitted Data
4	Power +		
Entradas restantes sin conexión			
		1	CD (Carrier Detect)
		4	DTR (Data Terminal Ready)
		6	DSR (Data Set Ready)
		7	RTS (Request To Send)
		8	CTS (Clear To Send)
		9	RI (Ring Indicator)

Formato de los mensajes del MPPT

El regulador de carga envía bloques de datos o "tramas" actualizadas cada segundo. Cada trama sigue el siguiente formato, según la hoja de información del protocolo VE.Direct (VictronEnergy (2019d)):

<Nueva línea> <ID de la trama> <Tab> <Valor de la trama>

Definiéndose los identificadores de la siguiente manera:

- <**Nueva línea**> Un retorno de carro seguido de un salto de línea (0x0D, 0x0A).
- <**ID de la trama**> Una etiqueta de tamaño arbitrario en el que se identifica el tipo de valor del que se trata.
- <**Tab**> Tabulación horizontal (0x09).
- <**Valor de la trama**> El valor de la trama en formato ASCII. El número de caracteres depende de la magnitud y el signo del propio valor.

Una muestra del mapeado de bytes del regulador de carga es como la que sigue a continuación:

```
b': A883778866478276388276BBBFFFFF828736728872\r\nPID\t0xA04B\r\
    nFW\t139\r\nSER#\tHQ1540TUFJ5\r\nV\t13310\r\nI\t0\r\nVPV\t0\r
    \nPPV\t0\r\nCS\t0\r\nMPPT\t0\r\nERR\t0\r\nLOAD\tOFF\r\nH19\
    t22700\r\nH20\t236\r\nH21\t734\r\nH22\t302\r\nH23\t875\r\
    nHSDS\t193\r\nChecksum\t\t'
```

El bloque de datos se divide en 17 tramas, separadas por el retorno de carro " \r " y el salto de línea " \n ". A su vez, cada trama se divide en la ID y el valor de la trama, separados por la tabulación " \t ", como se ha comentado antes.

La primera trama corresponde a un mensaje asíncrono en código hexadecimal. Según la estructura del código, el mensaje es distinto y es posible decodificarlo mediante la documentación proporcionada de los códigos hexadecimales del protocolo (VictronEnergy (2019b)). Aún así, como esta trama no suele emitirse frecuentemente, a no ser que haya información de algún estado del dispositivo (sobrecalentamientos, pérdida de potencia, conexiones, etc) y, como dichos estados ya se reflejan en otras tramas (código de error o estado de operación del dispositivo), se ha decidido no profundizar en los mensajes asíncronos.

Las demás tramas contienen valores retenidos (más conocidos en inglés como "holding registers") y son actualizadas cada segundo por el dispositivo. Cada trama viene definido en la Tabla 4.3 según la información aportada por la ficha de datos del protocolo.

Tabla 4.3: Tramas del protocolo VE.Direct

Trama	**Unidades**	**Descripción**
V	mV	Voltaje del sistema de baterías
I	mA	Intensidad del sistema de baterías
VPV	mV	Voltaje de los módulos fotovoltaicos
PPV	W	Potencia de los módulos fotovoltaicos
PID	-	ID del producto
SER#	-	Número de Serie
HSDS	-	Número de Secuencia del día (0 a 365)
MPPT	-	Modo de Operación
ERR	-	Código de Error
CS	-	Estado de Operación
H19	0.01 kW/h	Potencia Total (Reseteable)
H20	0.01 kW/h	Potencia de hoy
H21	W	Potencia máxima de hoy
H22	0.01 kW/h	Potencia de ayer
H23	W	Potencia máxima de ayer

De las tramas definidas en la Tabla 4.3, las más significativas a nivel de lectura del sistema fotovoltaico son:

- El voltaje del sistema de baterías.
- La intensidad del sistema de baterías.
- El voltaje de los módulos fotovoltaicos.
- La potencia de los módulos fotovoltaicos.

Esto se debe a que, con dichos valores, se pueden definir las curvas V-I de los paneles solares y las del sistema de baterías del vehículo.

Hay que destacar que el MPPT no proporciona directamente la intensidad de los módulos, pero se puede obtener mediante la Ley de Watt (4.1) que enuncia que "la potencia consumida es directamente proporcional al voltaje suministrado y a la corriente que circula".

$$P = V \cdot I \rightarrow I = \frac{P}{V} \tag{4.1}$$

Donde P es la potencia de los módulos, y V e I son el voltaje y la intensidad de los módulos, respectivamente.

Los siguientes 2 tramos (PID y SER#) son valores fijos que definen al dispositivo. En este caso, la trama PID señala que se trata de un dispositivo 0xA04B, que, según la tabla de dispositivos del manual del protocolo VE.Direct (VictronEnergy (2019d)) define el modelo BlueSolar MPPT 150/35 rev2, con el número de serie HQ1540TUFJ5 (definido por la trama SER#).

Otra de las tramas de interés es el de los códigos de error (ERR), ya que facilita la identificación de qué es lo que está sucediendo en todo momento en el dispositivo. Mediante los códigos recogidos en el foro de Victron Energy (VictronEnergy (2019c)), se ha podido identificar los posibles errores expuestos en esta tabla:

Tabla 4.4: Códigos de error del MPPT

Código	**Descripción**	**Código**	**Descripción**
0	No hay errores	26	Terminales sobrecalentados
2	Voltaje de baterías demasiado alto	28	Problema en la fase de la potencia
3	Fallo en el sensor remoto de temperatura	33	Voltaje de entrada elevado (paneles)
4	Fallo en el sensor remoto de temperatura	34	Corriente de entrada elevada (paneles)
5	Fallo en el sensor remoto de temperatura (conexión perdida)	38	Entrada desconectada (voltaje de las baterías excesivo)
6	Fallo en el sentido remoto del voltaje de la batería	39	Entrada desconectada
7	Fallo en el sentido remoto del voltaje de la batería	65	Comunicación de advertencia
8	Fallo en el sentido remoto del voltaje de la batería (conexión perdida)	66	Dispositivo incompatible
17	Temperatura del cargador elevado	67	Conexión perdida con el BMS (Battery Monitor System)
18	Corriente elevada en el cargador	114	Temperatura de la CPU elevada
19	Corriente invertida en el cargador	116	Datos de los factores de calibración perdidos
20	Límite excedido en el tiempo de carga	117	Firmware no válido o no compatible
21	Fallo en el sensor de corriente (parcial o roto)	119	Opciones del usuario no válidos

Hay que remarcar que los códigos de error de la Tabla 4.4 sólo aparecen cuando el estado de operación del MPPT está en modo fallo.
El estado de operación se muestra en la trama "CS"del mapeado de bytes y puede estar en uno de los 5 estados mostrados en la Tabla 4.5

También se puede comprobar en qué tipo de seguidor está configurado el MPPT con el código que muestra en el tramo "MPPT". Las 3 opciones se exponen en la Tabla 4.6

Tabla 4.5: Códigos de los estados de operación del MPPT

Código	**Descripción**
0	Not charging (No está cargando)
2	Fault (En fallo)
3	Bulk (En carga)
4	Absorption (En absorción)
5	Float (En flotación)

Tabla 4.6: Códigos de los modos del MPPT

Código	**Descripción**
0	Apagado
1	Limitado en voltaje o corriente
2	Seguidor de punto de máxima potencia activa

Finalmente, la último trama identificada como "Checksum"siempre va a cerrar el paquete de tramas de una misma llamada. El valor que tiene esta trama es un único byte (no necesariamente imprimible como un caracter ASCII).
Para el caso del programa de lecturas, sirve para poder realizar los saltos de línea en los registros dentro del fichero.

4.2.2. Programación de la lectura de datos

Para realizar el programa de lectura de datos se ha empleado el lenguaje Python (en la versión 3.7.3) debido a que con ello se puede trabajar en ROS y tiene una curva de aprendizaje menos costosa que otros lenguajes como, por ejemplo, C++.

La misión de este programa es leer todos los datos procedentes del MPPT y guardalos de forma adecuada en una hoja de cálculo en Excel (fichero de extensión .xls). El módulo empleado para la comunicación con el regulador de carga y la forma de interpretar los tramos se ha inspirado en un programa publicado por un usuario en GitHub que tenía el mismo propósito pero su funcionamiento era inestable (karioja (2017)).

Los módulos de Python empleados en el programa son los siguientes:

- **logging**: para reportar los errores al usuario vía la consola.
- **serial**: para poder realizar la comunicación con el MPPT.
- **ctypes**: para enviar mensajes de texto al usuario.
- **time**: para pausar el programa al final de cada mapeado de bytes.
- **datetime**: para registrar y nombrar el fichero .xls en el momento de ejecutar el programa.
- **xlwt**: para toda la gestión de la hoja de cálculo.

La pausa del programa se realiza cada segundo ya que el regulador de carga actualiza los valores en ese intervalo. De esta forma se está evitando una lectura continua e innecesaria del programa, evitando valores repetidos.

Organización del programa

El programa se ha organizado en 3 archivos:

- `vemppt_main.py`
- `vemppt_parser.py`
- `vemppt_register.py`

El archivo principal del programa es `vemppt_main.py`. En este archivo se importan los módulos del programa definidos anteriormente y es la que define la estructura principal del programa. Su objetivo es crear el nuevo fichero de registros .xls, establecer la comunicación con el MPPT a través del puerto indicado del ordenador y enviar cada trama al archivo `vemppt_parser.py`.
En caso de haber finalizado el programa, este archivo se encarga de cerrar de forma correcta la comunicación con el regulador de carga y el fichero de registros generado.

El archivo `vemppt_parser.py` se encarga de procesar la trama enviada por el archivo principal de tal forma que pueda identificar de qué dato se trata.
Inicialmente decodifica la trama, posteriormente lo filtra por las 17 tramas definidas y, finalmente envía el dato al archivo de registros `vemppt_register.py` ó, en caso negativo, emite una notificación al usuario.
En una situación de conflicto en los datos o sobreescritura, también se notifica al usuario y se realiza un salto de línea en el fichero de registros para evitar que el programa falle.

El tercer archivo, llamado `vemppt_register.py`, es el encargado de procesar los datos filtrados por el archivo `vemppt_parser.py` enviándolos a la ruta definida por éste.

Los procesos del programa se pueden ver de una forma visual en el siguiente esquema, y el programa resultante se puede obtener en el repositorio publicado en GitHub (Poyatos (2019c)):

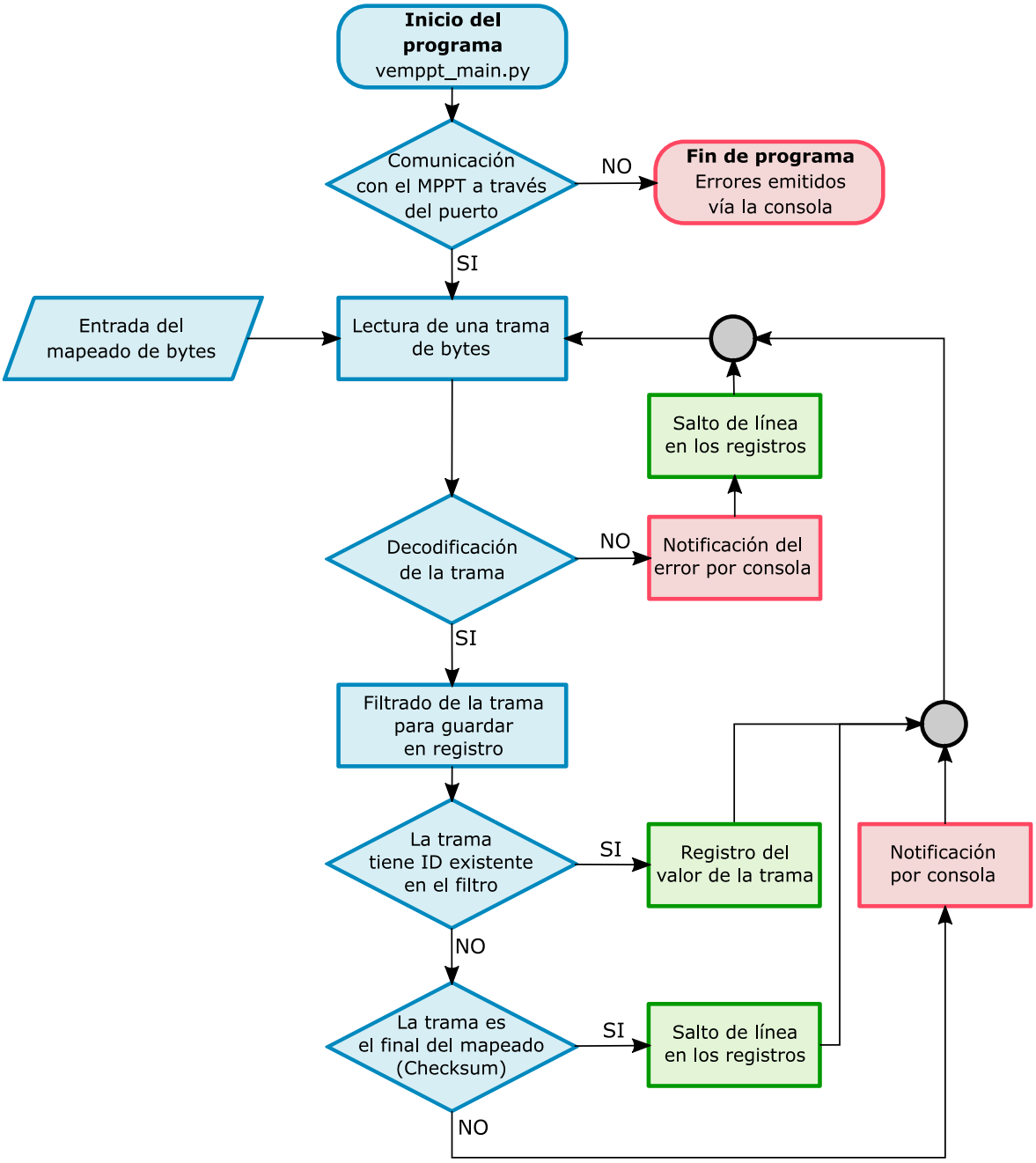


Figura 4.3: *Diagrama de procesos del programa de lectura* ***vemppt_reader***

La decodificación de las tramas es necesario para poder manejar e interpretar los identificadores y valores de cada tramo por el programa. La decodificación se realiza del conjunto de bytes a un formato de codificación de caracteres Unicode UTF-8 (de sus siglas en inglés: 8-bit Unicode Transformation Format).

4.3. Comunicación del regulador de carga con el sistema ROS del vehículo

4.3.1. Instalación del entorno ROS y definición del workspace

Se ha decidido realizar el paquete en Ubuntu Linux, debido a que es un sistema operativo familiar de la distribución Linux y soporta ROS.
En cuanto a la versión del sistema operativo, se ha optado por la 16.04 LTS (versión con soporte) debido a que en esta versión la instalación del entorno ROS se realiza con kinetic, que es la misma instalación que tiene el sistema ROS del vehículo eléctrico. Con ello se evita cualquier problema de compatibilidades, ya que en la versión más reciente de Ubuntu (18.04 LTS) requiere que se emplee el instalador melodic.

Para el caso concreto de este proyecto, se ha empleado una máquina virtual (VirtualBox) para poder trabajar en Ubuntu y realizar el paquete. Una máquina virtual es un software que permite gestionar los espacios del ordenador principal para ejecutar ordenadores virtuales con otros sistemas operativos. Así se permite trabar con Linux en un ordenador que tiene un sistema operativo diferente (como pueden ser Windows o MacOS).
También existe la posibilidad de integrarlo junto con otro sistema operativo, en un mismo ordenador. No obstante, para evitar cualquier fallo que pudiese suceder en la instalación del sistema operativo (pudiendo perder información del otro sistema operativo con el que se comparte) se ha decidido emplear la máquina virtual.

La instalación completa de ROS se ha seguido según se indica en la wikipedia oficial de ROS (ROS.org (2019)).

Instalación del workspace Catkin vía kinetic

Desde el terminal de Ubuntu, se ejecuta el comando de instalación de la versión kinetic:

```
sudo apt-get install ros-kinetic-catkin
```

Posteriormente se instala Catkin como cualquier otro paquete de CMake:

```
mkdir build && cd build && cmake ../ && make && sudo make
    install
```

Se ha comprobado que la instalación se ha realizado correctamente introduciendo el siguiente comando y revisando si coinciden las rutas:

```
$ source /opt/ros/kinetic/setup.bash
```

Una vez verificada la instalación, se procede a crear el workspace de catkin para poder crear el paquete indicado para la lectura del regulador de carga. Para ello se crean previamente la carpeta del workspace (con el nombre de "catkin_ws" por conveniencia) y dentro de ella, la carpeta src donde irán alojadas todas las funciones.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Al ejecutar catkin_make dentro de la carpeta src del workspace, se generará en su interior un CMakeLists.txt y dos carpetas llamadas "build" y "devel". Para configurar el workspace se ejecuta el setup.*sh de la carpeta devel:

```
$ source devel/setup.bash
```

Definición del workspace para el paquete ROS

Al quedar establecido el workspace de Catkin, se debe definir el apartado que va a emplear el paquete de lectura de datos del regulador.

