



UNIVERSIDAD
DE MÁLAGA



ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

**Funciones básicas de navegación ROS para un manipulador
móvil omnidireccional**

Grado en Ingeniería Robótica, Electrónica y Mecatrónica.

Autor: Francisco de Paula Carbonero Navarro

Tutor: Jesús Manuel Gómez de Gabriel

MÁLAGA, Septiembre de 2.023

DECLARACIÓN DE ORIGINALIDAD DEL TRABAJO FIN DE GRADO

D./ Dña.: Francisco de Paula Carbonero Navarro

DNI/Pasaporte: 46273634A Correo electrónico: f.carbonero.n@uma.com

Titulación: Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Título del Proyecto/Trabajo: Funciones básicas de navegación ROS para un manipulador móvil omnidireccional

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin.

Así mismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a 4 de septiembre de 2023

Fdo.:

RESUMEN

Esta memoria presenta y desarrolla una plataforma robótica omnidireccional. Es una continuación de un trabajo previo realizado por compañeros. El objetivo principal es que el usuario sea capaz de controlar la plataforma mediante un joystick. Para ello, se han usado unas controladoras RoboClaw, que permiten enviar comandos a las ruedas. También se ha propuesto utilizar un sensor LiDAR para proporcionar retroalimentación a la plataforma, de manera que pueda esquivar obstáculos de una forma básica. La herramienta utilizada para integrar todos estos dispositivos y que el conjunto funcione ha sido ROS, ampliamente refrendado en robótica.

El usuario podrá dirigir al robot, ajustar parámetros como su velocidad, el rango de detección del LiDAR y determinar la acción a realizar cuando se detecte un obstáculo.

También se ha continuado con el diseño y construcción del chasis de la plataforma, de tal modo que sea completamente funcional.

A largo plazo, el objetivo final es dotar a la plataforma de las funciones necesarias para poder ayudar a personas dependientes. A corto plazo, servir como plataforma de investigación facilitando numerosos experimentos en diversos campos.

PALABRAS CLAVE

Cinemática – Control – Joystick – LiDAR – Mecanum – Omnidireccional – Roboclaw – Robot - ROS

ABSTRACT

This report introduces and elaborates on the development of an omnidirectional robotic platform, which builds upon the work previously undertaken by colleagues. The primary objective is to enable user control of the platform via joystick. To achieve this, RoboClaw controllers have been utilized to send commands to the wheels. Additionally, the incorporation of a LiDAR sensor has been proposed to provide feedback to the platform, enabling basic obstacle avoidance capabilities. ROS, widely recognized in the field of robotics, has been the chosen tool for integrating all these devices and ensuring their seamless functionality.

Users will have the ability to steer the robot, fine-tune parameters such as its speed, the LiDAR's detection range, and determine the robot's response when an obstacle is detected.

Furthermore, the design and construction of the platform's chassis have progressed to be fully functional.

In the long term, the ultimate goal is to equip the platform with the necessary features to assist individuals with dependency. In the short term, it will serve as a research platform facilitating numerous experiments in various fields.

KEYWORDS

Control – Joystick – Kinematics – LiDAR – Mecanum – Omnidirectional – Roboclaw – Robot - ROS

AGRADECIMIENTOS

Quiero agradecer a mi tutor D. Jesús Manuel Gómez por toda la ayuda, paciencia y conocimientos que me ha aportado durante la realización de este proyecto. Agradecer también a mis padres especialmente y a toda mi familia por su apoyo incondicional. Y por último a mis amigos que me han animado durante todo este proceso y me han enseñado a dar lo mejor de mí.

ÍNDICE GENERAL

	Página
Índice de Figuras	xi
1 Introducción	1
1.1 Descripción del proyecto	1
1.2 Motivación y Justificación	2
1.3 Objetivos	2
1.4 Estado del arte	2
1.5 Estructura de la Memoria.....	7
2 Especificaciones, diseño y precedentes	9
2.1 Requerimientos	9
2.2 Diseño	10
2.3 Precedentes.....	12
3 Componentes de la plataforma robótica	15
3.1 Componentes eléctricos	16
3.2 Componentes mecánicos	19
3.3 Descripción del hardware externo utilizado	21
3.4 Descripción del Software.....	25
4 Desarrollo del trabajo	29
4.1 Primeros pasos y problemas	30
4.2 Modificación y creación de los nodos.....	30
4.3 Desarrollo mecánico y de hardware.....	43
5 Resultados y conclusiones	47
6 Líneas de Trabajo Futuras	55
Referencias	57

ÍNDICE DE FIGURAS

FIGURA	Página
1.1 Garmi, el robot de apoyo	3
1.2 Garmi, el robot de apoyo (2)	4
1.3 Nadine, el robot con comportamientos sociales.....	4
1.4 Gráfico del fenómeno <i>Uncanny Valley</i>	5
1.5 Foto del robot NAO.....	7
2.1 Diagrama de funcionamiento del proceso	10
2.2 Modelo básico del chasis.....	11
2.3 Modelo en Solidworks.....	12
2.4 Conexión de los encoders.....	12
2.5 Montaje de la plataforma	12
2.6 Esquema eléctrico de la plataforma robótica.....	13
2.7 Ubicación actual de las baterías.....	14
2.8 Imágenes de la pantalla exterior.....	14
3.1 Batería LiFePO4 modelo Winston LFP060AHA.....	16
3.2 Especificaciones técnicas de la batería LiFePO4	17
3.3 123SmartBMS gen3	17
3.4 Perfil estructural.	19
3.5 Alucobond	19
3.6 Rueda Mecanum.....	20
3.7 Controladora 2x30A de la marca RoboClaw.....	21
3.8 YDLIDARX4.....	22
3.9 Raspberry Pi 4 modelo B	23
3.10 Slimbook Pro X 14.....	23
3.11 Mando Dualshock PS4.....	24
3.12 Capturas de la aplicación Basicmicro Motion Studio	27
3.13 Capturas de la aplicación 123SmartBMS.....	28
4.1 Captura de las carpetas del workspace.....	31

