

UNIVERSIDAD DE MÁLAGA  
ESCUELA TÉCNICA SUPERIOR DE  
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN  
SISTEMA DE CONTROL DE ENERGÍA  
PARA UN ROBOT MÓVIL CON UN  
MANIPULADOR INDUSTRIAL.

GRADO EN INGENIERÍA DE  
SISTEMAS ELECTRÓNICOS

DAVID FLORES PEDRAJAS  
MÁLAGA, JUNIO DE 2023



## **DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE ENERGÍA PARA UN ROBOT MÓVIL CON UN MANIPULADOR INDUSTRIAL.**

**Autor:** David Flores Pedrajas

**Tutor:** Amalia Cristina Urdiales García

**Cotutor:** Jesús Manuel Gómez de Gabriel

**Departamento:** Ingenierías de sistemas y automática

**Titulación:** Grado en Ingeniería de Sistemas Electrónicos

**Palabras clave:** Baterías, sistema de control, energía, robótica, internet de las cosas.

### **Resumen**

El presente trabajo se centra en el diseño de un sistema de control de energía para un robot móvil con un brazo robótico. La fuente de energía se basa en baterías LiFePO<sub>4</sub>, las cuales deben cumplir con los requisitos de alimentación de la plataforma y el brazo. Se ha realizado el proyecto en tres bloques de implementación:

En el desarrollo mecánico se ha logrado una optimización del espacio disponible, además de haber mejorado algunas características de la plataforma, como la funcionalidad y la seguridad. En el desarrollo eléctrico se ha diseñado el circuito completo, así como la elección de sus componentes. En el desarrollo de software hemos implementado una interfaz para la monitorización y estudio de datos del sistema de energía, a nivel local y mediante una nube IoT.

Seguidamente, se realizaron pruebas y se documentaron los datos obtenidos en ciclos de carga y descarga. Para concluir, se han logrado los objetivos iniciales establecidos en el trabajo, y se han presentado algunas mejoras y posibles líneas de desarrollo futuro.



# **DESIGN AND IMPLEMENTATION OF AN ENERGY CONTROL SYSTEM FOR A MOBILE ROBOT WITH AN INDUSTRIAL MANIPULATOR.**

**Author:** David Flores Pedrajas

**Supervisor:** Amalia Cristina Urdiales García

**Co-supervisor:** Jesús Manuel Gómez de Gabriel

**Department:** Systems Engineering and Automation

**Degree:** Bachelor's Degree in Electronic Systems Engineering

**Keywords:** Batteries, control system, energy, robotics, Internet of Things

## **Abstract**

The present work focuses on the design and implementation of an energy control system for a mobile robot with a robotic arm. The energy source is based on LiFePO<sub>4</sub> batteries, which must meet the power requirements of the platform and the arm. The project has been carried out in three implementation blocks.

Mechanical development: Optimization of the available space has been achieved, along with improvements in platform functionality and safety.

Electrical development: The complete circuit has been designed, including the selection of components.

Software development: An interface has been implemented for monitoring and studying data from the energy system, both locally and through an IoT cloud.

Subsequently, tests were conducted, and data obtained from charge and discharge cycles were documented. In conclusion, the initial objectives of the work have been achieved, and some improvements and potential avenues for future development have been presented.



A mis padres,  
por ser mis pilares fundamentales,  
por su amor incondicional y sacrificio,  
para haberme permitido llegar a donde estoy.

*David*



# Acrónimos

<b>BMS</b>	Battery Management System
<b>IoT</b>	Internet of Things
<b>PHRI</b>	Physical Human-Robot Interaction
<b>ROS</b>	Robot Operating System
<b>SoC</b>	State of Charge



# Índice

<b>Resumen</b>	<b>III</b>
<b>Abstract</b>	<b>v</b>
<b>Acrónimos</b>	<b>IX</b>
<b>I Introducción</b>	<b>1</b>
<b>1 Introducción</b>	<b>3</b>
1.1 Contexto tecnológico . . . . .	4
1.2 Objetivos . . . . .	11
1.3 Estructura de la memoria . . . . .	12
<b>II Implementación del proyecto</b>	<b>13</b>
<b>2 Especificaciones del sistema</b>	<b>15</b>
2.1 Requisitos . . . . .	20
<b>3 Desarrollo del sistema</b>	<b>25</b>
3.1 Desarrollo Mecánico . . . . .	27
3.1.1 Ubicación y dimensionamiento de las baterías . . . . .	27
3.1.2 Chasis . . . . .	29
3.1.3 Partes funcionales . . . . .	30

3.1.4	Implementación final . . . . .	36
3.2	Desarrollo Eléctrico . . . . .	38
3.2.1	Alimentación del sistema . . . . .	39
3.2.2	Cargador . . . . .	42
3.2.3	BMS y sensores de corriente . . . . .	44
3.2.4	Relé . . . . .	47
3.2.5	Inversor . . . . .	51
3.2.6	Regulador . . . . .	53
3.2.7	Distribuidor de 24 V . . . . .	55
3.2.8	Dispositivo de visualización . . . . .	55
3.3	Desarrollo Software . . . . .	57
3.3.1	Interfaz BMS - Raspberry Pi . . . . .	59
3.3.2	ThingSpeak . . . . .	59
3.3.3	ROS . . . . .	69
3.3.4	App para Smartphones . . . . .	80
<b>4</b>	<b>Verificación y pruebas</b>	<b>83</b>
4.1	Sistema de pruebas . . . . .	83
4.2	Pruebas realizadas . . . . .	85
<b>III</b>	<b>Parte tercera.</b>	<b>95</b>
	<b>Conclusiones y líneas futuras</b>	<b>97</b>
<b>IV</b>	<b>Apéndices</b>	<b>101</b>
<b>A</b>	<b>Vistas del desarrollo Mecánico</b>	<b>103</b>
<b>B</b>	<b>Manual de uso</b>	<b>111</b>
<b>C</b>	<b>Presupuesto</b>	<b>117</b>





# Índice de figuras

1.1	Tipos de BMS. Fuente: [1] . . . . .	6
1.2	Curvas de descarga de litio vs ácido plomo vs LiFePO4. Fuente: [17]	8
1.3	Métodos de estimación del SOC . . . . .	8
1.4	Método OCV. . . . .	9
1.5	Ecuación del SOC. Fuente:[20] . . . . .	9
1.6	Representación del error de la integración de corriente. Fuente:[11]	10
1.7	Estimación del SOC con filtro Kalman. . . . .	11
2.1	Modelo de la plataforma omnidireccional. Fuente: [15] . . . . .	16
2.2	Brazo robótico Panda. Fuente: [5] . . . . .	17
2.3	Batería LiFePO4. Fuente: [9] . . . . .	17
2.4	Esquema sencillo de especificaciones . . . . .	18
2.5	Sensor de mapeo láser SLAMTEC. Fuente: [16] . . . . .	22
2.6	Diagrama de funcionamiento del sensor SLAMTEC. Fuente: [16] .	23
3.1	Diagrama global del sistema . . . . .	26
3.2	Disposición inicial de las baterías (488 x 114 x 203 mm) . . . . .	27
3.3	Reubicación de las baterías . . . . .	28
3.4	Estructura de la plataforma . . . . .	29
3.5	Conjunto mediana con cargador e inversor . . . . .	29
3.6	Panel interno . . . . .	30
3.7	Carcasa de la base, impresión 3D . . . . .	31
3.8	Carcasa de la parte superior, impresión 3D . . . . .	32

3.9 Rejillas de ventilación, impresión 3D . . . . .	33
3.10 Soporte para el relé y su localización en el sistema . . . . .	34
3.11 Pulsador de emergencia y soporte para el dispositivo de visualización . . . . .	35
3.12 Conector para el cargador de la batería . . . . .	35
3.13 Diseño del sistema de alimentación completo . . . . .	36
3.14 Diseño del sistema completo implementado . . . . .	37
3.15 Esquema eléctrico del sistema . . . . .	38
3.16 Especificaciones técnicas de la batería LiFePO4. Fuente:[9] . . . . .	39
3.17 Carga y descarga de la batería LiFePO4. Fuente:[21] . . . . .	40
3.18 Curva de voltaje vs SOC. Fuente:[9] . . . . .	41
3.19 Curva de descarga en distintas temperaturas ambientales. Fuente:[9]	41
3.20 Curva de descarga en temperaturas ambientales normales. Fuente:[9]	41
3.21 Curva de carga en temperaturas normales. Fuente:[9] . . . . .	42
3.22 Capacidad de descarga en temperaturas normales. Fuente:[9] . .	42
3.23 Cargador POW24V20A-D1. Fuente:[8] . . . . .	42
3.24 Características input del cargador. Fuente:[8] . . . . .	43
3.25 Características output del cargador. Fuente:[8] . . . . .	43
3.26 Curva de carga del POW24V20A-D1. Fuente:[8] . . . . .	43
3.27 Pasos para la instalación del BMS. Pasos 1-3. Fuente:[9] . . . . .	44
3.28 Pasos para la instalación del BMS. Pasos 4-5. Fuente:[9] . . . . .	45
3.29 Configuración de los sensores de corriente. Fuente:[9] . . . . .	45
3.30 Placas del BMS. Fuente:[9] . . . . .	46
3.31 Operaciones del BMS sobre los relés de carga/descarga . . . . .	47
3.32 Especificaciones del relé.[2] . . . . .	48
3.33 Conexionado del relé.[2] . . . . .	48
3.34 Conexionado del relé en el laboratorio . . . . .	49
3.35 Conexión del pulsador de emergencia . . . . .	50
3.36 Diseño del pulsador en SolidWorks. . . . .	50
3.37 Especificaciones del inversor TS-1000. Fuente:[22] . . . . .	52

3.38 Diagrama de uso del inversor TS-1000. Fuente:[22] . . . . .	52
3.39 Especificaciones mecánicas y curva de reducción de carga del inversor TS-1000. Fuente:[22] . . . . .	53
3.40 Esquema básico de un convertidor Buck. Fuente:[13] . . . . .	54
3.41 Regulador de tensión. . . . .	54
3.42 Modelo del distribuidor de 24 V utilizado. . . . .	55
3.43 Dispositivo de visualización utilizado. . . . .	56
3.44 Soporte de impresión 3D para el display. . . . .	56
3.45 Raspberry Pi 3B. Fuente:[12] . . . . .	57
3.46 Especificaciones Raspberry Pi 3B. Fuente:[12] . . . . .	58
3.47 Cable 123SmartBMS to USB. Fuente:[4] . . . . .	59
3.48 Configuración del canal ThingSpeak . . . . .	60
3.49 Apariencia del canal ThingSpeak . . . . .	61
3.50 smartbms.py . . . . .	65
3.51 main.py . . . . .	67
3.52 Esquema de funcionamiento ROS . . . . .	69
3.53 SSH con puTTY . . . . .	71
3.54 bms-publicador-v2.py . . . . .	73
3.55 Inicialización del nodo publicador en ROS . . . . .	74
3.56 Mensaje BatteryState . . . . .	75
3.57 Mensaje CustomBatteryState . . . . .	76
3.58 PlotJuggler . . . . .	78
3.59 Plotjuggler en el dispositivo de visualización de la plataforma. . . . .	79
3.61 App de 123 Electric para Smartphones . . . . .	81
4.1 Sistema de pruebas con el proyector . . . . .	84
4.2 Datos exportados de ThingSpeak en Excel . . . . .	85
4.3 Código Matlab . . . . .	87
4.4 Primera Carga - 12/06/23 . . . . .	88
4.5 Primera Descarga - 12/06/23 . . . . .	89
4.6 Alerta de Vmin - 12/06/23 . . . . .	90

---

4.7 Segunda Carga - 16/06/23 . . . . .	91
4.8 Segunda Descarga - 20/06/23 . . . . .	92
4.9 Análisis de Matlab en ThingSpeak . . . . .	93
4.10 Interfaz de <i>IoT Data Explorer, Matlab</i> . . . . .	94
4.11 Nodo suscriptor ROS . . . . .	99
A.1 Carcasa de la base . . . . .	104
A.2 Carcasa de la parte superior . . . . .	105
A.3 Soporte para el dispositivo de visualización . . . . .	106
A.4 Imágenes del sistema completo implementado . . . . .	106
A.5 Modelo del diseño mecánico completo . . . . .	107
A.6 Base de la plataforma y panel frontal . . . . .	108
A.7 Panel lateral . . . . .	109
A.8 Panel interno . . . . .	109
B.1 Descarga de las baterías . . . . .	112
B.2 Carga de las baterías . . . . .	113
B.3 Ajustes de la app . . . . .	114

# Índice de Tablas

2.1	Tabla de especificaciones y requisitos mecánicos . . . . .	20
2.2	Tabla de especificaciones y requisitos eléctricos . . . . .	21
2.3	Tabla de especificaciones y requisitos de monitorización . . . . .	24
2.4	Tabla de especificaciones y requisitos de visualización . . . . .	24
C.1	Tabla de presupuesto . . . . .	118



# **Parte I**

## **Introducción**



# Capítulo 1

## Introducción

### Contenido

---

1.1	Contexto tecnológico	4
1.2	Objetivos	11
1.3	Estructura de la memoria	12

---

### Sinopsis

El proyecto tiene un enfoque en la investigación sobre la interacción física robot/humano (pHRI), con la finalidad de poder desarrollar un robot capaz de proporcionar asistencia física a las personas, considerando aspectos como el agarre de objetos, el levantamiento de pesos y la movilidad omnidireccional en un entorno doméstico. Esto implica investigar y desarrollar técnicas y algoritmos que permitan al robot comprender y responder de manera adecuada a las necesidades y comandos de los usuarios humanos, asegurando una interacción segura, ergonómica y eficiente.

El desarrollo de un robot con un brazo de dimensiones moderadas, una controladora eficiente y una estructura estable es fundamental para lograr una interacción segura y efectiva entre el robot y los humanos. La estabilidad de la plataforma, en relación con el uso de baterías pesadas, es crucial para garantizar la seguridad durante la asistencia física.

## 1.1. Contexto tecnológico

**Robots móviles** El avance tecnológico en el campo de los robots móviles ha revolucionado diversas industrias y sectores, demostrando su potencial para realizar tareas automatizadas en entornos específicos. A diferencia de los robots industriales estáticos, que se encuentran ensamblados en una maquinaria fija, los robots móviles son capaces de desplazarse de manera autónoma dentro de una zona determinada.

Estos robots dinámicos han despertado un gran interés en la comunidad científica e industrial, convirtiéndose en el foco principal de investigación en laboratorios de todo el mundo. Se estima que los avances en la robótica móvil están transformando sectores como la industria, la milicia, los sistemas de salud y la seguridad, entre otros. Los robots móviles presentan una serie de ventajas y aplicaciones prácticas. Su capacidad de movimiento les permite adaptarse a diferentes entornos y realizar tareas en lugares de difícil acceso para los humanos. Además, su autonomía les permite llevar a cabo operaciones repetitivas y peligrosas sin poner en riesgo la seguridad de los trabajadores.

En la industria, los robots móviles están siendo utilizados para la automatización de procesos de producción y logística. Su capacidad para desplazarse de manera eficiente y realizar tareas de forma autónoma mejora la productividad y reduce los costos operativos.

**Tecnologías desarrolladas** En los últimos años, hemos sido testigos de mejoras significativas en la eficiencia y capacidad de las baterías utilizadas en los robots móviles. Las baterías de litio y las de polímero de litio (Li-Po) son ampliamente utilizadas debido a su alta densidad de energía, larga vida útil y capacidad de carga rápida, ya que ofrecen una mayor capacidad energética en comparación con las baterías de plomo-ácido o níquel-cadmio (Ni-Cd), lo que se traduce en una mayor autonomía de los robots móviles. Cabe destacar las baterías de litio ferrofósfato (LiFePO<sub>4</sub>), ya que se han convertido en una opción popular debido a su seguridad, alta densidad de energía, menor autodescarga y mayor vida útil en comparación con las tecnologías más tradicionales. Otra tecnología importante a considerar es el sistema de gestión de baterías (BMS). Un BMS es esencial para garantizar un rendimiento óptimo de las baterías y prolongar su vida útil. Proporciona funciones de monitorización y protección, como la supervisión del estado

de carga, la temperatura y la corriente, la protección contra sobrecargas y descargas excesivas, y la gestión del equilibrio de las celdas individuales. El uso de un BMS en una batería LiFePO<sub>4</sub> es especialmente importante debido a las características específicas de esta tecnología.

Tipos de BMS:

- Centralizado: el BMS se encuentra montado en la batería o en su proximidad. Se conecta un cable desde cada pin de entrada del BMS a cada celda del grupo de batería.
- Modular: utiliza la misma tecnología que el BMS centralizado, pero con varios módulos distribuidos en el grupo de baterías. Estos módulos recopilan y envían la información de cada celda al módulo maestro, que controla la comunicación entre ellos. Esta configuración reduce el ruido y la caída de tensión de las mediciones, lo que aumenta la precisión.
- Maestro-Esclavo: similar al BMS modular, utiliza varios módulos esclavos que realizan mediciones en varias celdas cada uno y envían la información al módulo maestro para su gestión. La fabricación y el costo son similares al BMS modular, aunque las placas esclavo son más económicas debido a la presencia de menos componentes de control.
- Distribuido: se coloca un pequeño circuito de medición en cada celda de la batería, obteniendo mediciones precisas y libres de ruido. Los datos de cada celda se envían a través de un cable de comunicación conectado a un circuito controlador, donde se procesan y gestionan. Esta configuración es la más precisa pero también la más costosa debido a la mayor cantidad de componentes requeridos.

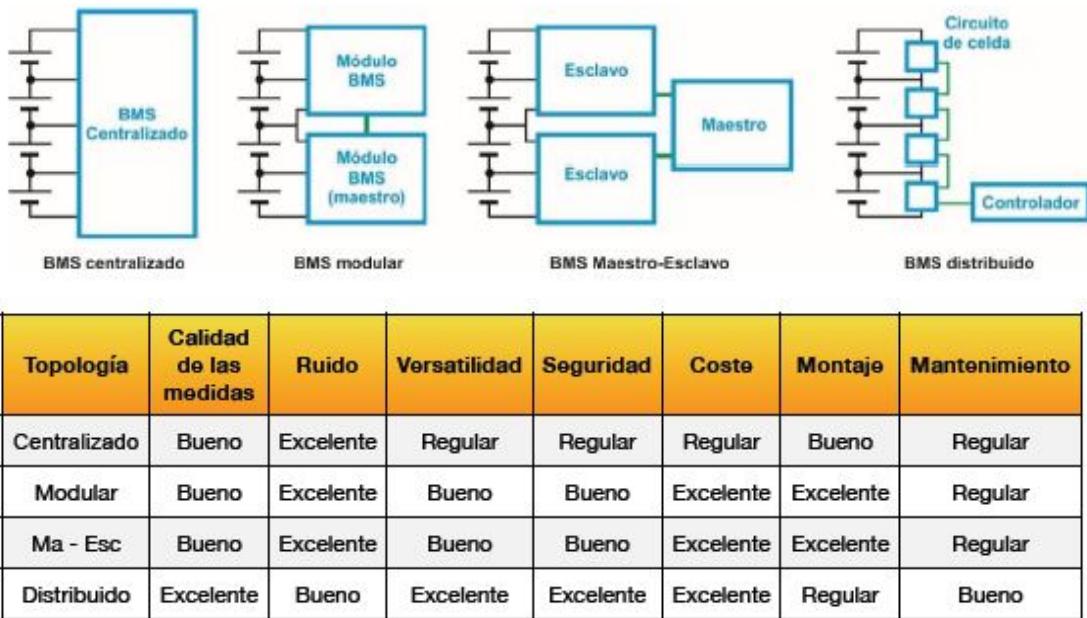


Figura 1.1: Tipos de BMS. Fuente: [1]

**Mercado actual** La competencia en el mercado de sistemas de gestión de baterías es intensa y existe una marcada polarización técnica. Con el creciente interés en los vehículos de nueva energía y el mercado en general, la tecnología de gestión de baterías ha ganado una gran atención en China. Numerosos fabricantes especializados, empresas de baterías eléctricas y fabricantes de vehículos han comenzado a investigar y producir en este campo. En China, hay una gran cantidad de fabricantes de BMS, y la ma-

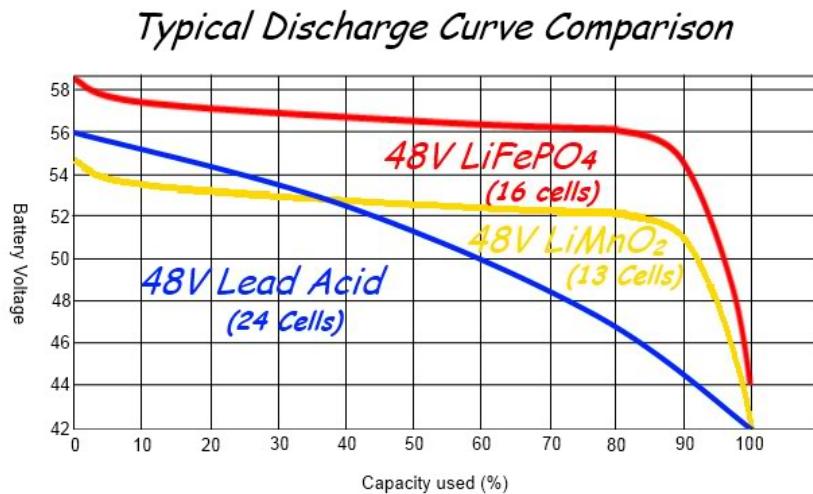
yoría de ellos son empresas privadas, lo que indica una competencia feroz en el mercado. Desde una perspectiva técnica, también existe una polarización significativa. Algunas empresas han dominado tecnologías avanzadas de sistemas de gestión de baterías a través de investigaciones propias o adquisiciones, como es el caso de Junsheng Electronics, mientras que otras solo han podido desarrollar productos BMS de baja gama. Actualmente, cada fabricante de sistemas tiene diversas opciones para seleccionar un BMS, lo que ha brindado oportunidades de desarrollo para la mayoría de los fabricantes de BMS. En el futuro, a medida que aumente

el número de empresas dedicadas a sistemas de gestión de baterías y los requisitos técnicos continúen evolucionando, se espera que la indus-

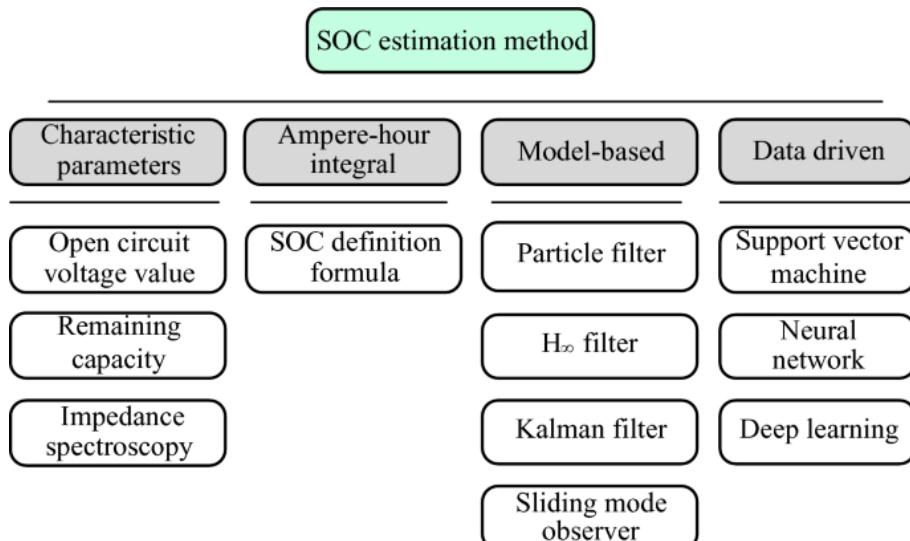
tria experimente una reorganización para adaptarse a estas demandas crecientes.[10]

**Estado de carga y cálculos en baterías LiFePO<sub>4</sub>** La estimación del Estado de Carga es una función importante del sistema de gestión de baterías y es la base para la implementación de las estrategias de control. En los últimos 50 años, los académicos han propuesto varios métodos clásicos para la estimación del nivel de batería en sus publicaciones, cada uno de los cuales tiene su propio ámbito de aplicación.[18] Todos los tipos de baterías comparten una característica común: el voltaje en sus terminales disminuye o aumenta en función de su nivel de carga. Cuando la batería está completamente cargada, el voltaje será más alto, y cuando esté vacía, será más bajo. La relación entre el voltaje y el estado de carga depende directamente de la tecnología de la batería utilizada. Por ejemplo, si comparamos las curvas de descarga de una batería de plomo y una batería de iones de litio, podemos observar diferencias significativas. Las baterías de plomo-ácido tienen una curva de descarga relativamente lineal, lo que facilita una buena estimación del estado de carga. Con base en una medición de voltaje, es posible estimar con precisión el valor del SoC asociado. Sin embargo, las baterías de iones de litio presentan una curva de descarga mucho más plana, lo que significa que el voltaje en los terminales de la batería cambia muy poco en un amplio rango de operación. A diferencia de otras tecnologías mencionadas, en la tecnología de LiFePO<sub>4</sub>, el voltaje permanece casi constante durante la mayor parte de su ciclo de descarga.

Esta característica dificulta la estimación precisa del SOC utilizando únicamente la medición de voltaje. Éste se puede calcular mediante algoritmos basados en modelos, que tienen en cuenta factores como la corriente de carga y descarga, la resistencia interna de la batería y las condiciones ambientales. Sin embargo, estos algoritmos pueden ser complejos y requieren un procesamiento de datos considerable.



**Figura 1.2: Curvas de descarga de litio vs ácido plomo vs LiFePO4. Fuente: [17]**



**Figura 1.3: Métodos de estimación del SOC**

**Open Circuit Voltage** El método OCV se refiere al voltaje de una batería cuando no hay corriente a través de ella y no está conectada a ninguna carga. Mide el potencial eléctrico de la batería en reposo, sin carga ni descarga activa.

Se necesita una curva de calibración para relacionar el OCV con el nivel de carga. La curva se obtiene midiendo el OCV de la batería en diferentes puntos de carga y descarga conocidos y luego estableciendo una correspondencia entre el OCV y el SOC medido en esos puntos.[3]

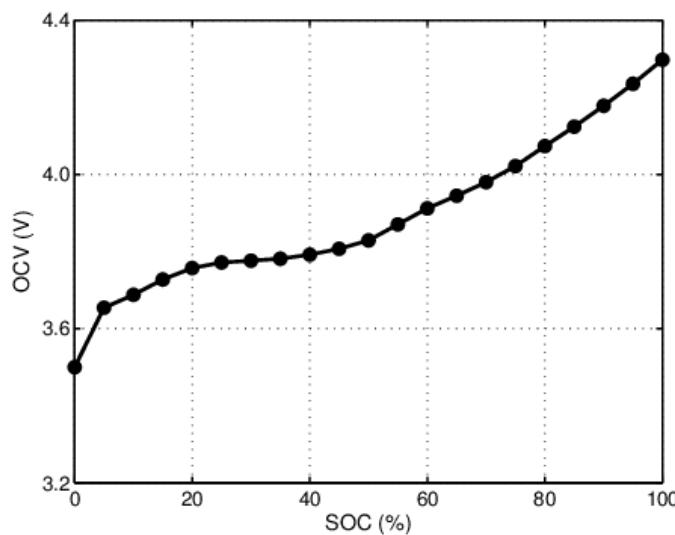


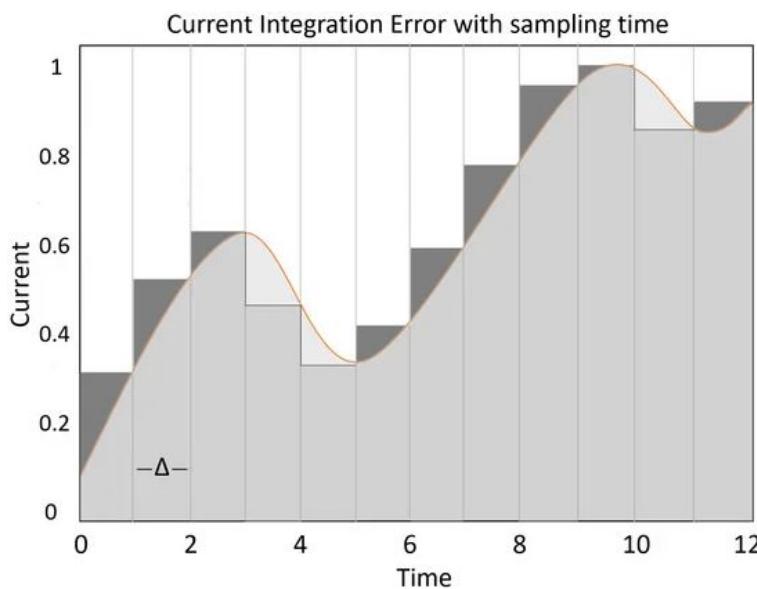
Figura 1.4: Método OCV.

**Conteo de Coulomb** En éste método, se determina la carga acumulada integrando las corrientes de carga y descarga durante el tiempo.

A diferencia del método OCV, este método es capaz de determinar la evolución del estado de carga durante el uso de la batería. No requiere que la batería esté en reposo para realizar una medición precisa.

$$SOC(t) = SOC(t - 1) + \int_0^t \frac{I}{C_{bat}} \cdot dt$$

Figura 1.5: Ecuación del SOC. Fuente:[20]



**Figura 1.6: Representación del error de la integración de corriente. Fuente:[11]**

El SOC en un determinado momento  $t$  se determina conociendo un estado de carga inicial  $SO(t - 1)$ ,  $I$  es la corriente y  $C_b$  es la capacidad total de la celda (Ah).

Un valor de SOC de 1 es equivalente a una celda completamente cargada, mientras que un valor de 0 representa la descarga.

Aunque el método de conteo de Coulomb es útil, puede sufrir de errores acumulativos a lo largo del tiempo debido a las inexactitudes en la medición de corriente y a las posibles pérdidas de energía.

**Filtro Kalman** El filtro de Kalman utiliza un modelo matemático del sistema y combina la información de las mediciones previas con las mediciones actuales para obtener una estimación óptima del SOC actual. El algoritmo funciona en dos pasos principales:

Paso de predicción, se utiliza el modelo matemático del sistema para predecir el estado futuro y su incertidumbre. Se estima cómo evolucionará el estado en el siguiente paso temporal sin tener en cuenta las mediciones actuales.

Paso de actualización, se compara la predicción del estado con la medición actual para corregir la estimación. Se utiliza una ponderación basada en la incertidumbre de la predicción y la incertidumbre de la medición para combinar ambas fuentes de información de manera óptima.

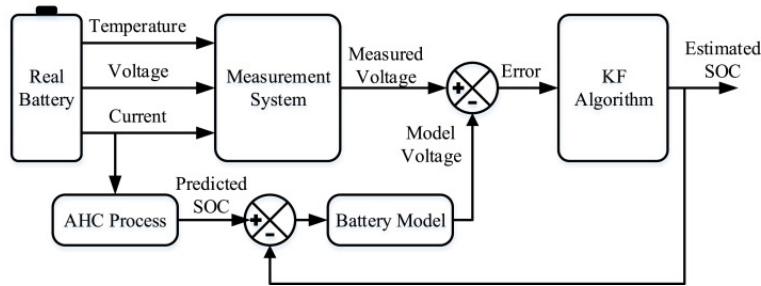


Figura 1.7: Estimación del SOC con filtro Kalman.

## 1.2. Objetivos

**Idea general** En este trabajo se propone diseñar e implementar un sistema de control de energía para un robot móvil con un manipulador industrial.

El diseño del sistema de control comprende las siguientes características: diseño del sistema eléctrico, diseño mecánico de la ubicación de las baterías, la electrónica de control, cableado e interfaz de usuario.

**Alimentación del sistema** La energía se almacena en baterías de tipo LiFePO4. A diferencia de las baterías de Li-ion, éstas tienen la ventaja de ser más seguras por su composición, pero la estimación del estado de carga es más compleja. Por ello, es necesario incluir un sistema que estime el SoC basado en las corrientes de entrada y salida. Asimismo, para proteger la vida útil de las celdas, el sistema incorpora relés de alta potencia para la desconexión de la energía del sistema.

**Interfaz** El sistema tendrá una interfaz que permita conocer el estado de carga, consumo de energía del robot, así como la potencia de carga en cada momento, a través de una interfaz local o mediante Internet of Things (IoT), además de una aplicación para smartphones.

### 1.3. Estructura de la memoria

Tras haber realizado una introducción sobre el sistema que desarrollaremos, se explicará a continuación los distintos puntos que trataremos para la realización de este proyecto.

**Especificaciones** Requisitos eléctricos y mecánicos, descripción de las funciones a conseguir con el desarrollo, soporte que conlleva, restricciones de partida.

**Desarrollo del sistema** descripción detallada de la implementación del sistema. Desarrollos mecánico, eléctrico y de software.

**Verificación y pruebas** sistema empleado para llevar a cabo las pruebas sobre el sistema, para verificar su correcto funcionamiento.