Para ello, desde la carpeta source, que es donde se alojan todos los paquetes que se requieran, se ejecuta el comando de "catkin_create_pkg". Al crear el paquete se le puede introducir las dependencias que va a tener.
Para el caso del paquete de lecturas, como se va a emplear Python y los mensajes personalizados para enviar la información a través de tópicos, se agregan las dependencias "rospy" y "std_msgs" respectivamente:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg ros_mppt rospy std_msgs
```

Con ello se crea una carpeta ros_mppt que contiene dos archivos package.xml y a CMakeLists.txt, que ya han sido parcialmente rellenados con la información dada al utilizar el comando de "catkin_create_pkg".

Para definir el paquete creado en el workspace de Catkin, se debe ejecutar de nuevo el setup de la carpeta devel. Previamente se debe compilar el paquete en el workspace con el comando "catkin_make" para que se generen los archivos necesarios:

```
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
```

4.3.2. Programación del paquete `ros_mppt`

Características del script `vemppt_ros`

Se desea que el nodo del paquete se encargue de establecer la comunicación con el regulador, mientras esté el sistema ROS del vehículo en ejecución. El script del paquete se encarga de leer las tramas recibidas por la comunicación, filtrarlas y registrar aquellos valores de interés y enviarlos a través de un tópico. Este procedimiento se puede ver de forma esquemática en la Figura 4.4. Los datos de interés son los ya comentados en el apartado 4.2.1, de los tramos posibles de la Tabla 4.3:

- Voltaje y corriente del sistema de baterías.
- Voltaje y potencia de los módulos fotovoltaicos.

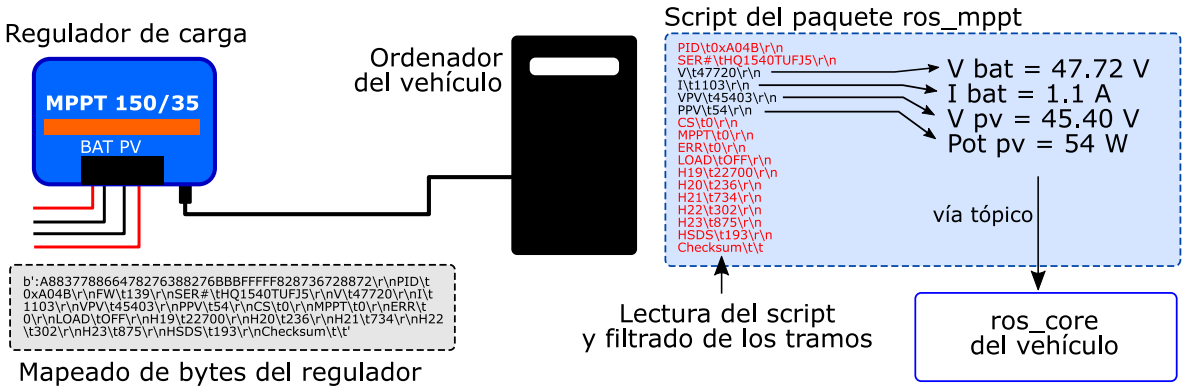


Figura 4.4: *Esquema de los procesos llevados a cabo por el paquete* ***ros_mppt***

Para ello se han creado los siguientes elementos en el paquete ya definido en el workspace de Catkin:

- La definición de las funcionalidades del paquete en los archivos package.xml y CMakelists.txt para un correcto funcionamiento del paquete, aparte de documentarlo de forma correcta.
- Un script en lenguaje Python que realice los procesos descritos.
- Un archivo .msg con cada valor definido que se va a enviar a través del tópico.

Definición de los archivos package.xml y CMakelists.txt

Previo a la realización del script se ha personalizado el archivo package.xml. En este archivo se introduce la información que define al paquete y la forma de compilar el paquete, detallando las herramientas que se van a emplear.
De forma más detallada, se debe cumplimentar la siguiente información:

- Versión del archivo.
- Datos del paquete: nombre, versión, descripción del paquete, correo del que mantiene actualizado el paquete y la licencia empleada.
- Nombre del autor y enlace a la documentación del paquete.
- Compilador empleado y detalles del compilador: dependencias, ejecuciones y exportaciones de diversas herramientas.

Dicha información se introduce en el archivo en forma de etiquetas, dependiendo del tipo de dato, teniendo en su interior el tipo de argumento definido. Para el caso de `ros_mppt`, las dependencias del compilador son la generación de mensajes (message_generation), el uso del lenguaje Python (ropsy) y el formato de envío de mensajes (std_msgs). Las exportaciones al compilar se limitan sólo a rospy y std_msgs ya que la generación de mensajes es interno del script vía la comunicación con el regulador de carga. En cuando a las ejecuciones del compilador, requiere el uso del gestor de mensajes (message_runtime) y el lenguaje del script (rospy).

Con ello definido, el archivo package.xml de `ros_mppt` queda definido de la siguiente manera:

```
<?xml version="1.0"?>
<package format="2">

  <name>ros_mppt</name>
  <version>0.1.1</version>
  <description>MPPT message sender package</description>
  <maintainer email="aaronpb@todo.todo">aaronpb</maintainer>
  <license>GNU GPL v3.0</license>

  <url type="website">http://wiki.ros.org/ros_mppt</url>
  <author>Aaron PB</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>message_generation</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>message_runtime</exec_depend>
  <exec_depend>rospy</exec_depend>

  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>

</package>
```

Código 4.1: Archivo package.xml del paquete `ros_mppt`

Para el caso del archivo CMakelists.txt, se define el funcionamiento del paquete y los comandos a ejecutar inicialmente al instalar y activar el paquete en cualquier otro dispositivo.
El archivo tiene multitud de opciones para cubrir todos los aspectos posibles para definir el paquete de la forma deseada, detallados en la wiki de ROS (ROS.org (2019)).

Para el caso del paquete `ros_mppt`, se ha definido lo siguiente:

- **Componentes requeridos en el workspace** de Catkin para que funcione adecuadamente el paquete: message_generation y std_msgs (para el envío de los mensajes a través del tópico), y rospy (ya que se emplea el lenguaje Python).
- **Archivos .msgs**: en este caso, el paquete contiene un grupo de mensajes personalizados llamado mppt.msgs.
- **Dependencias de la generación de mensajes**: tal y como se ha señalado en el archivo package.xml, el paquete depende de std_msgs, donde se van a definir el formato de los valores que irán a través del tópico.
- **Configuración específica de Catkin**: se agregan las dependencias del compilador señalados también en el archivo package.xml (message_runtime, ropsy y std_msgs).
- **Instalación de programas**: se definen los directorios que se van a construir y la ruta de los programas a instalar al descargar el paquete y ejecutarlo.

Lo descrito anteriormente se puede ver definido en el archivo de CMakelists.txt:

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_mppt)

find_package(catkin REQUIRED COMPONENTS
  message_generation
  rospy
  std_msgs
)

##############################################
## ROS messages, services and actions ##
##############################################

## Message files
add_message_files(
   FILES
   mppt.msg
)

## Generate added messages dependecies
generate_messages(
  DEPENDENCIES
  std_msgs
)

###################################
## catkin specific configuration ##
###################################

catkin_package(
  CATKIN_DEPENDS message_runtime
  CATKIN_DEPENDS rospy
  CATKIN_DEPENDS std_msgs
)

###########
## Build ##
###########

include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

#############
## Install ##
#############

## Mark executable scripts for installation
install(PROGRAMS
  scripts/vemppt_ros.py
  DESTINATION ${CATKIN_PACKAGE_PYTHON_DESTINATION}
)
```

Código 4.2: Archivo CMakelists.txt del paquete `ros_mppt`

Realización del script `vemppt_ros`

Para la realización del script, se ha tenido en cuenta las recomendaciones aportadas por la wiki (ROS.org (2019)) para que el programa realice las acciones requeridas dependiendo de si el sistema ROS está activo o no. Es decir, con la guía aportada en la documentación de ROS, se ha definido el bucle while del funcionamiento del script.

Antes de crear el archivo, se ha instalado lo definido en los archivos package.xml y CMakelists.txt, dentro del workspace:

```
$ source /opt/ros/kinetic/setup.bash
$ cd ~/catkin_ws
$ catkin_make
```

Para respetar la organización del paquete, dentro de la carpeta del paquete se ha creado la subcarpeta scripts para alojar el programa:

```
$ roscd ros\_mppt
$ mkdir scripts
```

Para realizar el código se ha empleado un editor llamado Sublime Text debido a la comodidad y la personalización que permite de la interfaz.

La wikipedia de ROS presenta el siguiente esquema básico para cualquier nodo en Python capaz de realizar procesos internos y enviar datos específicos:

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def talker():
  pub = rospy.Publisher('chatter', String, queue_size=10)
  rospy.init_node('talker', anonymous=True)
  rate = rospy.Rate(10) # 10hz
  while not rospy.is_shutdown():
    hello_str = "hello world %s" %rospy.get_time()
    rospy.loginfo(hello_str)
    pub.publish(hello_str)
    rate.sleep()

if __name__ == '__main__':
  try:
    talker()
  except rospy.ROSInterruptException:
    pass
```

Se recuerda que existen dos tipos de nodos, explicados en el capítulo 3; uno actúa enviando mensajes (talker), que es el que se desea realizar, y otro actúa leyendo los mensajes recibidos (listener) que, en este caso, es el propio sistema ROS.

Analizando cada parte del código, comenzando por las primeras líneas, se tiene que el nodo emplea dos módulos. El módulo rospy es necesario para que el programa sea capaz de poder ejecutarse y ser compatible dentro del sistema ROS. Para el caso de la importación String del archivo std_msgs.msg es debido a que dicho archivo contiene todos los formatos predefinidos y el ejemplo emplea una variable como cadena de texto con “hello world”.

En cuanto al funcionamiento dentro del método “talker()”, es el siguiente:

Inicialmente se definen 2 aspectos fundamentales que debe contener un nodo para poder comunicarse con el sistema ROS.
La primera es la interfaz que distingue el nodo del resto de nodos del sistema, definido en las líneas 7 y 8 del código. La línea 7 detalla que el nodo está publicando los mensajes al tópico llamado “chatter”, empleando un mensaje de tipo String. El último argumento limita la máxima cantidad de mensajes que se envían al sistema, en caso de que ningún otro nodo las esté recibiendo o las reciba lentamente.
La línea 8 es muy importante, debido a que manda a rospy el nombre del nodo (en este ejemplo: “talker”). En caso de que no se tuviera dicha información, el nodo no podría comunicarse con el sistema. Activando el modo anónimo se asegura que el nombre sea único, debido a que agrega números aleatorios después del nombre definido. Esto se puede visualizar en el rosgraph del nodo, como muestra la siguiente figura:

Figura 4.5: *RosGraph del nodo* ***vemppt_ros.py***

Como se puede observar en la Figura 4.5, se ha definido el nombre del nodo como `ros_mppt` y el tópico como `mppt_channel`.

El otro aspecto fundamental es la frecuencia con la que se realiza el bucle del proceso del nodo. Eso se indica en la línea 9 del ejemplo mostrado, en la que se crea un objeto Rate con la ayuda del método sleep() que contiene. Con el argumento 10, se le está definiendo una frecuencia al bucle de 0.1 segundos.

Estos dos aspectos fundamentales del nodo se han declarado de la siguiente manera en el script `vemppt_ros.py`:

```
1  #!/usr/bin/env python
2
3  import rospy
4  import time
5  import serial
6  import logging
7  from ros_mppt.msg import mppt
8
9  #Definicion del archivo de registros de errores
10 def sender():
11     pub = rospy.Publisher('mppt_channel', mppt, queue_size=10)
12     rospy.init_node('ros_mppt', anonymous=True)
13     r = rospy.Rate(10)
14     msg = mppt()
15     #Predefined values of msg
16     msg.v_bat = -1
17     msg.i_bat = -1
18     msg.v_pv = -1
19     msg.p_pv = -1
20
21 #Continuacion del codigo...
```

Lógicamente contiene más módulos, necesarios para la comunicación con el regulador de carga. Además, tiene definido su propio archivo con el formato de salida de los valores en el tópico, llamado "mppt.msg". Este archivo se ha creado dentro de otra subcarpeta dentro del paquete, llamado msg. Dentro se han definido que los 4 valores de interés salgan con un formato de tipo flotante.
El contenido del archivo se muestra a continuación:

```
1 float64 v_bat
2 float64 i_bat
3 float64 v_pv
4 float64 p_pv
```

Continuando con la explicación del ejemplo proporcionado por la wiki de ROS, el bucle while definido desde la línea 10 hasta la línea 14 muestra la estructura estándar de un nodo en ROS.
Dentro del bucle se realizan todos los procesos definidos por el usuario (en el caso del ejemplo una publicación de un "hola mundo" seguido del tiempo actual en el que se envía el mensaje), y finaliza con la publicación de los valores.
La función pub.publish permite enviar aquellos parámetros deseados al tópico definido. En cuanto a rospy.loginfo permite observar los valores emitidos por pantalla, a la vez que se registran en el sistema ROS y permite depurar el código. La última función, rate.sleep(), se encarga de imponer la frecuencia definida en el objeto Rate.

En el caso del programa del regulador, en el interior del bucle while se agrega la llamada a los tramos del mapeado de bytes una vez realizada la comunicación, posteriormente se agrega el filtrado de los tramos, y en caso de haber filtrado correctamente un valor, se procede a actualizarlo en el tópico sobreescribiendo el valor.

Además de ello, se ha definido un archivo que registra cualquier error que haya podido surgir durante la ejecución del nodo, mediante el módulo logging.
El archivo guarda a partir de un nivel de registro de info (los niveles de registro en escala ascendente son: DEBUG, INFO, WARNING, ERROR y CRITICAL), se ha nombrado como "ros_mppt.log" y en cada ejecución del programa se agregan los registros en la última línea anterior por la escritura tipo appending, es decir; en caso de que ya esté el archivo creado, se agregan las líneas del registro. En caso contrario, se genera el archivo con dicho nombre y se abre en modo escritura.
En el argumento del formato, se ha establecido que entre corchetes se presente la fecha y hora; y el nivel de advertencia del registro y, a continuación el mensaje registrado.

El archivo "ros_mppt.log" se ha definido de esa manera al principio del código:

```
1 logging.basicConfig(level=logging.INFO, filename='ros_mppt.log', filemode='
    a', format='[%(asctime)s - %(levelname)s]: %(message)s', datefmt='%d-%b
    -%y %H:%M:%S')
```

Y, finalmente, el tramo de procesos interno del bucle while y las comunicaciones iniciales, se muestran en el siguiente código:

```
#Inicio del programa explicado anteriormente

    while not rospy.is_shutdown():

        try:
            ve_read = ser.readline().decode("utf-8")
        except: logging.warning('Skipping decoding from ve_read line: ' +
    ve_read)

        if "V" in ve_read and "P" not in ve_read:
            try:
                ve_read = ve_read.split("\t")
                msg.v_bat = float(ve_read[1]) * 0.001
            except Exception as e: logging.error('Exception ocurred in V',
    exc_info=True)
        elif "I" in ve_read and "P" not in ve_read:
            try:
                ve_read = ve_read.split("\t")
                msg.i_bat = float(ve_read[1]) * 0.001
            except Exception as e: logging.error('Exception ocurred in I',
    exc_info=True)
        elif "VPV" in ve_read:
            try
                ve_read = ve_read.split("\t") * 0.001
                msg.v_pv = float(ve_read[1])
            except Exception as e: logging.error('Exception ocurred in VPV'
    , exc_info=True)
        elif "PPV" in ve_read:
            try:
                ve_read = ve_read.split("\t")
                msg.p_pv = float(ve_read[1])
            except Exception as e: logging.error('Exception ocurred in PPV'
    , exc_info=True)

        rospy.loginfo(msg)
        pub.publish(msg)
        r.sleep()

if __name__ == '__main__':
    try:
        ser = serial.Serial('/dev/ttyUSB0', 19200, timeout=10)
        sender()
    except rospy.ROSInterruptException: pass
    finally:
        ser.close()
```

Por último, para que el nodo se ejecute en ros, se debe hacer el archivo .py ejecutable vía:

```
$ chmod +x vemppt_ros.py
```

Para entender mejor el funcionamiento del nodo, se muestra el diagrama de procesos que realiza el script del paquete, publicado y documentado en GitHub (Poyatos (2019a)), en la Figura 4.6.

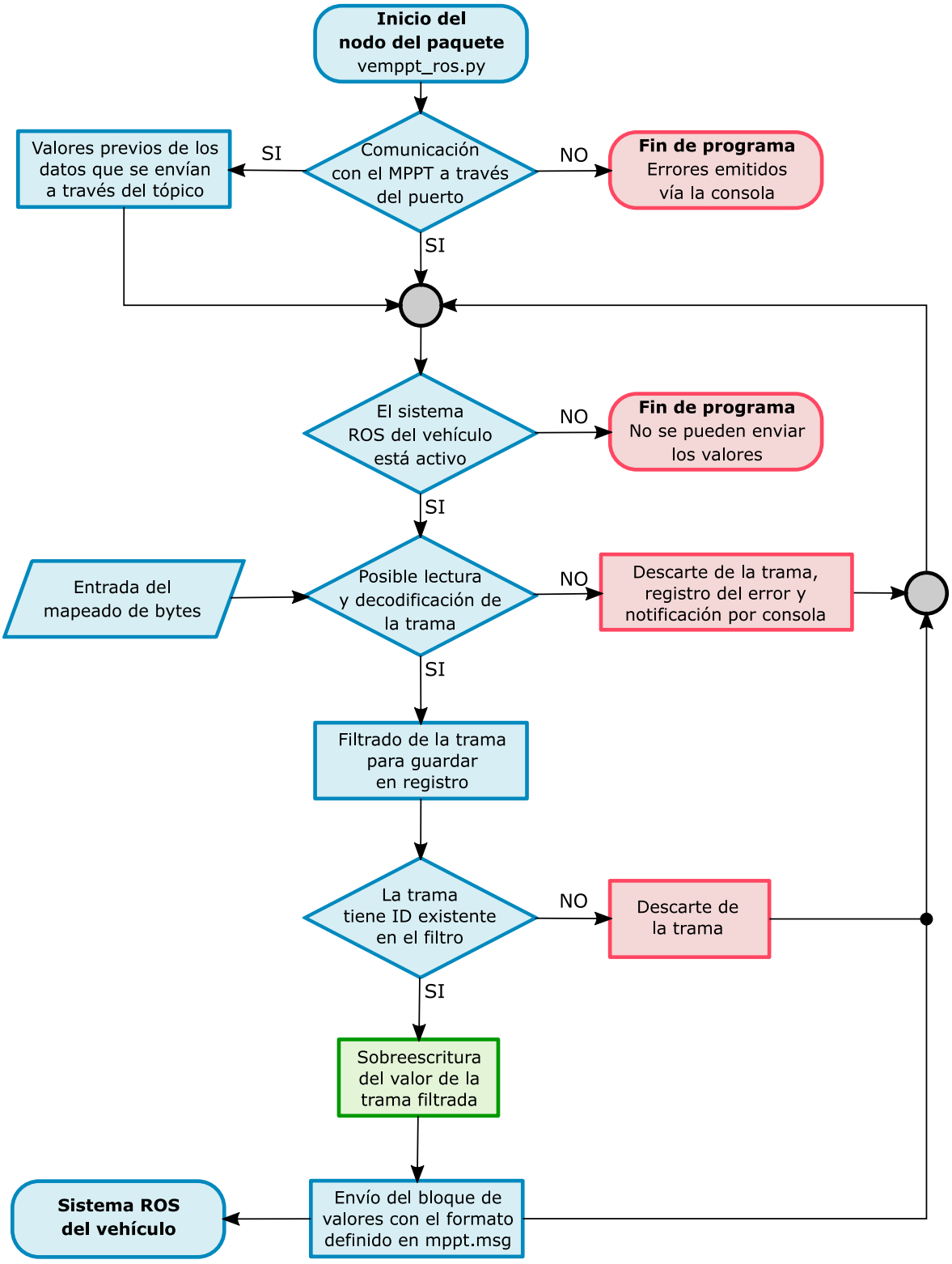


Figura 4.6: *Diagrama de procesos del programa* `vemppt_ros`

4.3.3. Comprobaciones de lectura y envío de datos al sistema

Durante el desarrollo del programa y para verificar su funcionamiento antes de instalarlo en el vehículo, se han realizado comprobaciones con un montaje simple definido en la Figura 4.7 y cuyo esquema es el que sigue en la Figura 4.8.

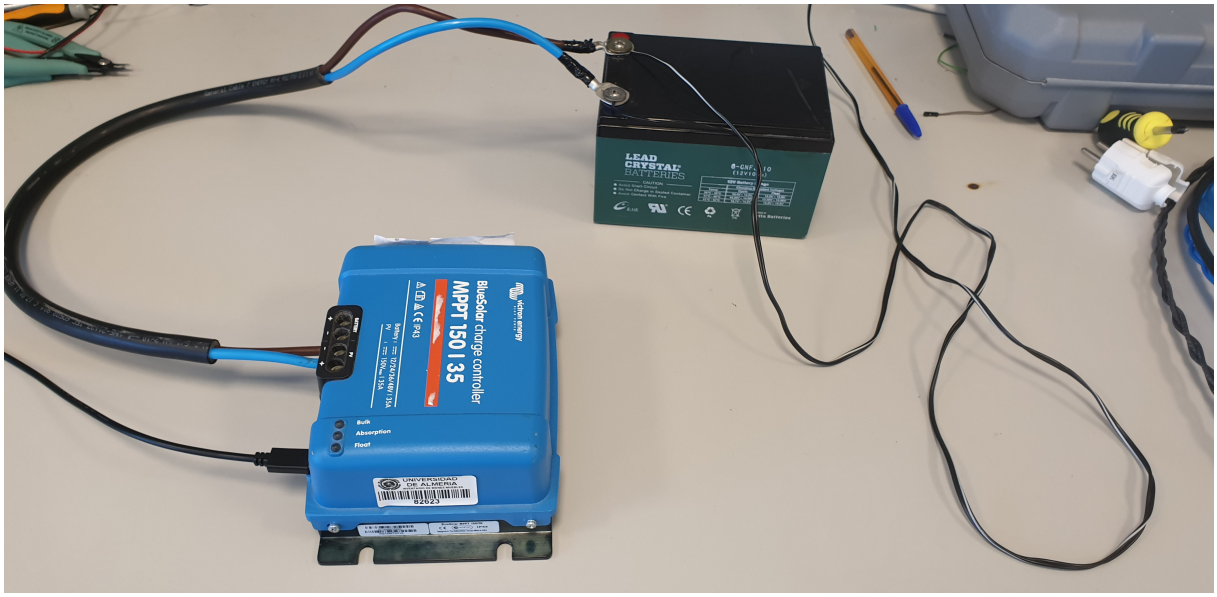


Figura 4.7: *Disposición del regulador de carga y la batería para las comprobaciones del nodo*

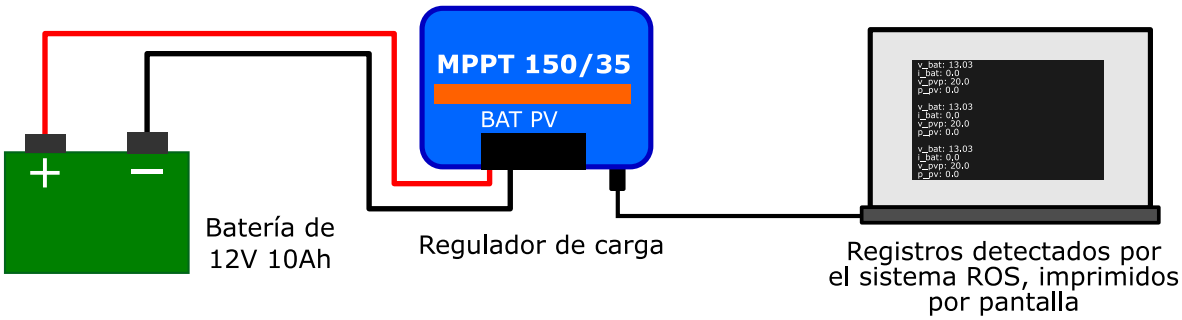


Figura 4.8: *Esquema del montaje diseñado para comprobar el funcionamiento del nodo*

Como elementos del sistema, solo es necesario incorporar un sistema de baterías para que el regulador de carga lo detecte y comience a funcionar. Al no tener un sistema de baterías similar al del vehículo, se ha decidido utilizar una batería disponible en el laboratorio del Cite IV de 12 V y 10 Ah, compatible con las configuraciones aceptadas por el regulador (acepta sistemas de batería de 12, 24 y 48 V).
Para cargar la batería se ha empleado un transformador de 230 V a 12 V.

El nodo creado se ha ejecutado en un sistema ROS vacío (sólo contiene el propio nodo creado), y se ha depurado con los datos del tópico que se muestran en pantalla gracias a la función rospy.loginfo dentro del programa.

Para realizar las comprobaciones se ha señalado el puerto donde se conecta el cable VE.Direct dentro del programa para establecer la comunicación del script con los registros del regulador. Posteriormente, se ejecuta el sistema ros y el script del paquete, en dos terminales diferentes:

```
$ rosrun roscore
$ rosrun ros_mppt vemppt_ros.py
```

En el terminal donde se ejecuta el script, se muestran los valores del tópico, como se muestra en la siguiente figura:

Figura 4.9: *Valores publicados en el tópico* `mppt_channel` *por el nodo programado*

4.3.4. Documentación y verificación del paquete `ros_mppt`

Una vez comprobado que se emiten los valores deseados por pantalla, para proporcionarle la solidez y optimizar al máximo el paquete, se ha decidido documentarlo y publicarlo en rosdistro. Rosdistro es una base que recopila todos los paquetes publicados por los usuarios. De esa manera, cualquier usuario puede descargar el paquete desde la propia plataforma de ROS e instalarlo en su sistema.
Aparte, una vez queda enlazado, se permite que el paquete se documente en la wiki de ROS.
Para publicarlo en rosdistro, se debe verificar previamente si los archivos del paquete están definidos correctamente. De ello se encarga catkin_lint, un programa que revisa mediante un análisis estático si los archivos package.xml y CMakelists.txt del paquete están cumplimentados de forma correcta (Röhling (2019)).

Para realizar la revisión, se ha empleado el siguiente comando donde se especifica al programa la ruta del paquete:

```
$ catkin_lint catkin_ws/src/ros_mppt
```

En caso de identificar algún error en el archivo como, por ejemplo, desorden en las dependencias si no están colocadas en orden alfabético o, como el caso de la Figura 4.10, se ha detectado un archivo ejecutable que no está incluido en CMakelists.txt.

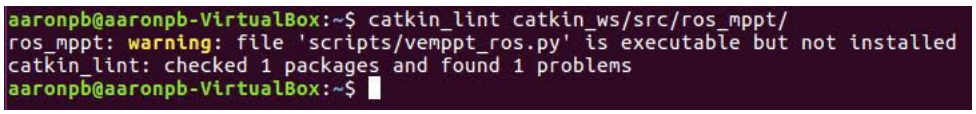


Figura 4.10: *Aviso generado por el programa catkin_lint al detectar que no está definido el archivo ejecutable en CMakelists.txt*

Una vez catkin_lint no ha emitido errores de ningún archivo del paquete, se ha solicitado la incorporación del mismo a la base de rosdistro vía el repositorio indicado en GitHub (rosdistro (2019)) y se ha documentado el paquete en la wiki oficial de ROS (Poyatos (2019b)).

Para que cualquier usuario pueda instalarlo desde la base de rosdistro es necesario crear un lanzamiento (más conocido como release) del paquete vía bloom. Bloom es una herramienta que facilita la generación de artefactos necesarios para lanzar un proyecto realizando dichos procesos de forma automática.
Para el lanzamiento del paquete se han seguido los siguientes pasos:

1. Previamente, se ha creado un repositorio en GitHub (llamado `ros_mppt-release`) en el que bloom generará los artefactos de lanzamiento del paquete ROS.

2. Se ha instalado bloom para poder ejecutar sus funciones:

   ```
   $ sudo apt-get install python-bloom
   ```

3. Posteriormente, se ha preparado el lanzamiento del paquete vía el workspace de catkin, creando previamente el archivo de registro de acciones del paquete (es recomendable crearlo para tener control en todo momento de los procesos del propio paquete):

   ```
   $ catkin_generate_changelog
   $ catkin_prepare_release
   ```

4. Finalmente, se ha ejecutado el lanzamiento a través de bloom, indicando que la versión de ROS empleada es la kinetic y el nombre del repositorio registrado en la base de rosdistro es `ros_mppt`:

   ```
   $ bloom-release --rosdistro kinetic --track kinetic ros_mppt --edit
   ```

   El argumento –edit sirve para indicar a bloom qué URI tomar de referencia para actualizar futuros desarrollos del paquete (en este caso, al máster branch del repositorio principal del paquete), aparte de poder modificar todos los parámetros de lanzamiento si es necesario (como distros de ROS con los que el paquete es compatible, nombre del repositorio, descripción del lanzamiento, etc.).

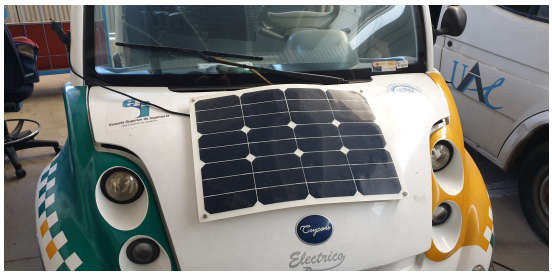
4.4. Integración del sistema fotovoltaico

La ubicación de los paneles solares ya estaba definido por Javier Guerrero en su trabajo fin de grado (Guerrero (2016)), por lo que se han vuelto a instalar en el vehículo ahora debidamente conectados, empleando cables de 4 mm2 de sección que cumplen de sobra con el dimensionado requerido para evitar sobrecalentamientos, como se muestra en la siguiente tabla:

Tabla 4.7: Valores de intensidad y potencia en función de la sección de los cables (Prieto (2012))

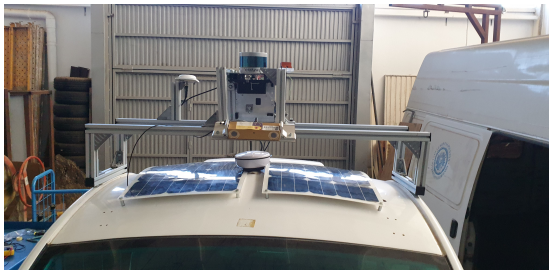
Sección del cable	**Intensidad máxima**	**Pot. máx en 12 Vcc**	**Pot. máx en 24 Vcc**	**Pot. máx en 48 Vcc**	**Pot.máx en 220 Vac**
1.5 mm2	11 A	132 W	264 W	528 W	2 420 W
2.5 mm2	15 A	180 W	360 W	720 W	3 300 W
4 mm2	20 A	240 W	480 W	960 W	4 400 W
6 mm2	25 A	300 W	600 W	1 200 W	5 500 W
10 mm2	34 A	408 W	816 W	1 632 W	7 480 W
16 mm2	45 A	540 W	1 080 W	2 160 W	9 900 W
25 mm2	59 A	708 W	1 416 W	2 832 W	12 980 W

La instalación de los paneles solares ha quedado de la siguiente forma:



(a) *Panel delantero instalado*

(b) *Panel trasero instalado*



(c) *Paneles superiores instalados*

(d) *Cableado interior de los paneles superiores*

Figura 4.11: *Instalación de los paneles solares en el vehículo*

La instalación del regulador de carga se ha decidido ubicarlo en el interior del maletero del vehículo ya que de esta manera está cerca del ordenador al que irá conectado el cable VE.Direct que envía los registros del dispositivo.
Se ha aprovechado algunos de los agujeros ubicados en la pared izquierda del maletero (véase la Figura 4.12) para colocar de forma correcta el regulador, empleando tornillos de 3 mm de diámetro.
También hay que destacar que dicha ubicación proporciona una accesibilidad ideal al regulador si en cualquier otro momento se decide mejorar o modificar el sistema fotovoltaico.

Figura 4.12: *Pared del maletero del vehículo donde se ha instalado el regulador de carga del sistema fotovoltaico*

En cuanto a la conexión del sistema, para prevenir cualquier daño hacia el regulador de carga, se ha instalado un fusible de 20 A (Figura 4.13) ya que las demandas solicitadas del vehículo no superan esa cantidad. Con el fusible se está agregando además, un elemento de seguridad que evita que se produzcan sobrecargas a cualquier dispositivo integrado en el vehículo.

Figura 4.13: *Fusible de 20 A instalado en el sistema de baterías del vehículo*

Otro elemento de seguridad incorporado ha sido un interruptor que aisle el sistema de baterías con el resto de elementos del vehículo para que, de esta manera, se pueda realizar cualquier manipulación de los dispositivos sin tener riesgos de descargas eléctricas indebidas.
El resultado es como sigue en las Figuras 4.14 y 4.15.

Figura 4.14: *Detalle de la instalación del regulador de carga y el enlace con los demás elementos del vehículo*

Como se puede observar en las figuras, se ha colocado el dispositivo boca-abajo. Se ha instalado con esa orientación para que haya una mejor accesibilidad a los contactos del regulador de carga.

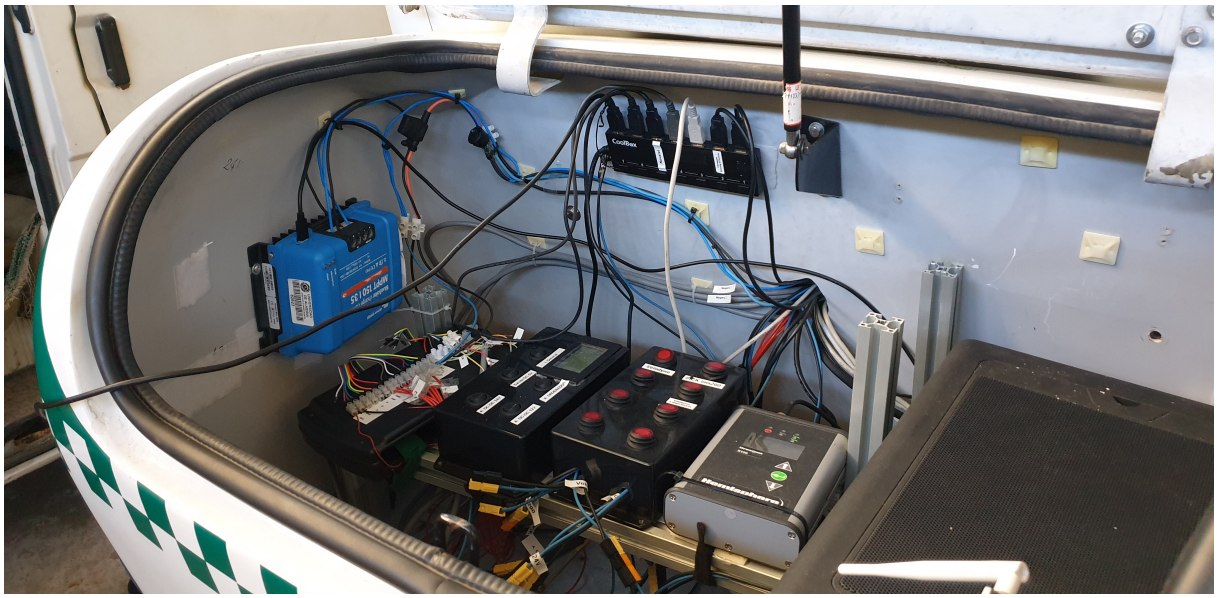


Figura 4.15: *Instalación del regulador de carga y el cableado con el sistema de baterías y el resto de elementos del vehículo*

Finalmente, se ha instalado el paquete `ros_mppt` en el ordenador del vehículo.
Para ello se han seguido las indicaciones publicadas en la documentación del paquete (Poyatos (2019b)):

1. Clonar el paquete publicado en GitHub desde el terminal:

```
$ git clone https://github.com/AaronPB/ros_mppt.git
```

2. Compilar el paquete en el workspace de catkin:

```
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
```

3. Señalar el puerto de comunicación donde se encuentra conectado el cable VE.Direct, dentro del programa `vemppt_ros.py`

Con ello, el sistema está totalmente integrado en el vehículo eléctrico UAL-eCARM.

Capítulo 5

Ensayos experimentales

5.1. Organización de los ensayos

Se han realizado 2 diferentes ensayos enfocados a la lectura del regulador de carga, ya que es el elemento del sistema fotovoltaico que se encarga de enviar los parámetros de interés para su posterior interpretación.

Los dos tipos de ensayos realizados en el trabajo son los siguientes:

- Lectura correcta de las tramas con el programa `vemppt_reader.py`
- Lectura y registros de los tópicos del MPPT vía el paquete `ros_mppt`, en el sistema ROS del vehículo.

Como se ha explicado en el capítulo anterior, con el primer tipo de ensayo se pretende conseguir un filtrado del mapeado de bytes del regulador sólido. Se entiende por solidez del código que el programa no quede afectado si hay una perturbaciones en el mapeado de bytes, como pueden ser: desplazamientos entre líneas de bytes, duplicidades, mensajes asíncronos repentinos, valores nulos o negativos en algunas variables, etcétera.

Finalmente, definido el paquete ROS, se da paso al segundo tipo de ensayo que consiste en, una vez implementado el sistema fotovoltaico en el vehículo, comprobar el funcionamiento del sistema y que los envíos de los datos se producen a través del paquete ROS.

5.2. Ensayos de lectura de tramas

5.2.1. Dispositivos y software empleados

Para este primer ensayo se ha empleado el programa `vemppt_reader` realizado en Python con los módulos descritos en el capítulo 4. Este programa se ha ido desarrollando a lo largo del período de este ensayo.

En cuanto a los elementos físicos, se ha empleado una batería de 12 V detallada en la Figura 5.1 (compatible con el regulador de carga, que acepta sistemas de 12, 24 y 48 V) debido a que no se tiene una réplica del sistema de baterías del vehículo. Como el ensayo tiene por objeto definir el filtrado de las tramas del regulador, esta diferencia no es determinante.

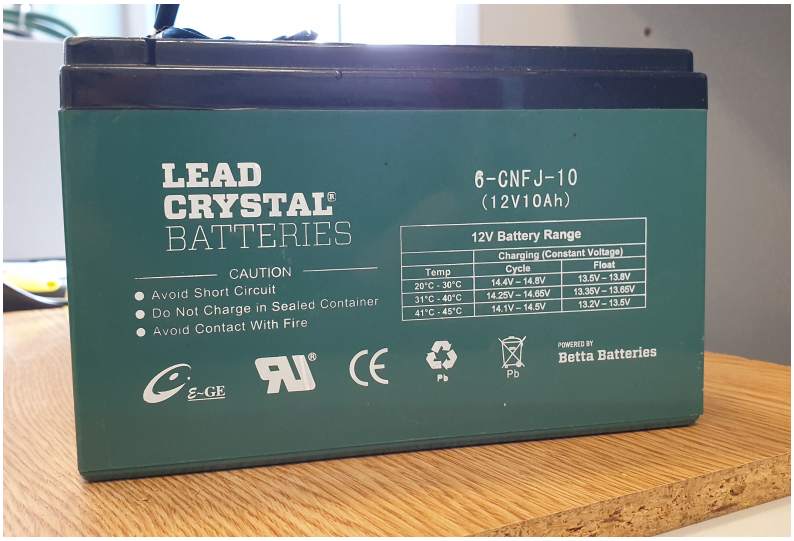


Figura 5.1: *Batería de 12V 10Ah modelo 6-CNFJ-10 de Lead Crystal Batteries*

También se utilizan los 4 módulos fotovoltaicos que se han implementado posteriormente en el vehículo.

Para descargar las baterías se ha empelado una carga programada de corriente continua, señalado en la Figura 5.2, capaz de realizar demandas de hasta 2.6 kW teniendo como intervalos de voltaje y corriente de 16 a 80 V y de 30 a 300 A, respectivamente.
Se tiene por objeto descargar las baterías para que el regulador de carga identifique que el sistema de baterías está descargándose y, por lo tanto, comience a aportar corriente de los módulos fotovoltaicos.

Figura 5.2: *Programador de cargas en corriente continua modelo 6301 de Chroma*

5.2.2. Banco de ensayos realizado

Para poner en funcionamiento este ensayo es necesario que el regulador de carga esté en funcionamiento, y eso se consigue cuando identifica el sistema de baterías al que aportar la demanda. Por ello se ha creado un banco de ensayos que simula el esquema del sistema fotovoltaico que se mostró en el capítulo 3, en la Figura 3.4.
El banco de ensayos y su esquema quedan representadas en las Figuras 5.3 y 5.4:

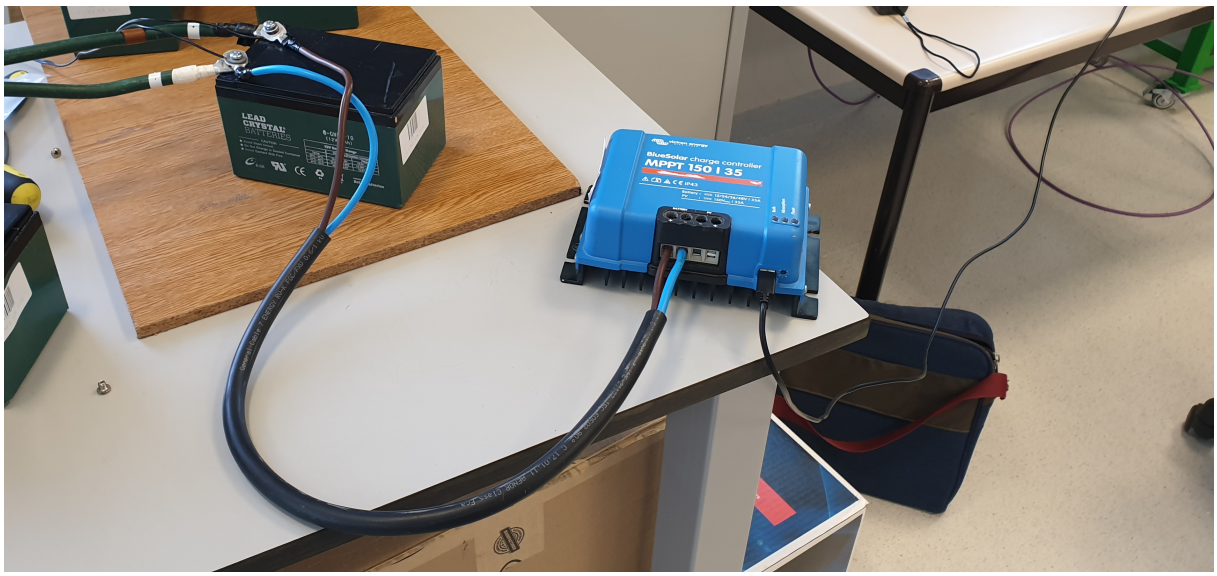


Figura 5.3: *Banco de ensayos para realizar la lectura del regulador de carga*

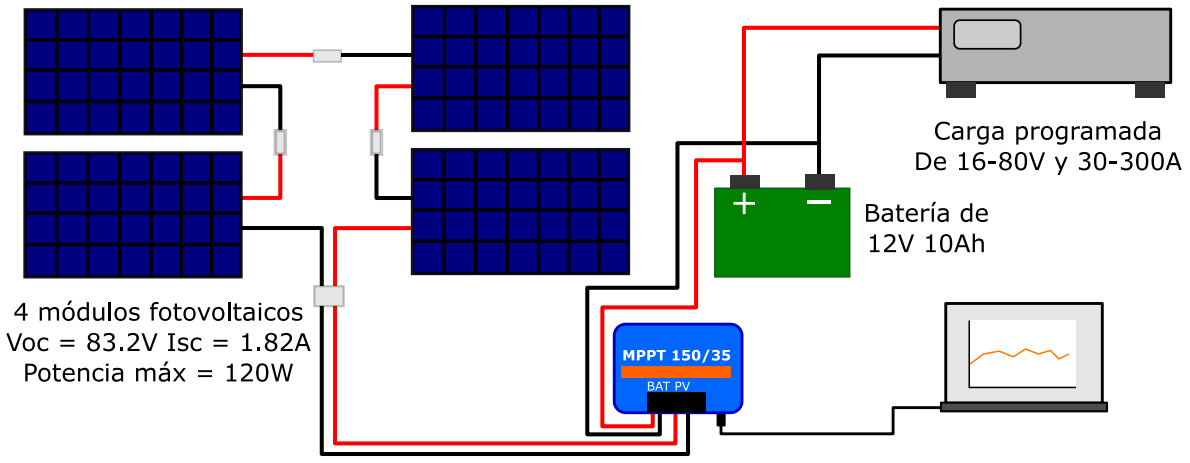


Figura 5.4: *Esquema del banco de ensayos*

El regulador de carga solo se ha empleado para aquellos casos donde se ha requerido una descarga puntual de la batería, y los módulos se han empleado en pruebas finales del programa para observar si las lecturas se realizan de forma correcta.
Para la mayoría de las pruebas se ha empleado un transformador de 230 V a 12 V para cargar las baterías.

5.2.3. Observaciones y mejoras en el programa `vemppt_reader`

Lectura del mapeado de bytes

Al principio, el programa estaba estructurado como un solo archivo ya que el objetivo era entender el comportamiento de los datos que enviaba el regulador.
La lectura de bytes se realizaba de la siguiente manera:

```
#Reading process
try:
    while True:
        line = line + 1

        with serial.Serial('COM5', 19200, timeout=10) as ser:
            ve_read = ser.read(220)          #Bytes read
            ser.close()
        #Data splitter
        ve_read_lines = ve_read.split(b'\n')
        dataPID = splitter(1)        # Product ID
        dataFW = splitter(2)         # Firmware Version
        dataSER = splitter(3)        # Serial Number
        dataV = splitter(4)          # Battery Voltage            [V]
        dataI = splitter(5)          # Current                    [A]
        dataVPV = splitter(6)        # PV Voltage                 [V]
        dataPPV = splitter(7)        # PV Power                   [W]
        dataCS = splitter(8)         # Mode
        dataMPPT = splitter(9)       #
        dataERR = splitter(10)       # Error codes
        dataLOAD = splitter(11)      # Load
        dataH19 = splitter(12)       # Yield Total
        dataH20 = splitter(13)       # Yield Today                [kWh]
        dataH21 = splitter(14)       # Maximum Power Today        [W]
        dataH22 = splitter(15)       # Yield Yesterday            [kWh]
        dataH23 = splitter(16)       # Maximum Power Yesterday
        dataHSDS = splitter(17)      # Day Sequence Number

        #print("Split line\n",ve_read_lines)

        dataI = float(dataI[1]) * 0.001
        dataV = float(dataV[1]) * 0.001
        dataVPV = float(dataVPV[1]) * 0.001
        dataPPV = int(dataPPV[1])
        timer = datetime.now()
        datatimer = timer.strftime("%H:%M:%S")

        print("Read number",line,"at",datatimer,"\nV",dataV,"I",dataI,"VPV"
    ,dataVPV,"PPV",dataPPV)

        sheet.write(line, 0, datatimer)
        sheet.write(line, 1, dataV)
        sheet.write(line, 2, dataI)
        sheet.write(line, 3, dataVPV)
        sheet.write(line, 4, dataPPV)

        time.sleep(1)
except KeyboardInterrupt:
    Mbox("Finished!","Data reader has been finished properly!",64)
except Exception as e:
    print("Byte line\n",ve_read)
```

```
51      print("Split line\n",ve_read_lines)
52      Mbox("An error ocurred","The communication port does not connect to a
        valid VE Device or is not defined properly!",16)
53      logging.exception("An error ocurred")
54  finally:
55      register_values.save(initial_date.strftime("MPPT Read %d_%m_%Y - %Hh %
        Mmin.xls"))
```

Un inconveniente era la saturación en la comunicación, debido a que en cada solicitud al regulador de carga, se realizaba las siguientes acciones en bucle:

1. El establecimiento del contacto con el regulador de carga a través del puerto señalado.
2. La solicitud de los valores registrados.
3. El cierre de la comunicación.

Esto se pudo solucionar haciendo que el programa establezca la comunicación al inicio de ejecutarlo, y cerrando la comunicación cuando el programa finaliza:

```
1  def main():
2      #Detalles iniciales
3
4      #Main thread
5      try:
6          ser = serial.Serial('COM5', 19200, timeout=10)      #Change here
       COM port
7          #Mas detalles en el try
8
9          while True:
10             #Estructura del programa
11
12     except Exception as e:
13         logging.exception("An error ocurred")
14         Mbox("An error ocurred","Something went wrong!",16)
15
16     finally:
17         ser.close()
18         vemppt_register.xls_close(initial_date)
19
20 if __name__ == '__main__':
21     main()
```

De esta manera, la comunicación durante la lectura de los datos siempre permanecerá abierta y, en caso de error, finally se encarga de cerrar correctamente el puerto de comunicación y la hoja excel generada.

Otro inconveniente en la lectura, era la forma de leer las tramas:

```
1              ve_read = ser.read(220)          #Bytes read
```

Esta línea de código indica que, en cada lectura, se leen los primeros 220 bytes de los registros del regulador de carga. Esto quiere decir que finaliza la lectura cortando el mapeado de bytes, por lo que en la siguiente lectura, el programa comienza a leer desde el último trozo dividido, generando errores al no identificarlo como una línea de bytes. Además, al estar dividido, pueden haber valores que en este trozo de la trama no se muestran, provocando errores en el filtrado inicial de los valores.

Este inconveniente fue el motivo por el que se cambió la manera de filtrar las tramas. Se observó que el módulo pyserial también tiene la opción de leer tramas con la función "ser.readline", por lo que se definió que el bucle de lectura solicitase todas las líneas de registros, una a una, enviándolas posteriormente al archivo de filtrado `vemppt_parser`

```
def main():
    #Detalles iniciales

    #Main thread
    try:
        ser = serial.Serial('COM5', 19200, timeout=10)      #Change here
    COM port
        #Mas detalles en el try

        while True:
            ve_read = ser.readline()
            line = vemppt_parser.parser(ve_read,line)

    #Detalles posteriores
```

Parseado de bytes

Se solventaron también varios problemas en el filtrado de los datos y es que, no todos las tramas que se leen del regulador de carga son posibles de decodificar para que lo pueda interpretar el programa. Es el caso de los mensajes asíncronos que envía de vez en cuando el regulador o valores nulos en algunas tramas, perdiendo también la estructura de la misma.

Para asegurar la decodificación segura de cada trama, se introdujo en el método de filtrado el siguiente código inicial:

```
def parser(parse_line,line):
    try:
        parse_str = parse_line.decode("utf-8")
    except:
        print("[!] Cannot decode this:",parse_line,"\n    Skipping to next
    line...")
        time.sleep(1)
        line = line+1
        return line
    #Continuacion del metodo...
```

Para evitar solape en valores iguales que ya han sido anotados en la hoja de excel (que causan errores de sobreescritura en el archivo), se cambia de línea y se realiza una parada de 1 segundo.

Si las tramas superan la decodificación, se les da paso a un filtrado preciso siguiendo el orden de la Tabla 4.3. Para ello se observa si el nombre de la trama coincide con la sección que registra el valor. Si coincide, se procede a desglosar la parte del ID de la trama con la parte del valor con la función split y se envía al archivo de registros. En caso de que no coincida en todo el filtrado, la trama se descarta avisando al usuario.

Un trozo de este filtrado es como el siguiente trozo del método:

```
def parser(parse_line,line):
    #Filtrado previo...