4.2	Captura de las carpetas de src	31
4.3	Diagrama de archivos dentro de joy_ros.....	33
4.4	Captura de joy.config.yaml.	34
4.5	Captura de teleop.launch	34
4.6	Captura del programa Jstest-gtk.	35
4.7	Diagrama de archivos dentro de ydlidar.....	36
4.8	Diagrama de archivos dentro de roboclaw_ros.....	38
4.9	Grafo de funcionamiento de roboclaw_ros	39
4.10	Captura de código del archivo roboclaw.launch.....	40
4.11	Cinemática de la plataforma robótica	40
4.12	Captura de código del archivo roboclaw_node.py.....	41
4.13	Diagrama de funcionamiento de lidar_avoider.....	42
4.14	Captura de código del archivo lidar_avoider.launch	42
4.15	Imágenes del brazo robótico colocado en la plataforma.....	43
4.16	Ubicación del hub desechado	44
4.17	Diagrama de conexión de las RoboClaw	44
4.18	Diagrama de conexión de la Logic Battery de las RoboClaw	45
4.19	Imagen real de una controladora RoboClaw	45
4.20	Ubicación de las controladoras en la plataforma	46
4.21	Imagen de los enchufes interiores.....	46
4.22	Panel lateral con el enchufe del cargador y las rejillas de ventilación	46
5.1	Nodos activos con la plataforma en funcionamiento	48
5.2	Estado de los dispositivos conectados.....	49
5.3	Grafo de funcionamiento de roboclaw_ros	50
5.4	Medidas del LiDAR en RViz	51
5.5	Mapeo con hector_slam en RViz.....	51
5.6	Salida de /cmd_vel con rqt_plot.....	52
5.7	Robot funcional.....	53



INTRODUCCIÓN

1.1 Descripción del proyecto

El envejecimiento de la población es uno de los mayores problemas demográficos al que se enfrentan los países del primer mundo. Con el propósito de abordar los desafíos que surgen de esta situación, aparece la necesidad de explorar soluciones innovadoras que mejoren la calidad de vida de las personas mayores y brinden el apoyo necesario para una vida independiente y saludable. Debido a esto, cada vez más fondos y proyectos se están destinando a crear soluciones que permitan a personas de la tercera edad vivir su vida con calidad e independencia.

Para ello, se pretende crear un robot de asistencia para personas en situación de dependencia. La idea es que el robot cuente con un brazo robótico y que sea capaz de realizar diversas tareas de apoyo, como coger objetos y llamar a emergencias o a algún teléfono de contacto. El objetivo no es otro que poder aportar seguridad, libertad y privacidad.

Este proyecto corresponde al Trabajo de Fin de Grado (TFG) del Grado en Ingeniería Robótica, Electrónica y Mecatrónica (GIERM) de la Universidad de Málaga. El proyecto a realizar consiste en dotar a una plataforma robótica de movilidad, permitiendo al usuario manejarla mediante un joystick.

Este trabajo forma parte de un camino para poder conseguir el objetivo final comentado anteriormente, además la plataforma robótica también sería un elemento de investigación sobre el que poder desarrollar y probar nuevas ideas.

1.2 Motivación y Justificación

La realización de este proyecto se origina en la necesidad de una plataforma de investigación funcional, que permita avanzar en la búsqueda de la innovación y poder servir como base para que compañeros puedan desarrollar sus ideas e inquietudes, y que pueda llegar a ser un robot de apoyo.

Por supuesto, nace del deseo personal también, el deseo de poner en práctica los conocimientos aprendidos en el grado y adquirir nuevas competencias, de realizar una primera toma de contacto con un tema tan interesante como es la robótica auxiliar, y de enfrentar un reto real usando múltiples herramientas.

1.3 Objetivos

Este proyecto pretende centrarse en el diseño, movilidad y control de la plataforma. Por lo que los objetivos que definen este trabajo son:

1. Dotar a la plataforma robótica de movimiento: Conseguir que la plataforma robótica pueda desplazarse por el entorno.
2. Conseguir un control preciso usando el joystick: Al ser un robot pensado principalmente para entornos domésticos, es necesario que, dado su gran tamaño, sea capaz de moverse precisamente por el espacio sin peligro para el usuario, el entorno, y él mismo.
3. Utilizar el LiDAR para realizar una simple evasión de obstáculos: Asegurarnos de que tenemos un sensor láser funcional y el robot es capaz de leer estos datos. Se comprobará mediante un programa sencillo.
4. Diseñar y crear el chasis de la plataforma: Es necesario, componer el armazón que protegerá el interior de la plataforma.

Como podemos observar, el objetivo general es dar el primer paso para que el robot puede contar con un sistema de navegación tanto autónoma como teleoperada y permitir un movimiento total para desarrollar otras funcionalidades.

1.4 Estado del arte

La siguiente revisión de la actualidad señala los avances más significativos en el desarrollo de robots de apoyo para personas mayores. A medida que la tecnología evoluciona y las necesidades cambiantes de la población mayor se vuelven más pronunciadas, este análisis subraya cómo los

robots de apoyo están contribuyendo a mejorar la calidad de vida y promover la independencia en este grupo demográfico.

En primer lugar, presentamos a "Garmi", un robot humanoide que se mueve sobre una plataforma con ruedas y que cuenta con una pantalla negra en la que se muestran dos círculos azules que simulan ser sus ojos. Más allá de poder realizar diagnósticos en pacientes, también puede brindar atención y tratamiento.



Figura 1.1: Garmi, el robot de apoyo.

Para su construcción, "Garmi" requirió de aproximadamente una docena de científicos, y se desarrolló en Alemania. El país más poblado de Europa es una de las sociedades del mundo que envejece más rápidamente. Los investigadores consideraron importante concebir robots que puedan hacerse cargo de algunas de las tareas de los enfermeros, cuidadores y médicos. El proyecto no se queda solo allí, sino que el robot podría ofrecer un servicio personalizado en una casa o en un hogar de ancianos, sirviendo las comidas, abriendo botellas de agua y pidiendo ayuda en caso de una caída, pero también organizando videollamadas con familiares y amigos.



Figura 1.2: Garmi, el robot de apoyo (2).

También, hay robots mucho más especializados en comportamientos sociales como es el caso de "Nadine". "Nadine" se ha creado a imagen y semejanza de su creadora Nadia Thalmann, que lleva tres décadas investigando a los robots. Además de su aspecto humano, también destacan sus capacidades: posee empatía y se comporta según las emociones que lee en el lenguaje verbal y no verbal de las personas. Es capaz de estar seria, alegre o triste según la necesidad. De momento no se comercializa, pero su creadora asegura que robots como "Nadine" podrían ser utilizados algún día como acompañantes para personas con demencia. Actualmente, "Nadine" trabaja como recepcionista en la universidad Nanyang Technological de Singapur, donde se creó.



Figura 1.3: Nadine, el robot con comportamientos sociales.

Sin embargo, este robot por ejemplo, puede caer dentro de un fenómeno interesante para este proyecto como puede ser *Uncanny Valley*. Se traduce al español como “valle inquietante” y sirve para describir al fenómeno en el cual una persona siente rechazo por aquellos robots con una

apariencia muy similar a los humanos, pero que no llega a ser lo suficientemente convincentes como para parecerse a una persona real.