**Conclusiones** aportaciones del trabajo así como sus posibles extensiones, y las líneas futuras que permitirían su mejora

**Apéndices** información complementaria

## **Parte II**

### **Implementación del proyecto**



# Capítulo 2

## Especificaciones del sistema

### Contenido

---

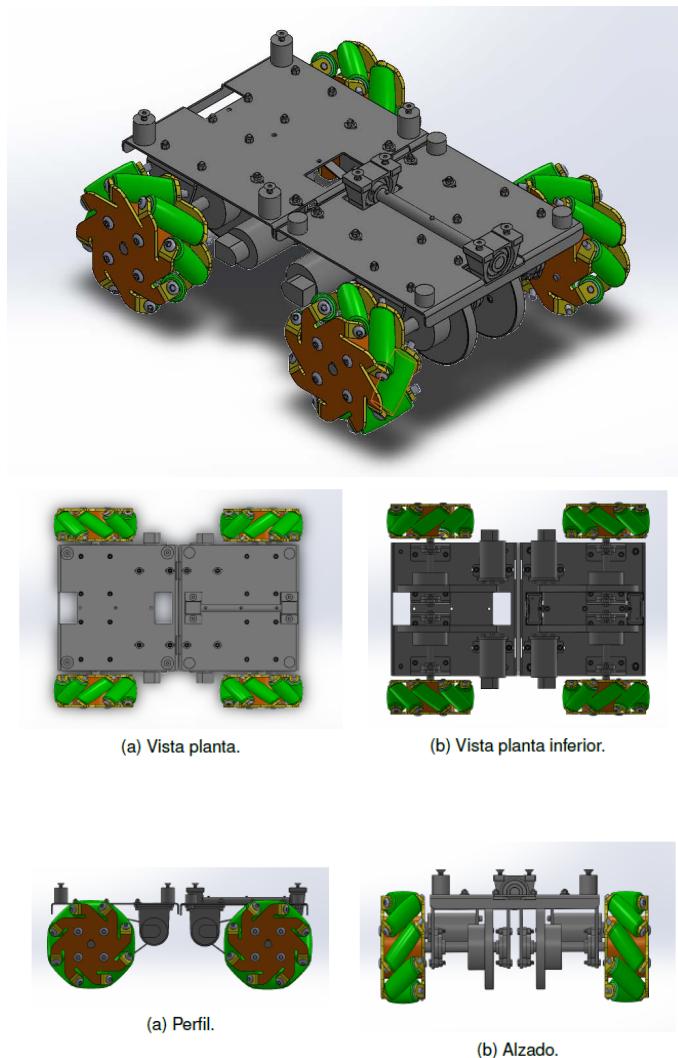
2.1 Requisitos . . . . .	20
--------------------------	----

---

### Sinopsis

Para llevar a cabo la descripción de las funciones a conseguir con el desarrollo del sistema, partimos de la necesidad de implementar un sistema de control de energía para un robot móvil con un manipulador robótico. Se detallarán las especificaciones técnicas y funcionales del sistema, cuyo objetivo principal es garantizar un funcionamiento seguro y eficiente del sistema, cumpliendo con los requisitos establecidos. Las especificaciones abarcan aspectos mecánicos, eléctricos, así como de interfaz de visualización de datos.

**Plataforma omnidireccional.** Robot omnidireccional con cuatro ruedas Mecanum, diseñado y desarrollado por Alonso Serrano Flores en su Trabajo Fin de Grado [15]. Estas ruedas están alimentadas, cada una, con un motor de 24 V que consume 60 W.



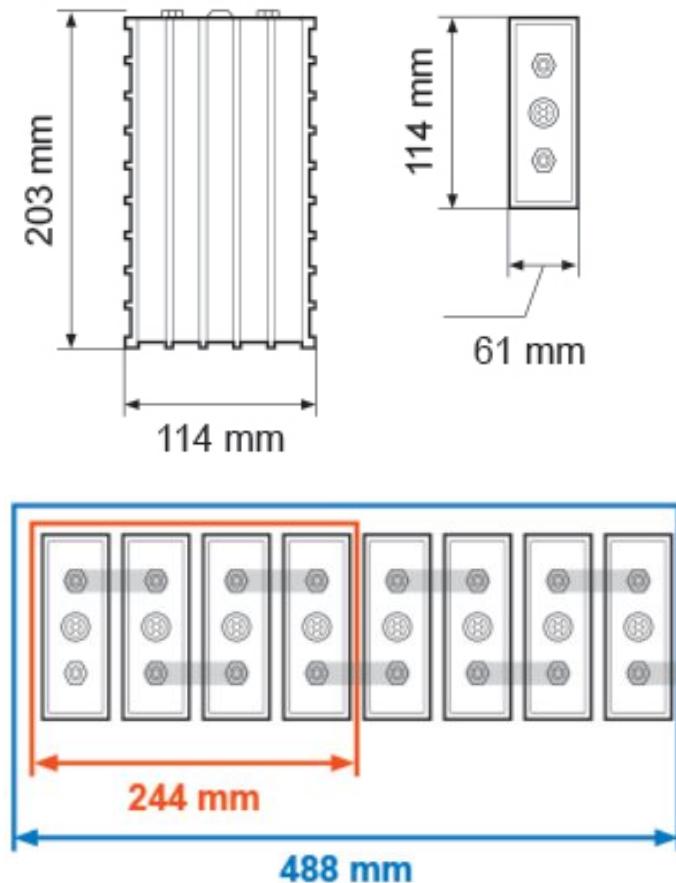
**Figura 2.1: Modelo de la plataforma omnidireccional. Fuente: [15]**

**Manipulador robótico.** Robot articulado de 7 ejes, modelo "Panda" de la marca FRANKA EMIKA. Éste brazo robótico requiere una alimentación de entre 100-240 V en corriente alterna (AC), consume una potencia máxima de 350 W y tiene una capacidad de carga de 3 Kg.

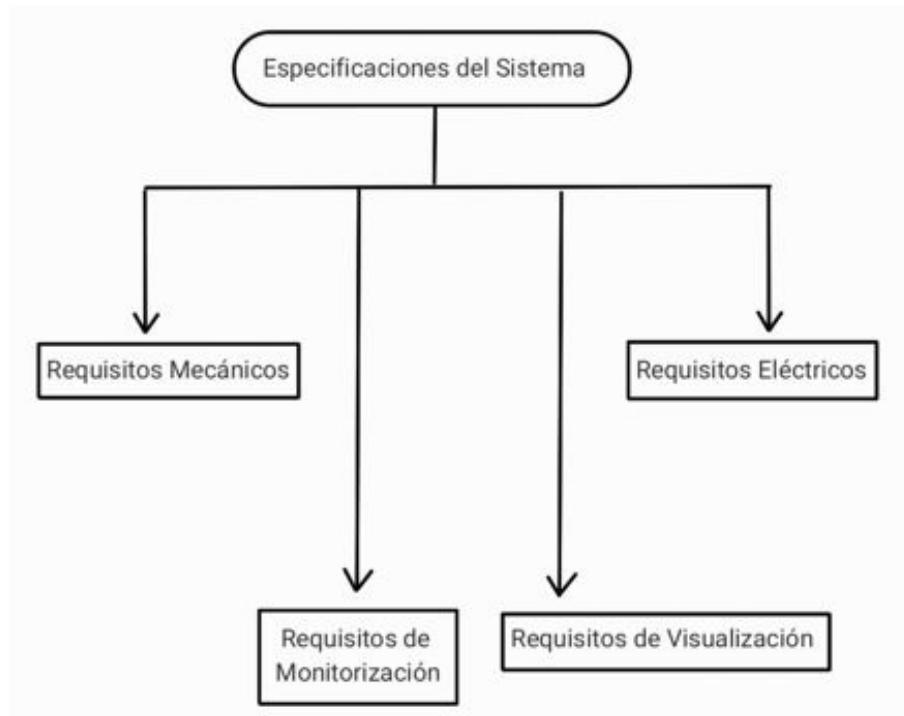


**Figura 2.2: Brazo robótico Panda. Fuente: [5]**

**Sistema energético.** Disponemos de ocho celdas de baterías de tipo LiFePO4, modelo "Winston LFP060AHA" de la marca GWL.



**Figura 2.3: Batería LiFePO4. Fuente: [9]**



**Figura 2.4: Esquema sencillo de especificaciones**

**Requisitos mecánicos.** Crucial para asegurar la correcta integración del sistema de energía en el robot móvil y una distribución equilibrada de los componentes.

**Requisitos eléctricos.** Esenciales para el funcionamiento óptimo del sistema energético. La capacidad de las baterías, así como su compatibilidad con los demás componentes del sistema, deben ser cuidadosamente evaluadas.

**Requisitos de monitorización.** Garantizan un funcionamiento seguro, permiten optimizar el rendimiento y prolongar la vida útil de la batería.

**Requisitos de visualización.** La inclusión de un dispositivo de visualización local en el panel frontal del sistema permite a los usuarios acceder de manera rápida, en tiempo real y sin conexión a Internet, además de una app y visualización de datos mediante IoT.



## 2.1. Requisitos

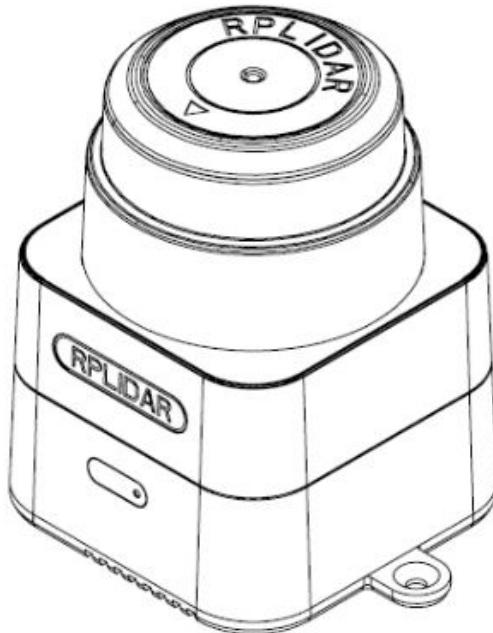
Requisitos Mecánicos			
Id.	Nombre	Descripción	Tipo
1	Orientación baterías	Las baterías deben colocarse de forma vertical, con la válvula de seguridad y los terminales hacia arriba.	Seguridad
2	Dimensiones físicas	El sistema de energía debe ajustarse a las dimensiones específicas (44 cm de largo, 27.5 cm de ancho y 23.4 cm de alto) para garantizar su integración en el robot móvil.	Diseño
3	Pulsador emergencia	El sistema debe incluir un accionador para activar/desactivar la alimentación de salida de las baterías.	Seguridad
4	Integración cargador interno	El sistema debe incluir un cargador interno que permita cargar las baterías de forma conveniente y funcional, sin necesidad de desconectarlas del robot móvil.	Funcional
5	Fijación baterías	Las baterías y otros componentes del sistema deben ser compactados y aislados eléctricamente mediante carcasa impresa en 3D para garantizar su seguridad y protección.	Diseño
6	Rejillas de ventilación	El sistema debe incluir rejillas de ventilación adecuadas para permitir una adecuada disipación de calor y mantener una temperatura óptima de funcionamiento.	Funcional
7	Dispositivo de visualización local	El panel frontal del sistema debe contar con un dispositivo de visualización, como interfaz de usuario local, que muestre información relevante sobre el estado de las baterías y su monitorización.	Funcional
8	Tapa transparente	El sistema debe contar con una tapa transparente en la parte superior que permita la visualización del estado de las baterías y el BMS esté funcionando correctamente.	Diseño

Tabla 2.1: Tabla de especificaciones y requisitos mecánicos

Requisitos Eléctricos			
Id.	Nombre	Descripción	Tipo
1	Tensión de las baterías	Las baterías deben suministrar un voltaje de 24 V DC. Con ésta tensión debemos alimentar los motores de la plataforma móvil	Funcional
2	Voltaje AC	Se requiere un inversor para convertir el voltaje de 24 V DC a 220 V AC.	Funcional
3	Voltaje DC	El sistema debe incluir un regulador de tensión (24 V a 5 V) para la alimentación de otros dispositivos necesarios, como sensores y otros elementos.	Funcional
4	Potencia	El sistema debe suministrar una potencia pico de 1000 W para poder abastecer los requisitos de consumo del brazo robótico y los motores de la plataforma, así como otros dispositivos que consuman alta potencia.	Funcional
5	Protección contra sobrecorriente	Se deben utilizar fusibles adecuados para prevenir daños en caso de sobrecorriente..	Seguridad
6	Relé de alta potencia	El sistema debe incluir un relé que controle el flujo de corriente hacia las cargas eléctricas, permitiendo gestionar la energía de forma conveniente y segura.	Funcional
7	Aislamiento eléctrico	Se deben utilizar componentes y conexiones adecuadas para asegurar un correcto aislamiento eléctrico y evitar problemas de interferencia o descargas.	Fiabilidad
8	Eficiencia energética	Se debe buscar la eficiencia energética en el diseño del sistema con el fin de aprovechar al máximo la energía y prolongar la duración de las baterías.	Eficiencia
9	Batería segura	La batería debe cumplir con estándares de seguridad, evitando riesgos de explosiones, fugas o sobrecalentamiento..	Seguridad
10	Baja autodescarga	La batería debe tener una baja tasa de autodescarga para mantener su carga durante largos períodos de inactividad.	Funcional

Tabla 2.2: Tabla de especificaciones y requisitos eléctricos

Haciendo referencia a los requisitos de obtener una alimentación DC menor (5 V) gracias al uso del regulador, se empleará un sensor de mapeo láser, como tecnología para la navegación autónoma y la percepción del entorno en el robot móvil. Estos sensores permiten al robot capturar información del entorno circundante y generar un mapa detallado y preciso.



**Figura 2.5: Sensor de mapeo láser SLAMTEC. Fuente: [16]**

El mapeador SLAMTEC es un nuevo tipo de sensor láser, el cual tiene funciones integradas de localización y mapeo simultáneos (SLAM) y es adecuado para muchas aplicaciones, como la navegación y posicionamiento de robots, mapeo ambiental y medición portátil.

Utiliza un motor de optimización de mapas SLAM de alto rendimiento y la tecnología de mapeo fino SharpEdge, que puede detectar y corregir activamente bucles cerrados, lograr mapas y posiciones de alta precisión.

Puede funcionar sin sensores adicionales ni entrada de datos. Gracias al sistema de navegación inercial de 9 grados de libertad incorporado, en el modo de mapeo portátil puede funcionar normalmente en entornos fluctuantes con inclinación, para garantizar la mejor calidad de los datos del mapa.

Funcionamiento: El kit del mapeador SLAMTEC utiliza el algoritmo de optimización SLAM único de SLAMTEC y un LIDAR<sup>1</sup> de alto rendimiento para fusionar datos de mapas más de 10 veces por segundo y construir áreas de mapeo de hasta 100000 metros cuadrados. El LIDAR realiza 9200 mediciones por segundo, y la distancia máxima de alcance puede alcanzar los 40 metros.

Además de sus funciones de mapeo y localización incorporadas, SLAMTEC Mapper también puede ser utilizado como un sensor láser. Los usuarios pueden integrar rápidamente los datos LIDAR en su sistema existente a través del SDK<sup>2</sup> o el nodo ROS<sup>3</sup>.

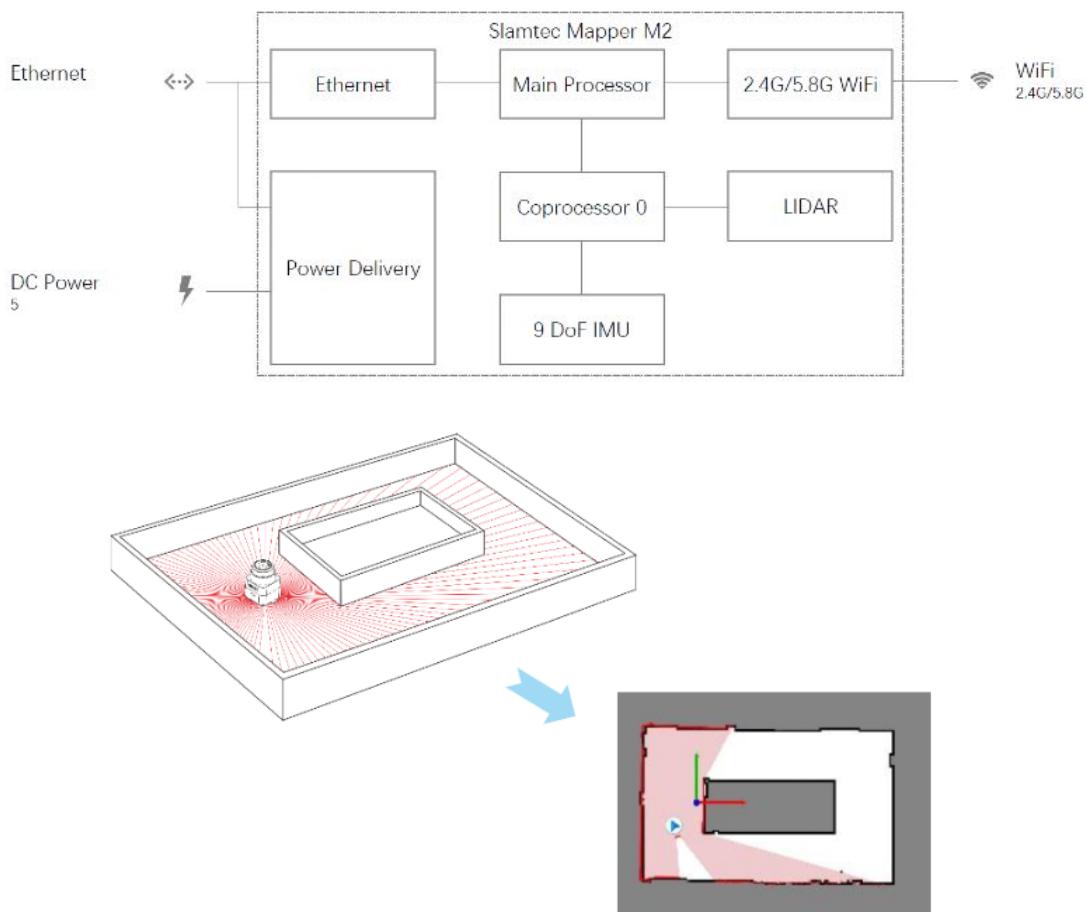


Figura 2.6: Diagrama de funcionamiento del sensor SLAMTEC. Fuente: [16]

<sup>1</sup>Laser Imaging Detection and Ranging

<sup>2</sup>Software Development Kit

<sup>3</sup>Robot Operating System

Requisitos de Monitorización			
Id.	Nombre	Descripción	Tipo
1	Monitorización de SoC	El sistema debe ser capaz de monitorizar el estado de carga de las baterías en tiempo real.	Funcional
2	Monitorización de voltaje individual	El sistema debe monitorizar el voltaje de cada celda de la batería individualmente.	Funcional
3	Monitorización de corrientes	Se requiere la monitorización de la corriente de carga y descarga de la batería.	Funcional
4	Monitorización de temperatura	Se necesita la monitorización de la temperatura de la batería para evitar sobrecalentamiento.	Funcional
5	Notificación de fallos	Se requiere la detección y notificación de cualquier fallo o error en el sistema.	Funcional
6	Generación de Informes y Registros Periódicos	El sistema debe generar informes y registros periódicos sobre el rendimiento y la salud de las baterías	Funcional
7	Monitorización de energía	El sistema debe monitorizar la energía consumida y almacenada.	Eficiencia

Tabla 2.3: Tabla de especificaciones y requisitos de monitorización

Requisitos de Visualización			
Id.	Nombre	Descripción	Tipo
1	Interfaz gráfica local	El sistema debe contar con una interfaz gráfica local para visualizar los datos de monitorización sin necesidad de conexión a internet.	Funcional
2	Visualización en la Nube mediante IoT	El sistema debe ser compatible con tecnología de IoT para permitir la visualización de datos en la nube.	Funcional

Tabla 2.4: Tabla de especificaciones y requisitos de visualización

# Capítulo 3

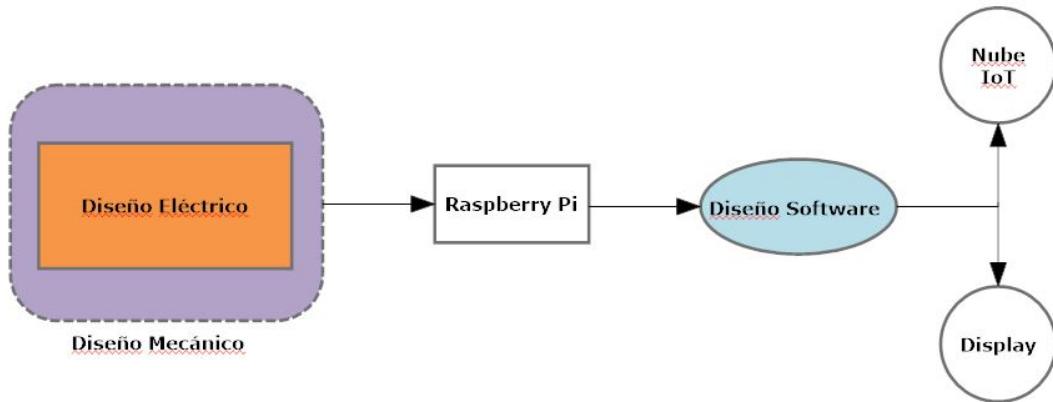
## Desarrollo del sistema

### Contenido

---

<b>3.1 Desarrollo Mecánico</b> . . . . .	<b>27</b>
3.1.1 Ubicación y dimensionamiento de las baterías . . . . .	27
3.1.2 Chasis . . . . .	29
3.1.3 Partes funcionales . . . . .	30
3.1.4 Implementación final . . . . .	36
<b>3.2 Desarrollo Eléctrico</b> . . . . .	<b>38</b>
3.2.1 Alimentación del sistema . . . . .	39
3.2.2 Cargador . . . . .	42
3.2.3 BMS y sensores de corriente . . . . .	44
3.2.4 Relé . . . . .	47
3.2.5 Inversor . . . . .	51
3.2.6 Regulador . . . . .	53
3.2.7 Distribuidor de 24 V . . . . .	55
3.2.8 Dispositivo de visualización . . . . .	55
<b>3.3 Desarrollo Software</b> . . . . .	<b>57</b>
3.3.1 Interfaz BMS - Raspberry Pi . . . . .	59
3.3.2 ThingSpeak . . . . .	59
3.3.3 ROS . . . . .	69
3.3.4 App para Smartphones . . . . .	80

---



**Figura 3.1: Diagrama global del sistema**

La fase de desarrollo de este trabajo representa un paso fundamental para llevar a cabo los objetivos propuestos. En este capítulo, se llevará a cabo la implementación y construcción del sistema de control de energía para el robot móvil con el manipulador robótico.

Éste capítulo implica la sinergia de diferentes componentes y tecnologías, así como la implementación de algoritmos y sistemas de control que permitirán el monitoreo y la gestión eficiente de la energía utilizada por el robot. Se buscará optimizar el rendimiento energético y garantizar el suministro adecuado de energía en cada etapa del funcionamiento del robot.

Para lograr éste objetivo, realizaremos un análisis detallado de los requisitos, tomando en consideración aspectos mecánicos, eléctricos y de software. Se seleccionarán los componentes más adecuados, como las baterías LiFePo4 y el BMS, que cumplirán con las necesidades específicas del sistema.

Asimismo, se explorarán tecnologías emergentes y se emplearán herramientas de desarrollo avanzadas, como el uso de IoT y ROS, con la finalidad de facilitar la integración de los diferentes elementos del sistema y la comunicación eficiente entre ellos.

Tras finalizar el capítulo de desarrollo, continuaremos con el proceso de pruebas del sistema para garantizar el cumplimiento de los requisitos establecidos y la correcta funcionalidad del sistema.

## 3.1. Desarrollo Mecánico

### 3.1.1. Ubicación y dimensionamiento de las baterías

La reubicación de las baterías representa un aspecto fundamental en este apartado de desarrollo mecánico. Inicialmente, las baterías se localizaban en una posición con limitaciones en términos de espacio, comprometiendo la funcionalidad y seguridad del sistema.

Por lo tanto, nos surge la necesidad de encontrar una nueva ubicación debido a la falta de espacio en la plataforma del robot móvil y a la necesidad de lograr una configuración más compacta y segura. La disposición inicial de las baterías no permitía aprovechar de forma eficiente el espacio disponible, lo que afectaba la integridad estructural y el rendimiento en general del sistema.

Además, se debe considerar la accesibilidad para el mantenimiento y reemplazo de las baterías, así como la implementación de medidas de seguridad para prevenir cortocircuitos o sobrecalentamiento.

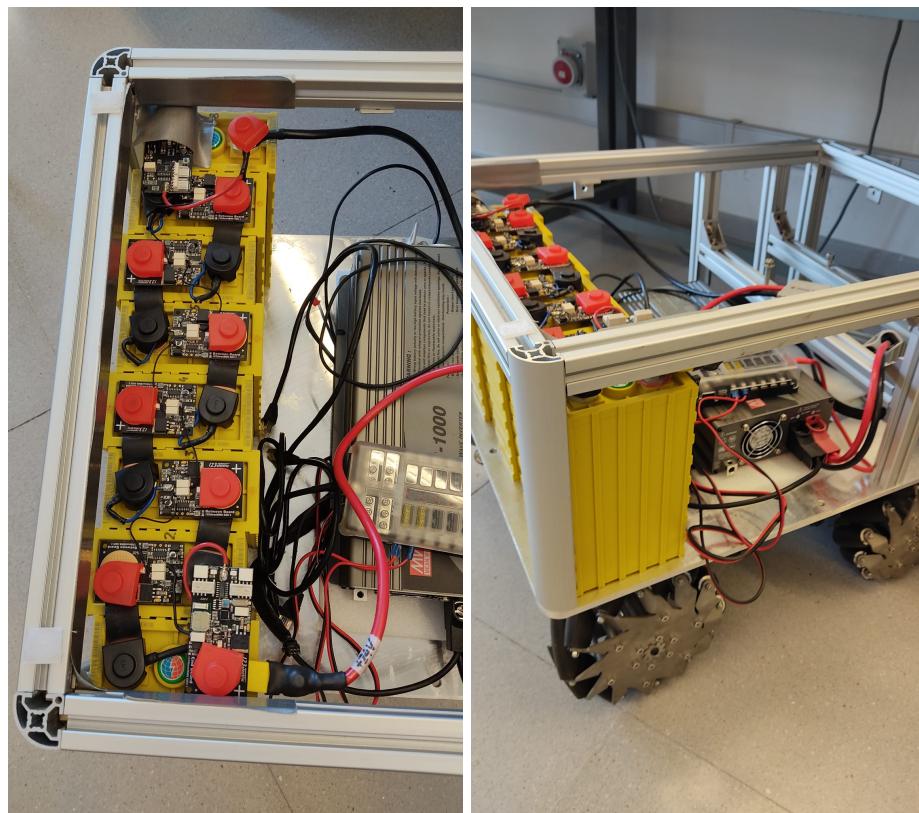
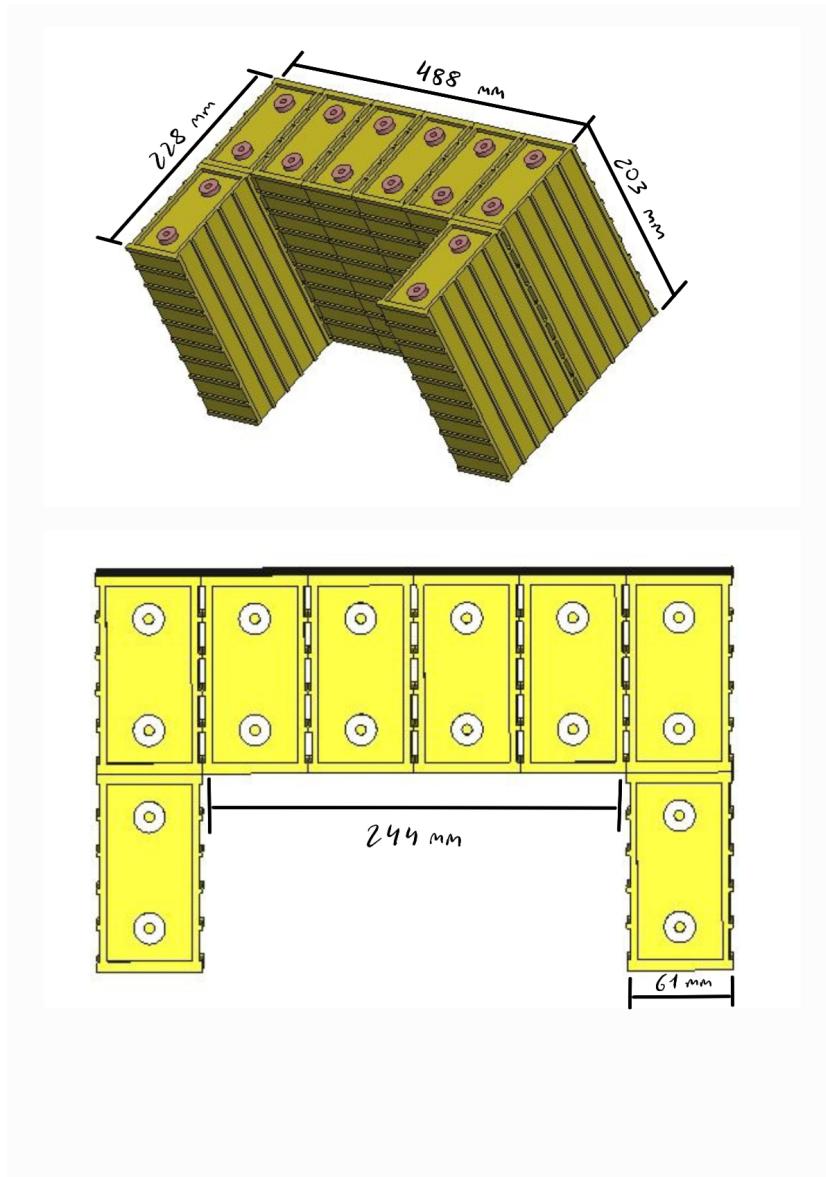


Figura 3.2: Disposición inicial de las baterías (488 x 114 x 203 mm)

La nueva disposición de las baterías debe cumplir con los requisitos de seguridad establecidos y optimizar el espacio disponible en la plataforma. El objetivo trata de buscar una configuración que permita una instalación más eficiente, minimizando la interferencia con otros componentes y permitiendo la colocación del relé de alta potencia requerido para el control de las cargas conectadas a las baterías.



**Figura 3.3: Reubicación de las baterías**

### 3.1.2. Chasis

La inclusión de paneles tiene como objetivo principal proteger los componentes internos del robot de posibles impactos, vibraciones y otros factores adversos. Dichos paneles permiten proporcionar una barrera física que protege los circuitos, las baterías y otros elementos críticos de daños externos. Además, ayudan a minimizar la interferencia electromagnética y reducen los riesgos de cortocircuitos o fugas eléctricas.

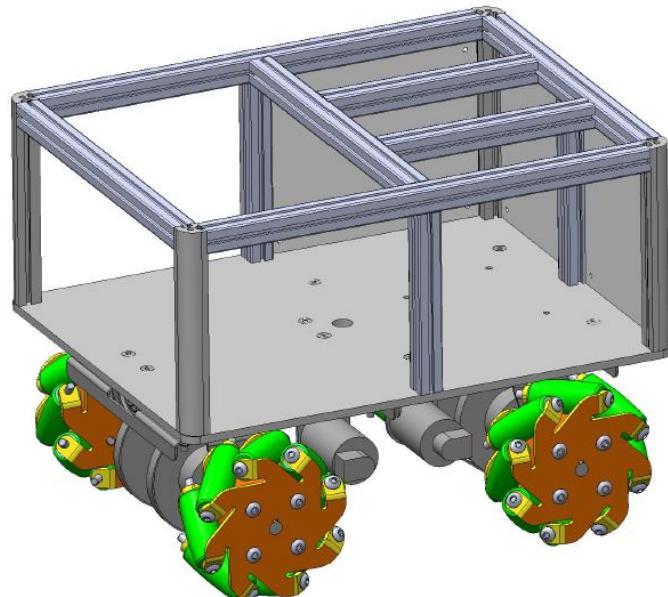


Figura 3.4: Estructura de la plataforma

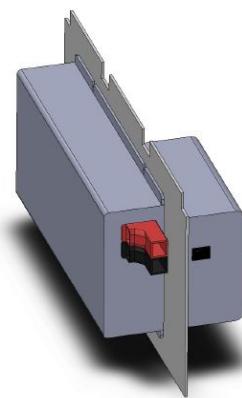
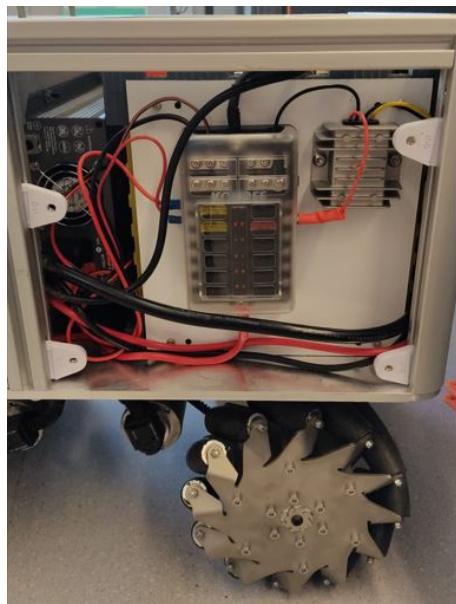


Figura 3.5: Conjunto mediana con cargador e inversor



**Figura 3.6: Panel interno**

El panel frontal está diseñado para contener el pulsador de emergencia y el dispositivo de visualización de datos, la interfaz local con el usuario. Además de estos paneles, se utilizará una tapa transparente en la parte superior que permita comprobar el correcto funcionamiento de las baterías y el BMS.