    #= Battery voltage =#
    elif "V" in parse_str and "P" not in parse_str: #For VPV and PPV cases
        parse_str = parse_str.split("\t")
        packet = float(parse_str[1]) * 0.001
        vemppt_register.xls_write(packet,1,line)
        return line

    #= Current =#
    elif "I" in parse_str and "P" not in parse_str: #For PID cases
        parse_str = parse_str.split("\t")
        packet = float(parse_str[1]) * 0.001
        vemppt_register.xls_write(packet,2,line)
        return line

    #= PV Voltage =#
    elif "VPV" in parse_str:
        parse_str = parse_str.split("\t")
        packet = float(parse_str[1]) * 0.001
        vemppt_register.xls_write(packet,3,line)
        return line

    #= PV Power =#
    elif "PPV" in parse_str:
        parse_str = parse_str.split("\t")
        packet = int(parse_str[1])
        vemppt_register.xls_write(packet,4,line)
        return line

    #Continuacion del filtrado...

    #= NULL =#
    else:
        print("[!] Unrecognised data:",parse_line)
        return line
```

5.3. Ensayos de lectura y registros con el vehículo UAL-eCARM

5.3.1. Dispositivos y software empleados

Para este ensayo se ha empleado el propio vehículo UAL-eCARM del grupo de investigación ARM, con el sistema fotovoltaico ya instalado.

A nivel de software, se ha utilizado el roscore del vehículo, que ya trae de otros proyectos de investigación diversos paquetes de lectura de sensores instalados en el mismo. A dichos paquetes se añade el paquete creado en este trabajo, que lee y registra los valores de voltaje e intensidad del sistema de baterías y el voltaje y la potencia de los paneles fotovoltaicos.

Para el guardado de datos se ha usado otra herramienta de ROS llamado "rosbag". Con dicha herramienta se pueden guardar todos los tópicos deseados que se envían a través del sistema ROS para, posteriormente, poder representar gráficamente los valores de interés. Para ejecutar el sistema ROS y realizar el guardado de los datos se ha usado el ordenador que trae el vehículo ya implementado, con el sistema operativo Ubuntu Linux, como se muestra en la Figura 5.5.

Figura 5.5: *Escritorio del ordenador instalado en el vehículo UAL-eCARM*

Para el posterior tratado de los datos guardados vía rosbag, se ha utilizado un programa "Bag2Txt.sh" creado por un compañero del grupo de investigación, Francisco José Mañas Álvarez, que permite la decodificación de los tópicos (que están almacenados en un solo archivo de extensión .bag) en archivos de texto con la extensión .txt.

Finalmente, para obtener las gráficas de voltaje e intensidad del sistema de baterías y de los módulos fotovoltaicos se ha creado un programa en matlab que lee los datos del archivo de texto y los representa en dos gráficas V-I, con la tendencia de descarga en las baterías (mediante una regresión lineal de los valores del voltaje de las baterías) para comparaciones posteriores en los resultados.

5.3.2. Ensayos realizados en el campus de la universidad

Procedimiento de los ensayos

Cada vez que se ha realizado un ensayo en esta última etapa, se ha seguido el siguiente procedimiento para asegurar una correcta lectura de los datos:

1. Conexión o desconexión de los módulos fotovoltaicos con el regulador de carga, dentro de la nave, para el caso en el que se desee realizar una lectura sin los paneles fotovoltaicos.

2. Encendido de los dispositivos (regulador de carga y ordenador).

3. Ejecución del sistema ROS del vehículo vía:

```
$ roslaunch ual_ecar_config steering_low_level.launch
```

4. Ejecución del script del paquete de lectura del regulador, para que envíe los tópicos al sistema ROS:

```
$ rosrun ros_mppt vemppt_ros.py
```

5. Retiro del vehículo de la nave y ubicación en el punto inicial de la ruta.

6. Iniciar la grabación de los tópicos del sistema ROS vía la herramienta rosbag, dentro de la carpeta datasets:

```
$ cd datasets
$ rosbag record -a
```

7. Realización de la ruta hasta finalizarla. Mientras se realiza, se va observando los mensajes que se obtienen del tópico del mppt.

8. Guardado de la ruta cerrando rosbag, obteniendo el archivo .bag

9. Posteriormente se renombra el archivo de los registros para obtener los archivos en extensión .txt vía el programa Bag2Txt.sh:

```
$ cd datasets/ecarm_2019_06_11_Ensayo01_Paneles
$ bash ../Bag2Txt.sh
```

10. Se repite el procedimiento del 4 al 9 hasta obtener los resultados programados para dicho día.

11. Al finalizar la sesión de ensayos, se vuelve a introducir el vehículo en la nave y se procede a cerrar los programas: el paquete `ros_mppt` y el sistema ROS. Por último, se apagan todos los dispositivos.

Al tratarse de ensayos en las vías del campus, se ha tenido por supuesto consideración en todo momento con los peatones, así como de mantener la distancia de seguridad y la velocidad adecuada para evitar cualquier tipo de accidente.
Los recorridos se detallan a continuación en cada día de ensayo, a excepción del primer día que se ha diseñado una ruta más libre para poder observar con mayor detenimiento si el programa envía los tópicos de manera correcta.

Ensayos del tópico `rostopic_mppt_channel`

Tabla 5.1: Información del día 1 de ensayo con el vehículo eléctrico

Fecha	**Nº de ensayo**	**Uso de paneles**	**Duración (min)**
11/06/2019	01	Si	18
	02	Si	45
	02	Si	18

Durante los tres ensayos realizados en este primer día, se ha corregido parte del código del paquete `ros_mppt` ya que ha surgido un inconveniente que no se ha dado en los ensayos anteriores. Se trata de valores nulos en las tramas definidas en el filtro, debido a que la trama de bytes ya superado el decodificado. Sin embargo, al no esperarse un valor nulo en el interior del filtro (al realizar la separación del ID de la trama y el valor de la misma) el programa genera errores.

Este inconveniente se pudo solucionar empleando el bloque try/catch en el código del script `vemppt_ros.py`. Como ejemplo, se muestran las modificaciones realizadas en el filtrado del voltaje del sistema de baterías:

```
while not rospy.is_shutdown():

    try:
      ve_read = ser.readline().decode("utf-8")
    except: print('Skipping decoding from ve_read line: ' + ve_read)

    #Filtro sin el bloque try/catch

    if "V" in ve_read and "P" not in ve_read:
      ve_read = ve_read.split("\t")
      msg.v_bat = float(ve_read[1]) * 0.001