Se trata de un concepto creado por el profesor de robótica Masahiro Mori en 1970, y desde entonces ha sido muy usado en la industria. Es, además, algo que muchas personas padecen a la hora de observar, sobre todo, a robots humanoides. En la siguiente gráfica podemos observar la gráfica que realizó Mori sobre este término:

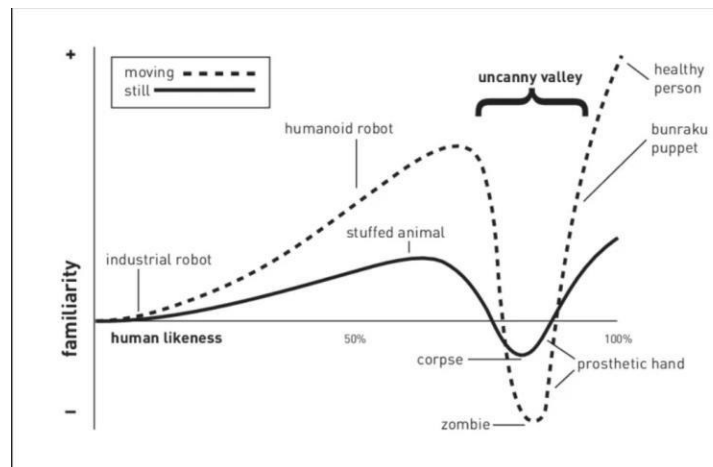


Figura 1.4: Gráfico del fenómeno *Uncanny Valley*.

Como podemos observar, hay un umbral en el que a partir de esa semejanza a un humano, además de la movilidad que presente el elemento, se genera un rechazo hacia el elemento por parte de las personas.

Realmente, lo que produce incomodidad a las personas que observan los robots son esas pequeñas imperfecciones que podemos encontrar en estos dispositivos y que dejan claro que, pese a que intenta imitar a un ser humano, no es realmente un humano. Un robot, por ejemplo, puede tener expresiones mucho más mecánicas que la de un ser humano, una piel visiblemente más artificial, o unos ojos cuyas pupilas son considerablemente más grandes que la de una persona real.

Este fenómeno, por tanto, no hace más que ofrecer una sensación negativa en cuanto a la robótica por parte de los usuarios particulares, por lo que es importante tenerlo en cuenta a la hora de desarrollar un robot.

El objetivo es evitar un rechazo social, de forma que las personas no se sientan incómodas al interactuar con un dispositivo que aporte y sea útil en su vida.

Por último vamos a ver el robot NAO V6, uno de los robots humanoides más avanzados del

mundo. Es un robot autónomo que destaca por incorporar tecnología avanzada en ingeniería mecánica y software. Con ella es capaz de reconocer el rostro de las personas y detectar sus emociones, a la vez que escuchar, hablar e interactuar con ellas.

NAO puede ser programado para realizar diferentes funciones, aunque principalmente ha sido diseñado para fomentar el desarrollo y la interacción entre personas y máquinas. En este caso, no nos vamos a centrar en personas mayores, si no que vamos a ver funcionalidades también aplicada a niños pequeños. El robot NAO está indicado para niños a partir de 7 años, lo que comprende desde la Educación infantil, básica, media y educación superior. Una de las principales características de NAO es que detecta el estado de ánimo de una persona y expresa emociones como tristeza, alegría, sorpresa y miedo.

NAO ofrece múltiples beneficios como pueden ser el incremento de la participación del estudiante que repercute directamente en alcanzar los objetivos académicos. Les permite unir en los proyectos la práctica con la teoría. Ayuda a fomentar el trabajo en equipo, la resolución de problemas y el desarrollo de habilidades. Resulta especialmente útil para proyectos de ciencias, matemáticas, tecnología e ingeniería robótica. También dispone de una plataforma de pruebas perfecta para modelos teóricos y conceptuales, además de un software con programación en varios idiomas que favorece la interacción entre humano y robot.

Este robot resulta especialmente interesante también porque está orientado a trabajar con niños dentro del espectro autista y con necesidades especiales, favoreciendo la interacción con estos de manera visual, verbal y táctil. Además, NAO reconoce a las personas con las que habla y puede recordar sus nombres transcurridos unos días.

En el ámbito del hogar puede convertirse en un componente más, capaz de contar cuentos e historias, recordar hábitos saludables o simplemente vigilar la casa.

Para programarlo, tiene varias opciones, uso de lenguajes más avanzados como *C++* o *Python*, y también con un sistemas intuitivos para gente sin conocimientos de programación.



Figura 1.5: Foto del robot NAO

Como podemos observar, especialistas prestigiosos de todo el mundo están poniendo su conocimiento al servicio de la sociedad, para poder superar retos del presente y del futuro y plantear soluciones que sean capaces de mejorar la vida de los individuos de nuestra sociedad.

1.5 Estructura de la Memoria

La memoria se encuentra estructurada de la siguiente manera:

- En el capítulo "2: ESPECIFICACIONES, DISEÑO Y PRECEDENTES" se describen las especificaciones requeridas y el diseño planteado para poder alcanzarlas. Además, se hará una breve descripción del trabajo previo de otros compañeros.
- En el capítulo "3: COMPONENTES DE LA PLATAFORMA ROBÓTICA" se presentan los distintos componentes claves que han formado parte de este proyecto"

1. INTRODUCCIÓN

- En el capítulo "4 :DESARROLLO DEL TRABAJO" se detalla el avance del proyecto tanto a nivel de software como a nivel mecánico y electrónico.
- En el capítulo "5: RESULTADOS Y CONCLUSIONES" se pueden ver capturas del programa en funcionamiento y gráficas.
- En el capítulo "6: LÍNEAS DE TRABAJO FUTURAS" se detallan las posibles líneas de desarrollo futuras y posibles soluciones a problemas actuales.



ESPECIFICACIONES, DISEÑO Y PRECEDENTES

Los objetivos globales del proyecto se pueden consultar en la sección 1.3. Sin embargo, dentro de este marco, se han propuesto unas especificaciones concretas y cuantificables que permiten alcanzar esos objetivos abstractos de momento. En esta sección quedarán definidos los requerimientos para superar el proyecto y las decisiones de diseño tomadas para ello. También se detallará el trabajo realizado por otros compañeros que ha permitido el desarrollo de este proyecto.

2.1 Requerimientos

Los requerimientos para definir el éxito y la validez del proyecto serán los siguientes:

1. El robot es completamente funcional, es capaz de moverse por la habitación de manera precisa mediante control por joystick.
2. El robot es capaz de detenerse si detecta un obstáculo frontal entre -30 y 30 grados. Se ha elegido este rango debido a que el robot cuenta con un brazo robótico y el LiDAR está a un lado de este. Con este rango se evita su interferencia y su posible detección como obstáculo.
3. La velocidad del robot permite su uso doméstico sin comprometer la seguridad de las personas. Se establece una velocidad lineal máxima de 1 m/s.
4. El chasis del robot queda terminado aun pudiendo recibir modificaciones futuras.
5. El proyecto queda bien documentado para que otros compañeros puedan continuar hacia el objetivo final.