Respecto al panel interno unido a las carcasa 3D de las baterías, tiene la función de acoplar el distribuidor de 24 V y el regulador de 24 - 5 V, con el objetivo de que no queden sueltos y sin sujetar dentro de la plataforma. Éste panel tiene unas dimensiones de 24.7 cm de base y 22.1 cm de altura.

### 3.1.3. Partes funcionales

Las carcasa fabricadas mediante impresión 3D permiten a las baterías y otros componentes del sistema optimizar el espacio disponible y aislarlos eléctricamente.

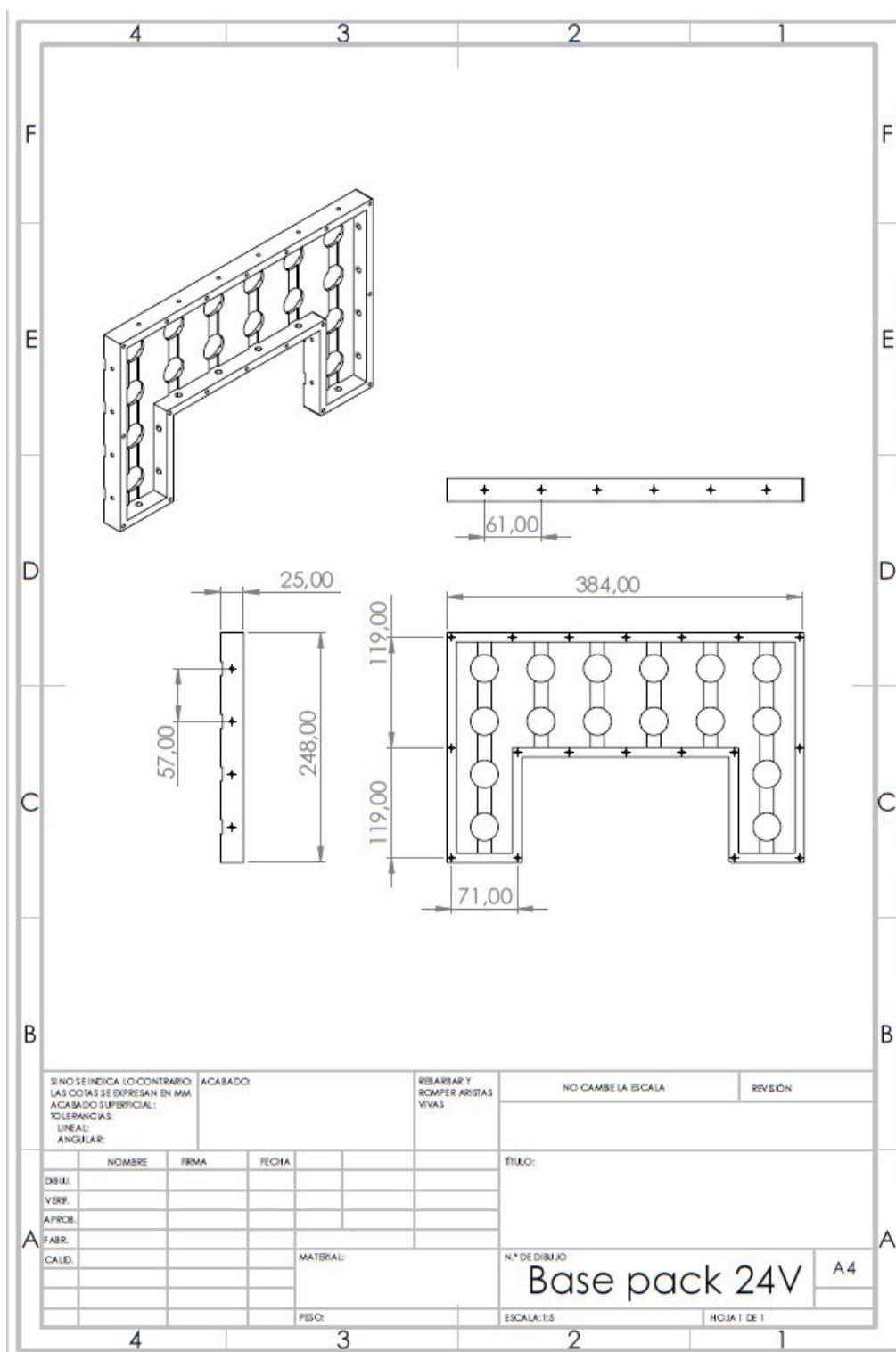
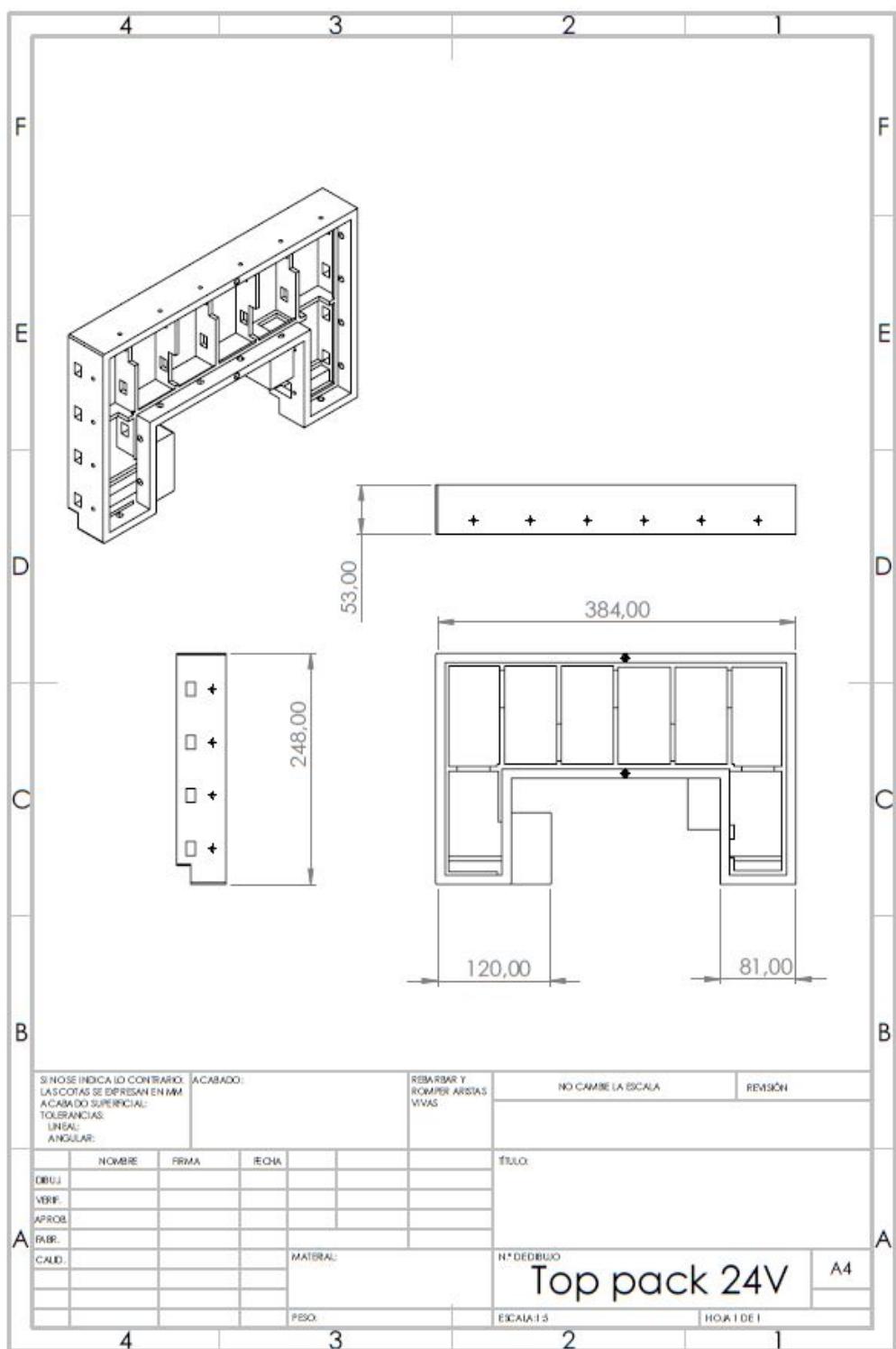
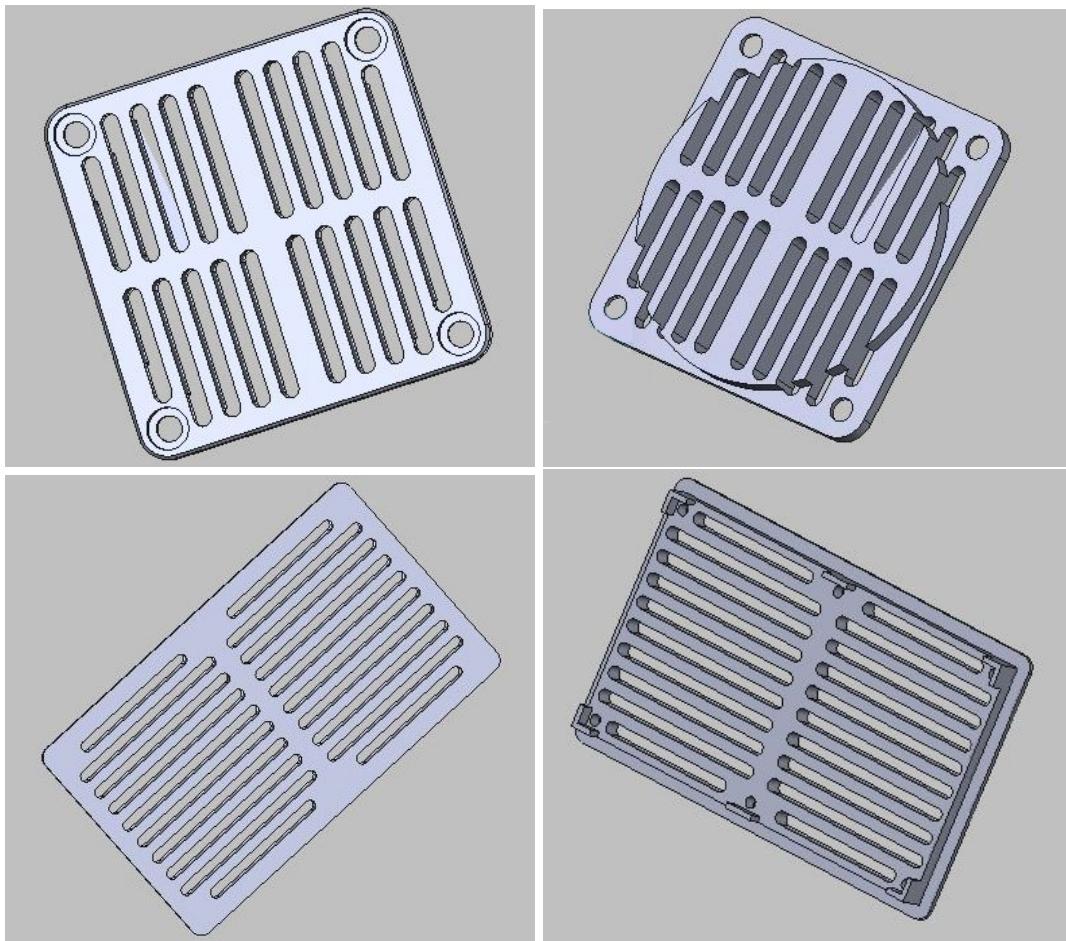


Figura 3.7: Carcasa de la base, impresión 3D



**Figura 3.8: Carcasa de la parte superior, impresión 3D**

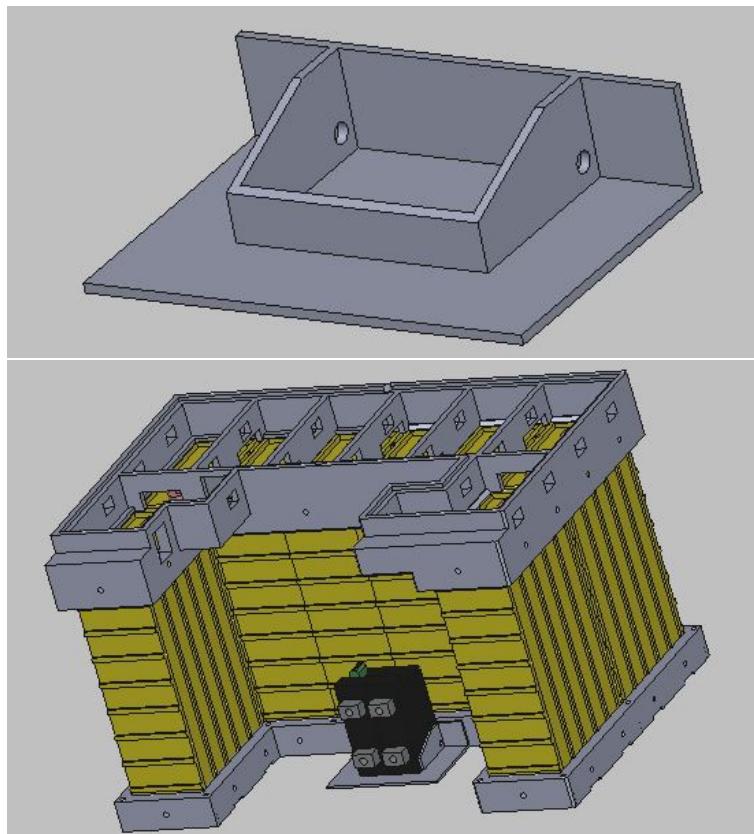
Las rejillas de ventilación son elementos clave para garantizar una adecuada disipación del calor generado por los componentes electrónicos y las baterías. Estas rejillas permiten y facilitan una circulación de aire óptima, evitando el sobrecalentamiento y asegurando un funcionamiento seguro y eficiente del sistema. Además, contribuyen a mantener una temperatura estable y prolongar la vida útil de los componentes.



**Figura 3.9: Rejillas de ventilación, impresión 3D**

En el diseño mecánico, se ha considerado cuidadosamente la ubicación de componentes clave como el relé de alta potencia, el dispositivo de visualización de datos y el botón de emergencia.

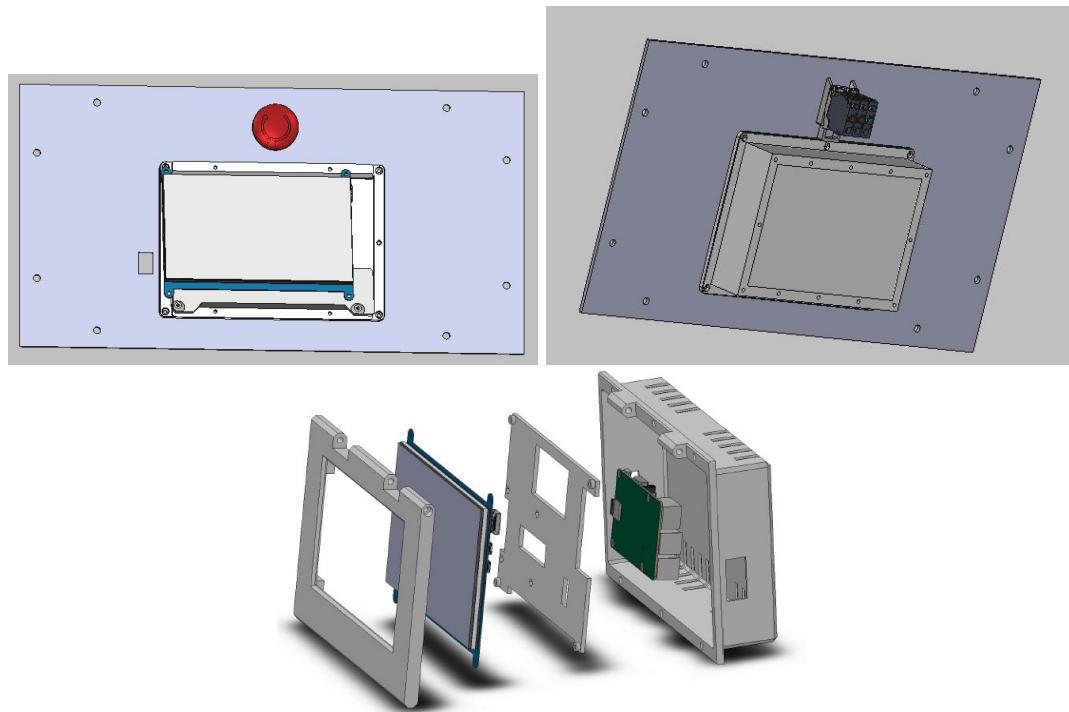
El relé se ha colocado de forma estratégica en el espacio generado entre las baterías, lo que permite un aprovechamiento eficiente del espacio disponible, gracias a la reubicación de las baterías. Para asegurar su sujeción de manera segura y estable, se ha utilizado un soporte específico diseñado para éste propósito.



**Figura 3.10: Soporte para el relé y su localización en el sistema**

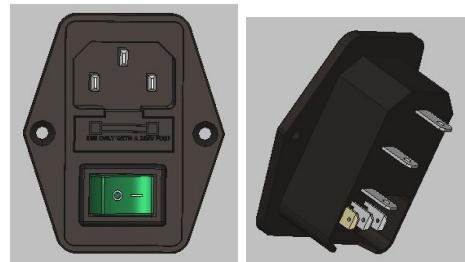
El pulsador de emergencia se ubica en el panel frontal, proporcionando una forma fácil y rápida de detener el sistema en caso de emergencia, asegurando la seguridad y cumpliendo con las regulaciones aplicables.

Por otro lado, el dispositivo de visualización de datos se ha colocado también en el panel frontal. Permite un acceso conveniente para lograr una visualización clara de los datos relevantes del sistema.



**Figura 3.11: Pulsador de emergencia y soporte para el dispositivo de visualización**

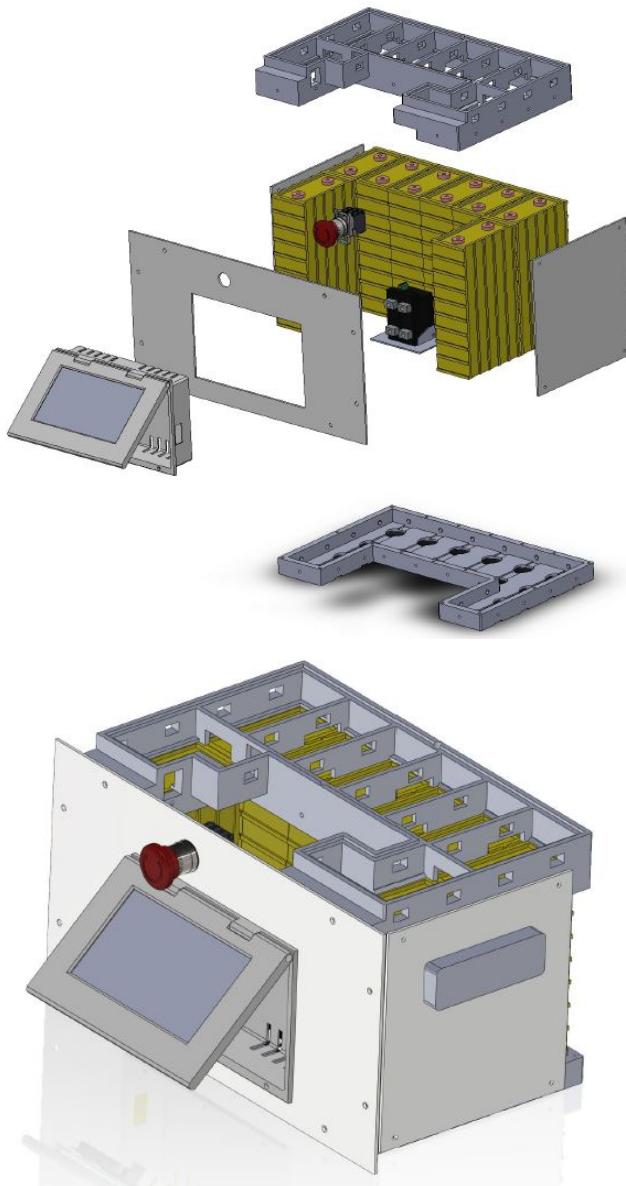
Además, el conector del cargador de la batería se ha insertado en el panel lateral de la plataforma, lo que facilita la conexión y desconexión del mismo, de manera rápida y sencilla. Al estar integrado, el cargador no ocupa espacio adicional y no está expuesto a posibles golpes o daños externos. Así, conseguimos prolongar la vida útil del cargador y mantener la apariencia ordenada y compacta de la plataforma.



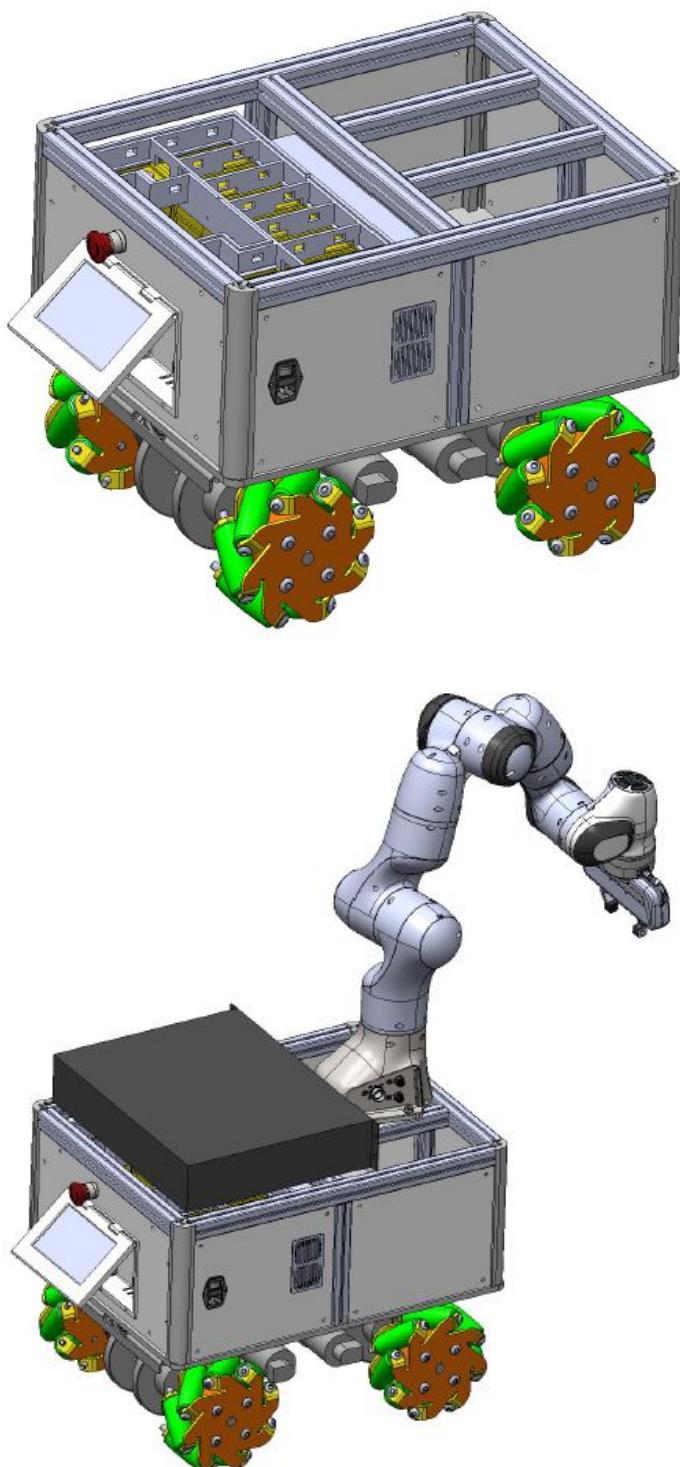
**Figura 3.12: Conector para el cargador de la batería**

### 3.1.4. Implementación final

Para finalizar éste capítulo, se muestra el diseño del proyecto, el cual ha sido cuidadosamente planificado y desarrollado para asegurar la óptima funcionalidad y seguridad del proyecto. La imagen ofrece una vista detallada de la estructura y disposición de los componentes principales de la parte mecánica.



**Figura 3.13: Diseño del sistema de alimentación completo**



**Figura 3.14: Diseño del sistema completo implementado**

## 3.2. Desarrollo Eléctrico

Ésta sección aborda el desarrollo eléctrico completo de nuestro proyecto, que engloba desde la selección de los componentes electrónicos más adecuados hasta su implementación y conexión con el sistema. El objetivo principal es garantizar un diseño eléctrico eficiente y seguro para lograr el funcionamiento óptimo del sistema.

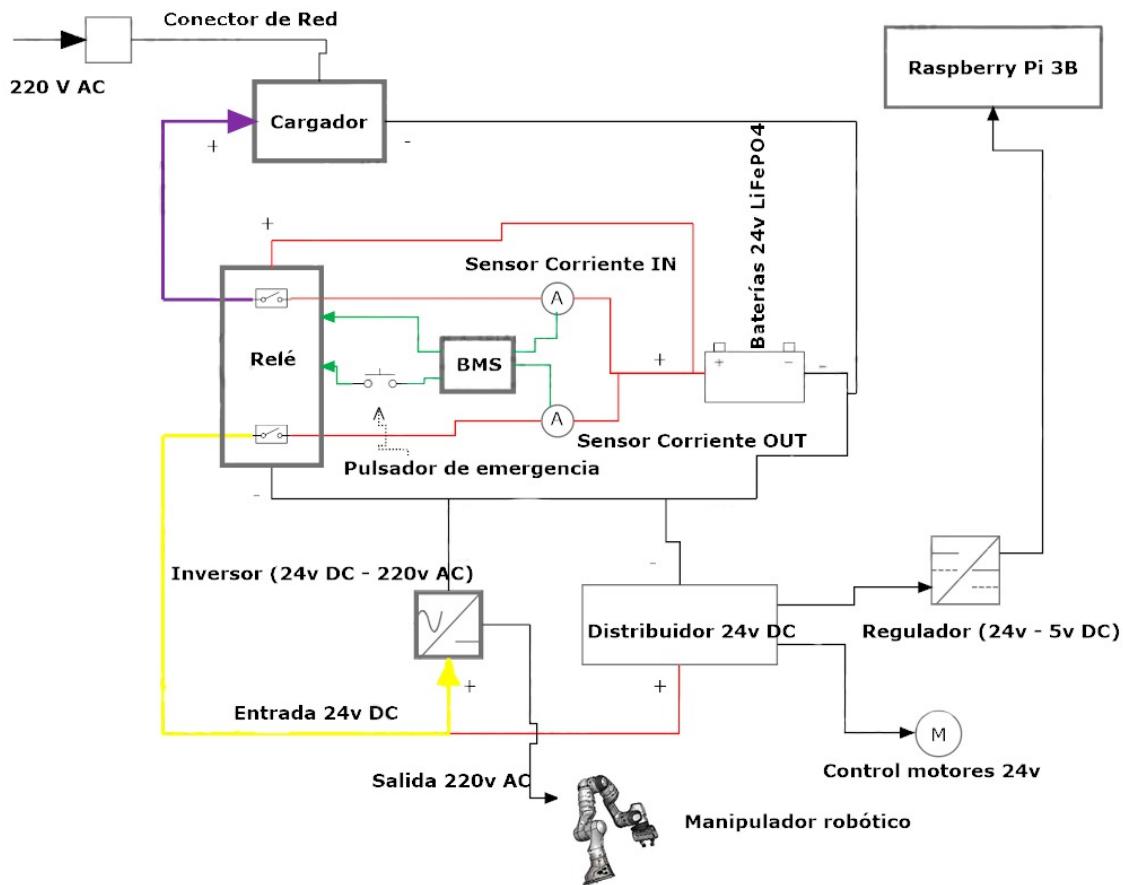


Figura 3.15: Esquema eléctrico del sistema

Diseño del circuito eléctrico completo, creado con SmartDraw.

Para comenzar, la elección de los componentes debe cumplir rigurosamente los requisitos eléctricos impuestos. Esto implica evaluar las características técnicas de cada componente y su compatibilidad con los demás elementos del sistema.

En segundo lugar, realizaremos el conexionado de cada componente, siguiendo las especificaciones y diagramas indicados por los fabricantes. Es de suma importancia seguir las pautas de diseño y las recomendaciones de seguridad para evitar posibles cortocircuitos, interferencias electromagnéticas o daños en los componentes.

Durante la implementación, debemos considerar la ubicación física de los componentes dentro del sistema, siguiendo las pautas establecidas en el desarrollo mecánico.

### 3.2.1. Alimentación del sistema

El sistema de alimentación del proyecto consiste en ocho celdas de baterías LiFePO<sub>4</sub>, modelo Winston LFP060AHA de la marca GWL.

Teniendo en cuenta que la conexión de las ocho celdas se hace en serie, el sistema de alimentación completo aporta 26.4 V de tensión nominal y tiene una capacidad total de 480 Ah.

Model name	LFP060AHA	Alternative product marking TS-LFP60AHA, WB-LYP60AHA
Nominal voltage	3.3 V	Operating voltage under load is 3.0 V
Capacity	60 AH	+/- 5%
Operating voltage	max 4.0V - min 2.8V	At 80% DOD
Deep discharge voltage	2.5 V	The cells is damaged if voltage drops below this level
Maximal charge voltage	4 V	The cells is damaged if voltage exceeds this level
Optimal discharge current	< 30 A	0.5 C
Maximal discharge current	< 180 A	3 C, continuous for max 15 minutes from full charge
Max peak discharge current	< 600 A	10 C, maximal 5 seconds in 1 minute
Optimal charge current	< 30 A	0.5 C
Maximal charge current	< 180 A	< 3 C with battery temperature monitoring
Maximal continuous operating temperature	80 °C	The battery temperature should not increase this level during charge and discharge
Dimensions	114 x 203 x 61	Millimeters (tolerance +/- 2 mm)
Weight	2.3 kg	Kilograms (tolerance +/- 150g)
Optimal torque force	12 - 15 Nm	

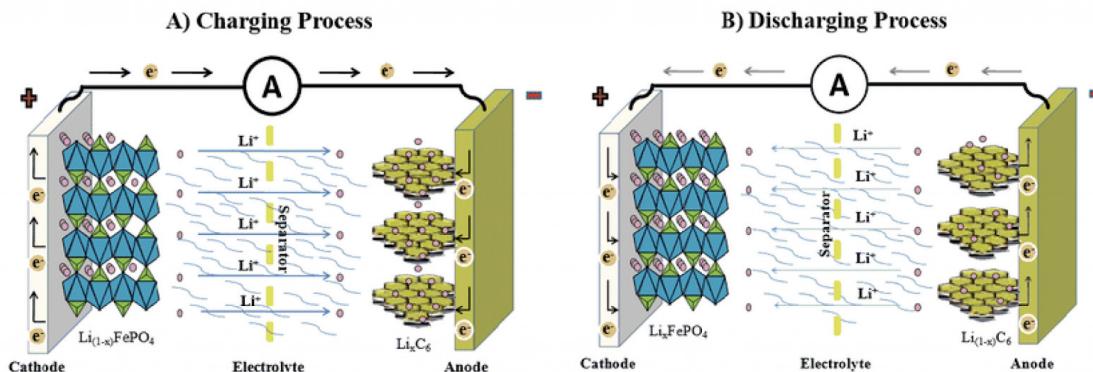
**Figura 3.16: Especificaciones técnicas de la batería LiFePO<sub>4</sub>. Fuente:[9]**

Caben destacar las siguientes características de las baterías:

- La carga inicial y posterior recomendada es de 3,65 V.**
- El voltaje mínimo es de 2,8 V.**
- El voltaje máximo es de 3,8 V.**
- La corriente de descarga máxima es de 3C continuamente.**
- Temperatura de funcionamiento desde -45 °C hasta 85 °C**
- La densidad energética es de 83,47 Wh/kg**
- No hay efecto de autodescarga.**
- No tiene efecto memoria.**
- No tiene combustión espontánea.**
- No reacciona con la humedad ni con el oxígeno.**

Las baterías LiFePO<sub>4</sub> son un tipo de batería de iones de litio, compuestas por litio, hierro y fosfato como material catódico, junto con un electrodo de carbono y grafito con soporte metálico como ánodo.