    #Filtro con el bloque try/catch

    if "V" in ve_read and "P" not in ve_read:
      try:
        ve_read = ve_read.split("\t")
        msg.v_bat = float(ve_read[1]) * 0.001
      except: print('Error in Battery Voltage')
```

Al haber podido solucionar el inconveniente durante el mismo día del ensayo, se han logrado obtener gráficas en cada número de ensayo realizado. Dichas gráficas son las mostradas a continuación, en las Figuras 5.6, 5.7 y 5.8.

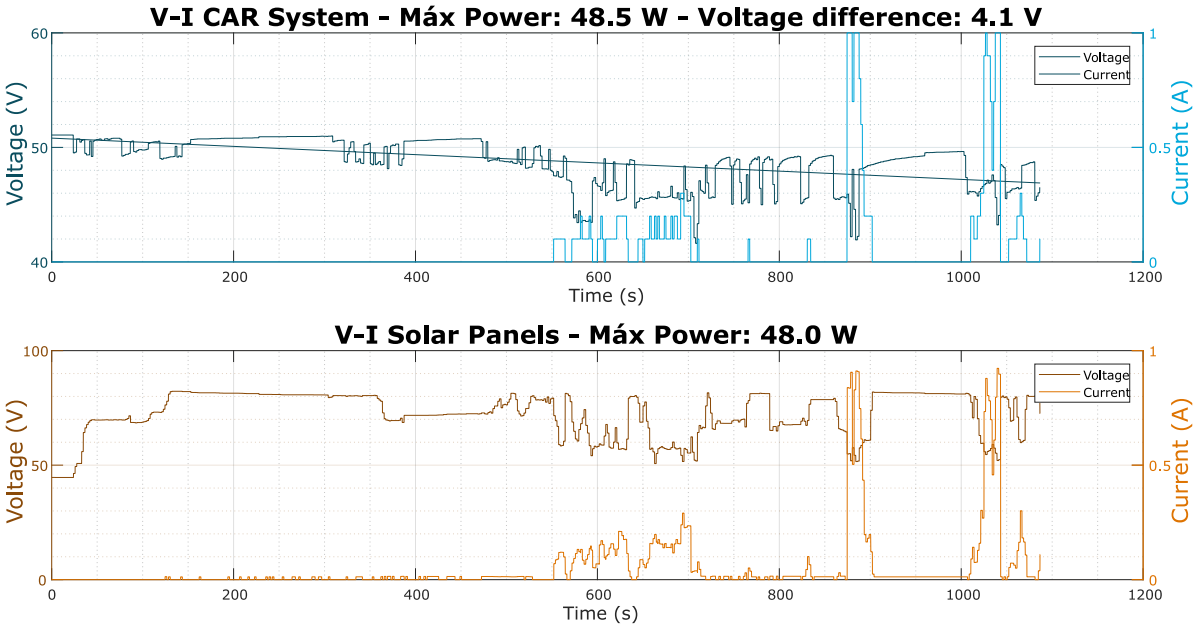


Figura 5.6: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 01 con Paneles Activos*

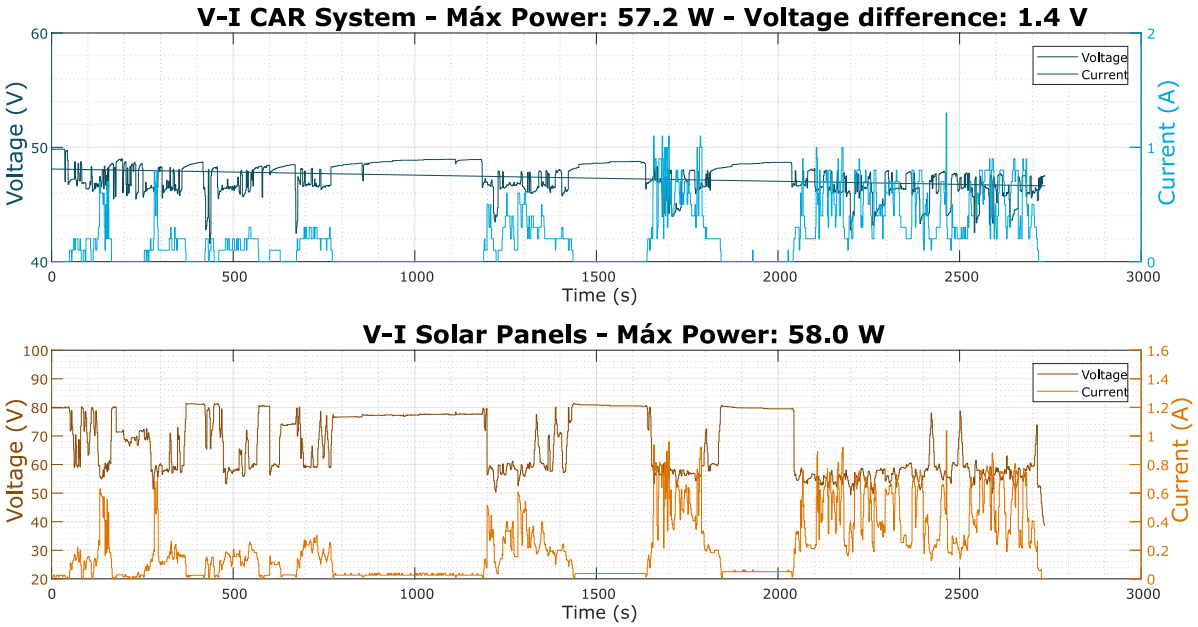


Figura 5.7: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 02 con Paneles Activos*

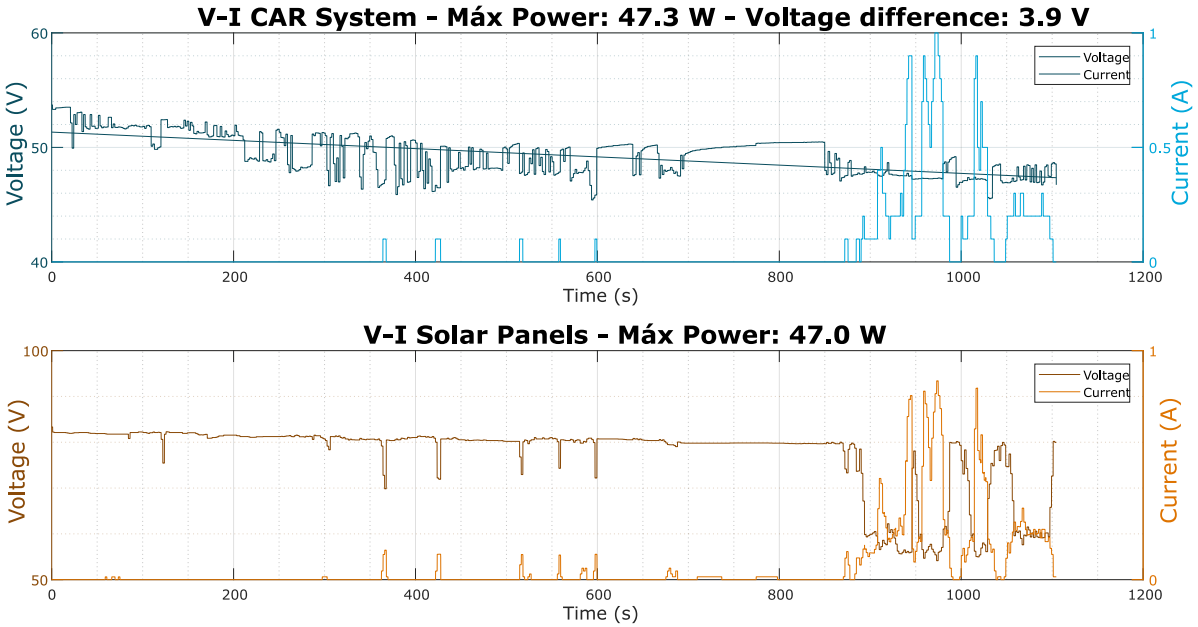


Figura 5.8: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 03 con Paneles Activos*

Ensayos de larga duración

Tabla 5.2: Información del día 2 de ensayo con el vehículo eléctrico

Fecha	**Nº de ensayo**	**Uso de paneles**	**Duración (min)**
12/06/2019	01	No	41
	02	Si	37

Para este día de ensayo se ha planificado dos rutas de larga duración (de 40 minutos aproximadamente), una de ellas se ha realizado con paneles solares y la otra sin ellas para poder observar las diferencias y el aporte que realiza el sistema fotovoltaico instalado en el vehículo.

Para tener una mayor precisión en la comparación de las gráficas obtenidas, se ha realizado la misma ruta tanto para el ensayo sin paneles fotovoltaicos conectados como para en el que sí están conectados.
La ruta planificada es la que sigue en las Figuras 5.9 y 5.10, y las gráficas obtenidas son las que siguen en las Figuras 5.11 y 5.12

Figura 5.9: *Primera parte de la ruta planificada, con el inicio (I) y la continuación (A) del ensayo realizado el día 12/09/2019*

Figura 5.10: *Segunda parte de la ruta, con la continuación (A) y el final de la ruta (F) del ensayo realizado el día 12/09/2019*

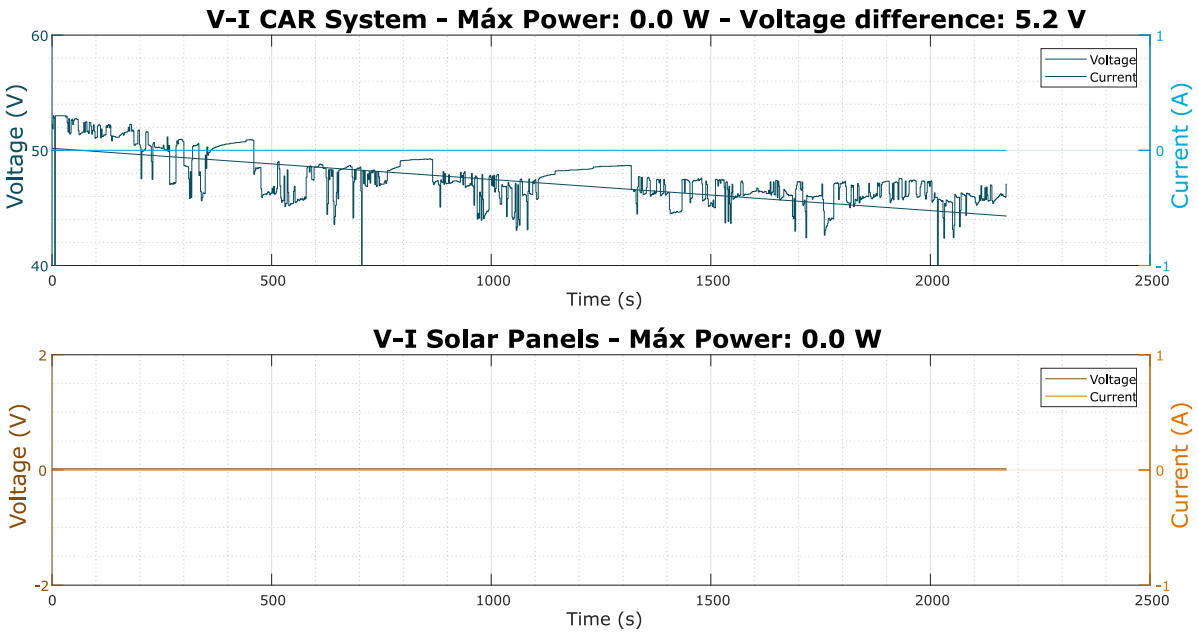


Figura 5.11: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 01 sin Paneles Activos*

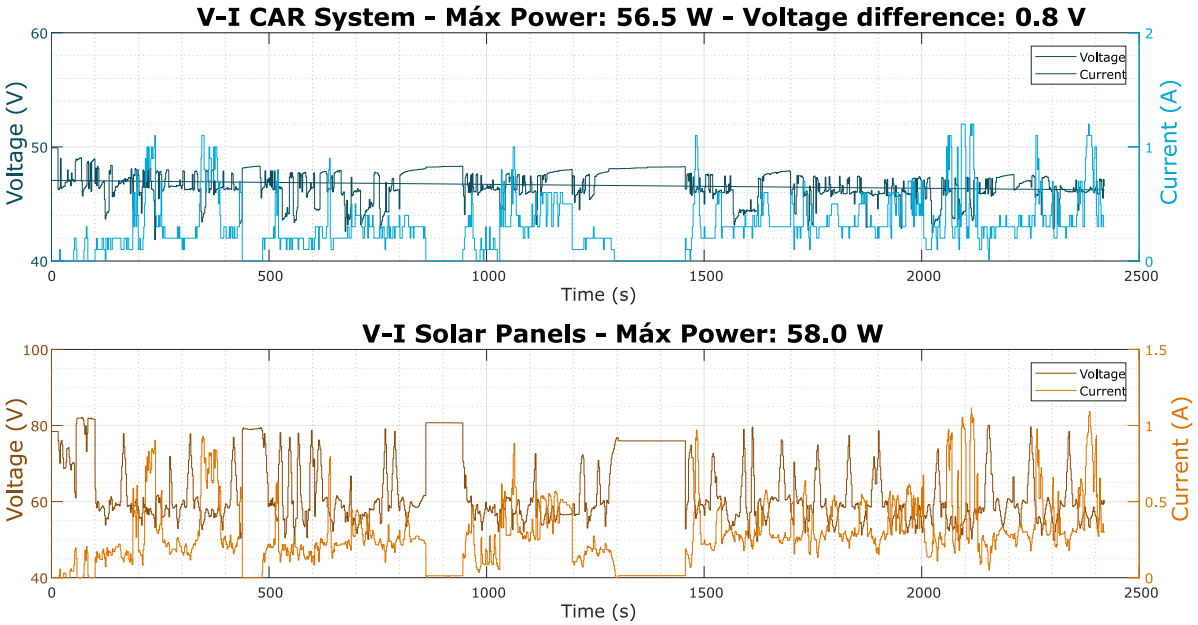


Figura 5.12: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 02 con Paneles Activos*

Ensayos de corta duración

Tabla 5.3: Información del día 3 de ensayo con el vehículo eléctrico

Fecha	**Nº de ensayo**	**Uso de paneles**	**Duración (min)**
17/06/2019	01	Si	13
	02	No	13
	03	Si	9
	04	No	7

Para este día de ensayo se ha planificado cuatro rutas de corta duración (de unos 10 minutos aproximadamente). El motivo por el que se realiza este último ensayo es para comprobar si ha tenido influencia el voltaje del sistema de baterías sobre el comportamiento de la potencia aportada por el sistema fotovoltaico.
Realizando ensayos en corta duración, se evita una descarga mayor de las baterías, pudiendo comparar los casos de aporte y no aporte de energía solar en una mejor igualdad de condiciones.

Al ser ensayos de tiempo reducido se ha decidido planificar 4 ensayos, dos de ellos con paneles activos y dos sin paneles activos. También se han definido dos rutas más cortas que el caso anterior, ambos en el campus de la universidad.
Cada ruta se repite de la misma manera tanto por una configuración con los módulos fotovoltaicos activos, como por otro sin los módulos activos.

La primera ruta planificada es la que se representa a continuación en la Figura 5.13 y las dos gráficas V-I obtenidas de los dos casos de configuración, se pueden observar en las Figuras 5.14 y 5.15.

La segunda ruta realizada es la que sigue posteriormente en la Figura 5.16 y las dos gráficas V-I obtenidas de los dos casos de configuración, en las Figuras 5.17 y 5.18.

Figura 5.13: *Primera ruta planificada, con el inicio (I) y y el final (F) del ensayo realizado el día 17/09/2019*

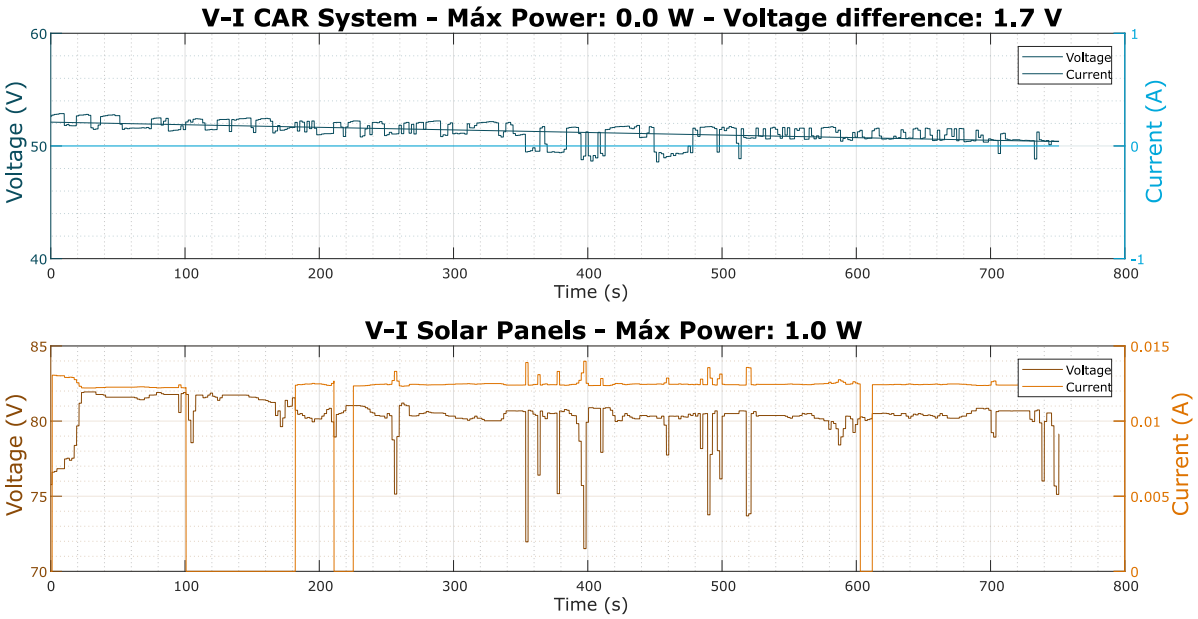


Figura 5.14: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 01 con Paneles Activos*

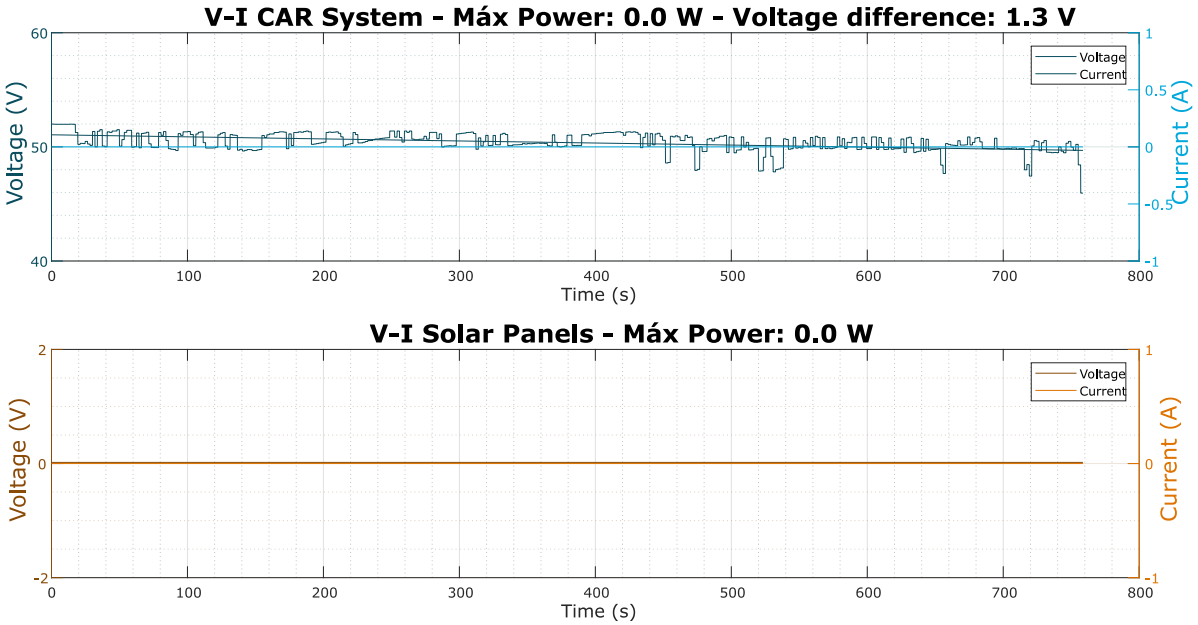


Figura 5.15: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 02 sin Paneles Activos*

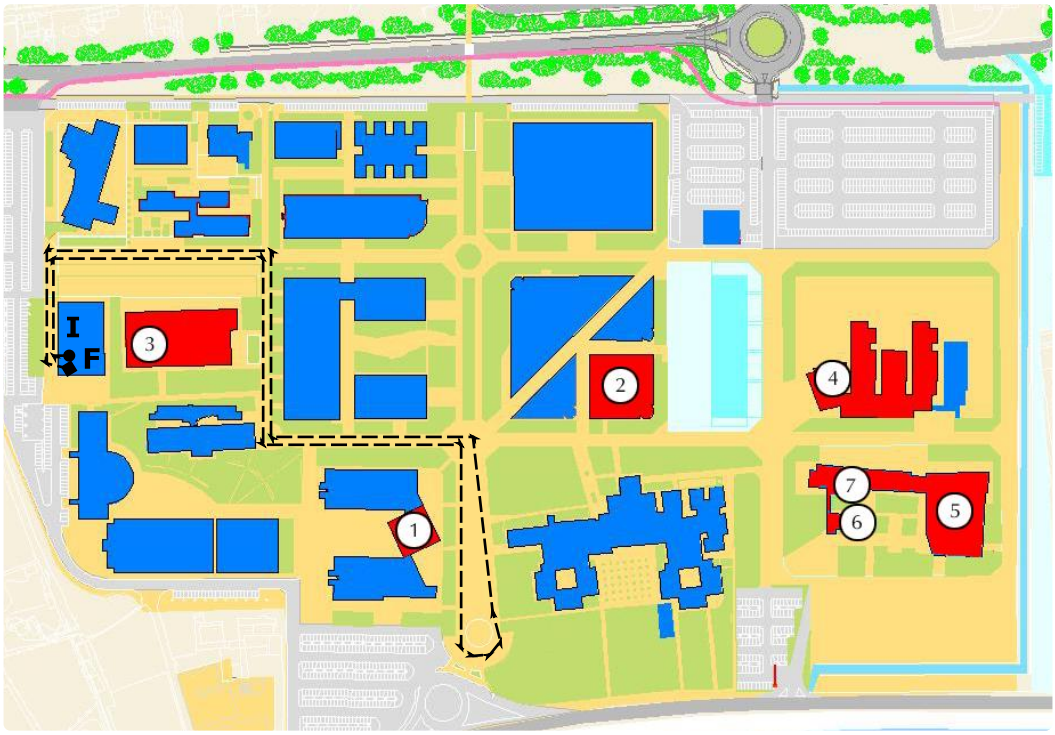


Figura 5.16: *Segunda ruta planificada, con el inicio (I) y y el final (F) del ensayo realizado el día 17/09/2019*

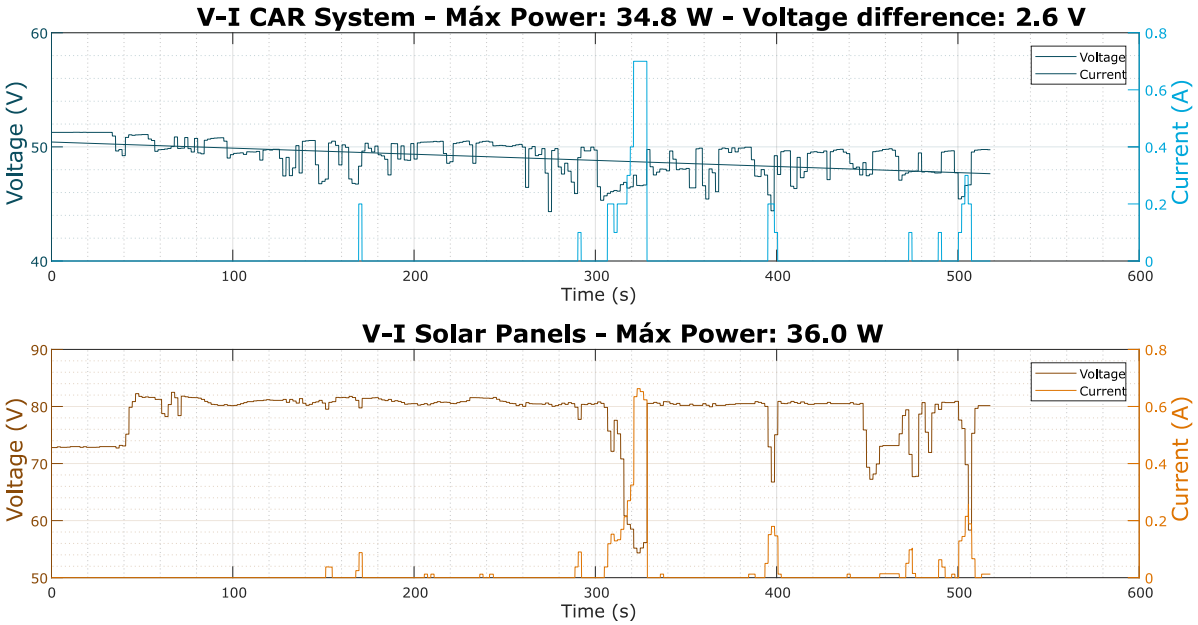


Figura 5.17: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 03 con Paneles Activos*

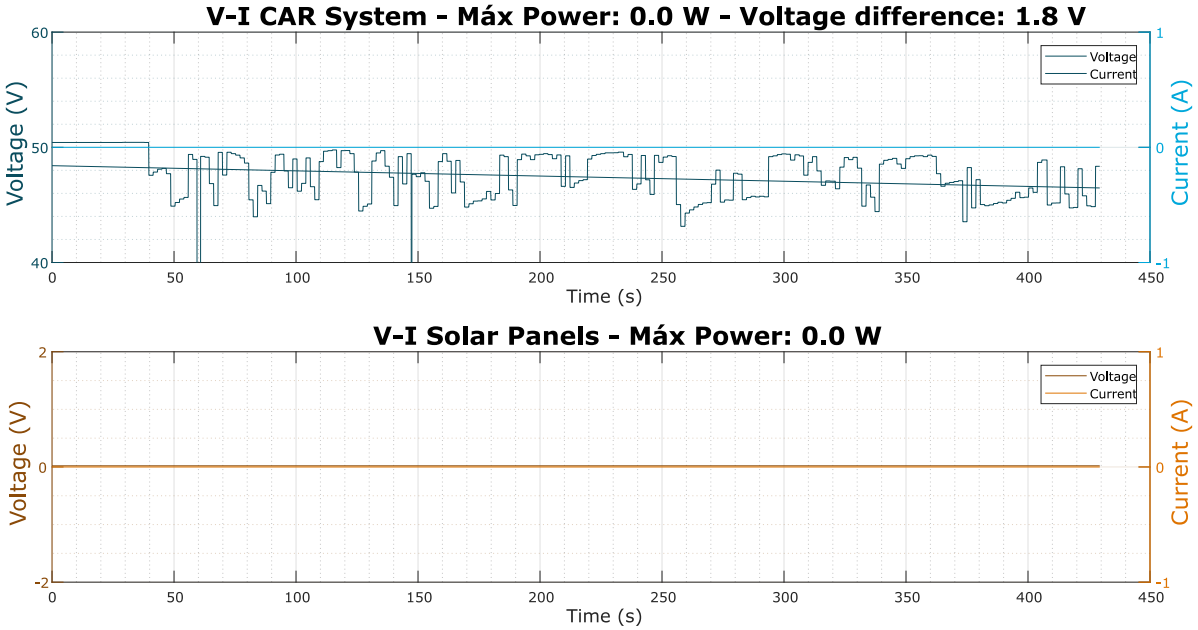


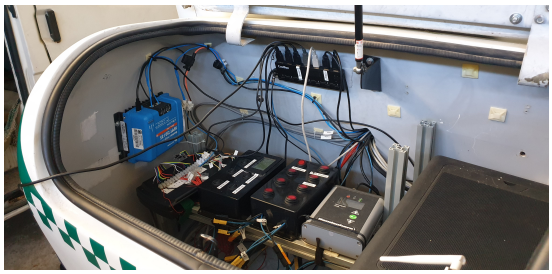
Figura 5.18: *Gráficas V-I del sistema de baterías y de los módulos fotovoltaicos del Ensayo 04 sin Paneles Activos*

Capítulo 6

Resultados y conclusiones

6.1. El sistema fotovoltaico integrado y las comunicaciones

El resultado principal del trabajo ha sido la integración de los módulos fotovoltaicos y del regulador de carga en el sistema eléctrico del vehículo UAL e-CARM.
Para tener una mejor referencia y simplificar la búsqueda, se ha vuelto a colocar la Figura 4.15, mostrada en el capítulo 4 (ahora Figura 6.1a), que representa la instalación física de los elementos en el vehículo. También se expone un esquema gráfico en la Figura 6.2, que facilita el detalle de las conexiones eléctricas, la comunicación con el MPPT y la disposición de los elementos del sistema.



(a) *MPPT y sus conexiones*

(b) *Vehículo con los paneles solares instalados*

Figura 6.1: *Resultados de la integración del sistema fotovoltaico en el vehículo eléctrico*

También se ha cumplido con el subobjetivo de la comunicación y adquisición de señales vía el diseño y la programación de un software; gracias al paquete `ros_mppt`. La comunicación del sistema ROS del vehículo con el regulador de carga para la lectura de los valores del sistema fotovoltaico implementado se ha realizado de forma correcta como se puede observar en los ensayos realizados en el apartado 2 del capítulo 5 y en el rosgraph del propio sistema ROS, donde se puede localizar el paquete creado (véase la Figura 6.3).

Como conclusión, se puede destacar la solidez y la flexibilidad del paquete `ros_mppt`, debido a que se ha desarrollado y modificado ante varias imprevisiones de la lectura de las tramas, y también porque se ha creado un filtrado completo de las posibles tramas que puede emitir el MPPT, pudiendo agregar o eliminar aquellas que interesen registrar o no, en el sistema ROS del vehículo.

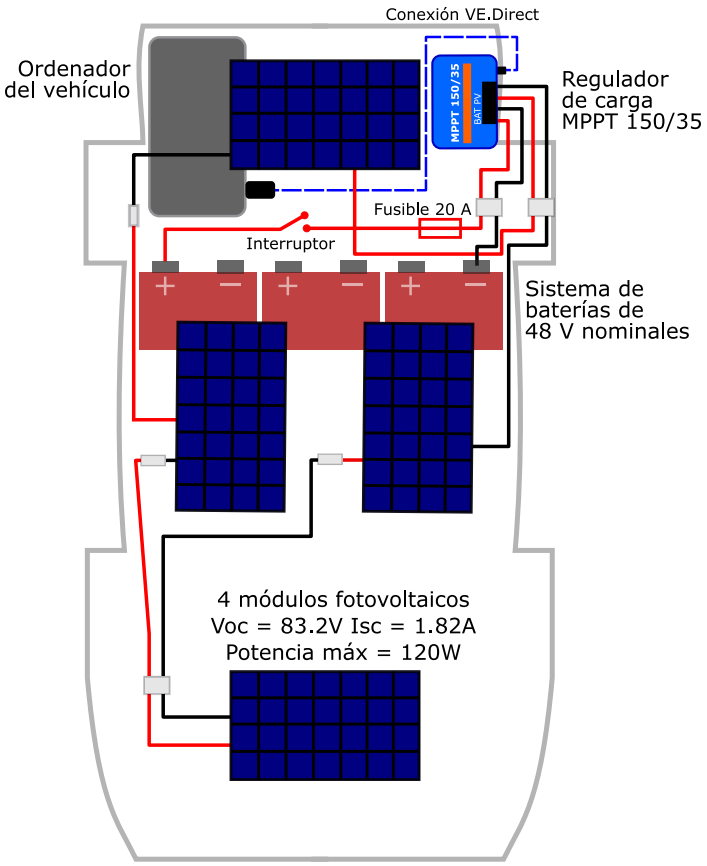


Figura 6.2: *Esquema gráfico de las conexiones y componentes eléctricas, además de la disposición de los elementos físicos en el vehículo en una vista en planta*

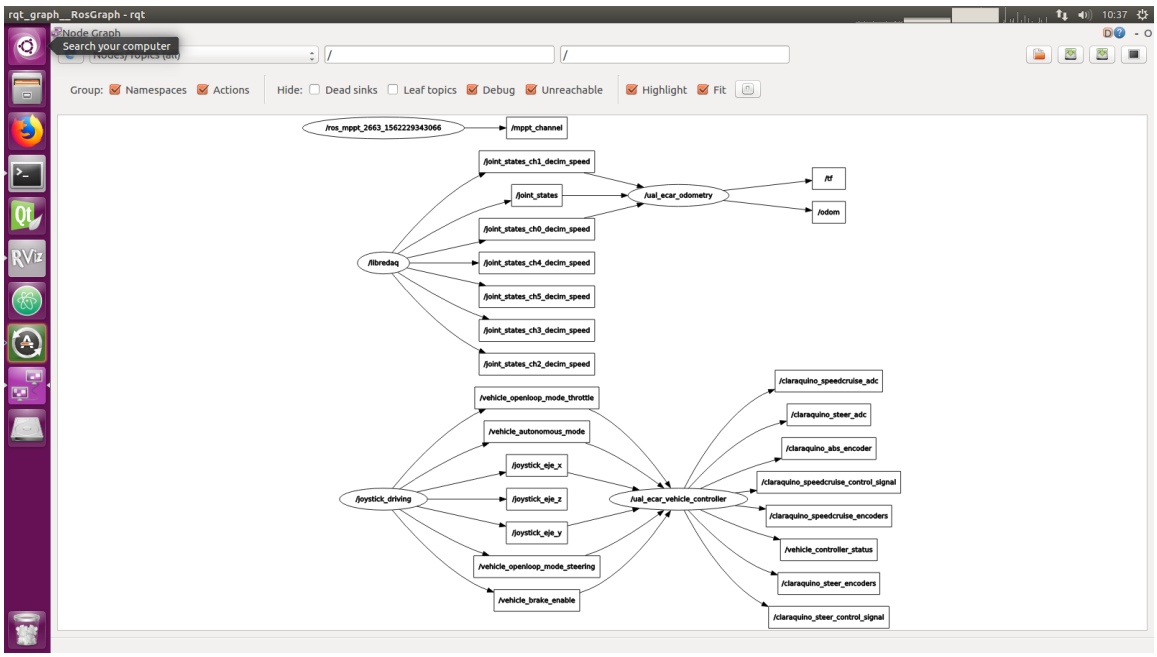


Figura 6.3: *RosGraph de todos los nodos que actúan en el sistema ROS con el roslaunch low del vehículo*

6.2. Análisis y ensayos realizados

De acuerdo con los subobjetivos propuestos en este trabajo, se ha llevado a cabo el análisis energético y el estudio del sistema fotovoltaico integrado en el vehículo eléctrico. Esta parte se ha desarrollado en el último tramo del trabajo, al realizar los ensayos con el vehículo eléctrico.

Con los ensayos realizados en el vehículo eléctrico, se pueden observar varias tendencias en las gráficas obtenidas.
El sistema fotovoltaico proporciona una mayor estabilidad en el proceso de carga y descarga del sistema de baterías del vehículo. Esto se ha podido observar en las Figuras 5.11 y 5.18, ya que han sido ensayos donde se han desconectado los módulos solares para observar el comportamiento del sistema de baterías y se pueden detectar bajadas de tensión que llegan prácticamente a 0 V (cortadas al colocar el límite de la gráfica en 40 V).
También se ha detectado una mayor caída de tensión en los ensayos sin módulos fotovoltaicos. Esto quiere decir que el sistema fotovoltaico influye de manera positiva en la reducción de la descarga de las baterías. Se puede apreciar además, que cuanto más activo es el sistema fotovoltaico, menor es la caída (Figura 5.12). Estas diferencias de caída de tensión, dependiendo de la actividad del sistema fotovoltaico, se pueden observar mejor comparando las Figuras 5.6, 5.7 y 5.8

Otro de los resultados obtenidos a partir de los valores de los ensayos es que el regulador de carga estima de forma correcta el punto de máxima potencia (MPP) de los módulos fotovoltaicos. Según los resultados de Javier Guerrero, el punto de máxima potencia se ubica en torno a los 60 V, como se presenta en la siguiente figura:

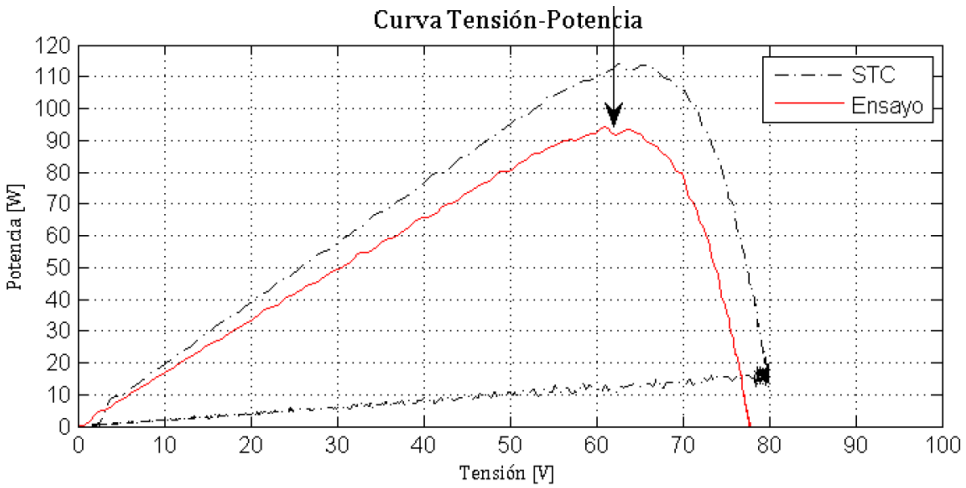


Figura 6.4: *Curva V-P característica del sistema de 4 módulos fotovoltaicos TechnoSun dispuestos en serie (Guerrero (2016))*