2.2 Diseño

Lo principal a la hora de realizar un diseño es tener claro el funcionamiento a implementar. Se ha realizado un simple diagrama de flujo para ilustrar el proceso a alcanzar.

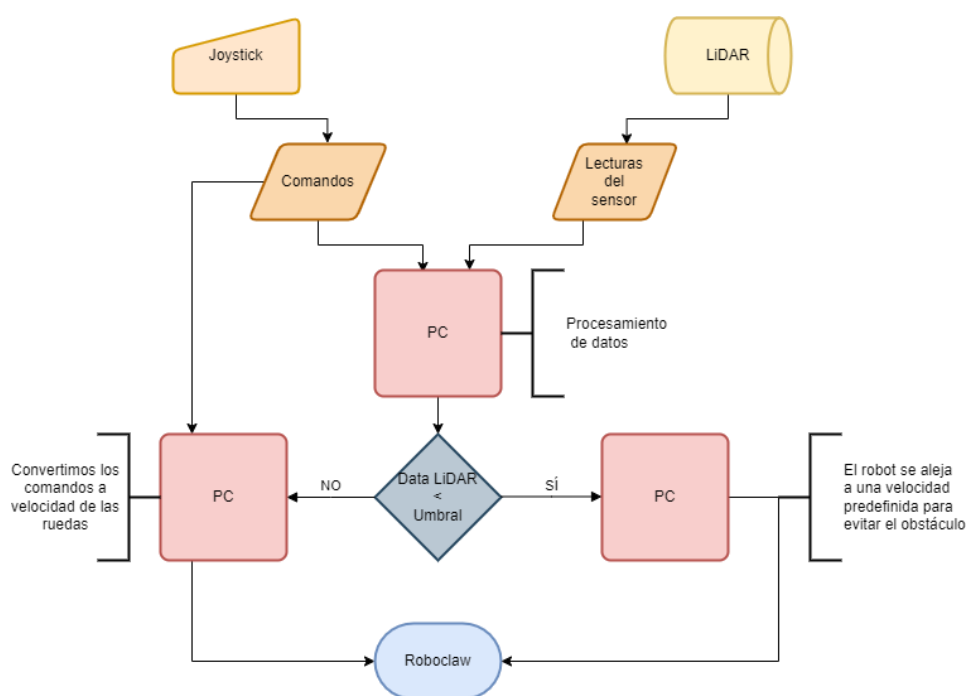


Figura 2.1: Diagrama de funcionamiento del proceso.

El objetivo es que el usuario introduzca comandos desde el joystick, mientras el LiDAR está en funcionamiento, que el PC procese esta información y dependiendo de si el robot se encuentra demasiado cerca de un objeto o no, use la información.

La idea es que tanto los controladores RoboClaw, como el LiDAR y el joystick, estén alimentados por el PC, mediante cables USB.

Por otra parte, se diseñó también el chasis. La primera idea para cumplir las especificaciones es un diseño muy básico del chasis, de forma compartimentada, añadiendo los paneles y el brazo robótico posteriormente:

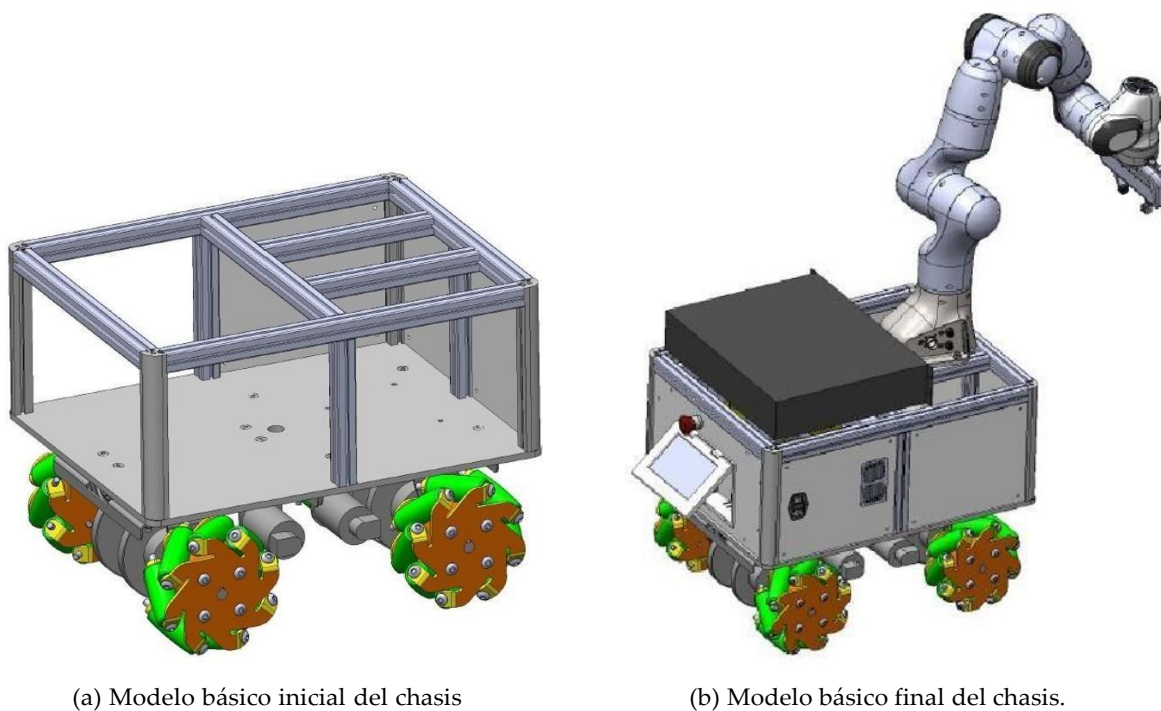


Figura 2.2: Modelo básico del chasis

Este diseño del chasis permite alojar tanto las baterías como el resto de componentes. En los paneles laterales se encuentran elementos que se usarán con frecuencia como el enchufe del cargador, y algunas rejillas de ventilación. Las barras superiores están diseñadas para situar el brazo robótico en la zona central. Todas las barras cuentan con un perfil estructural en el caso de tener que rediseñar el interior de la plataforma. El compartimento más grande está diseñado principalmente para las baterías.

Estos son los requerimientos y primeros diseños en los que se ha basado el trabajo posterior. En el capítulo 4 se describirán y analizarán las decisiones y cambios que han llevado al resultado final del proyecto.

2.3 Precedentes

La plataforma robótica de este trabajo es fruto del trabajo previo de otros compañeros.

El diseño y montaje de la plataforma básica, las ruedas, los motores y los *encoders* lo realizó Alonso Serrano en su Trabajo Fin de Grado.