El primer modelo de batería de litio ferrofosfato se fabricó tras el descubrimiento del fosfato como material catódico para su uso en baterías de iones de litio en 1996.



**Figura 3.17: Carga y descarga de la batería LiFePO<sub>4</sub>. Fuente:[21]**

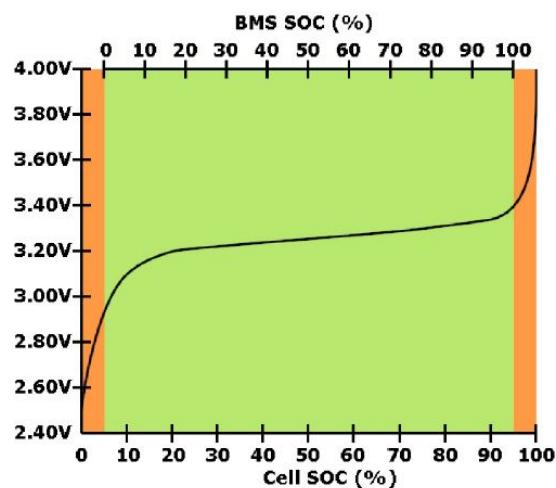


Figura 3.18: Curva de voltaje vs SOC. Fuente:[9]

El voltaje depende del porcentaje de energía restante en la celda.

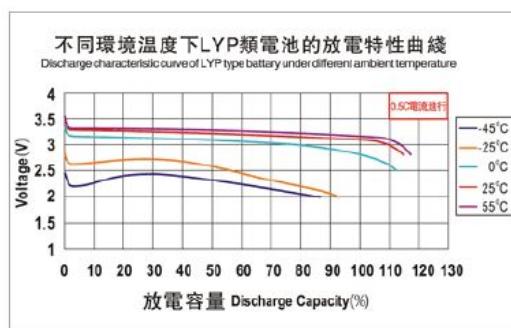


Figura 3.19: Curva de descarga en distintas temperaturas ambientales. Fuente:[9]

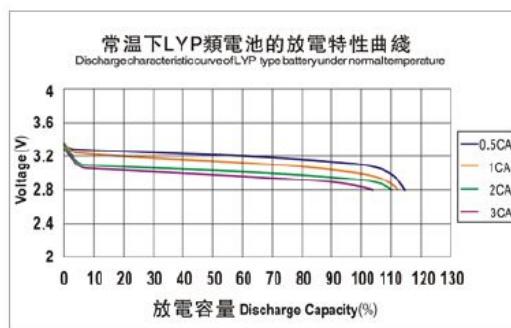
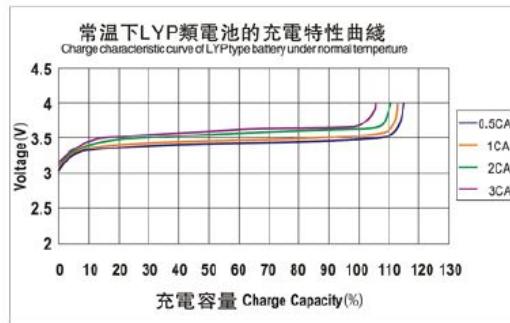
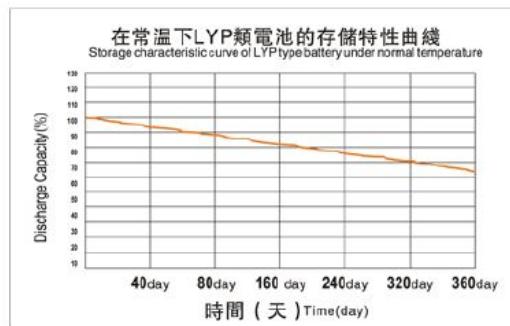


Figura 3.20: Curva de descarga en temperaturas ambientales normales. Fuente:[9]



**Figura 3.21: Curva de carga en temperaturas normales. Fuente:[9]**



**Figura 3.22: Capacidad de descarga en temperaturas normales. Fuente:[9]**

### 3.2.2. Cargador

El cargador disponible para del sistema de alimentación es el modelo POW24V20A-D1 de la marca GWL.

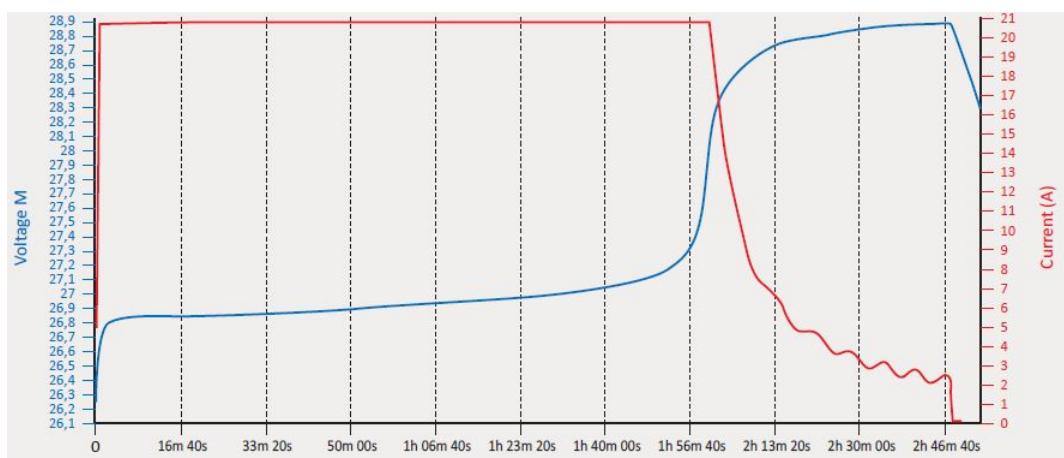


**Figura 3.23: Cargador POW24V20A-D1. Fuente:[8]**

No.	Item	Technical specification	Unit	Remark
1-1	Rated input voltage	230V	Vac	
1-2	Input voltage range	180V – 264V	Vac	
1-3	AC input voltage frequency	47 - 63	Hz	
1-4	Inrush current	< 50 A	A	@ 264Vac start-up in cold condition
1-5	Max input current	6 A	A	

**Figura 3.24: Características input del cargador. Fuente:[8]**

No.	Item	Technical specification	Unit	Remark
2-1	Nominal charge voltage	24V	Vdc	8 LiFePO4 cells @ 3.00V 2 LiFePO4 batteries @ 12.00V 10 LTO cells @ 2.40V
2-2	Fast charge voltage (V-max) LiFePO4 mode	29.2V	Vdc	Tolerance to stop charging according to the battery capacity $\pm 0.5$ V, see table 2. for how to set this mode.
2-3	Fast charge voltage (V-max) LTO mode	26.5V	Vdc	Tolerance to stop charging according to the battery capacity $\pm 0.5$ V, see table 2. for how to set this mode.
2-4	Fast charge voltage (V-max) Optional mode	30.4V	Vdc	Tolerance to stop charging according to the battery capacity $\pm 0.5$ V, see table 2. for how to set this mode. Caution – this mode will result in overcharging LiFePO4 or LTO cells!
2-5	Constant current (I-CC)	20A	A	Maximal current during full charge
2-6	Deep voltage level (V-deep)	21V ( $\pm 1$ V)	Vac	Deep discharge voltage level, bellow this voltage, the current is limited to I-min
2-6	Deep discharge current (I-min)	2A	A	The limited current bellow V-Deep ( $\pm 0.5$ A)
2-7 opt	BMS limit current (I-BMS)	approx 2A ( $\pm 0.5$ A)	A	The limited current for cell balancing (controlled by BMS)
2-8	Power efficiency	>80%		@ 230Vac

**Figura 3.25: Características output del cargador. Fuente:[8]****Figura 3.26: Curva de carga del POW24V20A-D1. Fuente:[8]**

### 3.2.3. BMS y sensores de corriente

#### Instalación

El BMS utilizado en nuestro proyecto es el 123/SmartBMS de 3<sup>a</sup> generación, de la marca GWL.

Debemos considerar la alta prioridad de uso del BMS por aspectos como el estado de carga y la seguridad de las baterías.

Al monitorizar parámetros clave como la temperatura, tensión y corriente, ya que evita situaciones peligrosas como sobrecargas, sobredescargas, cortocircuitos y condiciones fuera de rango que podrían provocar daños en las baterías o incluso riesgos de incendio.

El módulo BMS de nuestro proyecto se caracteriza por tener una alta precisión de medida del SoC.

Utiliza sensores de corriente de doble rango para medir una amplia gama de corrientes.

Tiene un registro de errores para ver cuándo, qué celda y qué error ocurrió.

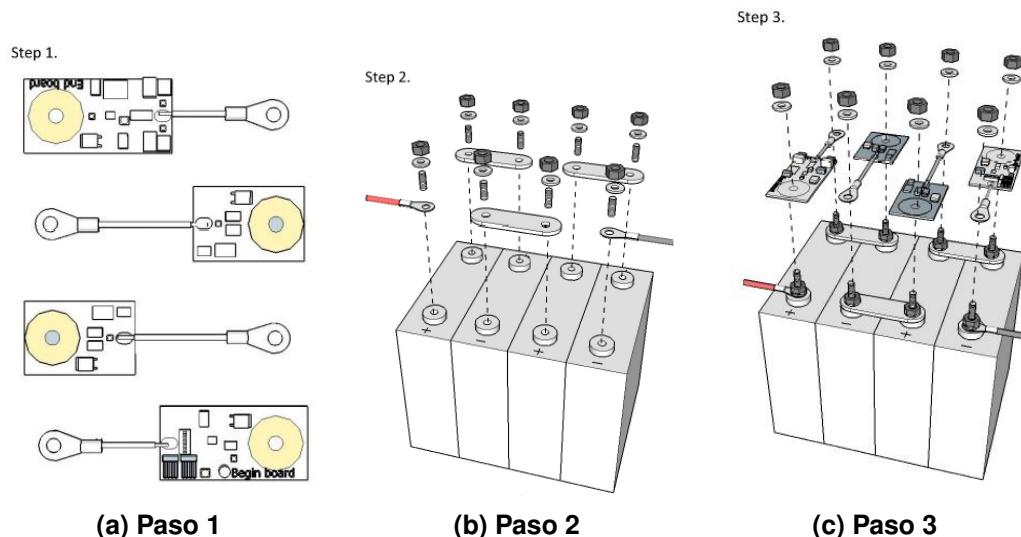


Figura 3.27: Pasos para la instalación del BMS. Pasos 1-3. Fuente:[9]

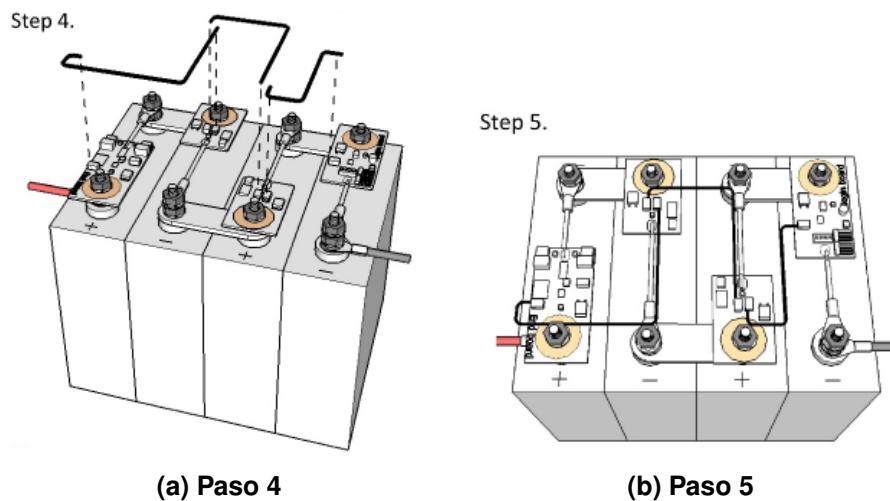


Figura 3.28: Pasos para la instalación del BMS. Pasos 4-5. Fuente:[9]

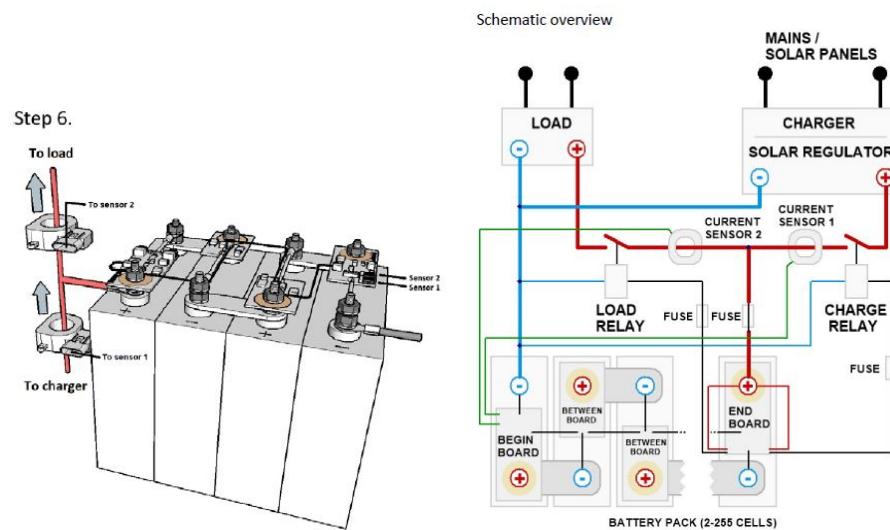
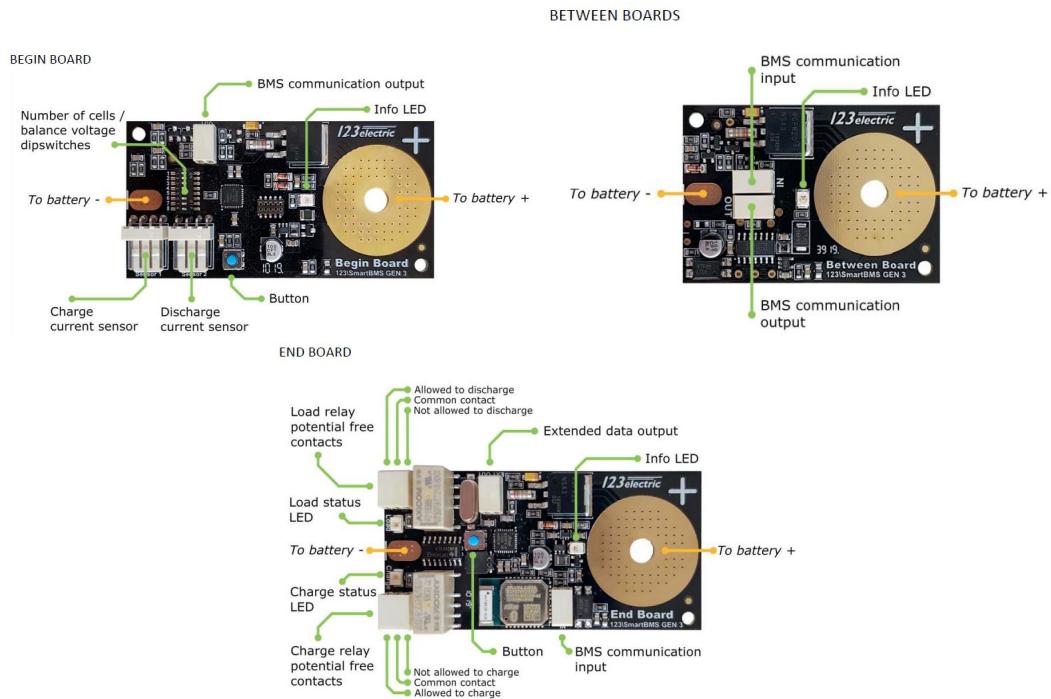


Figura 3.29: Configuración de los sensores de corriente. Fuente:[9]



**Figura 3.30: Placas del BMS. Fuente:[9]**

### Estimación del SOC

El BMS determina el estado de carga mediante el conteo de Coulomb<sup>1</sup>. El SOC se recalibra cada vez que el paquete está lleno y vacío, con la finalidad de reducir los errores en éste cálculo.

En el modo normal seleccionado para el proyecto, el SOC se establece en 100 % cuando todas las tensiones de las celdas son mayores o iguales que V-balance.

Cuando una de las tensiones de las celdas está por debajo de V-min durante un período prolongado sin permitir la descarga, el SOC se establece en 0 %.

<sup>1</sup>El conteo de Coulomb se basa en la medición constante de las corrientes entrantes y salientes y la integración de éstas corrientes.

Charge		Discharge/load	
Enable	Disable	Enable	Disable
All cell voltages < V-balance	Cell voltage $\geq$ V-max	All cell voltages $>$ V-min	Cell voltage $\leq$ V-min
AND SOC < chargerestart	All cell voltages $\geq$ V-balance	AND SOC $\geq$ discharge restart	Cell voltage $\leq$ V-min
AND cell temperature $>$ T-min	Cell temperature $<$ T-min	AND cell temperature $>$ T-min	Cell temperature $<$ T-min
AND cell temperature $<$ T-max	Cell temperature $>$ T-max	AND cell temperature $<$ T-max	Cell temperature $>$ T-max
AND cell communication	No cell communication	AND cell communication	No cell communication

Figura 3.31: Operaciones del BMS sobre los relés de carga/descarga

### 3.2.4. Relé

El elemento que permite controlar la carga o descarga de las baterías es el relé que utilizaremos. Se trata de un relé inteligente, modelo 123 smartRelay, de la marca 123 electric.

Este componente es un módulo de relé dual con un consumo extremadamente bajo de energía propia. Se puede utilizar para conmutar corrientes altas e incluso señales pequeñas. El diseño garantiza un funcionamiento prolongado, incluso con baterías muy pequeñas. Aporta una colaboración perfecta con el BMS que utilizaremos, ya que nos permite conmutar inversores y/o cargadores.

Un microcontrolador integrado asegura un funcionamiento confiable en entornos difíciles, combinado con un consumo de energía muy bajo. Dos LED muestran el estado actual de cada relé. Un destello significa que el relé está apagado, dos destellos indican que el relé está encendido.

### Electrical specifications

Description	Value / range	
Operating supply voltage range	9.0V to 70.0V	
Operating temperature range	-40 to 70°C	
Maximum current consumption average	@12V	<0.3mA
	@24V	<0.4mA
Minimum signal input voltage to turn on	5V	
Maximum switching current	@12VDC	120A
	@24VDC	100A
	@230VAC	120A
Maximum switching time	0.3s	

### Mechanical dimensions

Description	Value
Total dimensions LxWxH (mm)	62x63x80
Relay terminal	M6
Mounting bolt top	M5
Mounting bolt bottom	M4
Power supply and inputs max cable (mm <sup>2</sup> )	0.75mm <sup>2</sup>

Figura 3.32: Especificaciones del relé.[2]

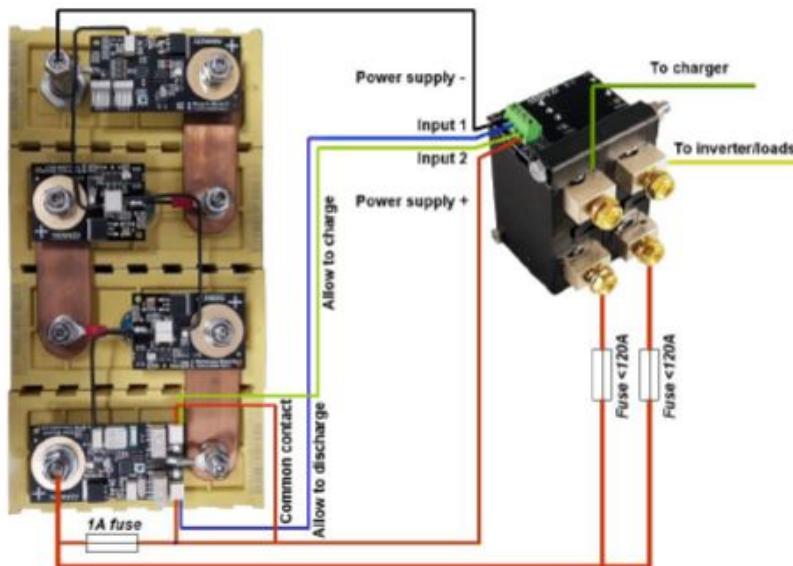
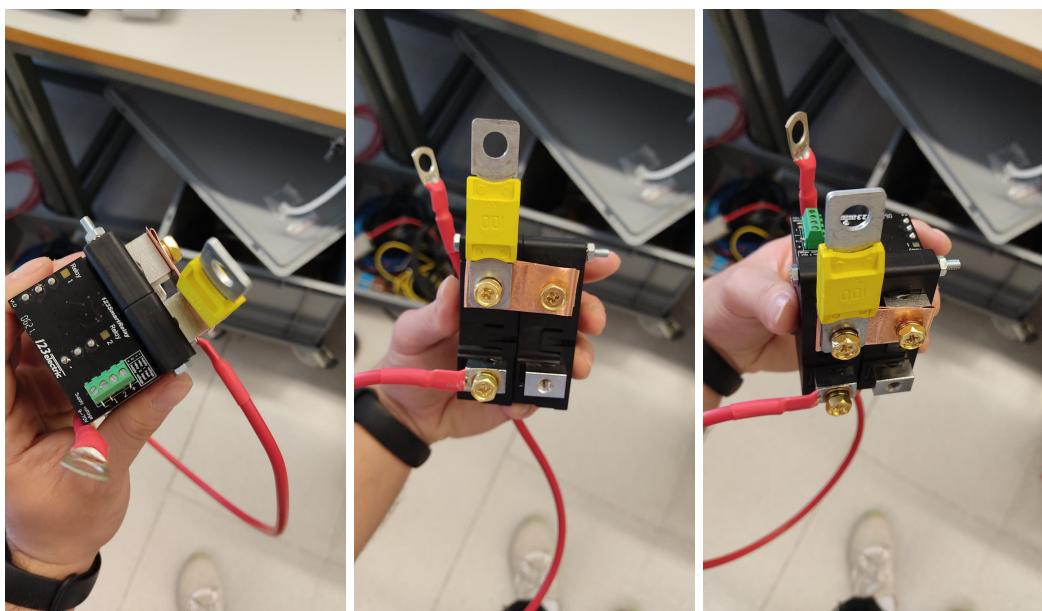


Figura 3.33: Conexionado del relé.[2]

El relé está alimentado por la batería. Recibe dos señales de control, provenientes del BMS.

La primera de ellas, Input 1, controla la descarga. Es decir, controla la salida que alimentará al inversor.

La segunda, Input 2, controla la carga. Es decir, permite la conexión con el cargador de las baterías.



**Figura 3.34: Conexión del relé en el laboratorio**

Utilizamos fusibles de 100 A para interrumpir la corriente eléctrica en caso de que se produzca una sobrecarga.

Cuando la corriente que circula por el circuito excede el valor nominal del fusible, el filamento se calienta y se funde, interrumpiendo así el flujo de corriente eléctrica y evitando posibles daños o incluso riesgos de incendio.

### Pulsador de emergencia

La modificación realizada para garantizar la seguridad del sistema consiste en conectar el pulsador de emergencia a la señal del BMS que permite la descarga hacia el inversor, y por lo tanto, al sistema alimentado completo.

Se caracteriza por tener una conexión normalmente cerrada, de manera que cuando lo activamos (pulsamos), abrimos el circuito y se detiene la alimentación del inversor.

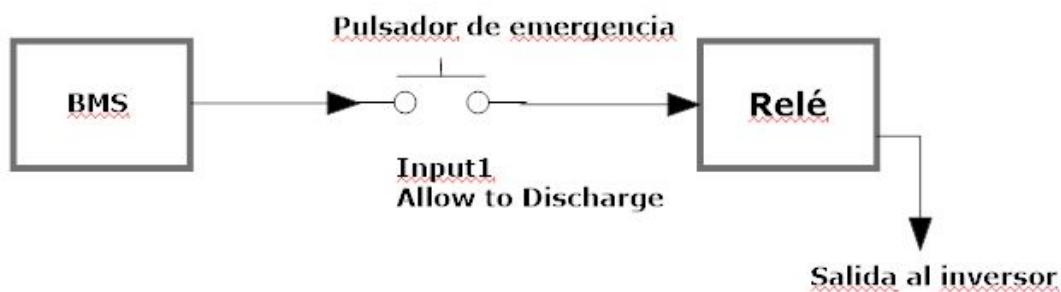


Figura 3.35: Conexión del pulsador de emergencia

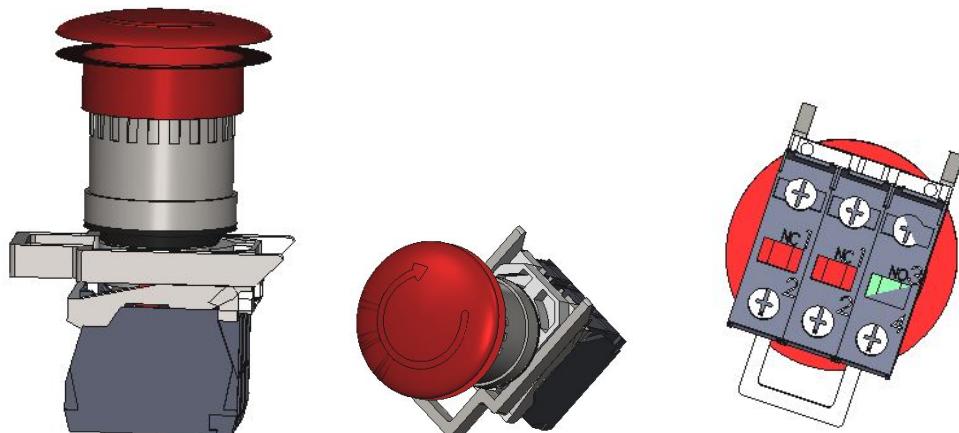


Figura 3.36: Diseño del pulsador en SolidWorks.

### 3.2.5. Inversor

Un inversor es un dispositivo que permite transformar la corriente continua en alterna

El inversor digital simple consta de un oscilador que controla un transistor, el cual se utiliza para interrumpir la corriente entrante y generar una onda rectangular. Esta onda rectangular se envía a través de un transformador que suaviza su forma, aproximándola a una onda senoidal y generando la tensión de salida requerida. Idealmente, la forma de onda de salida de un inversor debería ser sinusoidal. Una técnica efectiva para lograr esto es utilizar la modulación por ancho de pulso (PWM) para asegurar que la componente principal senoidal sea mucho mayor que las armónicas superiores.

Los inversores más modernos han adoptado dispositivos más avanzados, como tiristores, triacs, IGBT y MOSFETs.

Actualmente, los inversores más eficientes emplean múltiples filtros electrónicos para lograr una forma de onda que se asemeje a una onda senoidal en la entrada del transformador, en lugar de depender exclusivamente del transformador para suavizar la onda.

Partiendo del requisito de transformar los 24 V DC suministrados por las baterías, en 220 V AC, el inversor que mejor se adapta a nuestras necesidades es el modelo TS-1000 de la marca Mean Well.

El TS-1000 es un inversor de onda sinusoidal pura, totalmente controlado de forma digital por una avanzada CPU. Puede proporcionar 1000W de forma continua y 1150W durante 3 minutos, con una potencia de sobrecarga de 2000W.

Salida de onda sinusoidal verdadera ( $\text{THD}^2 < 3.0\%$ ).

Potencia nominal de 1000W.

Alta eficiencia, de hasta el 92 %.

Indicación LED completa para el estado de funcionamiento.

Alarma e indicador de batería baja.

Permite seleccionar la tensión de salida/frecuencia.

---

<sup>2</sup>La distorsión armónica total o THD (Total Harmonic Distortion) es el parámetro que indica el porcentaje de contenido armónico de la onda de tensión de salida del inversor.

Es adecuado para una amplia gama de aplicaciones, como PC, equipos de TI, vehículos, yates, electrodomésticos, motores, herramientas eléctricas, equipos de control industrial, sistemas de audio y vídeo, entre otros.

TS-1000

	Model	112	124	148	212	224	248
O U T P U T	Rated power	1000W max. continuously, 1150W max. for 180 seconds, 1500W max. for 10 seconds					
	2000W (30cycles)						
	AC voltage	110Vac, 60Hz (Factory setting)	230Vac, 50Hz (Factory setting)				
		100/110/115/120Vac (Selectable by setting button)	200/220/230/240Vac (Selectable by setting button)				
T	Waveform	True sine wave (THD < 3.0%) at rated input voltage					
	Protection	AC short · Overload · Over Temperature					
I N P U T	Bat. voltage range	10.5 ~ 15.0V	21.0 ~ 30.0V	42.0 ~ 60.0V	10.5 ~ 15.0V	21.0 ~ 30.0V	42.0 ~ 60.0V
	DC current	100A	50A	25A	100A	50A	25A
	Efficiency	88%	89%	90%	90 %	91%	92%
	Off mode current draw	Under 1.0mA at power switch OFF					
	Protection	Over current · battery polarity reverse by fuse · battery low shutdown · battery low alarm					

Figura 3.37: Especificaciones del inversor TS-1000. Fuente:[22]

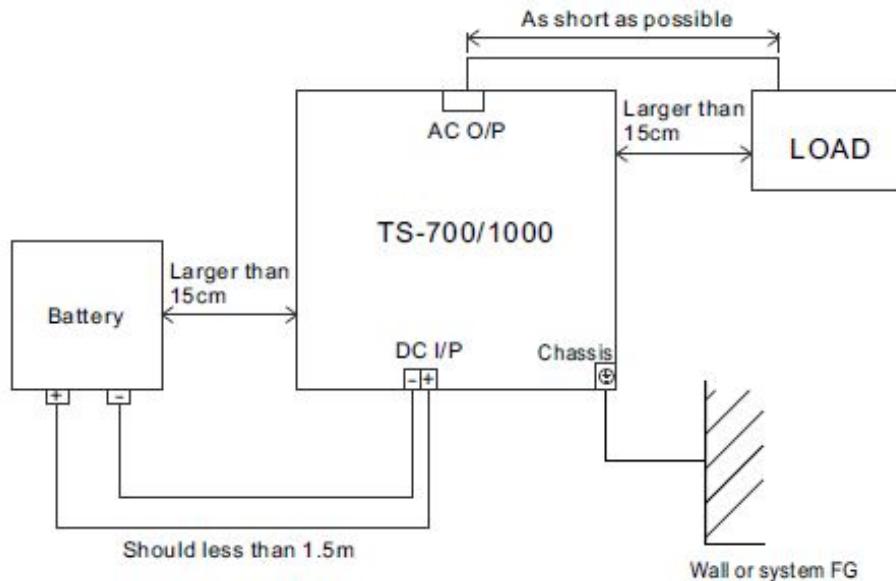


Figura 3.38: Diagrama de uso del inversor TS-1000. Fuente:[22]

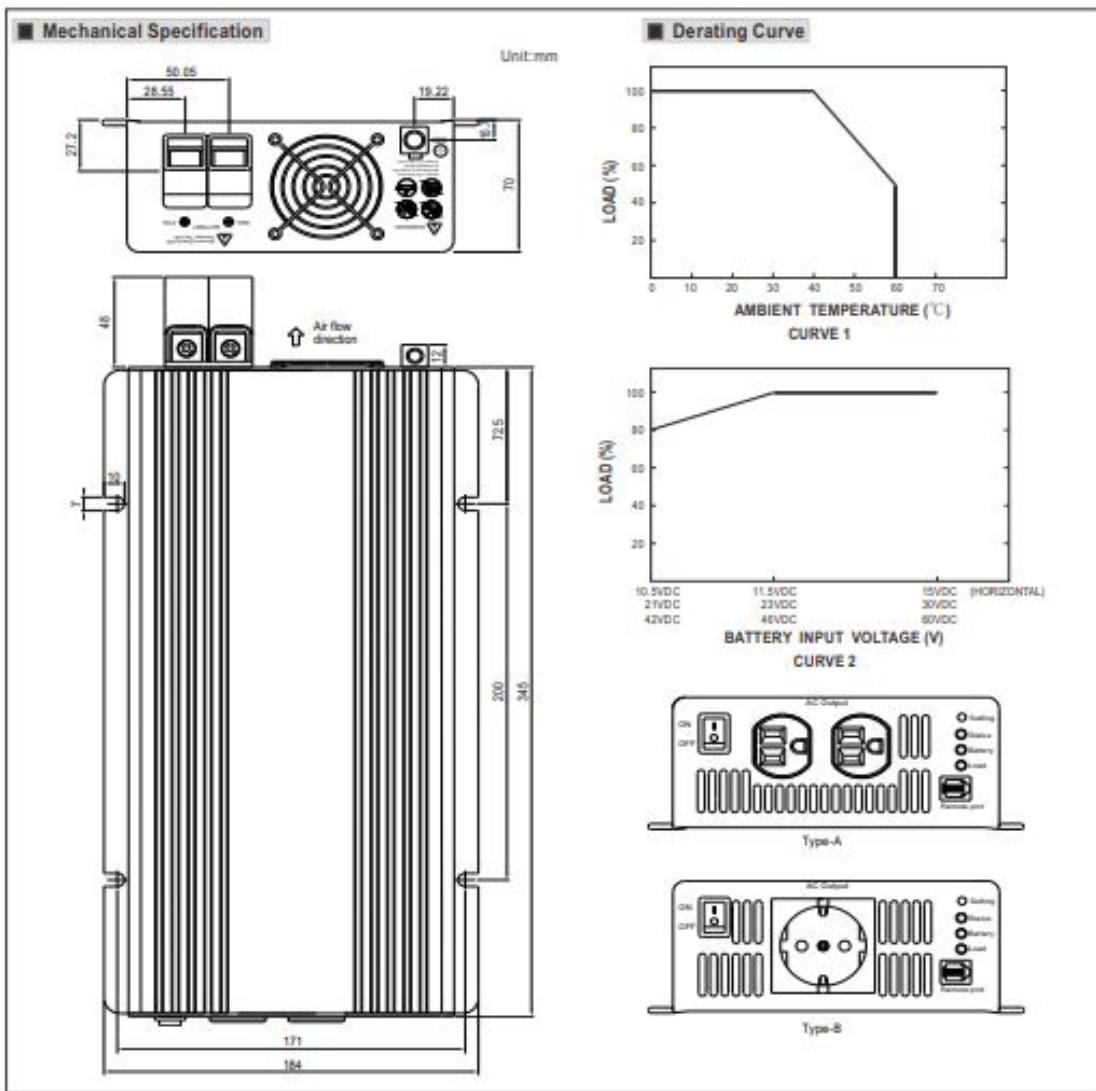


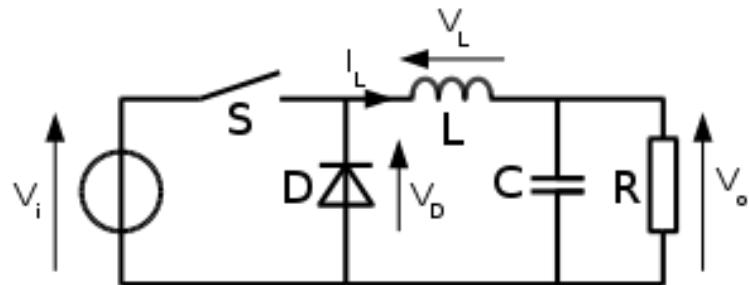
Figura 3.39: Especificaciones mecánicas y curva de reducción de carga del inversor TS-1000. Fuente:[22]

### 3.2.6. Regulador

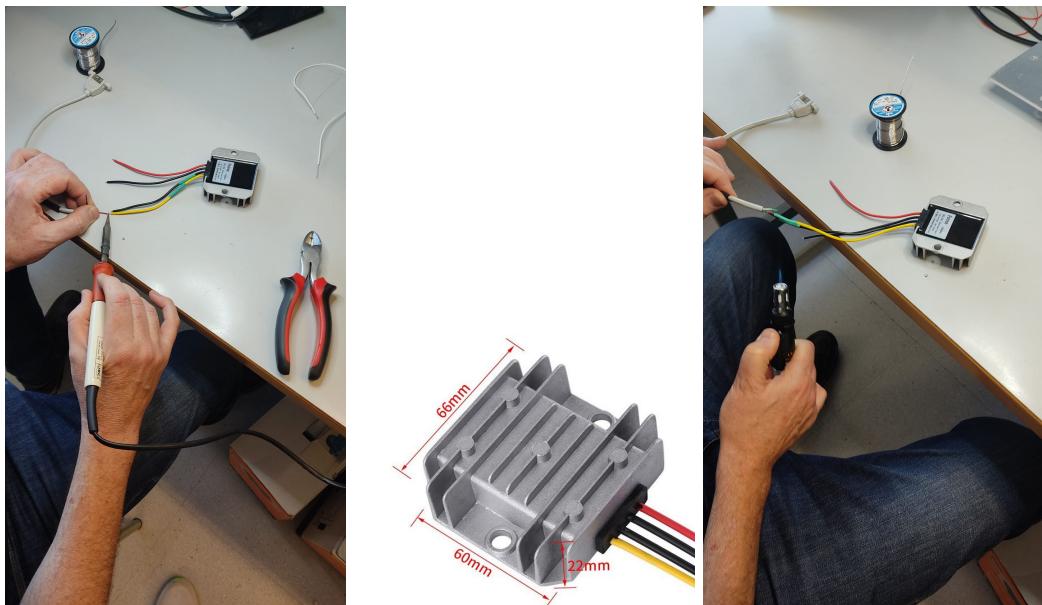
Partimos del requisito de alimentar componentes como el dispositivo de visualización, la Raspberry Pi y la controladora de los motores de la plataforma, los cuales deben recibir una tensión de entrada de 5 V.