Representando en una gráfica los puntos de potencia de cada tramo voltaje-intensidad registrado de cada ensayo y estableciendo la media aritmética para los valores de potencia ubicados en el intervalo de 50 a 65 V (puntos donde la potencia aportada por los módulos es bastante superior) se tiene como resultado el punto de tendencia de carga del regulador.
A continuación se presentan dos gráficas V-P de los dos ensayos más duraderos, ya que se puede observar una mejor tendencia (Figuras 6.5 y 6.6).

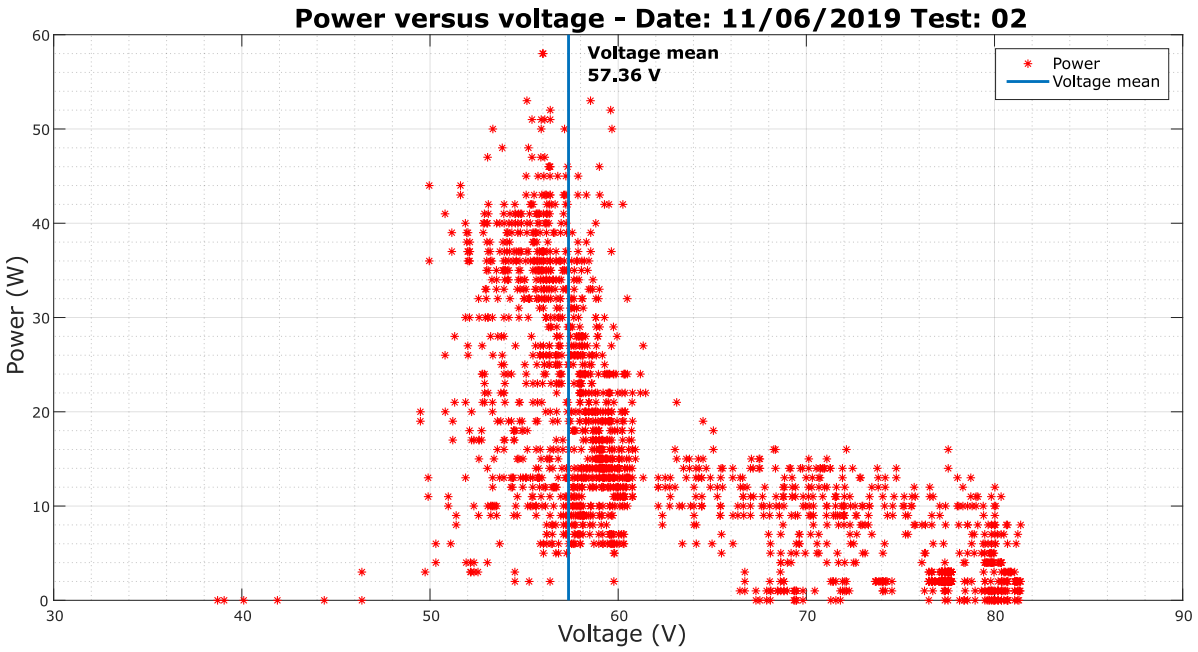


Figura 6.5: *Gráfica V-P del ensayo 02 realizado el 11/06/2019, con la tendencia de carga señalada por la recta azul*

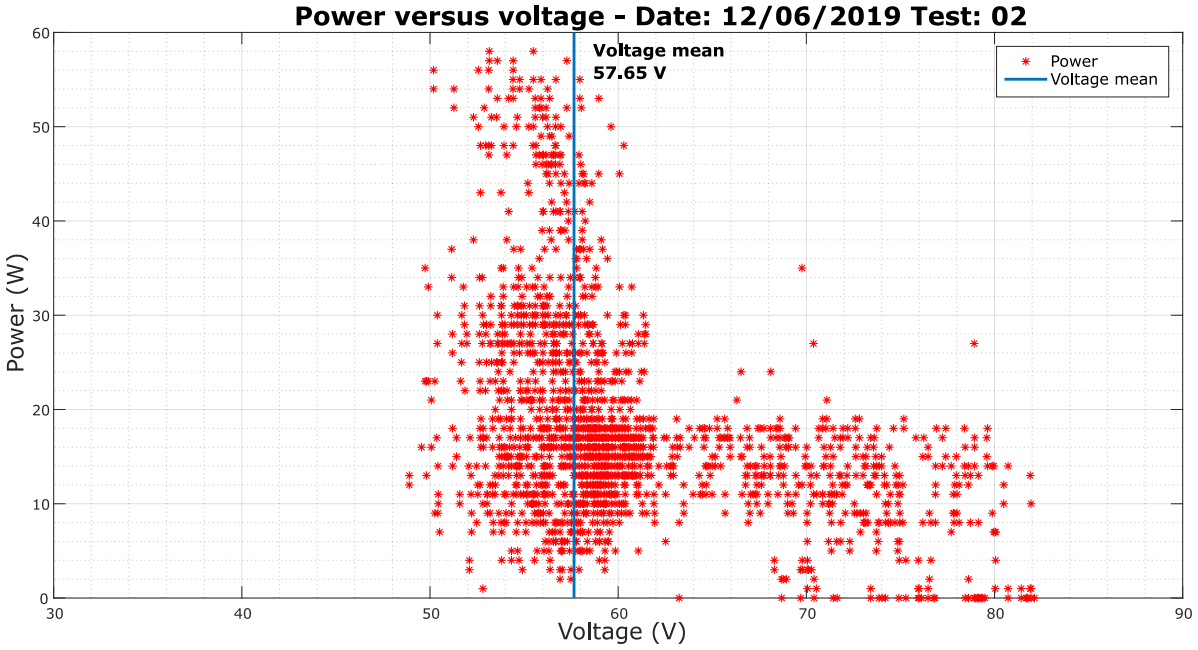
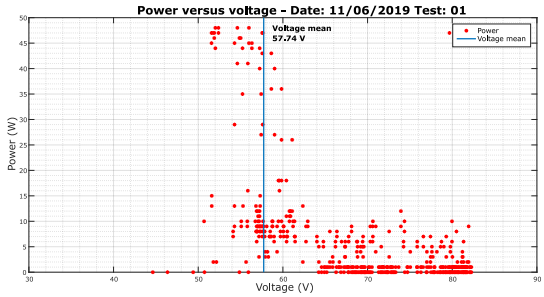
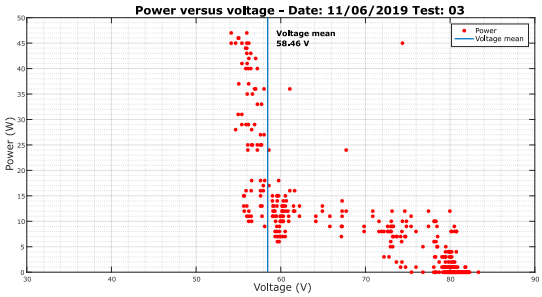


Figura 6.6: *Gráfica V-P del ensayo 02 realizado el 12/06/2019, con la tendencia de carga señalada por la recta azul*

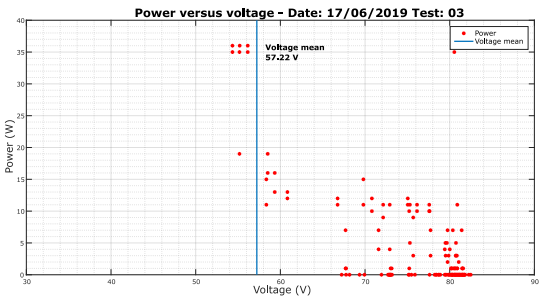
Se puede observar por las gráficas aportadas que, efectivamente, la tendencia de carga del regulador está en torno a los 58 V. La pequeña diferencia de 3 o 4 V con respecto al MPP resultante de la Curva V-P de Javier Guerrero puede ser debido a las sombras provocadas por diversos elementos del vehículo durante los ensayos (sobre todo los dos módulos instalados en la parte superior del vehículo), reduciendo el MPP real del sistema fotovoltaico instalado. También se puede deber a pérdidas de tensión en el cableado eléctrico o errores en la estimación del MPP por parte del algoritmo del regulador.
Aún así, la tendencia se puede observar en todos los ensayos realizados con el vehículo, como se puede apreciar en las siguientes dos gráficas representadas en la Figura 6.7:



(a) *Gráfica V-P Ensayo 01 del 11/06/2019*
Tendencia de carga: 57.74 V



(b) *Gráfica V-P Ensayo 03 del 11/06/2019*
Tendencia de carga: 58.46 V



(c) *Gráfica V-P Ensayo 03 del 17/06/2019*
Tendencia de carga: 57.22 V

Figura 6.7: *Gráficas V-P de ensayos realizados, con la tendencia de carga señalada por la recta azul*

En cuanto al comportamiento del propio sistema fotovoltaico, con los ensayos realizados en el capítulo 5, se ha podido comprobar que las características del montaje y los tipos de módulos empleados son determinantes. Con una corriente más elevada se podrían obtener unos resultados que caractericen al sistema en sí.
Como se ha detallado en el apartado 3.2.3 del capítulo de materiales y métodos, el regulador de carga interrumpe el periodo de absorción si cae por debajo de 2 A que, para el caso de este sistema fotovoltaico, la corriente ha estado siempre por debajo de dicho valor debido a las características del montaje en serie (definido en el apartado 3.2.2 del mismo capítulo). Se recuerda que la segunda opción del montaje en paralelo de los módulos no sería posible, ya que el voltaje en circuito abierto no supera en 5 V el voltaje nominal del sistema de baterías (41.6 V $<$ 48 V).

Para obtener unos resultados más acertados en cuanto a la caracterización del sistema se deben reemplazar los módulos fotovoltaicos por otros que generen una potencia más elevada de tal forma que la corriete del sistema fotovoltaico supere los 2 A, teniendo precaución con la sección del cableado ya instalado en el vehículo.

6.3. Trabajos futuros

Como trabajo futuro se puede realizar una caracterización más precisa del sistema fotovoltaico implementado, ya sea reemplazando los módulos fotovoltaicos actuales como aumentando la potencia haciendo uso de paneles fotovoltaicos externos. Esto último se puede realizar en puntos de recarga que funcionen con energía solar, si se quiere cernir a un caso más práctico.

También está la posibilidad de desarrollar una interfaz gráfica que muestre por pantalla diversos estados del sistema fotovoltaico, así como la posibilidad de implementar nuevos dispositivos (como sensores de temperatura, voltaje o corriente, balanceadores de carga de baterías, etcétera).

Por último, aprovechando que está abierta una línea de investigación acerca de la navegación autónoma de vehículos eléctricos, se puede emplear la arquitectura para aportar datos y estados de decisión al vehículo. Gracias a las tramas proporcionadas del regulador de carga se puede obtener información acerca del estado de carga de las baterías y el aporte de los módulos fotovoltaicos, pudiendo desarrollar modelos de control capaces de determinar cuándo el vehículo presenta la necesidad de ir a un punto de carga. Dicha duración puede servir al vehículo para trazar la ruta necesaria e ir autoconducido mediante sistemas de navegación. Además, existe la posibilidad de que el propio vehículo autogestione la energía aportada por el sistema fotovoltaico dependiendo de la situación, ya que puede establecer comunicación con el regulador de carga, alterando el algoritmo o modificando el modo de operación del mismo.

Bibliografía

[Donaldson 2008, c2009] Donaldson, Toby: *Python*. 2nd ed. Berkeley, Calif : Peachpit Press, 2008, c2009 (Visual quickstart guide). – ISBN 0-321-59098-8

[Guerrero 2016] Guerrero, Javier: *Análisis e instalación de un sistema fotovoltaico en el vehículo eléctrico UAL-eCARM*. 7 2016. – Trabajo Fin de Grado, Universidad de Almería

[IPCC 2016] IPCC: *Cambio Climático: Informe de síntesis. Guía Resumida del Quinto Informe de Evaluación del IPCC*. 2016. – URL `https://www.miteco.gob.es/es/ceneam/recursos/mini-portales-tematicos/guia-sintesis-resumida_tcm30-376937.pdf`. – consultado el 28-06-2019

[karioja 2017] karioja: *vedirect*. 2017. – URL `https://github.com/karioja/vedirect`. – consultado el 10-06-2019

[Liu und Makaran 2009] Liu, Ke ; Makaran, John: Design of a solar powered battery charger, 11 2009, S. 1 – 5

[Morgan Quigley 2015] Morgan Quigley, William D. S.: *1*. Bd. 1: *Programming Robots with ROS*. 1. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 11 2015. – ISBN 978-1-4493-2389-9

[Nayak Bhukya und Reddy Kota 2019] Nayak Bhukya, Muralidhar ; Reddy Kota, Venkata: A quick and effective MPPT scheme for solar power generation during dynamic weather and partial shaded conditions. In: *Engineering Science and Technology, an International Journal* 22 (2019), 6, Nr. 2, S. 869–884

[Poyatos 2019a] Poyatos, Aarón: *MPPT ROS Package*. 2019. – URL `https://github.com/AaronPB/ros_mppt`. – consultado el 18-07-2019

[Poyatos 2019b] Poyatos, Aarón: *MPPT ROS Package Documentation*. 2019. – URL `http://wiki.ros.org/ros_mppt`. – consultado el 18-07-2019

[Poyatos 2019c] Poyatos, Aarón: *VE MPPT Reader*. 2019. – URL `https://github.com/AaronPB/vemppt_reader`. – consultado el 18-07-2019

[Prieto 2012] Prieto, Raúl: *Cómo hacer el cálculo de la sección de los cables en una instalación eléctrica*. 9 2012. – URL `https://energias-renovables-y-limpias.blogspot.com/2012/09/calculo-seccion-cables-instalacion-electrica.html`. – consultado el 22-05-2019

[Röhling 2019] Röhling, Timo: *catkin lint documentation*. 2019. – URL `http://fkie.github.io/catkin_lint/`. – consultado el 16-05-2019

[rosdistro 2019] ROSDISTRO: *rosdistro repositories list*. 2019. – URL `https://github.com/ros/rosdistro`. – consultado el 20-05-2019

[ROS.org 2019] ROS.ORG: *Robot Operating System Documentation*. 2019. – URL `http://wiki.ros.org/`. – consultado el 10-07-2019

[SunFields 2018] SUNFIELDS: *Caja de conexiones y diodos de protección (bypass)*. 3 2018. – URL `https://www.sfe-solar.com/noticias/articulos/modulo-fotovoltaico-caja-conexiones-diodos-proteccion-bypass/`. – consultado el 24-07-2019

[VictronEnergy 2014] VICTRONENERGY: *Which solar charge controller: PWM or MPPT*. 1. 1351 JG Almere, The Netherlands: Victron Energy B.V (Veranst.), 6 2014

[VictronEnergy 2019a] VICTRONENERGY: *BlueSolar charge controller MPPT 150/35*. 3. 1351 JG Almere, The Netherlands: Victron Energy B.V (Veranst.), 7 2019

[VictronEnergy 2019b] VICTRONENERGY: *BlueSolar HEX protocol MPPT*. 3. 1351 JG Almere, The Netherlands: Victron Energy B.V (Veranst.), 7 2019

[VictronEnergy 2019c] VICTRONENERGY: *MPPT Solar Charger Error Codes*. 2019. – URL `https://www.victronenergy.com/live/mppt-error-codes`. – consultado el 17-07-2019

[VictronEnergy 2019d] VICTRONENERGY: *VE.Direct Protocol*. 3. 1351 JG Almere, The Netherlands: Victron Energy B.V (Veranst.), 7 2019. – Versión del manual: 3.25

Apéndice A

Programas realizados en Matlab

A.1. Representación gráfica V-I de los ensayos

El programa realizado en Matlab para representar gráficamente los valores obtenidos del tópico `mppt_channel` se divide en un script, donde se llama a su vez a una función que realiza la recta de regresión lineal en la gráfica V-I del sistema de baterías para estimar la caída de tensión de la misma.
El script ha quedado de la siguiente manera:

```
% Topic archives lectures
% AaronPB
clearvars;
clc;
set(0, 'DefaultFigureRenderer', 'painters');

%Data Manager
filename = 'rostopic_mppt_channel.txt';
delimiterIn = ',';
headerlinesIn = 1;
file = importdata(filename,delimiterIn,headerlinesIn);

t_raw = transpose(file.data(:,1));
d = size(t_raw);

%Time modifier to seconds values
init = t_raw(1);
for k = 1:d(2)
    t(k) = (t_raw(k)-init)*10^-9;
end

%Plot 1 - V-I CAR System
v_bat = transpose(file.data(:,2));
i_bat = transpose(file.data(:,3));

for k = 1:d(2)
    p_bat(k) = v_bat(k)*i_bat(k);
end

[regx,regy] = regresion(t,v_bat);
subplot(2,1,1);
[hAx,hLine1,hLine2] = plotyy(t,[v_bat(:) regy(:)],t,i_bat);

```

```
34 title(sprintf('V-I CAR System - Max Power: %0.1f W - Voltage difference:
       %0.1f V', max(p_bat), max(regy)-min(regy)));
35
36 set(hAx(1),'ylim',[40,60],'ytick',40:10:60);
37
38 xlabel('Time (s)');
39 ylabel(hAx(1),'Voltage (V)','color','red');
40 ylabel(hAx(2),'Current (A)','color','blue');
41 set(hAx(1),'ycolor',[0.04 0.3 0.37]);
42 set(hAx(2),'ycolor',[0 0.66 0.86]);
43 set(hLine1,'LineStyle','-');
44 set(hLine1,'color',[0.04 0.3 0.37]);
45 set(hLine2,'LineStyle','-');
46 set(hLine2,'color',[0 0.66 0.86]);
47 legend('Voltage','Current');
48 grid on;
49 grid minor;
50
51 %Plot 2 - V-I PV Panels
52 v_pv = transpose(file.data(:,4));
53 p_pv = transpose(file.data(:,5));
54
55 for k = 1:d(2)
56     i_pv(k) = file.data(k,5)/file.data(k,4);
57 end
58
59 subplot(2,1,2);
60 [hAx,hLine1,hLine2] = plotyy(t,v_pv,t,i_pv);
61
62 title(sprintf('V-I Solar Panels - Max Power: %0.1f W', max(p_pv)));
63
64 xlabel('Time (s)');
65 ylabel(hAx(1),'Voltage (V)','color','red');
66 ylabel(hAx(2),'Current (A)','color','blue');
67 set(hAx(1),'ycolor',[0.54 0.29 0.03]);
68 set(hAx(2),'ycolor',[0.87 0.45 0]);
69 set(hLine1,'LineStyle','-');
70 set(hLine1,'color',[0.54 0.29 0.03]);
71 set(hLine2,'LineStyle','-');
72 set(hLine2,'color',[0.87 0.45 0]);
73 legend('Voltage','Current');
74 grid on;
75 grid minor;
```

Código A.1: Script lectura_topic.m creado en Matlab

Del script se puede destacar que se ha tenido que forzar el modo de renderizado de Matlab a "painters" debido a que, en algunas ocasiones, al haber tantos elementos en las gráficas (sobre todo en los casos de ensayos largos) el auto-render de Matlab no ofrecía la posibilidad de guardarlo en formato vectorial .svg.

También se han tenido que realizar dos bucles para obtener la potencia de las baterías en cada dato obtenido de voltaje e intensidad (para visualizar la máxima potencia enviada) y para calcular la corriente que obtiene el regulador de carga a través de los módulos fotovoltaicos.

En cuanto a la función que realiza la regresión lineal, se ha programado empleando el ajuste lineal por el método de los mínimos cuadrados que es uno de los métodos más utilizados para averiguar si un conjunto de N datos (x_i, y_i) tienen un comportamiento lineal, y para determinar la ecuación de la recta (A.1) que mejor se ajusta a dicho conjunto.

$$y = a + bx \tag{A.1}$$

Donde a es la ordenada en el origen (definido por la ecuación A.2) y b es la pendiente de la recta (ecuación A.3):

$$a = \frac{\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i y_i}{N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2} \tag{A.2}$$

$$b = \frac{N \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2} \tag{A.3}$$

El archivo de la función ha quedado de la siguiente manera (donde x e y son tiempo y voltaje del sistema de baterías, respectivamente):

```
function [regx,regy] = regresion(x,y)
%Initial definitors

n = length(x);
A11 = 0;
A12 = 0;
A22 = n;
B1 = 0;
B2 = 0;

%Linear Regression operations
for i=1:n
    A11 = A11+x(i)^2;
    A12 = A12+x(i);
    A21 = A12;
    B1 = B1+x(i)*y(i);
    B2 = B2+y(i);
end

a = (B1*A22-B2*A12)/(A11*A22-A21*A12);
b = (B2*A11-B1*A21)/(A11*A22-A21*A12);

%Outputs
regx = x;
regy = a.*regx+b;
end
```

Código A.2: Función regresion.m creado en Matlab

A.2. Representación gráfica de la tendencia de carga

Este script se ha creado con el propósito de representar de una forma gráfica la tendencia de carga del regulador. Esta tendencia se consigue superponiendo el voltaje medio (en el tramo donde se ha detectado una potencia superior a la habitual) con la representación de la potencia transmitida al sistema a través de los módulos fotovoltaicos frente al voltaje de los mismos.
La representación gráfica se realiza a partir de los valores de potencia (p_pv) y voltaje (v_pv) registrados en cada momento por el tópico. El script queda definido a continuación.

```
% Topic archives lectures
% AaronPB
clearvars;
clc;
tend = 1;
set(0, 'DefaultFigureRenderer', 'painters');

%Data Manager
filename = 'rostopic_mppt_channel.txt';
delimiterIn = ',';
headerlinesIn = 1;
file = importdata(filename,delimiterIn,headerlinesIn);

t_raw = transpose(file.data(:,1));
d = size(t_raw);

%V-I PV Panels
v_pv = transpose(file.data(:,4));
p_pv = transpose(file.data(:,5));

%Voltage mean function
for k = 1:d(2)
    i_pv(k) = file.data(k,5)/file.data(k,4);
    if (v_pv(k) >= 50) && (v_pv(k) <=65)
        tendencia(tend) = v_pv(k);
        tend = tend+1;
    end
end

%Plot
plot(v_pv,p_pv,'r*')
title('Power versus voltage - Date: 17/06/2019 Test: 03');
line([mean(tendencia) mean(tendencia)], get(gca, 'ylim'), 'LineWidth', 2);
legend('Power','Voltage mean');
info={'Voltage mean', [num2str(mean(tendencia)),' V']};
text(mean(tendencia)+1,max(p_pv)-1,info,'FontSize',12,'FontWeight','bold');
xlabel('Voltage (V)');
ylabel('Power (W)');
xlim([30 90]);
grid on;
grid minor;
```

Código A.3: Script diagramapv.m creado en Matlab

El cálculo del voltaje medio se obtiene realizando la media de aquellos valores de voltaje que se hayan detectado en un intervalo definido. En este caso, se ha aplicado un intervalo de 50 a 65 V debido a que en dicha región se han observado las potencias máximas (que es al fin y al cabo la función del regulador de carga para optimizar en todo momento el sistema).
El bucle for en la línea 22 del script pasa por el filtro del intervalo definido (línea 24) todos los valores de voltaje. En caso de estar en dicho intervalo, se registra en el conjunto de valores (definido en la línea 25 con el vector "tendencia") del que posteriormente se calcula la media con la función `mean()` de Matlab, como se puede observar en las líneas 33 y 35 del código.

Se ha limitado la representación gráfica de 30 a 90 V debido a que en valores inferiores de voltaje, los registros de potencia no son significativos en comparación con los obtenidos en el intervalo.