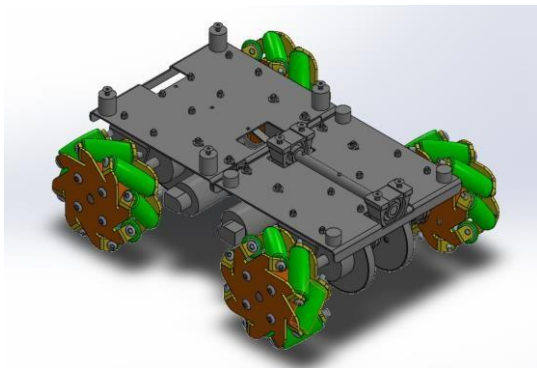


Figura 2.3: Modelo en Solidworks



Figura 2.4: Conexión de los encoders

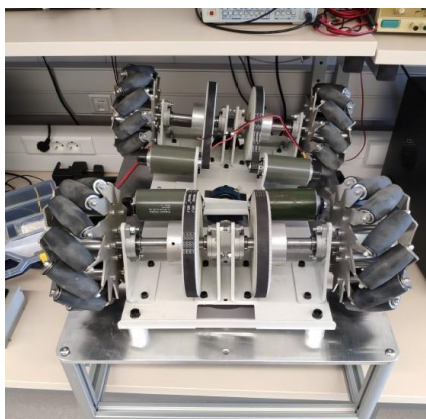


Figura 2.5: Montaje de la plataforma

El diseño y las conexiones eléctricas las ha desarrollado David Flores en su Trabajo de Fin de Grado.

Para entender el circuito general de la plataforma, se mostrará y se analizará brevemente el esquema eléctrico general.

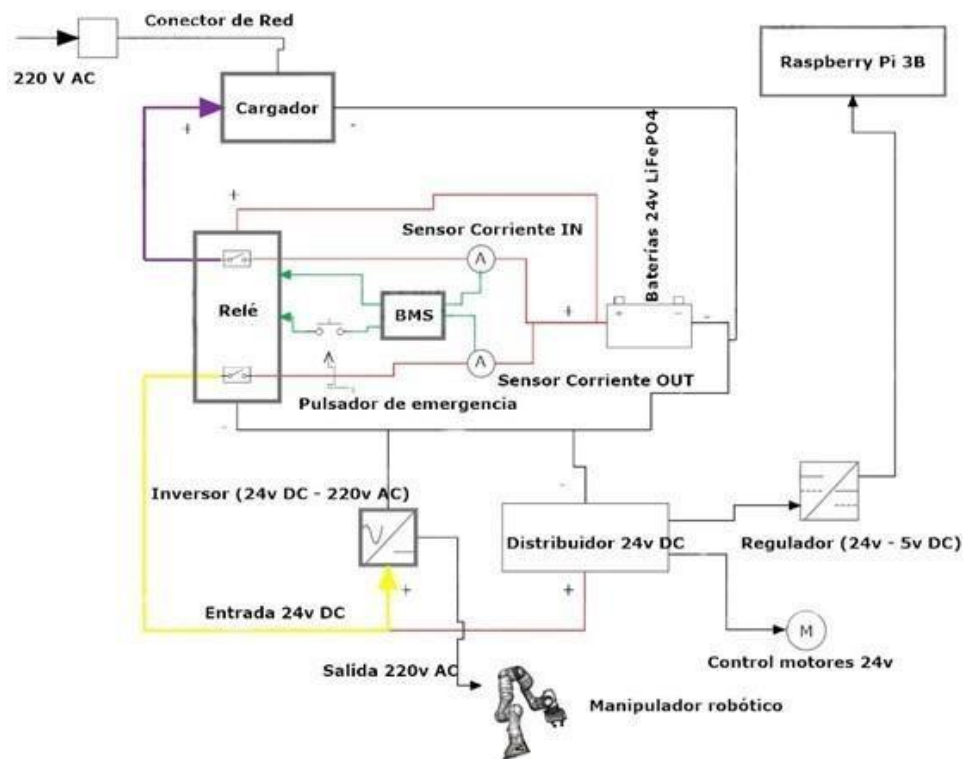


Figura 2.6: Esquema eléctrico de la plataforma robótica.

Como podemos observar, al estar enchufado para realizar la carga de las baterías, el circuito superior del relé es el que tiene que estar activado. Una vez cargado, podemos activar el circuito inferior del relé, permitiendo conectar el resto de la plataforma a la fuente de alimentación, en este caso, las baterías. Muy importante en la parte inferior es el pulsador de emergencia, que permite cortar el suministro eléctrico a todos los dispositivos entre los que se encuentran los motores de las ruedas.

Además del montaje y conexión de los elementos del esquema, cabe destacar que se reubicaron las baterías como podemos observar:

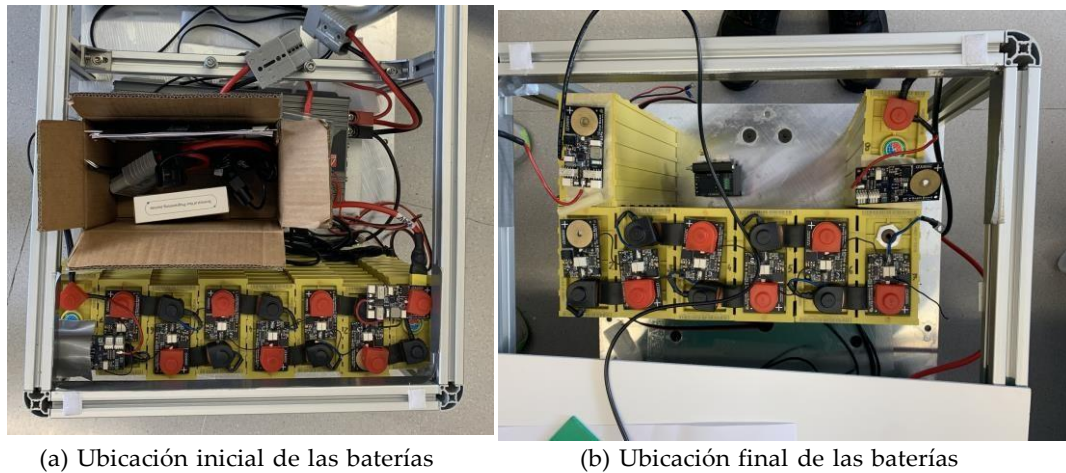


Figura 2.7: Ubicación actual de las baterías

Esto se debe a que permite la colocación de elementos accesibles en paneles laterales, y además, permite la colocación de la pantalla en el sitio planeado sin comprometer la seguridad tanto de la pantalla como de la raspberry¹.