Por ello, se requiere el uso de un regulador de tensión (convertidor Buck), alimentado por los 24 V DC de las baterías, para obtener una salida de 5 V DC.

El diseño consiste en una fuente conmutada con dos dispositivos semiconductores (transistor S y diodo D), un inductor L y opcionalmente un condensador C a la salida.



**Figura 3.40: Esquema básico de un convertidor Buck. Fuente:[13]**



**Figura 3.41: Regulador de tensión.**

Especificaciones	Valores
Voltaje de entrada	DC 12V/24V
Voltaje de salida máximo	DC 5V
Corriente de salida	5A
Potencia de salida	25W
Eficiencia	96 %

### 3.2.7. Distribuidor de 24 V

Este dispositivo lo utilizaremos para dividir o distribuir la señal de voltaje proveniente de las baterías, en múltiples salidas. Además, el modelo cuenta con fusibles como medida de protección.

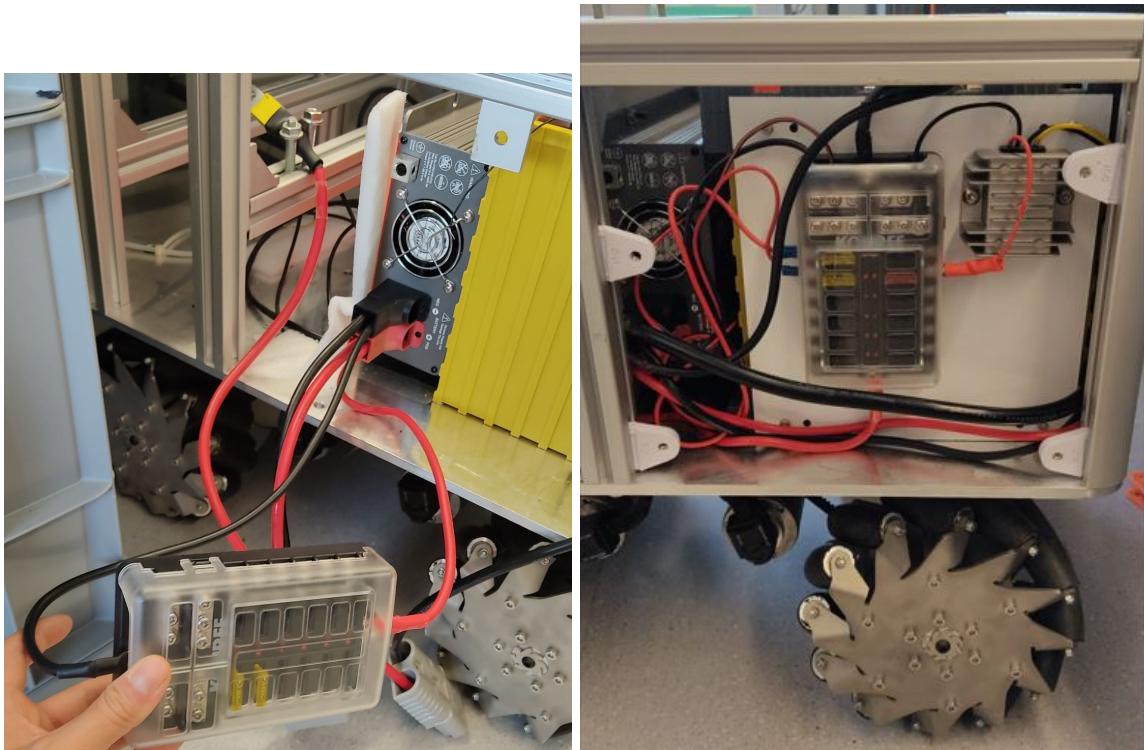


Figura 3.42: Modelo del distribuidor de 24 V utilizado.

### 3.2.8. Dispositivo de visualización

La interfaz local para poder visualizar los datos de forma gráfica se realiza mediante un display táctil HDMI de 7 pulgadas, con USB touch para Raspberry Pi, con una resolución de 1024x600 píxeles. Su alimentación es de 5 V DC.



Figura 3.43: Dispositivo de visualización utilizado.

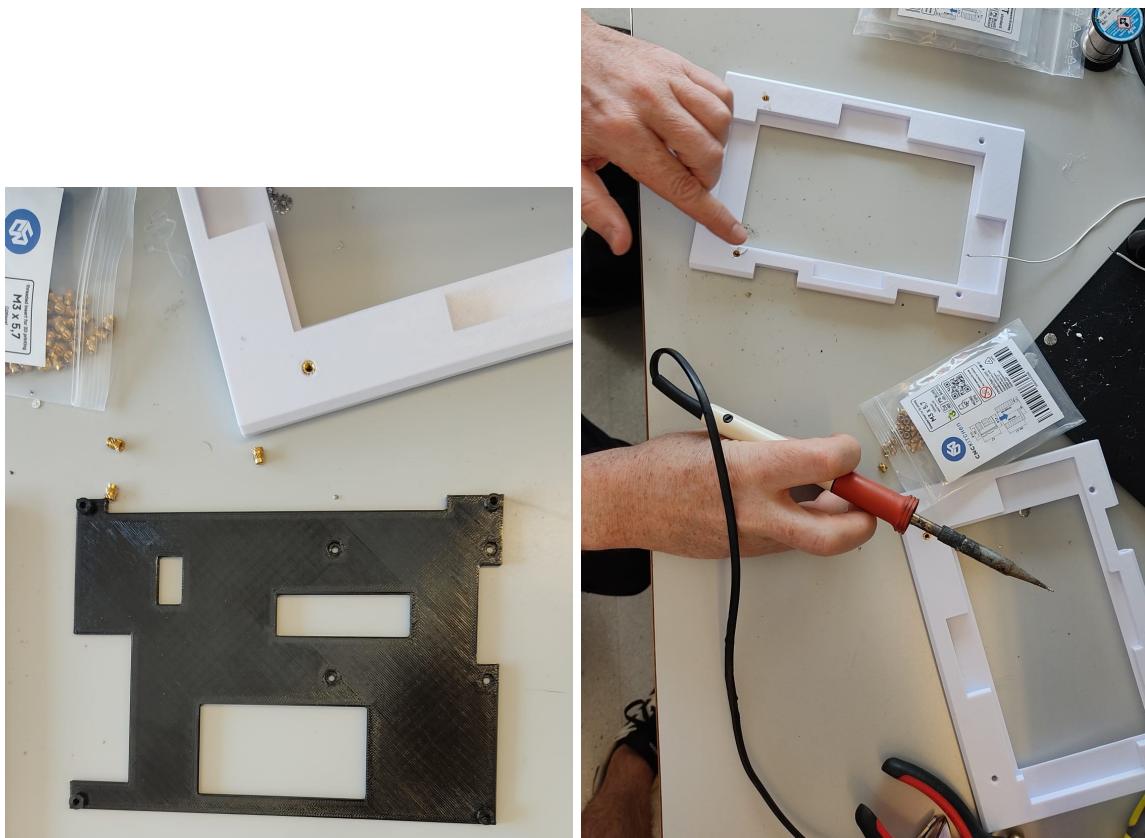


Figura 3.44: Soporte de impresión 3D para el display.

### 3.3. Desarrollo Software

#### Adquisición y procesamiento de datos con Raspberry Pi

Se trata de una placa de desarrollo de tamaño reducido que se ha convertido en un componente muy popular en el ámbito de la informática y la electrónica.

El modelo que utilizaremos es la Raspberry Pi 3B, debido a que sólo estaba disponible éste en el laboratorio.

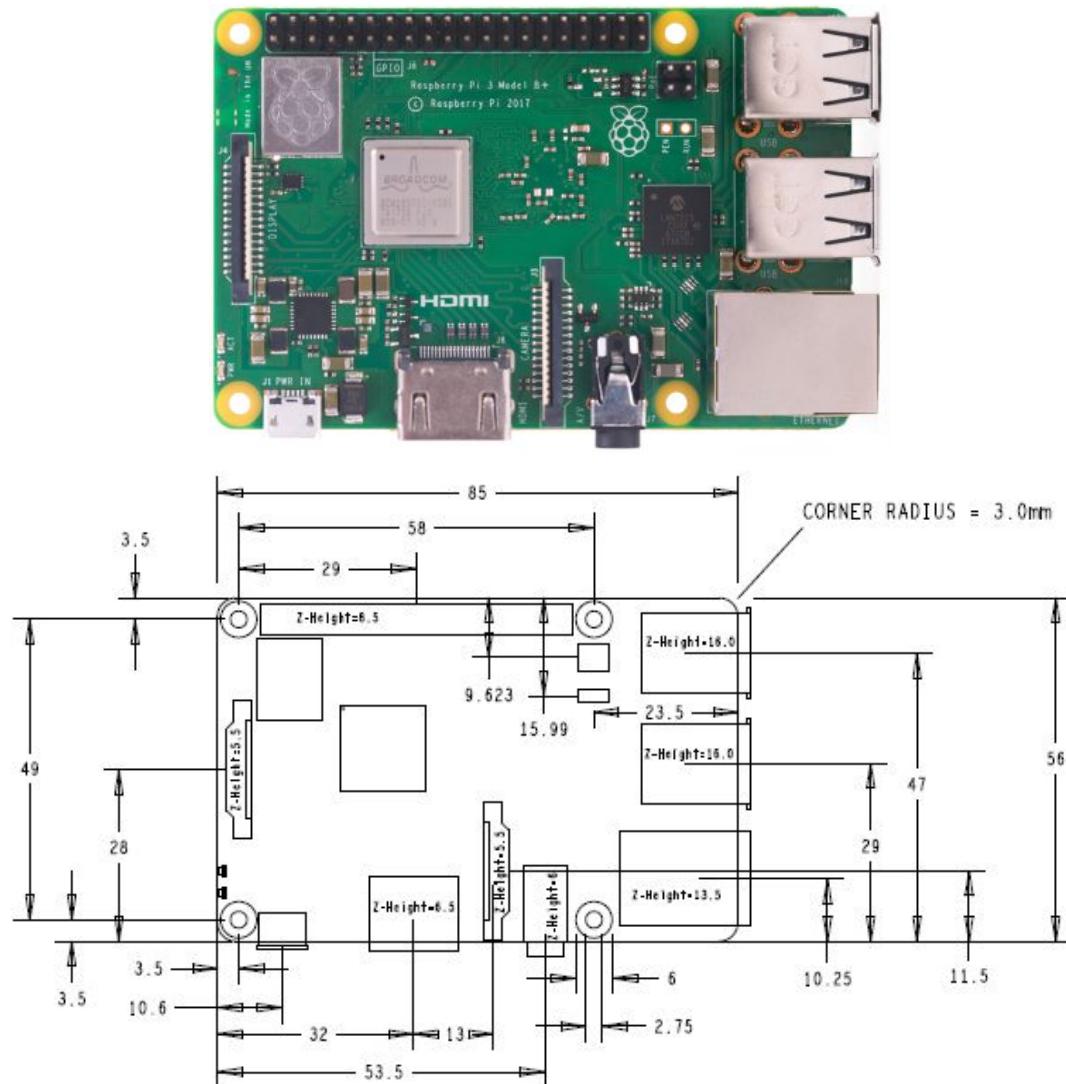


Figura 3.45: Raspberry Pi 3B. Fuente:[12]

La Raspberry Pi destaca principalmente por su versatilidad. Puede ejecutar diferentes sistemas operativos, además de disponer de una amplia gama de puertos de entrada y salida (GPIO) que permiten la conexión de sensores, actuadores y otros dispositivos electrónicos.

Se utiliza en una gran variedad de aplicaciones, desde proyectos de automatización del hogar, sistemas de vigilancia, hasta proyectos de robótica, Internet de las cosas (IoT), etc.

Otra ventaja clave es su conectividad. Con puertos Ethernet, Wi-Fi integrado y Bluetooth, se puede conectar la placa a redes y dispositivos externos, lo que permite el intercambio de datos y la comunicación con otros sistemas. Esto es especialmente útil en un entorno de IoT o en una red de sensores.

Gracias a su bajo costo y facilidad de uso, se ha convertido en una opción popular tanto para aficionados como para profesionales.

#### **Specifications**

<b>Processor</b>	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
<b>GPU</b>	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode.  Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
<b>Memory</b>	1GB LPDDR2
<b>Operating System</b>	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
<b>Dimensions</b>	85 x 56 x 17mm
<b>Power</b>	Micro USB socket 5V1, 2.5A
<b>Connectors:</b>	
<b>Ethernet</b>	10/100 BaseT Ethernet socket
<b>Video Output</b>	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
<b>Audio Output</b>	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
<b>GPIO Connector</b>	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
<b>Camera Connector</b>	15-pin MIPI Camera Serial Interface (CSI-2)
<b>Display Connector</b>	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
<b>Memory Card Slot</b>	Push/pull Micro SDIO

**Figura 3.46: Especificaciones Raspberry Pi 3B. Fuente:[12]**

### 3.3.1. Interfaz BMS - Raspberry Pi

El fabricante 123electric creó una interfaz, un cable que conecta la salida del BMS a un USB, con la finalidad de mostrar los datos que se monitorizan de las baterías en proyectos, usando para ello la Raspberry Pi.



Figura 3.47: Cable 123SmartBMS to USB. Fuente:[4]

La instalación es muy sencilla. Se enchufan los dos cables (no tienen polaridad) a las patillas *Ext Out* de placa *End Board* del BMS.

### 3.3.2. ThingSpeak

ThingSpeak es una plataforma de análisis IoT en la nube, que combina un servidor web, una base de datos y una API, lo que nos permite almacenar y transmitir datos utilizando el protocolo HTTP. Una de las características de ThingSpeak es que es de código abierto, lo que significa que podemos descargar su código y crear una instancia de ThingSpeak en nuestro propio servidor..

La simplicidad es uno de los aspectos destacados de ThingSpeak, ya que nos permite desarrollar rápidamente una aplicación visual para mostrar nuestros datos en cuestión de minutos, lo que nos permite centrarnos principalmente en la parte de hardware del proyecto.

Además, ThingSpeak, al ser desarrollado por Mathworks, ofrece una integración total con una herramienta ampliamente conocida como Matlab, lo cual permite aprovechar las capacidades y funciones de Matlab para realizar análisis avanzados y procesamiento de los datos almacenados en ThingSpeak.[19]

## Configuración del canal

Una vez creado el canal, utilizamos los campos necesarios para visualizar los datos. En nuestro caso, utilizaremos los ocho campos disponibles.

The screenshot shows the configuration page for a ThingSpeak channel. At the top, the Channel ID is 2023938 and the Name is bms\_uma. Below this, there is a large text area for the Description. A table lists eight fields, each with a checked checkbox indicating they are selected:

Field	Value	Status
Field 1	Pack Voltage	<input checked="" type="checkbox"/>
Field 2	Pack Current	<input checked="" type="checkbox"/>
Field 3	SOC	<input checked="" type="checkbox"/>
Field 4	Lowest Cell Temperati	<input checked="" type="checkbox"/>
Field 5	Highest Cell Temperat	<input checked="" type="checkbox"/>
Field 6	Charge Current	<input checked="" type="checkbox"/>
Field 7	Discharge Current	<input checked="" type="checkbox"/>
Field 8	Cell Comm. Error // Se	<input checked="" type="checkbox"/>

Below the fields, there are sections for "Write API Key" and "Read API Keys".

**Write API Key**

- Key: 37Q00CEM [REDACTED]
- Generate New Write API Key

**Help**

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

**API Keys Settings**

- Write API Key: Use this key to write data to a channel. If you feel your key has been compromised, click Generate New Write API Key.
- Read API Keys: Use this key to allow other people to view your private channel feeds and charts. Click Generate New Read API Key to generate an additional read key for the channel.
- Note: Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

**Read API Keys**

- Key: LC94F3LJT [REDACTED]
- Note: [REDACTED]
- Save Note
- Delete API Key
- Add New Read API Key

**API Requests**

- Write a Channel Feed  
GET [https://api.thingspeak.com/update?api\\_key=37Q00CEM&field1=0](https://api.thingspeak.com/update?api_key=37Q00CEM&field1=0)
- Read a Channel Feed  
GET <https://api.thingspeak.com/channels/2023938/feeds.json?results=2>
- Read a Channel Field  
GET <https://api.thingspeak.com/channels/2023938/fields/1.json?results=2>
- Read Channel Status Updates  
GET <https://api.thingspeak.com/channels/2023938/status.json>

**Figura 3.48: Configuración del canal ThingSpeak**

Con el channel ID y las claves API de lectura y escritura, podemos subir los datos que nos interesan.



**Figura 3.49: Apariencia del canal ThingSpeak**

Para poder analizar los datos de las baterías, utilizamos un programa en Python, desarrollado por AlbertroniDev.

## Instalación del Software

1. Descargar el archivo *install.sh* en la Raspberry Pi.
2. En la ubicación donde tenemos el script (/home/pi) ejecutamos el comando:  
*sudo bash install.sh* - Esto ejecuta el instalador con todos los permisos necesarios. El instalador descarga todos los archivos requeridos desde Github y los instala en /home/pi/smartbms-thingspeak.
3. Durante la instalación, se pide el puerto al que está conectado el BMS. En nuestro caso es /dev/ttyUSB0.
4. Después de esto, se pide la clave del canal de Thingspeak donde nos interesa subir los datos.

El sistema está configurado para actualizar los datos del BMS cada minuto.

```
○ ○ ○ smartbms.py

1 import serial_asyncio
2 import time
3
4 BMS_COMM_GAP = 300
5 BMS_COMM_TIMEOUT = 10000
6 BMS_COMM_BLOCK_SIZE = 58
7
8 class BMS(object):
9     def __init__(
10         self,
11         loop,
12         port
13     ):
14         self._loop = loop
15         self._serial_loop_task = None
16         self._port = port
17         self._last_received = 0
18         self._pack_voltage = 0
19         self._charge_current = 0
20         self._discharge_current = 0
21         self._pack_current = 0
22         self._soc = 0
23         self._lowest_cell_voltage = 0
24         self._lowest_cell_voltage_num = 0
25         self._highest_cell_voltage = 0
26         self._highest_cell_voltage_num = 0
27         self._lowest_cell_temperature = 0
28         self._lowest_cell_temperature_num = 0
29         self._highest_cell_temperature = 0
30         self._highest_cell_temperature_num = 0
31         self._cell_count = 0
32         self._cell_communication_error = 1
33         self._allowed_to_charge = 0
34         self._allowed_to_discharge = 0
35
36     async def connect(self):
37         self._serial_loop_task = self._loop.create_task(self._serial_read(self._port))
38
39     async def disconnect(self):
40         self._serial_loop_task.cancel()
41
42     @property
43     def pack_voltage(self):
44         return self._pack_voltage
45
46     @property
47     def charge_current(self):
48         return self._charge_current
49
50     @property
51     def discharge_current(self):
52         return self._discharge_current
53
54     @property
55     def pack_current(self):
```

```
56         return self._pack_current
57
58     @property
59     def soc(self):
60         return self._soc
61
62     @property
63     def lowest_cell_voltage(self):
64         return self._lowest_cell_voltage
65
66     @property
67     def lowest_cell_voltage_num(self):
68         return self._lowest_cell_voltage_num
69
70     @property
71     def highest_cell_voltage(self):
72         return self._highest_cell_voltage
73
74     @property
75     def highest_cell_voltage_num(self):
76         return self._highest_cell_voltage_num
77
78     @property
79     def lowest_cell_temperature(self):
80         return self._lowest_cell_temperature
81
82     @property
83     def lowest_cell_temperature_num(self):
84         return self._lowest_cell_temperature_num
85
86     @property
87     def highest_cell_temperature(self):
88         return self._highest_cell_temperature
89
90     @property
91     def highest_cell_temperature_num(self):
92         return self._highest_cell_temperature_num
93
94     @property
95     def cell_count(self):
96         return self._cell_count
97
98     @property
99     def cell_communication_error(self):
100        return self._cell_communication_error
101
102    @property
103    def serial_communication_error(self):
104        if(self._millis() > self._last_received + BMS_COMM_TIMEOUT):
105            return True
106        else:
107            return False
108
109    @property
110    def allowed_to_charge(self):
111        return self._allowed_to_charge
112
113    @property
114    def allowed_to_discharge(self):
115        return self._allowed_to_discharge
116
117    async def _serial_read(self, port):
118        reader, _ = await serial.asyncio.open_serial_connection(url=port, baudrate=9600)
```

```

119         self._last_received = self._millis()
120         bytes_received = 0
121         buf = bytearray(BMS_COMM_BLOCK_SIZE)
122         while True:
123             rx_byte = await reader.readexactly(1)
124             if self._millis() > self._last_received + BMS_COMM_GAP:
125                 bytes_received = 0
126                 self._last_received = self._millis()
127
128             if bytes_received <= BMS_COMM_BLOCK_SIZE-1:
129                 buf[bytes_received] = rx_byte[0]
130
131             bytes_received += 1
132
133             if bytes_received == BMS_COMM_BLOCK_SIZE:
134                 bytes_received = 0
135                 checksum = 0
136                 for i in range(BMS_COMM_BLOCK_SIZE-1):
137                     checksum += buf[i]
138
139                 received_checksum = buf[BMS_COMM_BLOCK_SIZE-1]
140                 if (checksum & 0xff) == received_checksum:
141                     self._pack_voltage = self._decode_voltage(buf[0:3])
142                     self._charge_current = self._decode_current(buf[3:6])
143                     self._discharge_current = self._decode_current(buf[6:9])
144                     self._pack_current = self._decode_current(buf[9:12])
145                     self._soc = buf[40]
146                     self._lowest_cell_voltage = self._decode_voltage(buf[12:14])
147                     self._lowest_cell_voltage_num = buf[14]
148                     self._highest_cell_voltage = self._decode_voltage(buf[15:17])
149                     self._highest_cell_voltage_num = buf[17]
150
151                     self._lowest_cell_temperature = self._decode_temperature(buf[18:20])
152                     self._lowest_cell_temperature_num = buf[20]
153                     self._highest_cell_temperature = self._decode_temperature(buf[21:23])
154                     self._highest_cell_temperature_num = buf[23]
155                     self._cell_count = buf[25]
156                     self._cell_communication_error = True if (buf[30] & 0b00000100) else
157                         False
158                     self._allowed_to_discharge = True if (buf[30] & 0b00000010) else False
159                     self._allowed_to_charge = True if (buf[30] & 0b00000001) else False
160
161     def _decode_current(self, raw_value):
162         if raw_value[0] == ord('X'):
163             return 0
164         elif raw_value[0] == ord('-'):
165             factor = -1
166         else:
167             factor = 1
168         return factor*round(0.125*int.from_bytes(raw_value[1:3], byteorder='big',
169         signed=False),1)
170
171     def _decode_voltage(self, raw_value):
172         #if(len(raw_value) == 3):
173             # voltage = int.from_bytes(raw_value[0:3], byteorder='big', signed=False)
174         #else:
175             voltage = int.from_bytes(raw_value, byteorder='big', signed=False)
176             return round(0.005*voltage,2)
177
178     def _decode_temperature(self, raw_value):
179         return round(int.from_bytes(raw_value[0:2], byteorder='big',
180         signed=False)*0.857-232,0)
181
182     def _millis(self):
183         return int(time.time() * 1000)

```

Figura 3.50: smartbms.py

Este código establece una conexión serie asíncrona, lee datos del BMS conectado y decodifica los datos recibidos para almacenarlos en variables accesibles a través de propiedades de la clase BMS.

Se definen varias propiedades que permiten acceder a los valores de los datos recibidos, como voltaje del paquete, corriente de carga y de descarga y SOC entre otros.

```
○ ○ ○          _main_.py

1 import sys
2 from argparse import ArgumentParser
3 import asyncio
4 import aiohttp
5 from smartbms.smartbms import BMS
6
7 class ComInstance:
8     def __init__(
9         self,
10        port,
11        key,
12        bms
13    ):
14        self.port = port
15        self.key = key
16        self.bms = bms
17
18     async def program():
19         tasks = []
20         instances = []
21         loop = asyncio.get_running_loop()
22         print('123\\SmartBMS to Thingspeak\\r\\n')
23         parser = ArgumentParser(description='123\\SmartBMS to Thingspeak')
24         parser.add_argument('-p', '--port', nargs='+', help='Serial port(s). For multiple BMS, separate the ports with a space.')
25         parser.add_argument('-k', '--key', nargs='+', help="Thingspeak channel key. For multiple BMS, separate the keys with a space.")
26         args = parser.parse_args()
27
28         if(args.port is None or args.key is None):
29             print('At least one port and key are required as argument.')
30             sys.exit()
31
```

```
32     # In some cases, argparse does not split multiple values in one argument, so we have to do
33     # this manually
34     ports = ' '.join(args.port).split(' ')
35     keys = ' '.join(args.key).split(' ')
36
37     for index, port in enumerate(ports):
38         bms = BMS(loop, port)
39         await bms.connect()
40         com = ComInstance(port, keys[index], bms)
41         instances.append(com)
42
43     # sleep for 10 seconds so we have time to receive BMS data and for internet connection to
44     # start
45     await asyncio.sleep(10)
46
47     while(True):
48         async with aiohttp.ClientSession() as session:
49             try:
50                 # Customize the code below to the values you want to upload to Thingspeak.
51                 for instance in instances:
52                     params = {
53                         'field1': str(instance.bms.pack_voltage),
54                         'field2': str(instance.bms.pack_current),
55                         'field3': str(instance.bms.soc),
56                         'field4': str(instance.bms.lowest_cell_voltage),
57                         'field5': str(instance.bms.highest_cell_voltage),
58                         'field6': str(int(instance.bms.allowed_to_charge)),
59                         'field7': str(int(instance.bms.allowed_to_discharge)),
60                         'field8': str(int(instance.bms.cell_communication_error or
61                             instance.bms.serial_communication_error))
62                     }
63                     params['api_key'] = instance.key
64                     async with session.get('https://api.thingspeak.com/update', params=params)
65                         as resp:
66                             if(resp.status == 200):
67                                 await resp.text()
68                             elif(resp.status == 400):
69                                 print('Wrong Thingspeak key: {}'.format(instance.key))
70                             else:
71                                 print('Error updating data: unknown error')
72
73                             print('Updated values')
74
75             except Exception as e:
76                 print('Could not reach Thingspeak server: ', e)
77
78             await asyncio.sleep(60)
79
80     def main():
81         asyncio.run(program())
82
83     if __name__ == "__main__":
84         main()
```

**Figura 3.51:** main.py

Este código recopila datos de nuestro BMS y los envía periódicamente a Thingspeak para su visualización y estudio.

El programa utiliza la biblioteca *asyncio* para realizar operaciones asíncronas y llevar a cabo la comunicación con los dispositivos BMS a través de puertos serie. Se pueden configurar los datos que queremos visualizar, ya que el canal ThingSpeak solo tiene ocho campos de visualización.

Analiza los argumentos de la línea de comandos para obtener los puertos serie y las claves de acceso (API key) a Thingspeak.

Para cada puerto y clave proporcionados, se crea una instancia de BMS y se establece una conexión, aunque ésto no será necesario para el proyecto, ya que sólo estamos leyendo de un puerto serie, para un solo BMS.

Luego, en un bucle infinito, se envían los datos del BMS a Thingspeak utilizando la biblioteca *aiohttp* para realizar solicitudes HTTP<sup>3</sup>.

Se realiza un control de posibles errores y se espera un intervalo de tiempo antes de cada envío de datos. En resumen, el programa principal se ocupa de la recopilación y envío de datos del BMS a Thingspeak para su monitoreo y análisis.

El código mostrado lo configuramos para visualizar el voltaje y corriente totales, SOC, tensiones más altas y bajas, así como dos señales booleanas que indican si está permitida la carga o la descarga. El último campo (field8) está dedicado para una variable que indica si hay un error en la comunicación. Además de estas variables, están disponibles también las temperaturas máximas y mínimas, así como las corrientes de carga y descarga. Simplemente tendríamos que cambiar estos campos para ver los datos que nos interesen.