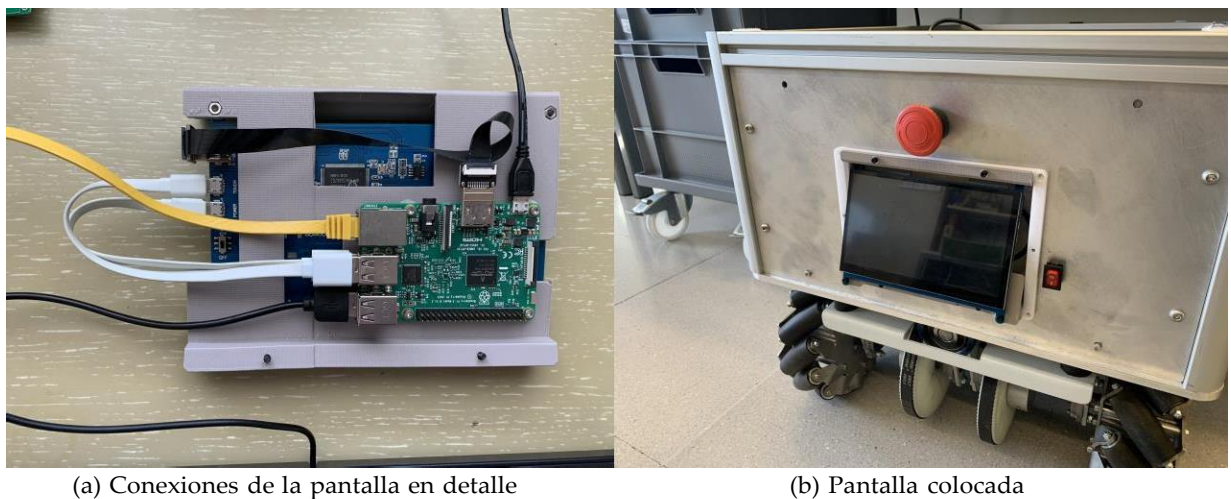


Figura 2.8: Imágenes de la pantalla exterior

Se recomienda para conocer mejor todos los elementos de la plataforma robótica realizar ojear el trabajo de los compañeros.

¹Raspberry Pi 3, no se entrará en detalle debido a que no forma parte directa de este proyecto.



COMPONENTES DE LA PLATAFORMA ROBÓTICA

Para cumplir los objetivos y especificaciones marcadas, es necesario contar con los elementos adecuados. En este capítulo se precisarán y describirán tanto los componentes como las herramientas que formarán parte del proyecto. Se realiza una división en componentes mecánicos, hardware y software.

3.1 Componentes eléctricos

El primer grupo de componentes a tratar van a ser los componentes eléctricos. El robot consta de numerosos elementos tales como distribuidor, inversor, relé, y más. No es objetivo de esta memoria tratar todos en profundidad, simplemente se describirán brevemente los más significativos para poder conocer los componentes del robot. Si se desea profundizar como se comentó en la sección 2.3, se pueden ver los trabajos de los compañeros en estas áreas.

3.1.1 Baterías

Las baterías son clave, puesto que conforman el sistema de alimentación de la plataforma robótica. Este consta de ocho celdas de baterías de LiFePO₄. El modelo concretamente es Winston LFP060AHA, de la marca GWL.

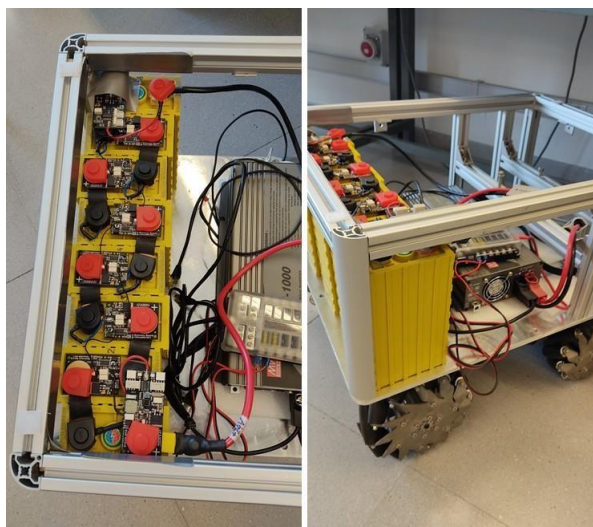


Figura 3.1: Batería LiFePO₄ modelo Winston LFP060AHA.

La batería de LiFePO₄ es una batería recargable compuesta por fosfato de hierro y litio. La principal ventaja de este tipo de batería es que ofrece tiempos de carga rápidos, alta densidad de potencia y una buena vida útil. La química también las hace más seguras que otros tipos de baterías de iones de litio porque no contienen tanta solución electrolítica inflamable.

Además, la esperanza de vida del LiFePO₄ es hasta diez veces superior al tiempo de vida de las baterías de plomo ácido y algunos tipos de baterías de ión litio. Las corrientes o potencias instantáneas aceptadas por el LiFePO₄ son significativamente superiores a las del Ión Litio (cobalto).

A continuación se muestran las especificaciones técnicas de la batería.

Specifications		
Model name	LFPS604HA	Alternative product marking TS-LFP604HA, WB-LYP604HA
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 3.2: Especificaciones técnicas de la batería LiFePO4.

Como podemos observar, al tener ocho baterías en serie, vamos a contar con una tensión nominal de 26.4V y una capacidad de 480Ah. Por lo que es importante contar con un buen sistema de seguridad, capaz de realizar una monitorización del estado de las baterías.

3.1.2 123SmartBMS gen3

BMS es un acrónimo para *Battery Management System*, es decir, un sistema de gestión de la batería. Es un conjunto de sensores, que mandan la información a una aplicación para su monitorización y control.



Figura 3.3: 123SmartBMS gen3 .

En nuestro caso, 123SmartBMS gen3, es un BMS modular, cuya función es mantener las baterías en condiciones óptimas, prolongando su vida útil. Se encarga de medir información importante como la tensión de las celdas que componen la batería, su temperatura y corrientes de entrada y salida con elevada precisión. El sistema balancea todas las celdas durante la carga

para que haya equilibrio.

Se pueden utilizar dos relés de señal integrados con contactos sin potencial para cambiar señales de habilitación/deshabilitación o relés de potencia para controlar la alimentación de cargadores/reguladores y/o inversores.

Por último, es muy sencillo conectar el móvil a BMS para visualizar información importante incluyendo el estado de carga, la energía consumida, la energía almacenada, la temperatura de las celdas, la corriente de entrada y salida, la tensión de las baterías, SoC (estado de carga) de la última semana y el historial de errores.

En la parte de software, se detallará la aplicación usada para la monitorización.

3.2 Componentes mecánicos

El segundo grupo de componentes a tratar van a ser los componentes mecánicos. Destacan el chasis y las ruedas.

3.2.1 Chasis

Como se ha comentado, las barras tienen un perfil estructural para poder reconfigurarlas. Son de aluminio de 30x30mm. En la siguiente foto podemos ver este tipo de perfil:



Figura 3.4: Perfil estructural.

Los cerramientos laterales están constituidos por panel sándwich de 0.2 mm de aluminio a dos caras con alma de polietileno, con un espesor total de 3 mm. El nombre comercial es Alucobond.

El Alucobond es ligero, posee gran rigidez a la flexión y excelente planeidad. Presenta escasos requisitos de construcción base y medios de fijación. Además es resistente a la intemperie, amortigua las vibraciones y constituye un excelente aislante térmico.

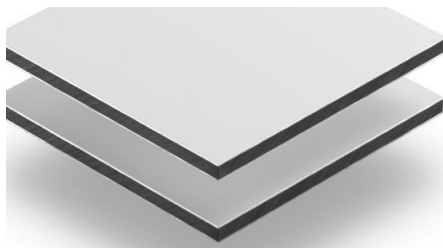


Figura 3.5: Alucobond.

3.2.2 Ruedas

La elección de las ruedas es fundamental para el desarrollo de este proyecto. Se han escogido las ruedas Mecanum.



Figura 3.6: Rueda Mecanum.

Las ruedas Mecanum son un tipo especial de rueda omnidireccional, muy útil en robótica para lograr una excelente maniobrabilidad y movimiento multidireccional. Las ruedas cuentan con una serie de rodillos colocados en un ángulo de 45 grados con respecto al eje de la rueda, esto permite al robot moverse en todas las direcciones en el plano.

Cuentan con un principio de rodadura especial. Cuando los rodillos en las ruedas giran en direcciones opuestas, generan fuerzas laterales que permiten que el robot se desplace en direcciones angulares. Al controlar la velocidad y dirección de giro de cada rueda individualmente, el robot puede lograr un movimiento muy versátil y preciso en cualquier dirección.

Estas ruedas son muy útiles para este tipo de aplicaciones porque permiten una maniobrabilidad excelente.

3.3 Descripción del hardware externo utilizado

A continuación veremos los distintos dispositivos electrónicos que forman parte del trabajo.

3.3.1 RoboClaw Motor Controller

En la siguiente figura se muestra la controladora seleccionada, RoboClaw 2x30A Motor Controller.

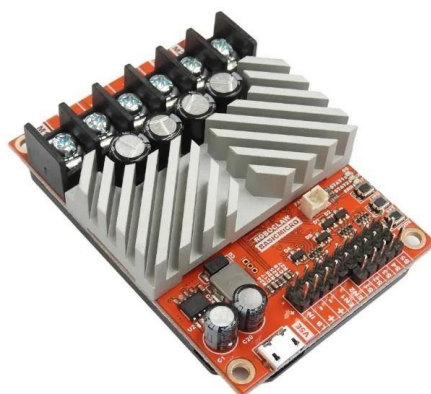


Figura 3.7: Controladora 2x30A de la marca RoboClaw.

Este controlador, está diseñado para controlar dos motores de DC con escobillas, es capaz de soportar 30A continuos por canal y picos del orden de los 60A. Tiene varios modos de conexión: USB, RC, Simple Serial, Packet serial y Analog. El modo de funcionamiento escogido ha sido Packet Serial debido a que es el típicamente usado para controlar RoboClaw desde un microcontrolador o PC.

Funciona con 3.3 o 5 voltios, y soporta interruptores de emergencia (muy útil para este trabajo). También, cuenta con soporte oficial para ROS, y un software que veremos posteriormente llamado Basicmicro Motion Studio, que permite calibrar y ajustar sencillamente todos los parámetros y establecer límites como la aceleración y desaceleración de los motores. Cuenta también con sistema de frenado regenerativo y sistemas de protección contra temperatura, corriente, voltaje,...

3.3.2 YDLIDARX4

La siguiente figura muestra el sensor escogido.



Figura 3.8: YDLIDARX4

Un LiDAR (de sus siglas en inglés “Light Detection And Ranging”) es una tecnología que permite medir distancias desde un emisor láser a un objeto o superficie. Emite un haz de luz que es reflejado en los objetos del entorno, el haz de luz rebota en los objetos (es reflejado) y vuelve al dispositivo (el LiDAR). El dispositivo entonces es capaz de medir el tiempo que tarda la luz en “viajar” desde tu mano hasta el objeto y volver, de ese modo es posible conocer a qué distancia se encuentra la superficie reflejada del dispositivo.

En nuestro caso, YDLIDAR X4 es un LiDAR que mide en 2 dimensiones con un rango de 360°. Está diseñado para alcanzar una alta frecuencia y una alta precisión en la medida.

3.3.3 PC

Se comentarán brevemente las dos alternativas disponibles y probadas. Finalmente se escogió el PC de la marca Slimbook.

3.3.3.1 Raspberry Pi 4 modelo B

Fue la primera opción, por su tamaño y fácil utilización. Equipada con un procesador ARM Cortex-A72 de cuatro núcleos a 1,50GHz, memoria LPDDR4-3200 4GB, con WiFi, Bluetooth 5.0,

4 puertos USB (dos USB 3.0 y dos USB 2.0), dos puertos micro HDMI y conector USB-C para alimentación de 5V.



Figura 3.9: Raspberry Pi 4 modelo B

3.3.3.2 Slimbook PC

Es un portátil muy ligero (1.1 Kg), con un procesador AMD Ryzen 7 5700U con 8 cores y 16 hilos, dos USB 3.1, un USB 3.1 tipo C y un USB 2.0. Memoria DDR4 no soldada 3200 Mhz y 8GB.



Figura 3.10: Slimbook Pro X 14

3.3.4 Joystick

El joystick elegido para las pruebas ha sido un mando inalámbrico de la *Play Station 4*. La elección de este mando se debe en que cómo se verá en el desarrollo, era necesario contar con dos palancas analógicas para controlar la velocidad.



Figura 3.11: Mando Dualshock PS4

3.4 Descripción del Software

Por último, vamos a presentar el software principal utilizado. Por un lado se encuentra ROS, que ha sido la herramienta utilizada para integrar todo el código y hardware, por otro Basicmicro Motion Studio, que ha servido para calibrar y monitorizar las controladoras RoboClaw, y por último 123SmartBMS para monitorizar las baterías y controlar la carga del robot.

3.4.1 ROS

ROS, que significa "Robot Operating System", es un meta sistema operativo de código abierto diseñado para facilitar el desarrollo, la programación y la operación de robots y sistemas robóticos. Provee de servicios que se esperarían de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, pasaje de mensaje entre procesos y manejo de paquetes. También brinda herramientas y librerías para obtener, construir, escribir y correr código a través y mediante varias computadoras.