---

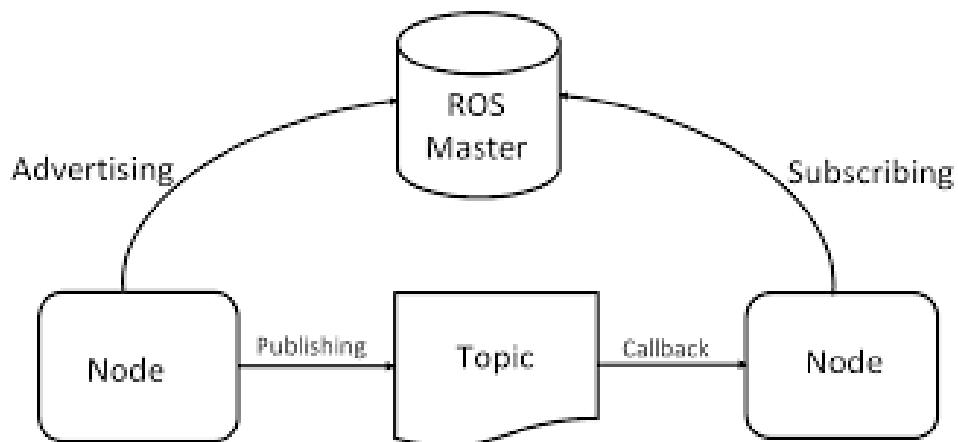
<sup>3</sup>Hypertext Transfer Protocol

### 3.3.3. ROS

El Robot Operating System (ROS) no es un sistema operativo, sino un framework<sup>4</sup>, por lo que requiere utilizar un sistema operativo compatible.

Se trata de una plataforma ampliamente utilizada en el campo de la robótica. Se ha convertido en un estándar de facto debido a su flexibilidad y modularidad, lo que permite a los desarrolladores crear y compartir software de manera eficiente. Una de las principales ventajas de ROS es su enfoque en la reutilización de código, lo que significa que los componentes y algoritmos desarrollados para un robot en particular pueden ser fácilmente transferidos y utilizados en otros robots con pocos o ningún cambio necesario.

La modularidad de ROS se basa en el concepto de nodos independientes, que son procesos ejecutables que se comunican entre sí a través de un sistema de mensajes. Esto significa que diferentes partes de un sistema robótico, como sensores, actuadores y algoritmos de control, pueden funcionar como nodos independientes y comunicarse entre sí de manera eficiente. Además, ROS proporciona una amplia variedad de bibliotecas y herramientas que facilitan el desarrollo de software robótico, como la gestión de paquetes, la simulación y la visualización.



**Figura 3.52: Esquema de funcionamiento ROS**

Existe una amplia comunidad de usuarios y desarrolladores de ROS que comparten sus conocimientos, bibliotecas y proyectos a través de repositorios en línea. Esto fomenta la colaboración y acelera el desarrollo de soluciones robóticas.

<sup>4</sup>Un framework es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos

al permitir a los usuarios aprovechar el trabajo realizado por otros y compartir sus propias contribuciones.[7]

De ésta forma, podremos lograr la transferencia de información a los procesos robóticos que lleve nuestro proyecto, como es el caso del brazo robótico.

### Versión de ROS

Para el proyecto, instalamos la versión ROS Noetic, ya que es la más actual y la que mejor funciona en la Raspberry Pi 3. Ésta versión está enfocada a Ubuntu 20.04. Para instalarlo, hemos seguido de forma rigurosa los pasos que se establecen en la guía oficial de ROS.[14]

Hemos de destacar el uso de SSH<sup>5</sup> con puTTY para la instalación de ROS, sus paquetes y todo lo que conlleva, debido a la sencillez y comodidad de ésta vía, además de no requerir el uso de un monitor y otros elementos de visualización.

Para usar puTTY, la Raspberry Pi y el ordenador deben estar conectados a la misma red. Simplemente se ingresa la dirección IP del dispositivo al que nos queremos conectar desde el ordenador.

---

<sup>5</sup>Secure Shell (SSH) es un protocolo cuya principal función es el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada.

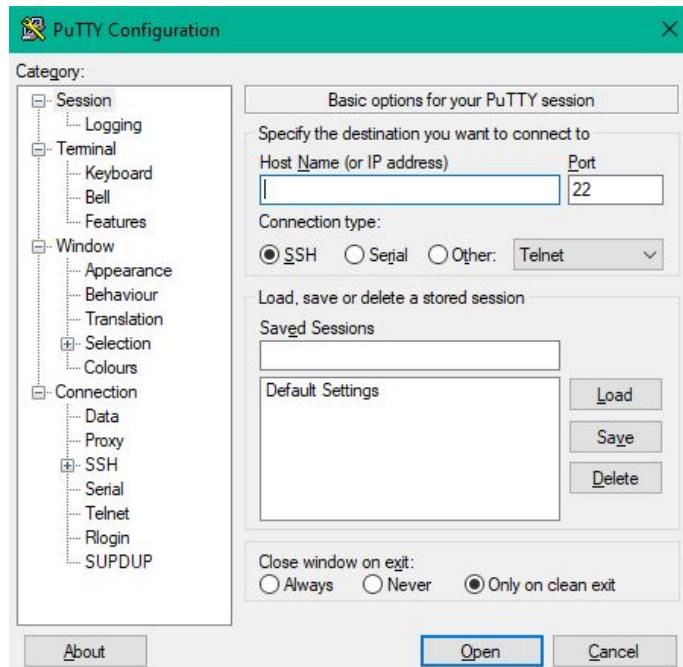


Figura 3.53: SSH con puTTY

## Sistema Operativo

El sistema operativo que instalamos en la Raspberry Pi es el Ubuntu 20.04.5 LTS, ya que ésta versión incluye grandes mejoras a nivel de rendimiento, interfaz de usuario y usabilidad, como un menor uso de CPU respecto a otras versiones de Ubuntu. Además, es el sistema operativo que mejor funciona para la versión de ROS que instalamos.

Para instalarlo, necesitamos escribir la imagen del sistema operativo en una tarjeta microSD, ya que es el tipo de almacenamiento que utiliza Raspberry Pi. Nosotros usamos una con capacidad de 16 GB.

## Paquete ROS

En primer lugar creamos un espacio de trabajo para añadir todos los paquetes necesarios. Tras crear este espacio, creamos un paquete que contendrá un nodo para publicar los datos de las baterías. De nuevo, para crear nuestro paquete seguimos las indicaciones que recomienda la página oficial.[14]

## Nodos ROS

Los nodos son los bloques fundamentales de un sistema en ROS y están diseñados para ser pequeños, modulares y especializados. Cada nodo está dedicado a una función particular, en nuestro proyecto, la función del nodo será recibir los datos de las baterías leídas a través del puerto serie y publicarlos en un topic.

La comunicación entre nodos se realiza mediante un sistema de publicación-suscripción.

```
○○○          bms_publicador_v2.py

1  #!/usr/bin/python3
2
3  import sys
4  import asyncio
5  import rospy
6  from std_msgs.msg import Float32
7  from sensor_msgs.msg import BatteryState
8
9  from smartbms.smartbms import BMS
10
11 class ComInstance:
12     def __init__(self, port, key, bms):
13         self.port = port
14         self.key = key
15         self.bms = bms
16
17     async def program():
18         instances = []
19         loop = asyncio.get_running_loop()
20         print('123\\SmartBMS to ROS\r\n')
21
22         port = '/dev/ttyUSB0' # Puerto a utilizar (puedes modificarlo según tus necesidades)
23
24         bms = BMS(loop, port)
25         await bms.connect()
26         com = ComInstance(port, None, bms)
27         instances.append(com)
28
29         # sleep for 10 seconds so we have time to receive BMS data and for internet connection to
30         # start
31         await asyncio.sleep(10)
32
33         # Inicializa nodo ROS y publicador
34         rospy.init_node('smartbms')
35         pub = rospy.Publisher('battery_state', BatteryState, queue_size=10)
```

```

36     while True:
37         try:
38             # Datos publicados a ROS.
39             for instance in instances:
40                 battery_voltage = instance.bms.pack_voltage
41                 battery_current = instance.bms.pack_current
42                 soc = instance.bms.soc
43                 lowest_cell_voltage = instance.bms.lowest_cell_voltage
44                 highest_cell_voltage = instance.bms.highest_cell_voltage
45                 allowed_to_charge = instance.bms.allowed_to_charge
46                 allowed_to_discharge = instance.bms.allowed_to_discharge
47                 cell_communication_error = instance.bms.cell_communication_error or
48                 instance.bms.serial_communication_error
49
50                 # Creamos mensaje BatteryState
51                 battery_state_msg = BatteryState()
52                 battery_state_msg.voltage = battery_voltage
53                 battery_state_msg.current = battery_current
54                 battery_state_msg.percentage = soc
55
56                 # Publicamos el topic battery state
57                 pub.publish(battery_state_msg)
58
59                 # Imprimimos los valores a través de la terminal de Linux
60                 print('Battery State: voltage={}, current={}, soc={}, lowest_cell_voltage={}',
61                     highest_cell_voltage={}, allowed_to_charge={}, allowed_to_discharge={},
62                     cell_communication_error={}'.format(
63                         battery_voltage,
64                         battery_current,
65                         soc,
66                         lowest_cell_voltage,
67                         highest_cell_voltage,
68                         allowed_to_charge,
69                         allowed_to_discharge,
70                         cell_communication_error
71                     ))
72
73             except Exception as e:
74                 print('Error publishing battery state to ROS:', e)
75                 await asyncio.sleep(60)
76
77         def main():
78             asyncio.run(program())
79
80     if __name__ == "__main__":
81         main()
82

```

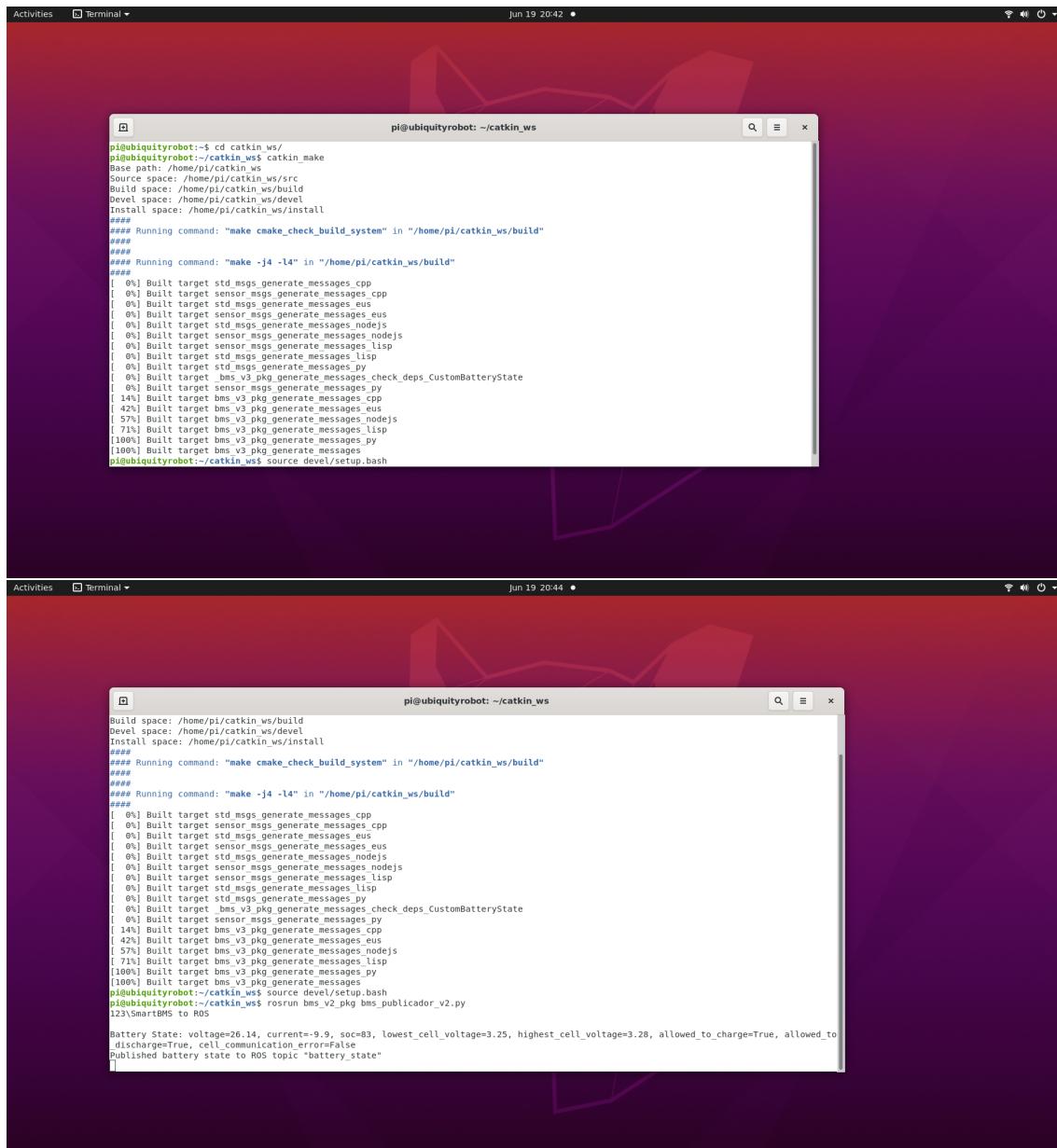
Figura 3.54: bms-publicador-v2.py

El primer nodo implementado que funcionó correctamente es el nodo *bms-publicador-v2.py* del paquete *bms-v2-pkg*. Se trata de un programa que reutiliza el código para la lectura de los datos adquiridos a través del BMS. Hemos importado el módulo *BMS*, donde se definen todos los datos de las baterías.

Creamos el nodo *smartbms* con la línea de comando *rospy.init\_node('smartbms')*.

Esta función se ocupa de configurar y registrar nuestro nodo en el sistema ROS, establecer las conexiones necesarias y preparar el entorno para la comunicación con otros nodos.

Posteriormente, creamos el publicador, con el comando `pub = rospy.Publisher('battery-state', BatteryState, queue_size=10)`, el cual publica mensajes de tipo `BatteryState` en el topic `battery-state`, con una capacidad de 10 mensajes en la cola.



```

pi@ubiquityrobot:~$ cd catkin_ws
pi@ubiquityrobot:~/catkin_ws$ catkin_make
Source space: /home/pi/catkin_ws/src
Build space: /home/pi/catkin_ws/build
Devel space: /home/pi/catkin_ws/devel
Install space: /home/pi/catkin_ws/install
#####
#### Running command: "make cmake_check_build_system" in "/home/pi/catkin_ws/build"
#####
#####
#### Running command: "make -j4 -l4" in "/home/pi/catkin_ws/build"
[ 0%] Built target std_msgs_generate_messages_cpp
[ 0%] Built target sensor_msgs_generate_messages_cpp
[ 0%] Built target std_msgs_generate_messages_eus
[ 0%] Built target sensor_msgs_generate_messages_eus
[ 0%] Built target std_msgs_generate_messages_nodejs
[ 0%] Built target sensor_msgs_generate_messages_nodejs
[ 0%] Built target sensor_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_py
[ 0%] Built target sensor_msgs_generate_messages_py
[ 0%] Built target check_deps_CustomBatteryState
[ 0%] Built target sensor_msgs_generate_messages_py
14% Built target bms_v3_pkg_generate_messages_cpp
42% Built target bms_v3_pkg_generate_messages_eus
50% Built target bms_v3_pkg_generate_messages_nodejs
71% Built target bms_v3_pkg_generate_messages_lisp
100% Built target bms_v3_pkg_generate_messages_py
100% Built target bms_v3_pkg_generate_messages
pi@ubiquityrobot:~/catkin_ws$ source devel/setup.bash

Build space: /home/pi/catkin_ws/build
Devel space: /home/pi/catkin_ws/devel
Install space: /home/pi/catkin_ws/install
#####
#### Running command: "make cmake_check_build_system" in "/home/pi/catkin_ws/build"
#####
#####
#### Running command: "make -j4 -l4" in "/home/pi/catkin_ws/build"
[ 0%] Built target std_msgs_generate_messages_cpp
[ 0%] Built target sensor_msgs_generate_messages_cpp
[ 0%] Built target std_msgs_generate_messages_eus
[ 0%] Built target sensor_msgs_generate_messages_eus
[ 0%] Built target std_msgs_generate_messages_nodejs
[ 0%] Built target sensor_msgs_generate_messages_nodejs
[ 0%] Built target sensor_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_py
[ 0%] Built target sensor_msgs_generate_messages_py
[ 0%] Built target check_deps_CustomBatteryState
[ 0%] Built target sensor_msgs_generate_messages_py
Built target bms_v3_pkg_generate_messages_cpp
42% Built target bms_v3_pkg_generate_messages_eus
57% Built target bms_v3_pkg_generate_messages_nodejs
71% Built target bms_v3_pkg_generate_messages_lisp
100% Built target bms_v3_pkg_generate_messages_py
100% Built target bms_v3_pkg_generate_messages
pi@ubiquityrobot:~/catkin_ws$ source devel/setup.bash
pi@ubiquityrobot:~/catkin_ws$ rosrun bms_v2_pkgs bms_publicador_v2.py
123SmartBMS to ROS
Battery State: voltage=26.14, current=-0.9, soc=83, lowest_cell_voltage=3.25, highest_cell_voltage=3.28, allowed_to_charge=True, allowed_to_discharge=True, cell_communication_error=False
Published battery state to ROS topic "battery_state"
]

```

Figura 3.55: Inicialización del nodo publicador en ROS

Primero se compila el espacio de trabajo. En segundo lugar se inicia el nodo maestro con el comando *roscore*. Finalmente se inicia el nodo, con el comando *rosrun [nombre paquete] [nombre nodo]*.

```
# Constants are chosen to match the enums in the linux kernel
# defined in include/linux/power_supply.h as of version 3.7
# The one difference is for style reasons the constants are
# all uppercase not mixed case.

# Power supply status constants
uint8 POWER_SUPPLY_STATUS_UNKNOWN = 0
uint8 POWER_SUPPLY_STATUS_CHARGING = 1
uint8 POWER_SUPPLY_STATUS_DISCHARGING = 2
uint8 POWER_SUPPLY_STATUS_NOT_CHARGING = 3
uint8 POWER_SUPPLY_STATUS_FULL = 4

# Power supply health constants
uint8 POWER_SUPPLY_HEALTH_UNKNOWN = 0
uint8 POWER_SUPPLY_HEALTH_GOOD = 1
uint8 POWER_SUPPLY_HEALTH_OVERHEAT = 2
uint8 POWER_SUPPLY_HEALTH_DEAD = 3
uint8 POWER_SUPPLY_HEALTH_OVERVOLTAGE = 4
uint8 POWER_SUPPLY_HEALTH_UNSPEC_FAILURE = 5
uint8 POWER_SUPPLY_HEALTH_COLD = 6
uint8 POWER_SUPPLY_HEALTH_WATCHDOG_TIMER_EXPIRE = 7
uint8 POWER_SUPPLY_HEALTH_SAFETY_TIMER_EXPIRE = 8

# Power supply technology (chemistry) constants
uint8 POWER_SUPPLY_TECHNOLOGY_NIMH = 0
uint8 POWER_SUPPLY_TECHNOLOGY_LION = 1
uint8 POWER_SUPPLY_TECHNOLOGY_LIPO = 2
uint8 POWER_SUPPLY_TECHNOLOGY_LIFE = 3
uint8 POWER_SUPPLY_TECHNOLOGY_NICD = 4
uint8 POWER_SUPPLY_TECHNOLOGY_LIMN = 5

Header header
float32 voltage           # Voltage in Volts (Mandatory)
float32 temperature        # Temperature in Degrees Celsius (If unmeasured NaN)
float32 current             # Negative when discharging (A) (If unmeasured NaN)
float32 charge              # Current charge in Ah (If unmeasured NaN)
float32 capacity            # Capacity in Ah (last full capacity) (If unmeasured NaN)
float32 design_capacity     # Capacity in Ah (design capacity) (If unmeasured NaN)
float32 percentage          # Charge percentage on 0 to 1 range (If unmeasured NaN)
uint8 power_supply_status   # The charging status as reported. Values defined above
uint8 power_supply_health    # The battery health metric. Values defined above
uint8 power_supply_technology # The battery chemistry. Values defined above
bool present                # True if the battery is present

float32[] cell_voltage      # An array of individual cell voltages for each cell in the pack
                           # If individual voltages unknown but number of cells known set each to NaN
float32[] cell_temperature   # An array of individual cell temperatures for each cell in the pack
                           # If individual temperatures unknown but number of cells known set each to NaN
string location             # The location into which the battery is inserted. (slot number or plug)
string serial_number         # The best approximation of the battery serial number
```

**Figura 3.56: Mensaje BatteryState**

La motivación de uso de este mensaje radica en las siguientes características:

Estándar de la industria: BatteryState es un mensaje estándar en ROS que permite representar datos relacionados con el estado de la batería. al tratarse de un mensaje estándar, logramos seguir una convención comúnmente aceptada en la comunidad de robótica y facilita la colaboración con otros sistemas y paquetes de software que también utilizan este estándar.

**Reutilización de código:** Al utilizar el mensaje BatteryState, nos permite aprovechar el código y las herramientas existentes que ya trabajan con este tipo de mensaje, ya que existen numerosos paquetes de software y controladores en ROS que están diseñados para interactuar con este mensaje, lo que nos permite simplificar la integración del sistema con otros componentes existentes.

**Claridad y consistencia:** El uso de mensajes estándar, como BatteryState, aporta una estructura bien definida y sólida para transmitir la información que se registra del BMS.

El contenido que publicamos consiste en asociar los campos de datos del BMS con las variables definidas en el mensaje estándar.

Por otro lado, logramos hacer el programa más práctico, imprimiendo todos los datos por la terminal, de forma que cada minuto se lanza el nodo con los valores actualizados.

Sin embargo, tenemos la necesidad de transmitir otros datos de la batería que actualmente no están disponibles en el mensaje, como son las tensiones máximas y mínimas o las corrientes de carga y descarga. El campo de la temperatura sí está implementado, pero no se ha podido configurar correctamente y no lo podemos utilizar.

Como solución, creamos otro paquete con un nuevo nodo publicador. En éste caso, creamos un mensaje llamado *CustomBatteryState*, el cual contiene tanto los campos nuevos como los que ya estaban disponibles en el mensaje estándar.

La desventaja en este caso deriva en que no se puede integrar y colaborar de una forma tan sencilla con otros sistemas y procesos robóticos.

```

○ ○ ○                               CustomBatteryState.msg

1 #CustomBatteryState.msg
2
3 float32 voltage
4 float32 current
5 float32 percentage
6
7 float32 charge_current
8 float32 discharge_current
9
10 float32 min_cell_voltage
11 float32 max_cell_voltage
12 float32 min_cell_temperature
13 float32 max_cell_temperature
14 bool allow_charge
15 bool allow_discharge

```

**Figura 3.57: Mensaje CustomBatteryState**

Para crear éste mensaje hemos seguido los tutoriales oficiales de ROS.

## Software de visualización

Actualmente, existen varias herramientas de visualización compatibles con ROS.

RQt es un programa muy completo que ofrece una amplia gama de funcionalidades, incluyendo la visualización de datos, el control de nodos, la depuración y la configuración de parámetros en tiempo real.

Permite crear interfaces gráficas personalizadas (GUI) mediante el uso de plugins, lo cual facilita la integración de diferentes vistas y herramientas en una única ventana.

Es muy extensible y admite una gran cantidad de plugins que amplían sus capacidades, lo que lo hace adecuado para una amplia gama aplicaciones en el desarrollo de robots.

En nuestro caso, para el proyecto desarrollado utilizaremos Plotjuggler, creado por Davide Faconti.

PlotJuggler se centra principalmente en la visualización de datos en tiempo real. Tiene una interfaz gráfica intuitiva y sensilla de usar para cargar, explorar y analizar datos de sensores y actuadores.

Hemos considerado usar este programa para el análisis y visualización de los datos de las baterías, debido a que consume menos recursos y es más sencillo que RQt en varios aspectos.

Además, permite la configuración interactiva de las gráficas, como ajustar los ejes, cambiar los colores y estilos de las líneas. Es compatible con distintos formatos de datos, como archivos CSV, rosbags y mensajes de ROS en formato binario.

Funciona como un nodo suscriptor a nuestro topic battery-state y permite graficar cualquier campo que sea de interés analizar. Otras fuentes de datos con las que trabaja Plotjuggler son a través de suscripción a fuentes de streaming como MQTT<sup>6</sup>, además de ser compatible con rosbags<sup>7</sup> y archivos CSV.[6]

---

<sup>6</sup>Message Queuing Telemetry Transport) es un protocolo de mensajería ligero y de código abierto diseñado para la comunicación entre dispositivos en redes de sensores o de Internet de las Cosas

<sup>7</sup>Formato de archivo de ROS para almacenar y reproducir datos. Permite grabar y reproducir información generada por diferentes nodos en un sistema ROS, como mensajes, tópicos, servicios y parámetros.

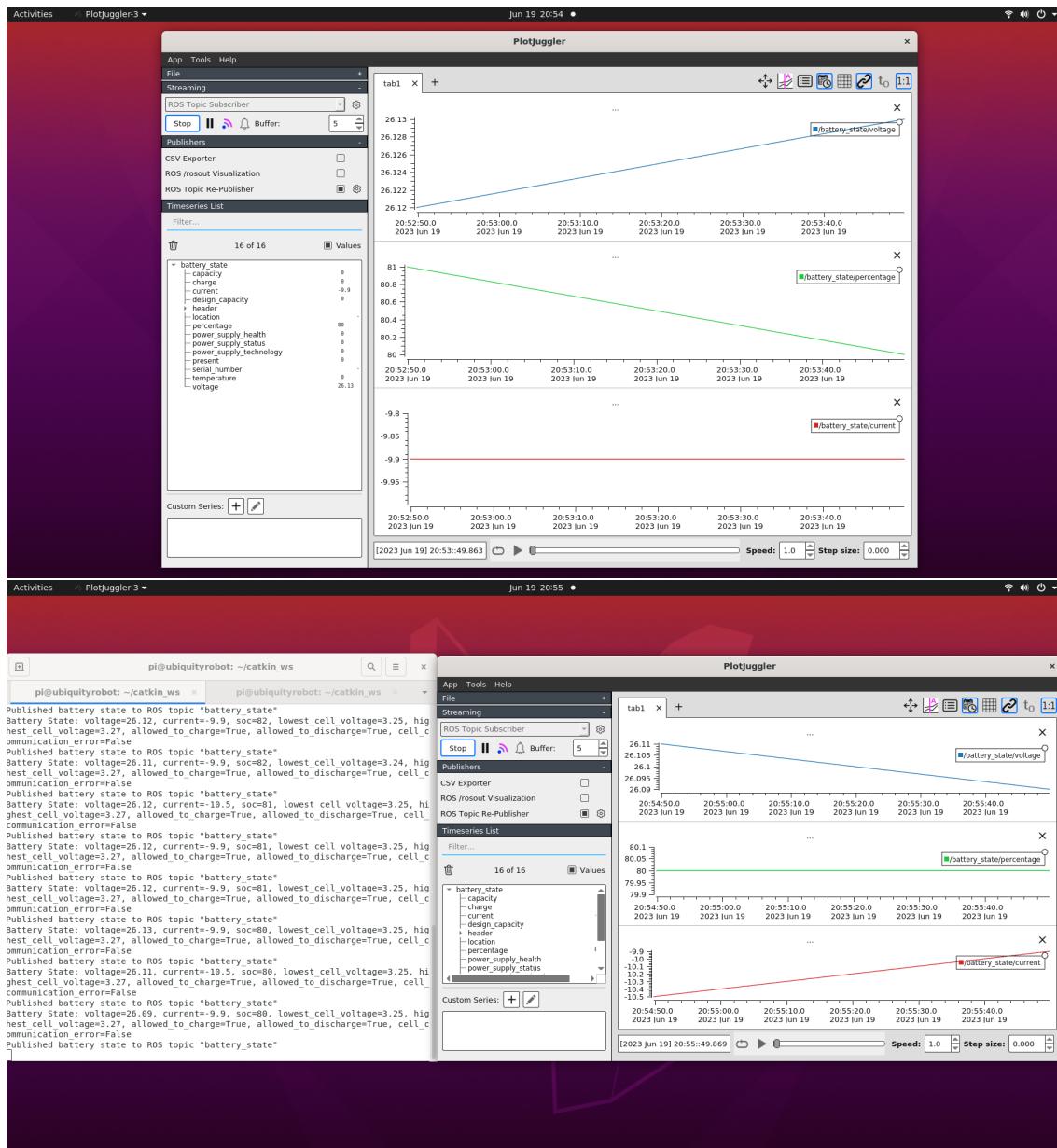


Figura 3.58: PlotJuggler

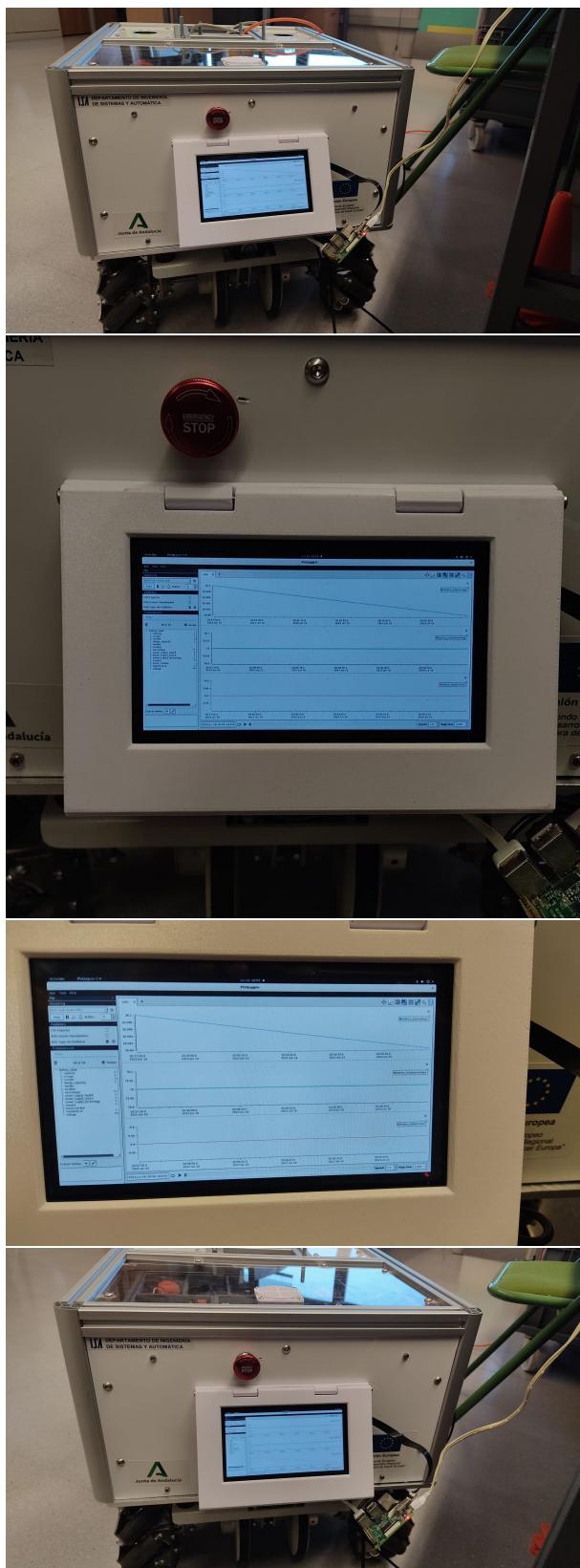
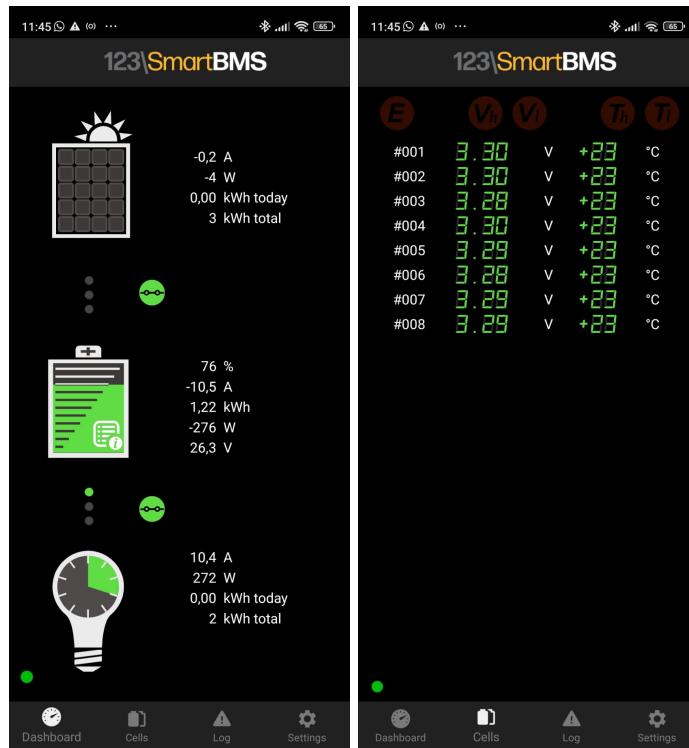


Figura 3.59: Plotjuggler en el dispositivo de visualización de la plataforma.

La interfaz es sencilla de usar y nos permite visualizar los datos que nos interesen. En la imagen observamos la tensión total, el SOC y la corriente total del sistema. Además, podemos ver los datos publicados en el topic, que se muestran en la terminal.

### 3.3.4. App para Smartphones

La compañía 123 Electric creó una aplicación para monitorizar el estado actual del sistema, si está cargando, descargando, energía en el momento, etc. Tiene un menú de ajustes que nos permite configurar los parámetros de tensiones máximas y mínimas permitidas, así como las temperaturas. Además, podemos controlar el relé para permitir la carga, la descarga o ambas.



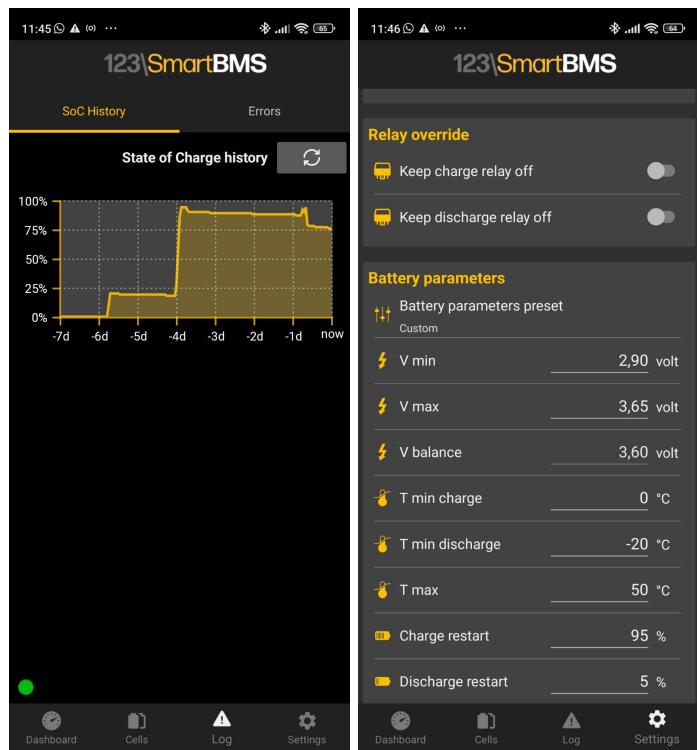


Figura 3.61: App de 123 Electric para Smartphones



# **Capítulo 4**

## **Verificación y pruebas**

En éste capítulo realizaremos algunas pruebas experimentales para comprobar cómo son los ciclos de carga y descarga de las baterías LiFePO<sub>4</sub> y los parámetros del SOC, corrientes, temperaturas, etc.

### **4.1. Sistema de pruebas**

El sistema seguido para hacer las pruebas se basa en cargar el sistema de energía en su totalidad, para descargarlo posteriormente.

En la descarga, necesitamos un dispositivo que consuma una potencia relativamente alta para poder monitorizar el proceso y adquirir los datos necesarios.

Disponemos de un proyector modelo *3M 9200 Overhead* que consume 275 W.



**Figura 4.1: Sistema de pruebas con el proyector**

## 4.2. Pruebas realizadas

Realizamos las pruebas en dos pasos:

1. Registraremos los datos en el canal ThingSpeak y los exportamos en formato .CSV de Excel. Al abrir los datos, nos aparecen en una única columna, separados por comas. Para trabajar con los datos adecuadamente, debemos seleccionar la pestaña de datos y convertir el texto en columnas.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	
	fecha	entry_id	field1	field2	field3	field4	field5	field6	field7	field8	latitude	longitude	elevation	status
1	2023-02-03T13:18:36+01:00	1	0	0	0	0	0	0	0	0	1			
2	2023-02-03T13:19:37+01:00	2	0	0	0	0	0	0	0	0	1			
4	2023-02-03T13:20:37+01:00	3	0	0	0	0	0	0	0	0	1			
5	2023-02-03T13:21:38+01:00	4	0	0	0	0	0	0	0	0	1			
6	2023-02-03T13:22:39+01:00	5	0	0	0	0	0	0	0	0	1			
7	2023-02-03T13:23:40+01:00	6	0	0	0	0	0	0	0	0	1			
8	2023-02-03T13:24:32+01:00	7	26.66	0.0	88 3.33	3.33		0	0	0				
9	2023-02-03T13:25:33+01:00	8	26.66	0.0	88 3.33	3.34		0	0	0				
10	2023-02-03T13:26:33+01:00	9	26.65	0.0	88 3.33	3.34		0	0	0				
11	2023-02-03T13:27:34+01:00	10	26.66	0.0	88 3.33	3.33		0	0	0				
12	2023-02-03T13:28:34+01:00	11	26.66	0.0	88 3.33	3.34		0	0	0				
13	2023-02-03T13:29:35+01:00	12	26.66	0.0	88 3.33	3.34		0	0	0				
14	2023-02-03T13:30:35+01:00	13	26.64	0.0	88 3.33	3.33		0	0	0				
15	2023-02-03T13:31:36+01:00	14	26.65	0.0	88 3.33	3.33		0	0	0				
16	2023-02-03T13:32:36+01:00	15	26.66	0.0	88 3.33	3.34		0	0	0				
17	2023-02-03T13:33:37+01:00	16	26.66	0.0	88 3.33	3.33		0	0	0				
18	2023-02-03T13:34:37+01:00	17	26.66	0.0	88 3.33	3.34		0	0	0				
19	2023-02-03T13:35:38+01:00	18	26.66	0.0	88 3.33	3.34		0	0	0				
20	2023-02-03T13:36:38+01:00	19	26.66	0.0	88 3.33	3.33		0	0	0				

**Figura 4.2: Datos exportados de ThingSpeak en Excel**

2. Creamos un programa en Matlab para leer el archivo .CSV, que nos permita representar gráficamente todos los campos de datos en función del tiempo, añadiendo además un filtro de fechas para poder analizar de forma concreta y eficaz.

```
% Leer la tabla CSV
data = readtable('feeds(9).csv', 'Delimiter', ';', 'TextType', 'string');

% Obtener las fechas como strings
fecha_str = data.created_at;

% Convertir las fechas a datetime
fecha = datetime(fecha_str, 'TimeZone', '+02:00', 'InputFormat', 'yyyy-MM-dd''T''HH:mm:ssXXX');
fecha.TimeZone = '';

% Rango de fechas para filtrar
fecha_inicio = datetime('19-Jun-2023 18:04:00');
fecha_fin = datetime('19-Jun-2023 21:00:00');

% Filtro lógica basada en el rango de fechas
mascara = fecha >= fecha_inicio & fecha <= fecha_fin;

% Filtrar los datos
data_filtrada = data(mascara, :);

% Obtener los datos para representación gráfica
y1 = data_filtrada.field1;
y2 = data_filtrada.field2;
y3 = data_filtrada.field3;
y4 = data_filtrada.field4;
y5 = data_filtrada.field5;
y6 = data_filtrada.field6;
y7 = data_filtrada.field7;
y8 = data_filtrada.field8;

y1_ = str2double(y1);
y3_ = str2double(y3);

% Crear un vector de fechas para el eje x
fechas_plot = fecha(mascara);

% Plotear los datos con las fechas en el eje x
figure()

subplot(2,2,1)
plot(fechas_plot,y1)
title('Voltaje Total')
xlabel('Fecha');
ylabel('V');
legend('Field 1');

subplot(2,2,2)
plot(fechas_plot,y2)
title('Corriente Total')
xlabel('Fecha');
ylabel('A');
legend('Field 2');
```

```
subplot(2,2,3)
plot(fechas_plot,y3)
title('SoC')
xlabel('Fecha');
ylabel('%');
legend('Field 3');

subplot(2,2,4)
plot(fechas_plot,y4)
title('Temperatura más baja')
%title('Voltaje más bajo')
xlabel('Fecha');
ylabel('°C');
%ylabel('V');
legend('Field 4');

figure()
subplot(2,2,1)
plot(fechas_plot,y5)
title('Temperatura más alta')
%title('Voltaje más alto')
xlabel('Fecha');
ylabel('°C');
%ylabel('V');
legend('Field 5');

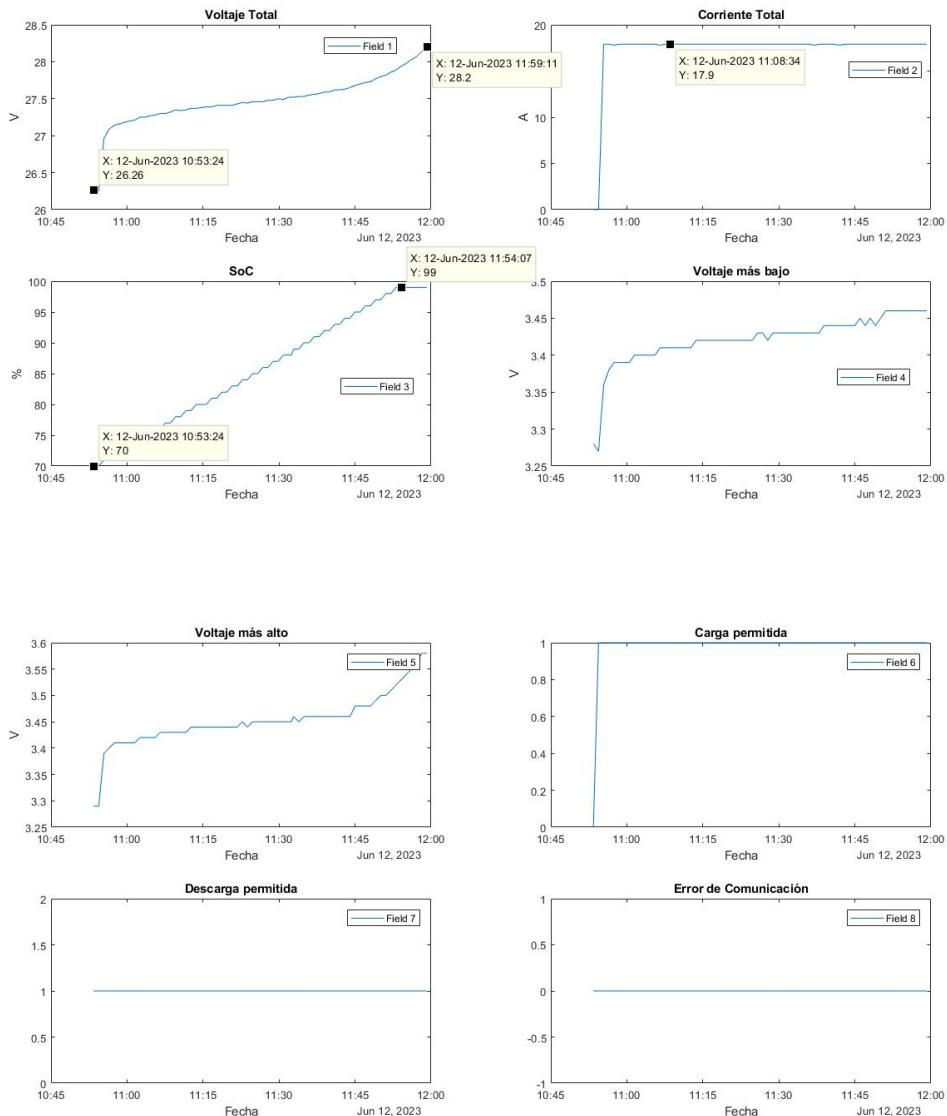
subplot(2,2,2)
plot(fechas_plot,y6)
title('Corriente de Carga')
%title('Carga permitida')
xlabel('Fecha');
ylabel('A');
legend('Field 6');

subplot(2,2,3)
plot(fechas_plot,y7)
title('Corriente de Descarga')
%title('Descarga permitida')
xlabel('Fecha');
ylabel('A');
legend('Field 7');

subplot(2,2,4)
plot(fechas_plot,y8)
title('Error de Comunicación')
xlabel('Fecha');
legend('Field 8');
```

**Figura 4.3: Código Matlab**

## Primera Carga



**Figura 4.4: Primera Carga - 12/06/23**

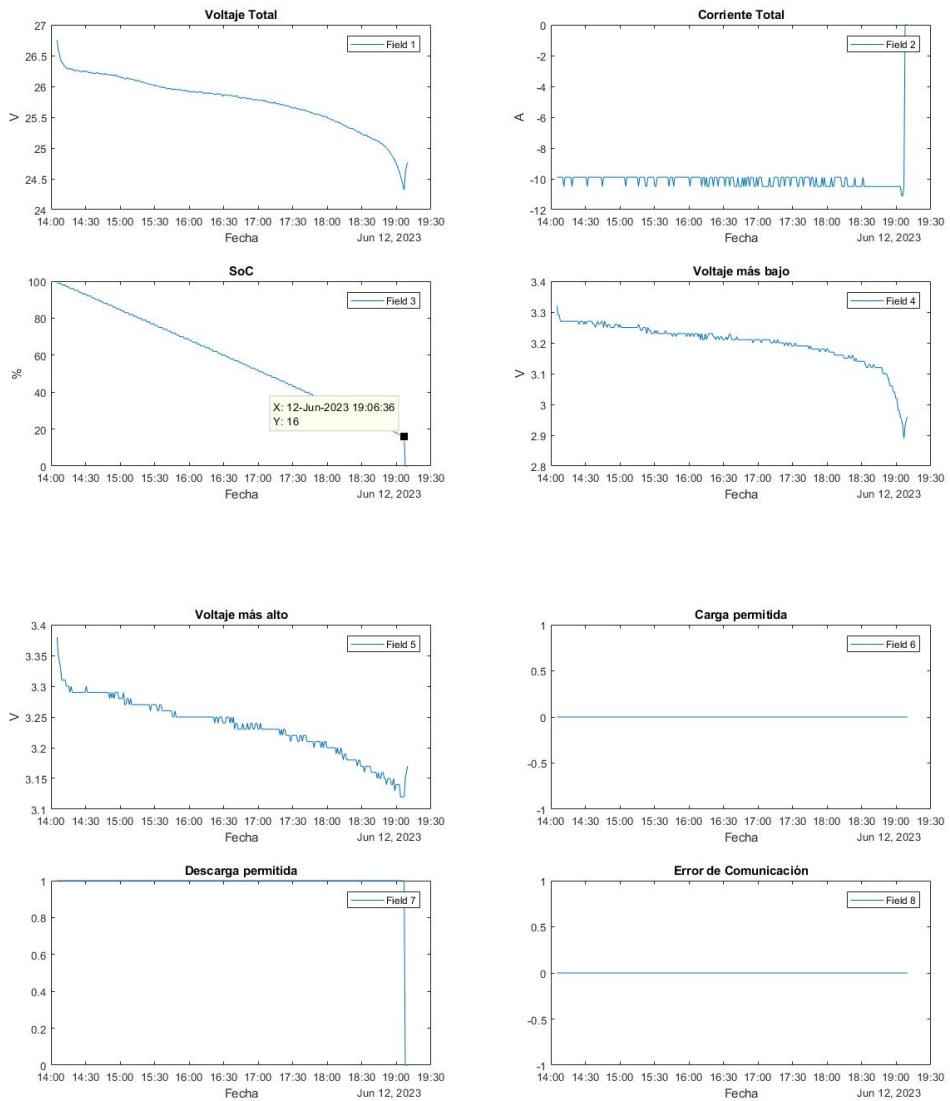
En ésta primera prueba de carga, partimos de un 70 % SOC y conseguimos alcanzar el 100 % en 1 hora, a la vista de los resultados prácticos. Podemos observar que la corriente total es positiva, 17.9 A, ya que está cargando el sistema.

Teóricamente, 1440 Wh (8 celdas x 3v x 60 Ah) corresponde con la capacidad máxima, 1008 Wh es el 70 %. Sabiendo ésto, para completar la capacidad total,

debemos cargar  $1440 - 1008 = 432$  Wh. La potencia de alimentación de la red es de 480 W. Por lo tanto, teóricamente tarda en cargarse  $432 \text{ Wh} / 480 \text{ W} = 0.9$  horas.

Por lo tanto, consideramos que la prueba realizada se adapta a los datos teóricos.

### Primera Descarga



**Figura 4.5: Primera Descarga - 12/06/23**

En el proceso de descarga, podemos ver cómo el SOC va disminuyendo,

hasta que en un instante alcanzó el 16 % y justo después se puso en 0.

Esto es debido a una de las celdas tenía una tensión inferior a 2.9 V (se puede ver en la gráfica de la derecha del SOC). En los ajustes de la App está configurada la  $V_{min} = 2.9$  V. Se puede observar que la tensión durante la descarga está en torno a 3.3 V, como podemos observar en la gráfica teórica de referencia: 3.18

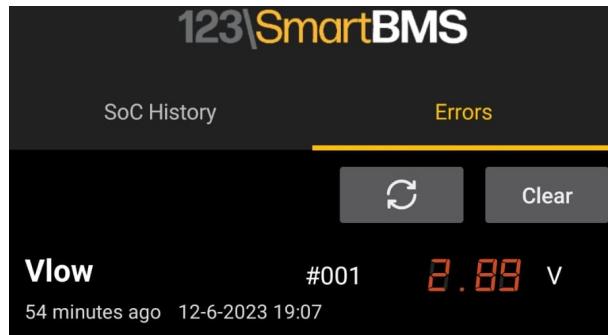


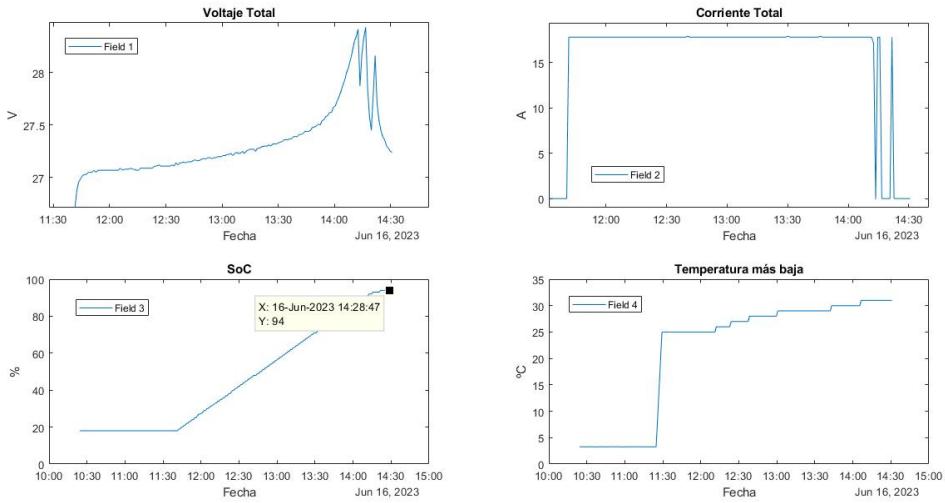
Figura 4.6: Alerta de  $V_{min}$  - 12/06/23

Cuando el BMS realiza la estimación del SOC, si una de las celdas alcanza  $V_{min}$  durante un instante cuando la descarga no está permitida, el SOC se establece en 0. [ 3.31].

Cuando una de las celdas tiene una tensión inferior a ésta, como ha ocurrido, el relé que controla la descarga se pondrá en OFF, impidiendo la descarga.

Por éste motivo, la corriente total pasa de ser negativa (descarga) a 0.

## Segunda Carga



**Figura 4.7: Segunda Carga - 16/06/23**

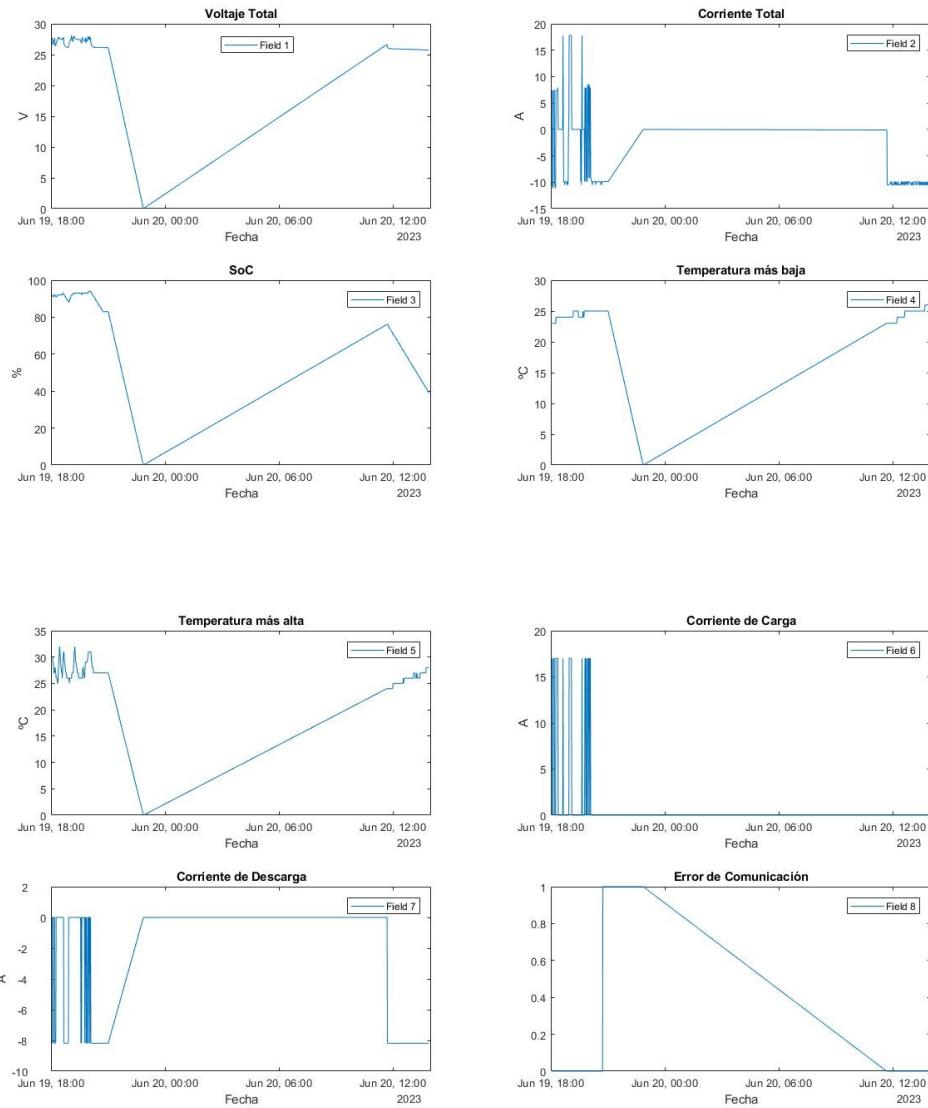
Cuando realizamos la segunda carga, partimos de un 18 % del SOC y logramos cargar hasta un 94 %, debido a que una de las celdas superó la tensión máxima configurada,  $V_{max} = 3.65v$ .

Cuando ésto ocurre, el sistema abre el relé que permite la carga. Después, la tensión de la celda que superó el límite disminuyó su valor y se volvió a reanudar la carga.

Posteriormente se repitió este suceso varias veces, lo podemos observar en los picos de la tensión y corriente total.

Básicamente, el relé que permite la carga estaba actuando de forma intermitente, debido a la condición de  $V_{max}$ .

## Segunda Descarga



**Figura 4.8: Segunda Descarga - 20/06/23**

En ésta segunda descarga, partimo del 94 %. Se ve un pico con una pendiente muy alta, ya que se desconectó el cable para la transmisión de datos y lo reanudamos el día 20 de junio.

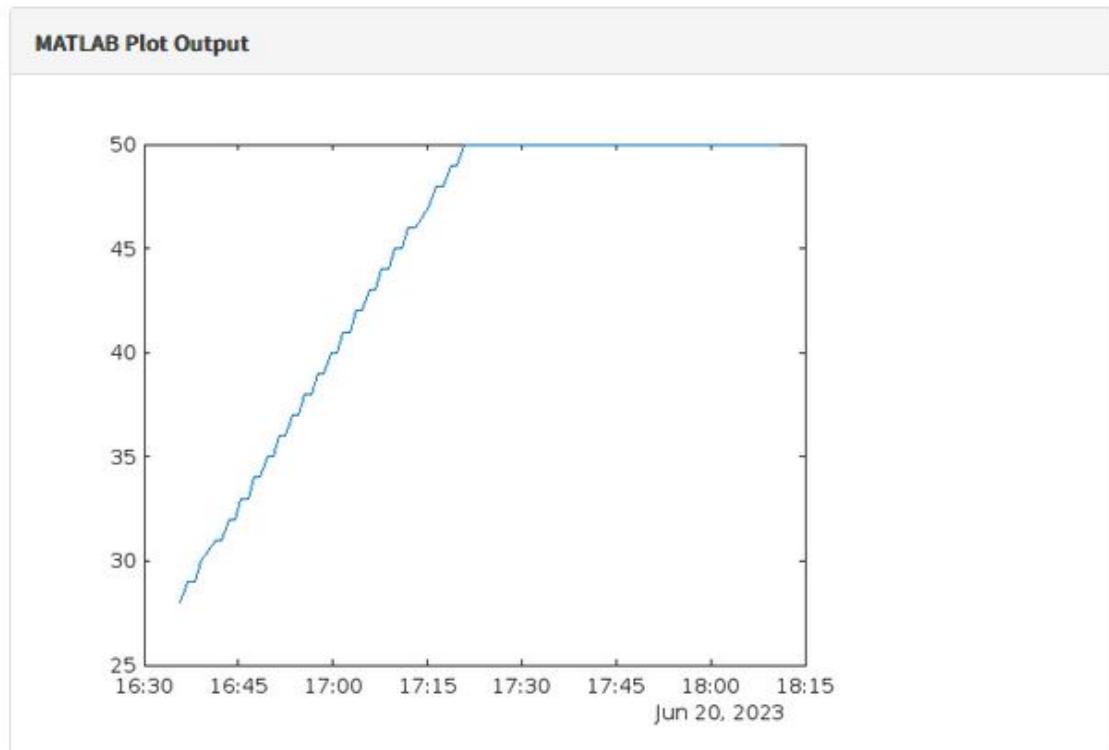
Además del programa creado, se han considerado otras opciones para la sección de verificaciones y pruebas.

a) ThingSpeak tiene la capacidad de hacer un análisis de forma online, ya que existe una función llamada *thingSpeakRead*. Esta función permite seleccionar los campos específicos de interés y especificar un rango temporal para obtener los datos. Proporciona flexibilidad y facilita la extracción de datos del canal de manera conveniente y eficiente.

```
readAPIKey = 'LC94F3LJ7ZJUN1IG';
readChannelID = 2023938;

[data2, time2] = thingSpeakRead(readChannelID, 'Field', fieldID1, 'NumPoints', 90,
'ReadKey', readAPIKey);

%% Visualize Data %%
plot(time2, data2)
```



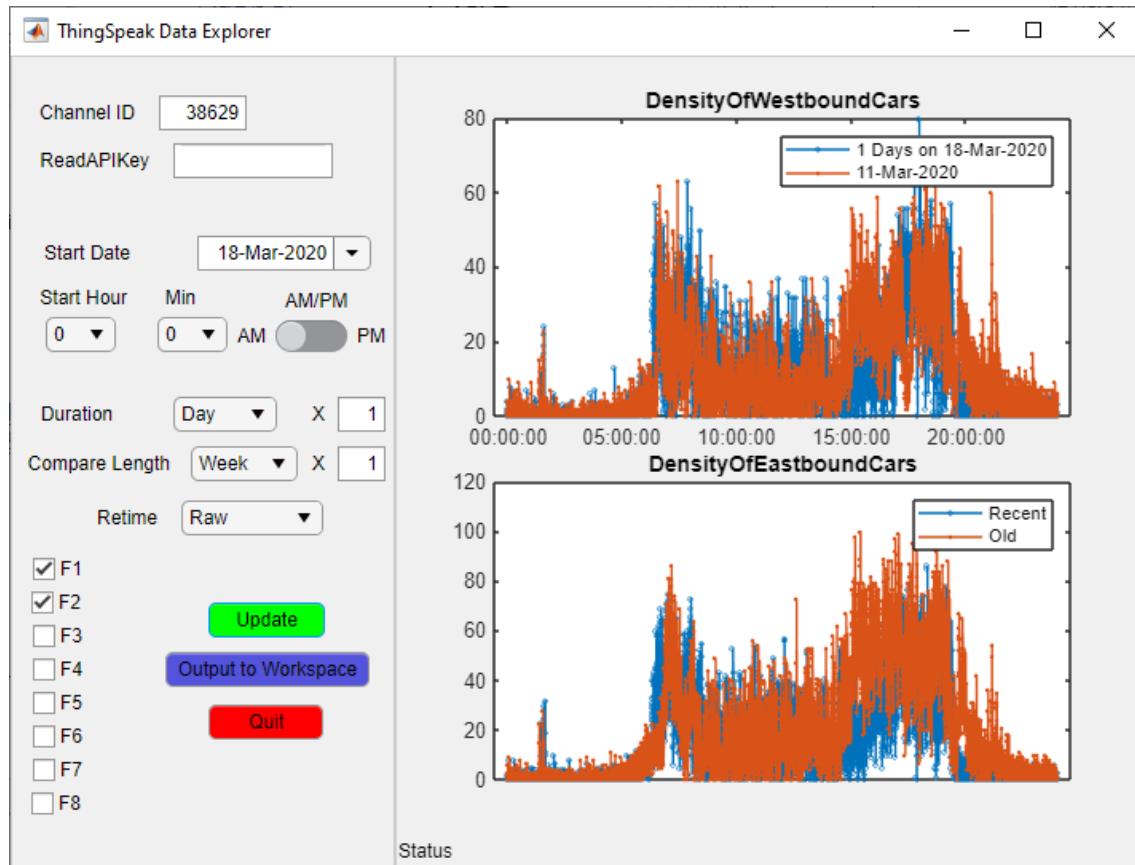
**Figura 4.9: Análisis de Matlab en ThingSpeak**

Como ejemplo, hemos representado 90 puntos (las últimas 90 muestras) del field 3, correspondiente al SOC.

Este método es muy ágil y permite analizar los datos en algún momento puntual. Por ello, hemos creado el programa en Matlab con la finalidad de analizar

los datos en más profundidad y con un mayor grado de precisión.

b) Matlab ha desarrollado una app llamada *IoT Data Explorer* para ThingSpeak que permite leer fácilmente los datos en sus canales de ThingSpeak y comparar datos de dos períodos de tiempo diferentes. En principio, ésta opción era la más sencilla y óptima de implementar en el sistema, pero tras instalarla y probarla daba errores que no pudimos solucionar.



**Figura 4.10: Interfaz de *IoT Data Explorer*, Matlab**

# **Parte III**

## **Parte tercera.**



# Conclusiones y líneas futuras

En éste proyecto, se ha abordado el diseño e implementación de un sistema de control de energía para un robot móvil con un manipulador industrial.

Originalmente, partíamos de una plataforma con una sencilla estructura sin completar, albergando ocho módulos de batería LiFePO4, donde el sistema de energía completo, incluyendo el resto de componentes que integra, debe suministrar una tensión de 24 V DC y 220 V AC, además de una potencia de salida de 1000 W. La disposición de estas celdas no era óptima y no se adaptaba a los requisitos de ubicación necesarios,

Para alcanzar el objetivo de éste proyecto, se han realizado tres grandes desarrollos.

Se ha realizado un desarrollo mecánico de la colocación de las baterías, junto con todos los elementos que se integran en el sistema de energía, con el objetivo de optimizar el espacio disponible, aportar características de seguridad y ganar la máxima eficiencia posible.

En lo que respecta al diseño eléctrico, se ha logrado la sinergia y compatibilidad perfecta entre todos los componentes elegidos, gracias a un diseño óptimo del circuito, a la vez que el cableado, para obtener los requisitos de energía del sistema. Podemos garantizar protección de la vida útil de las celdas mediante la incorporación de relés de alta potencia, los cuales permiten la desconexión de la energía cuando sea necesario, evitando daños y garantizando un funcionamiento seguro del sistema.

En lo que concierne al desarrollo de software, hemos logrado el control y adquisición de todos los datos necesarios, donde la Raspberry Pi es el componente central y fundamental en nuestro sistema de energía. gracias a su potente capacidad de procesamiento, versatilidad y facilidad de uso.

Para lograrlo, lo hemos realizado con una interfaz local en la plataforma, a través de ROS. Además, hemos implementado una interfaz online basada en la nube IoT de ThingSpeak.

### **Competencias adquiridas**

Debido a que se ha tratado de un proyecto de carácter mecatrónico, se han adquirido competencias de tipo eléctrico o electrónico, además de mecánico, en base al diseño del circuito y documentación de los distintos componentes, así como su instalación.

Hemos comprendido qué algoritmo de estimación del SOC sigue el BMS del sistema, el cual se base en el conteo de Coulomb, además de otras técnicas de estimación utilizadas.

En relación al software, con ROS, se ha aprendido a utilizar esta plataforma para el desarrollo de sistemas robóticos, además de haber adquirido experiencia en la configuración de nodos, la comunicación entre ellos y la implementación de algoritmos de control, para integrar los diferentes campos del sistema de control de energía.

En cuanto a ThingSpeak, se ha explorado el funcionamiento de esta plataforma de nube IoT, en base a la configuración de canales, gráficas y notificaciones, además de haber aprendido a utilizar sus herramientas proporcionadas para visualizar y analizar los datos recopilados, lo cual ha sido fundamental para el monitoreo y análisis del estado de carga, consumo de energía y potencia de carga del robot móvil.

### **Líneas futuras**

#### **1. Puerto serie USB**

La transmisión de datos desde las baterías hacia la Raspberry se realiza a través del puerto serie USB. Cuando procesamos los datos, automáticamente se envían a ThingSpeak y están listos para su visualización.

Si deseamos verlos de forma local, tenemos que iniciar el nodo ROS que publica dichos datos y mostrarlos por la pantalla integrada en la plataforma.

Sin embargo tenemos el problema de que en la conexión puerto serie, la información va de punto a punto y no puede enviarse a más de un destino, por lo tanto la el envío de datos a ThingSpeak falla y sólo podemos visualizar los datos de forma local, a no ser que se reinicie la conexión al puerto.

Mejora en desarrollo: crear un nodo ROS que se suscriba al topic de energía y suba los datos a ThingSpeak.

```
1 #!/usr/bin/python3
2 import sys
3 import asyncio
4 import rospy
5 from std_msgs.msg import Float32MultiArray
6 from sensor_msgs.msg import BatteryState
7 from bms_v3_pkg.msg import CustomBatteryState
8 import requests
9
10 def battery_state_callback(msg) :
11
12     battery_voltage = msg.voltage
13     battery_current = msg.current
14     battery_soc = msg.percentage
15     lowest_cell_voltage = msg.min_cell_voltage
16     highest_cell_voltage = msg.max_cell_voltage
17     lowest_cell_temperature = msg.min_cell_temperature
18     highest_cell_temperature = msg.max_cell_temperature
19
20     url = 'https://api.thingspeak.com/update'
21     api_key = KZWLB267YFY9TQWB
22
23     params = {
24         'field1': str(battery_voltage),
25         'field2': str(battery_current),
26         'field3': str(battery_soc),
27         'field4': str(lowest_cell_voltage),
28         'field5': str(highest_cell_voltage),
29         'field6': str(lowest_cell_temperature),
30         'field7': str(highest_cell_temperature)
31     }
32
33     response = requests.post(url + '?api_key=' + api_key, params=params)
34
35     if response.status_code == 200:
36         rospy.loginfo('Datos enviados a ThingSpeak con éxito')
37     else:
38         rospy.logwarn('Error al enviar los datos a ThingSpeak')
39
40 def main():
41     rospy.init_node('thingspeak_publisher')
42     rospy.Subscriber('battery_state', CustomBatteryState, battery_state_callback)
43     rospy.spin()
44
45 if __name__ == '__main__':
46     main()
```

Figura 4.11: Nodo suscriptor ROS

El nodo se suscribe al topic de energía con la función *callback* y publica los datos en ThingSpeak, haciendo uso de la librería *Requests*. Esta librería permite trabajar con el protocolo HTTP, de forma que puede publicar y solicitar contenido de datos en páginas web.

## 2. Router

Durante el desarrollo del proyecto en el laboratorio, la conexión de la Raspberry Pi a internet se realiza a través de un cable ethernet. Como mejora para que la conexión sea puramente funcional, se propone instalar un router en la plataforma, configurando una red local que utilice la red de eduroam universitaria. Así, el robot se podrá mover por todo el campus teniendo una conexión constante a internet.

# **Parte IV**

## **Apéndices**





## Apéndice A

### Vistas del desarrollo Mecánico

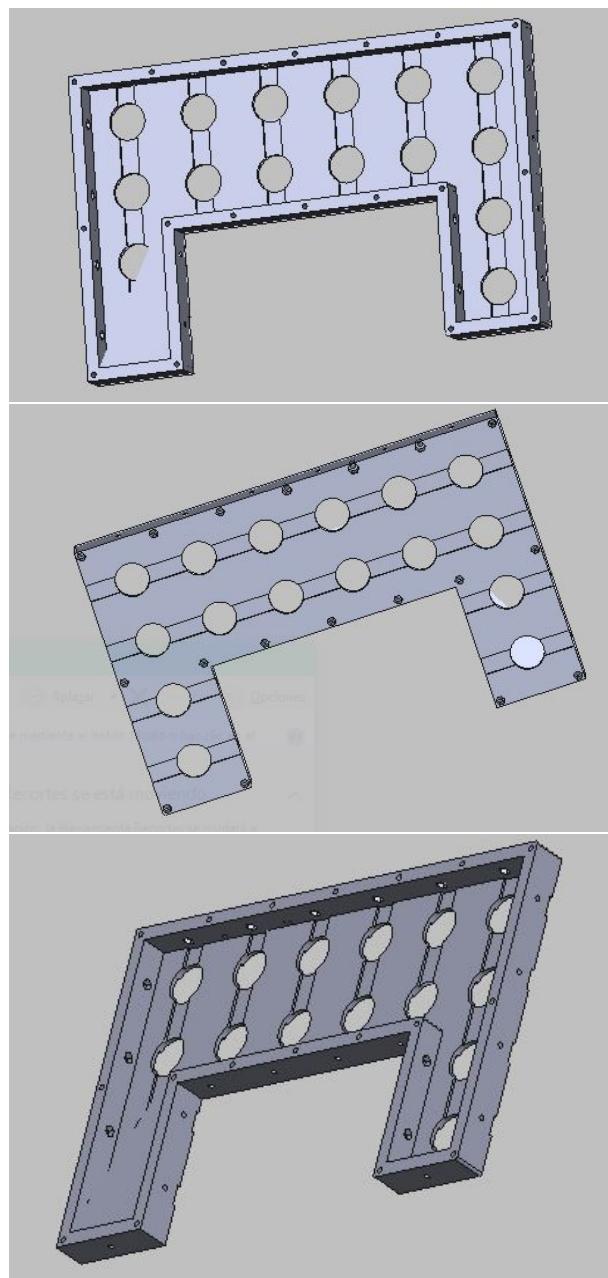
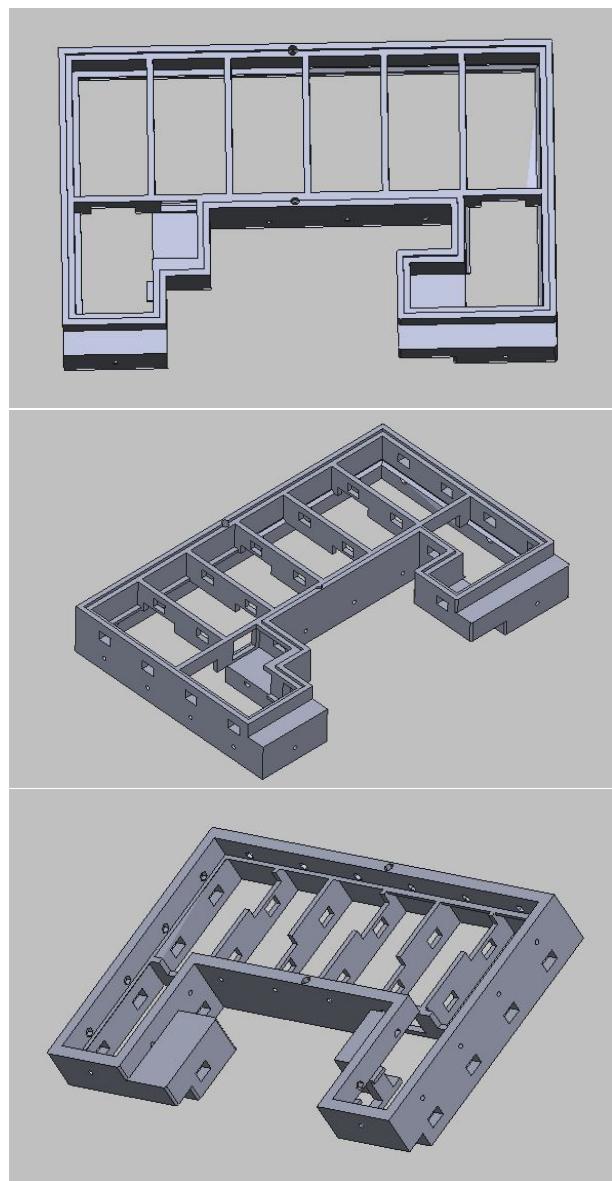
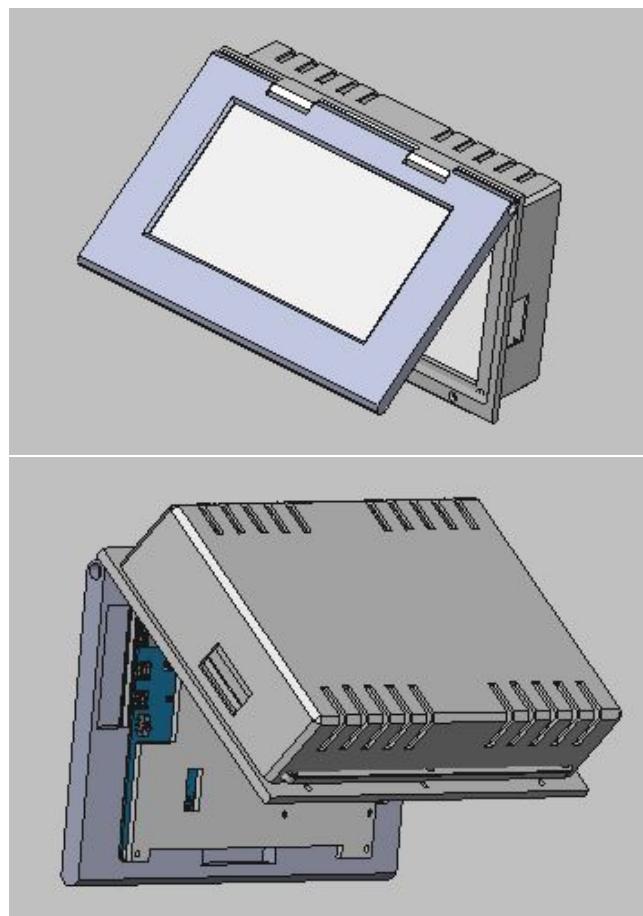


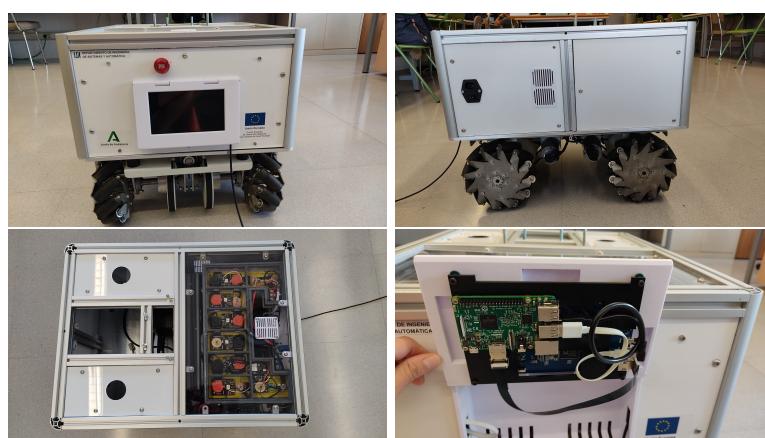
Figura A.1: Carcasa de la base



**Figura A.2: Carcasa de la parte superior**



**Figura A.3: Soporte para el dispositivo de visualización**



**Figura A.4: Imágenes del sistema completo implementado**

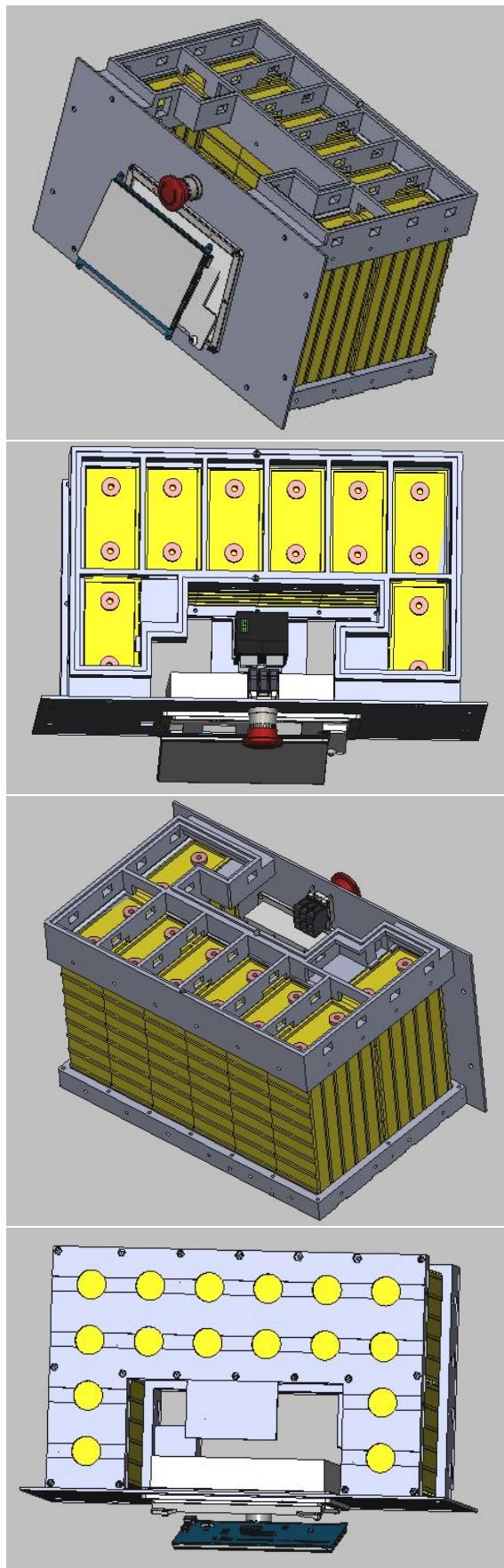
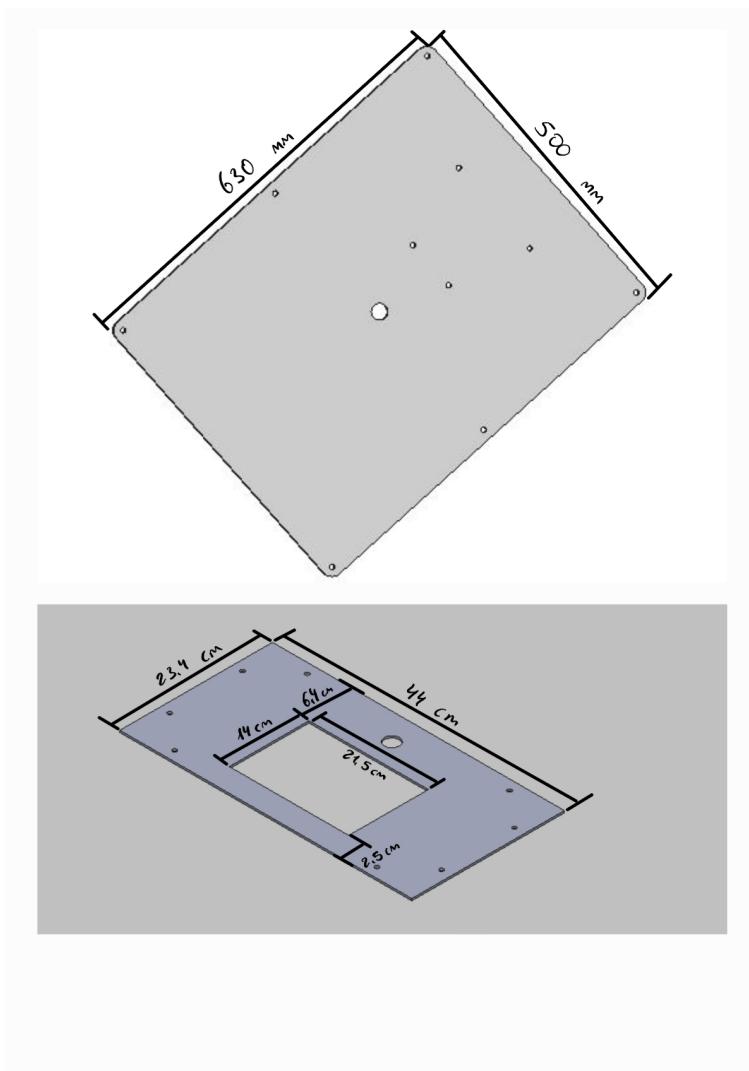
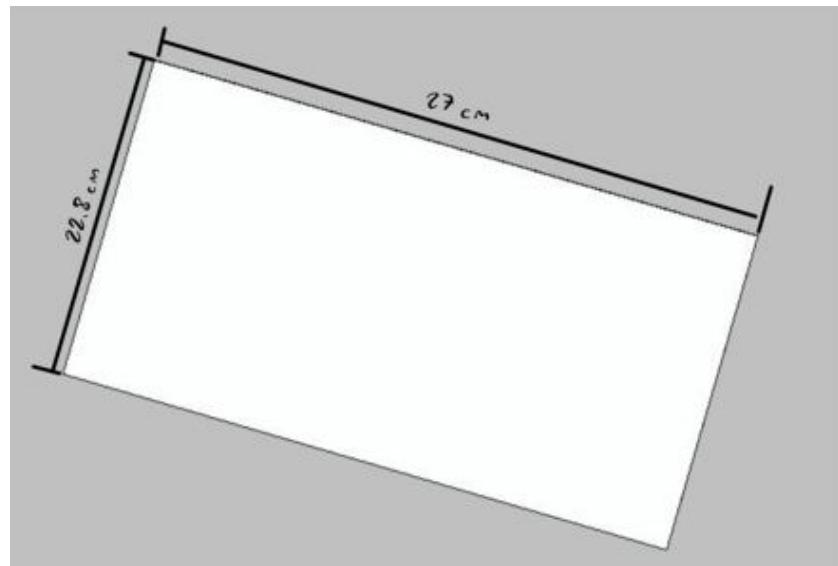


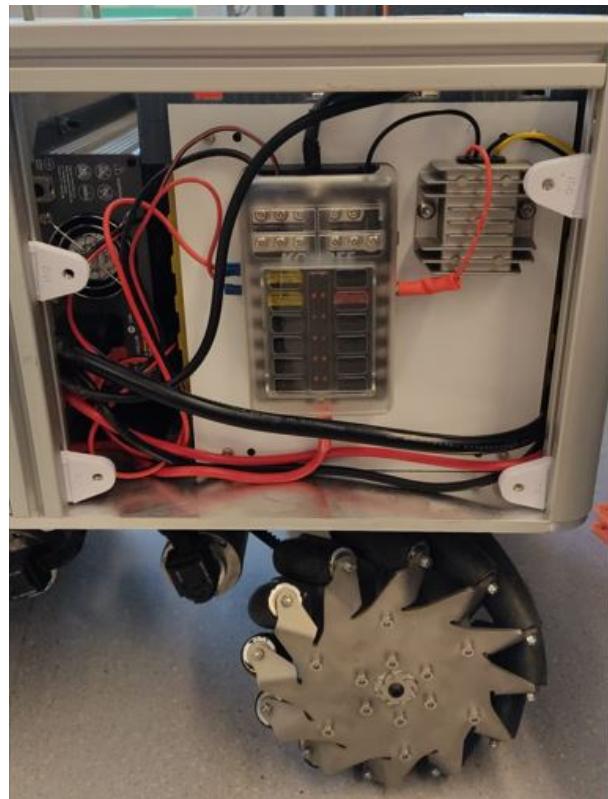
Figura A.5: Modelo del diseño mecánico completo



**Figura A.6:** Base de la plataforma y panel frontal



**Figura A.7: Panel lateral**



**Figura A.8: Panel interno**



# **Apéndice B**

## **Manual de uso**

Antes de utilizar el robot, se debe comprobar que el entorno de trabajo sea seguro y no suponga un riesgo para los humanos. Cerciorar que la superficie esté libre de obstáculos y no esté en mal estado, y halla ausencia de otros elementos que impidan el correcto desempeño.

### **Consideraciones previas**

En primer lugar, el pulsador que controla la salida del sistema de alimentación debe estar activado (circuito abierto), como medida de seguridad.

Antes de iniciar, se debe comprobar el estado de carga actual. Para ello, podemos hacerlo mediante distintas vías.

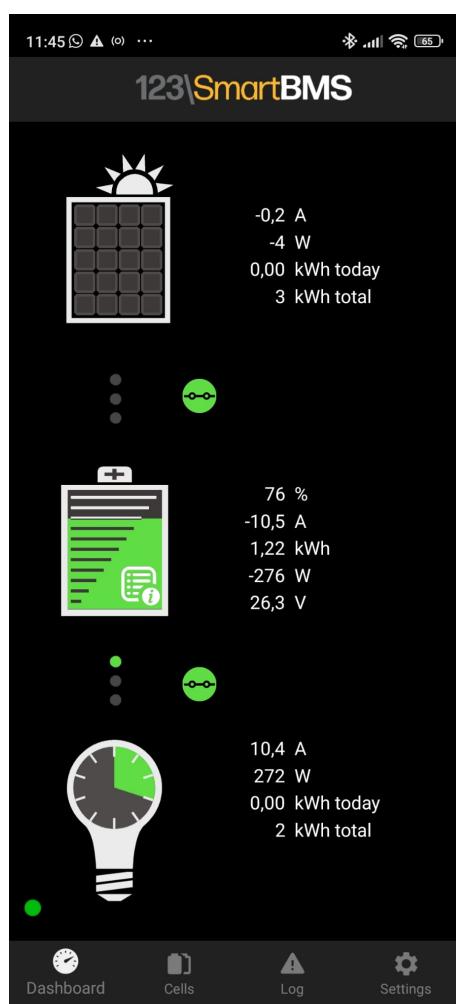
1. Con la aplicación móvil, podemos ver qué estado de carga tiene, la tensión total y la corriente total, la potencia, así como la energía almacenada en ese momento.
2. Con la nube IoT de ThingSpeak, podemos ver los datos que estén configurados para visualización. Tiene la ventaja de poder comprobar el estado del sistema de forma remota, a través de internet.
3. Con la interfaz local, a través de la pantalla incorporada en la plataforma. Tras iniciar el nodo publicador, los procesos robóticos que requieran conocer el estado de las baterías se suscriben al topic de energía.

Por motivo de seguridad, no cargar el sistema mientras el robot está en pleno funcionamiento.

## Inicio

1. Una vez comprobado el estado de carga, se debe estimar si el sistema de energía es capaz de cumplir los requisitos de potencia necesarios, así como una estimación del tiempo restante de la batería.
2. Después de la estimación, debemos desactivar el pulsador para cerrar el circuito y habilitar desde la app la activación del relé de salida, para alimentar el dispositivo deseado.

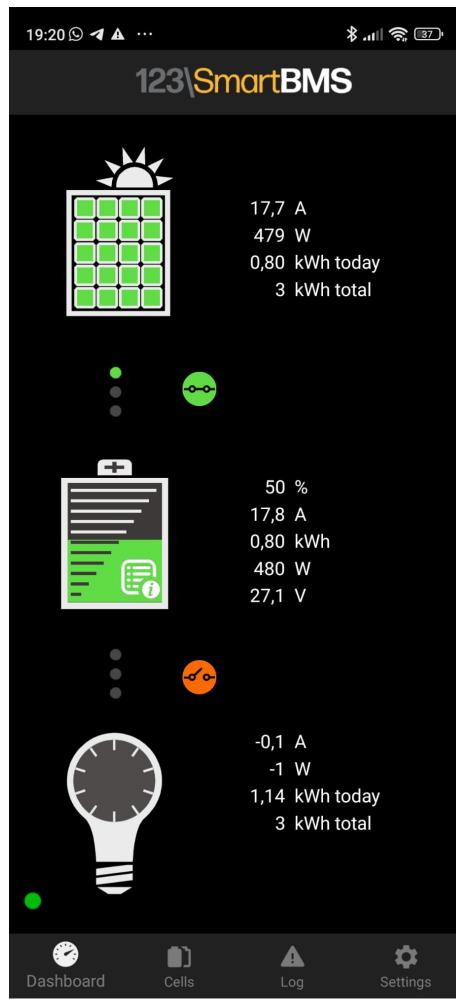
Tiempo de uso (descarga): la energía máxima de las baterías, al 100 %, es de 1440 wh (8 celdas x 60 Ah x 3 V). Si conectamos un dispositivo que consume 200 W, teóricamente la batería puede estar funcionando durante 5 horas, hasta que se descargue completamente.



**Figura B.1: Descarga de las baterías**

En este caso, el sistema tiene una energía almacenada de 1220 Wh y el dispositivo conectado consume 272 W. Por lo tanto, teóricamente tardaría en descargarse 4.48 horas ( $1220 \text{ Wh} / 272 \text{ W} = 4.48 \text{ horas}$ ).

Tiempo de carga: si se alimentan las baterías con 480 W de la red, tardaría 3 horas en cargarse completamente ( $1440 \text{ Wh} / 480 \text{ W} = 3 \text{ horas}$ ).



**Figura B.2: Carga de las baterías**

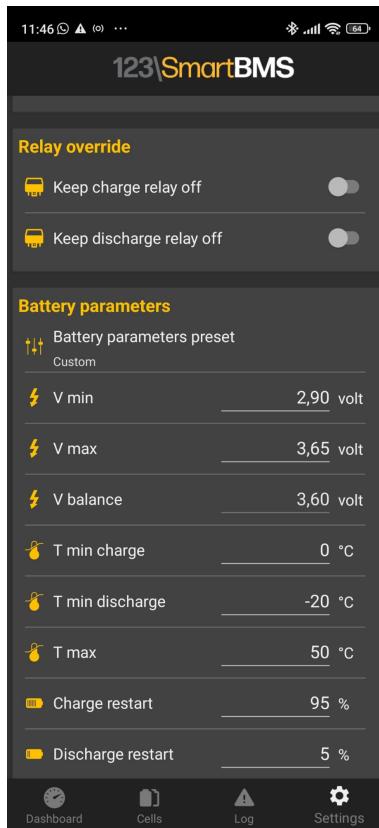
En este caso, el sistema tiene una energía almacenada de 800 Wh y está al 50 %. Para saber cuánto tiempo tarda en cargarse, sabemos que faltan aún 640 Wh ( $1440 \text{ Wh} - 800 \text{ Wh}$ ).

La red suministra 480 W para cargar la batería. Por lo tanto, tardaría teóricamente algo más de 1 hora en alcanzar el 100 % ( $640 \text{ Wh} / 480 \text{ W} = 1.33$

horas).

### Configuración de la App

La aplicación, además de mostrarnos un monitor de energía, tiene algunos ajustes para establecer valores de seguridad.



**Figura B.3: Ajustes de la app**

A continuación, mostraremos algunos parámetros destacados de los ajustes.

**V min:** Si una de las celdas cae por debajo de este umbral mínimo de voltaje, se encenderá el indicador de advertencia "Vl.<sup>en</sup>" en la pantalla de detalles de la batería. El relé "permitir descarga" para controlar dispositivos externos se apagará.

**V max:** Si una de las celdas supera este umbral máximo, se encenderá el indicador de advertencia "Vh.<sup>en</sup>" en la pantalla de detalles de la batería. El relé "permitir carga" para controlar dispositivos externos se apagará.

**V balance:** Este es el voltaje de equilibrado al que se desea que lleguen todas las celdas. Por encima de este voltaje, los módulos de las celdas comienzan a disipar 1 A para equilibrar las celdas.

T min: Si una de las celdas cae por debajo de este umbral mínimo de temperatura, se encenderá el indicador de advertencia "Tl.<sup>en</sup>" la pantalla de detalles de la batería. Ambos relés para controlar dispositivos externos se apagarán.

T max: Si una de las celdas supera este umbral máximo de temperatura, se encenderá el indicador de advertencia "Th.<sup>en</sup>" la pantalla de detalles de la batería. Ambos relés para controlar dispositivos externos se apagarán.

Charge restart: El relé de carga se vuelve a encender si la capacidad está por debajo de "reinicio de carga" programable y el BMS está en "modo normal". Esto es para evitar el cambio constante de relés.

Discharge restart: El relé de descarga se volverá a encender si la capacidad está por encima de "reinicio de descarga".





# Apéndice C

## Presupuesto

Presupuesto				
Componente	Coste unitario (€)	Unidades	Coste total (€)	
Set de baterías (WINSTON 24V, 1.44 kWh LiFeYPO4 Set With 60Ah Cells, BMS Mobile Monitoring)	988.88	1	988.88	
Placa Between Board (123ELECTRIC BMS123 Smart Gen3 - Single Cell Module)	17.63	4	70.52	
Cargador (POW24V20A-D1 Charger)	100	1	100	
Regulador (DC-DC 12V/24V a 5V, 5A, 25W. Módulo reductor de voltaje Buck)	12.89	1	12.89	
Inversor (Inverteere model TS-1000)	450	1	450	
Raspberry Pi 3B	55	1	55	
Pantalla (Raspberry Pi 7inch IPS 1024x600 Capacitive Touch Screen LCD Display HDMI Interface Supports)	70	1	70	
Cable USB ( 123ELECTRIC BMS123 Smart - USB cable)	24.66	1	24.66	
Relé (123SmartRelay - two 120A relays in one package)	77.06	1	77.06	
			1849.01	

Tabla C.1: Tabla de presupuesto

# Bibliografía

- [1] M. Ahulló. *DISEÑO Y PROGRAMACIÓN DE UN SISTEMA DE GESTIÓN DE BATERÍAS (BMS) PARA BATERÍAS LiFePO4*. PhD thesis, Universitat Politècnica de València (España), s.f.
- [2] Albertronic BV. 123electric bms123 smart - dual relay - datasheet. <https://shop.gwl.eu/en/By-Brand-Manufacturer/123electric/123ELECTRIC-BMS123-Smart-Dual-Relay.html?cur=1&listtype=search&searchparam=relay>.
- [3] Jae Jin J. Daehyun K., Keunhwi K. Second-order discrete-time sliding mode observer for state of charge determination based on a dynamic resistance li-ion battery model. [https://www.researchgate.net/publication/276036436\\_Second-Order\\_Discrete-Time\\_Sliding\\_Mode\\_Observer\\_for\\_State\\_of\\_Charge\\_Determination\\_Based\\_on\\_a\\_Dynamic\\_Resistance\\_Li-Ion\\_Battery\\_Model](https://www.researchgate.net/publication/276036436_Second-Order_Discrete-Time_Sliding_Mode_Observer_for_State_of_Charge_Determination_Based_on_a_Dynamic_Resistance_Li-Ion_Battery_Model), 2013.
- [4] 123 electric. 123smartbms to usb - manual.
- [5] FRANKA EMIKA. Panda - datasheet, 2019.
- [6] Davide Faconti. Plotjuggler. <https://github.com/facontidavide/PlotJuggler>, 2020.
- [7] J. Gutiérrez. *Introducción a ROS en Raspberry Pi*. PhD thesis, Universitat Oberta de Catalunya (España), Junio 2017.
- [8] GWL. Pow24v20a-d1 charger - datasheet.
- [9] GWL. Thundersky winston tswb-lyp60aha lifeypo4 - datasheet.
- [10] LARGE.net. Lista de los principales fabricantes mundiales de bms. <https://es.large.net/news/8vu43m9.html>, 2018.

- [11] A. Balasingam B. Pattipati K Movassagh, K. Raihan. Critical look at coulomb counting approach for state of charge estimation in batteries. <https://doi.org/10.3390/en14144074>, 2021.
- [12] Raspberry Pi. Raspberry pi 3b - datasheet.
- [13] C. Ramos. *Análisis de un convertidor DC/DC destinado al almacenamiento híbrido de energía*. PhD thesis, Universidad Politécnica de Madrid (España), Julio 2017.
- [14] ROS. Ubuntu install of ros noetic. <http://wiki.ros.org/noetic/Installation/Ubuntu>, 2023.
- [15] A. Serrano. *CONTROL DE UNA PLATAFORMA OMNIDIRECCIONAL PARA UN MANIPULADOR MOVIL*. PhD thesis, Universidad de Málaga (España), Junio 2021.
- [16] SLAMTEC. Slamtec mapper - datasheet, 2022.
- [17] Smart Energy. Comparación típica de la curva de descarga <http://www.smartnewenergy.com/info/sth-you-need-to-know-about-lifepo4-batteries-23935605.html>, 2018.
- [18] Xiaojunt Tan. *Battery management system and its applications*. John Wiley Sons, Incorporated, 2023.
- [19] Libre University. Thingspeak: Internet of things. [https://es.libre.university/lesson/HkS\\_Bx8kZ/5.%20ThingSpeak:%20Internet%20of%20Things](https://es.libre.university/lesson/HkS_Bx8kZ/5.%20ThingSpeak:%20Internet%20of%20Things), 2017.
- [20] Abbas ; Shateri Neda ; Auger Daniel Valencia, Nicolas ; Foto-uh. Development of a Hybrid Adaptive Neuro-fuzzy Inference System with Coulomb-Counting State-of-Charge Estimator for Lithium–Sulphur Battery. <https://link.springer.com/article/10.1007/s40815-022-01403-y>, 2023.
- [21] Ping He Wang, Yonggang and Haoshen Zhou. Olivine LiFePO<sub>4</sub>: Development and Future. <https://onlinelibrary.wiley.com/doi/full/10.1002/anie.200802539>, 2011.
- [22] Mean Well. TS-700/1000 Inverter Instruction Manual. <https://www.meanwell.com/Upload/PDF/TS-1000/TS-1000-SPEC.PDF>, 2010.