Algunas de las características principales de ROS son:

- **Arquitectura modular:** Está diseñado en forma de módulos interconectados, lo que permite a los desarrolladores construir sistemas robóticos de manera modular y reutilizar componentes.
- **Comunicación entre procesos:** Proporciona un sistema de comunicación entre procesos que permite a los diferentes componentes de un robot (sensores, actuadores, algoritmos de control, etc.) intercambiar información y comandos de manera eficiente.
- **Control de hardware:** Permite controlar diversos tipos de hardware, desde motores y sensores hasta cámaras y brazos robóticos, a través de interfaces estandarizadas.
- **Herramientas de desarrollo:** Ofrece una serie de herramientas para desarrollar, depurar y probar código robótico, lo que facilita el proceso de desarrollo.
- **Simulación:** Permite simular robots y entornos para probar y depurar algoritmos y sistemas antes de implementarlos en hardware real.
- **Comunidad activa:** Tiene una comunidad de desarrolladores muy activa que contribuye con bibliotecas, paquetes y soluciones a problemas comunes en la robótica.
- **Flexibilidad:** Aunque se originó en el contexto de la robótica móvil, ROS se ha utilizado en una amplia variedad de aplicaciones robóticas, desde robots industriales hasta drones y sistemas médicos.

3. COMPONENTES DE LA PLATAFORMA ROBÓTICA

En resumen, ROS es una plataforma esencial en la robótica moderna que facilita la creación y operación de sistemas robóticos complejos a través de su arquitectura modular y herramientas de desarrollo.

Actualmente ROS cuenta con dos versiones, ROS1 y ROS2. Estas cuentan con diferentes distribuciones (versiones específicas). En nuestro caso, usaremos ROS1 y la distribución ROS Noetic Ninjemys.

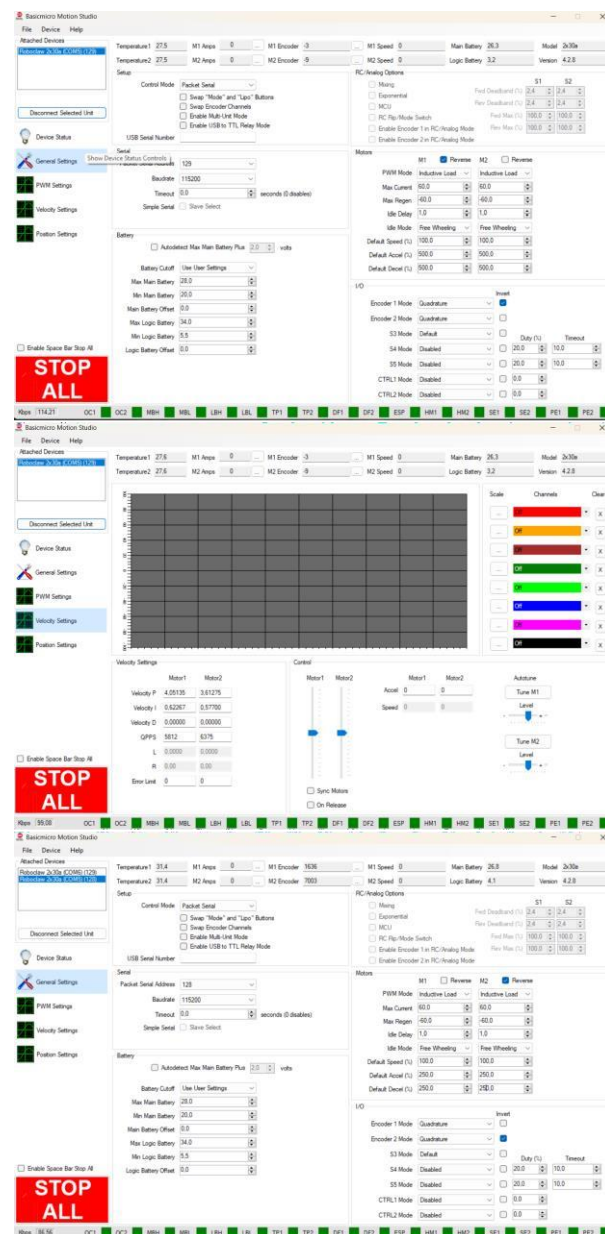
Con el objetivo de entender bien el contenido que se va a presentar, se va a aclarar también cómo funciona ROS y qué contenidos nos vamos a encontrar. Los distintos componentes de un proyecto en ROS son:

- **Nodos:** El componente fundamental de ROS es el nodo. Un nodo es un proceso de ejecución individual que realiza una tarea específica. Puede ser un controlador de hardware, un algoritmo de procesamiento, un sensor, un actuador, etc. Los nodos se comunican entre sí a través de topics, servicios y otros mecanismos.
- **Paquetes:** Son una unidad organizativa que agrupa nodos, bibliotecas, también archivos de configuración y otros recursos relacionados. Los paquetes facilitan la gestión y distribución de componentes robóticos, lo que permite la reutilización y el desarrollo modular en proyectos de robótica y automatización.
- **Topics:** Son un canal de comunicación que permite que los nodos intercambien mensajes. Los nodos pueden publicar mensajes en un topic y otros nodos pueden suscribirse a ese topic para recibir y procesar los mensajes. Los topics son fundamentales para compartir información.
- **Servicios:** Son un mecanismo de comunicación que permite que un nodo solicite una acción o información específica a otro nodo y reciba una respuesta. En este proyecto no se van a usar.
- **Master:** ROS Master es un nodo central que actúa como registro y facilitador de la comunicación entre los nodos. Mantiene un registro de nodos, topics, servicios y parámetros en el sistema, permitiendo que los nodos encuentren y se comuniquen entre sí de manera eficiente. En ROS2 se ha eliminado, pero en ROS1 es necesario su funcionamiento para que los elementos del sistema interactúen entre sí.
- **Parámetros:** Son valores configurables que permiten ajustar el comportamiento de los nodos y componentes. Pueden ser usados para definir constantes, configuraciones y valores de calibración en tiempo de ejecución. Los parámetros son accesibles por los nodos y se pueden modificar desde la línea de comandos o archivos de configuración. En este caso se configurarán desde los archivos ".launch".

- Herramientas: ROS incluye un conjunto de herramientas de línea de comandos y gráficas para depurar, visualizar y supervisar los nodos y la comunicación entre ellos. Ejemplos de herramientas son "rostopic", "roslaunch", "rqt_graph", entre otros.

3.4.2 Basicmicro Motion Studio

Aplicación que podemos descargar en la página oficial de Basicmicro. Sirve para cambiar y guardar parámetros en los controlados RoboClaw, además permite realizar su monitorización, y también nos permite calcular automáticamente el PID del robot.



3.4.3 123SmartBMS

Aplicación de la Google Play Store, que permite monitorizar el software al que le hemos conectado los sensores comentados en la sección 3.1.2.

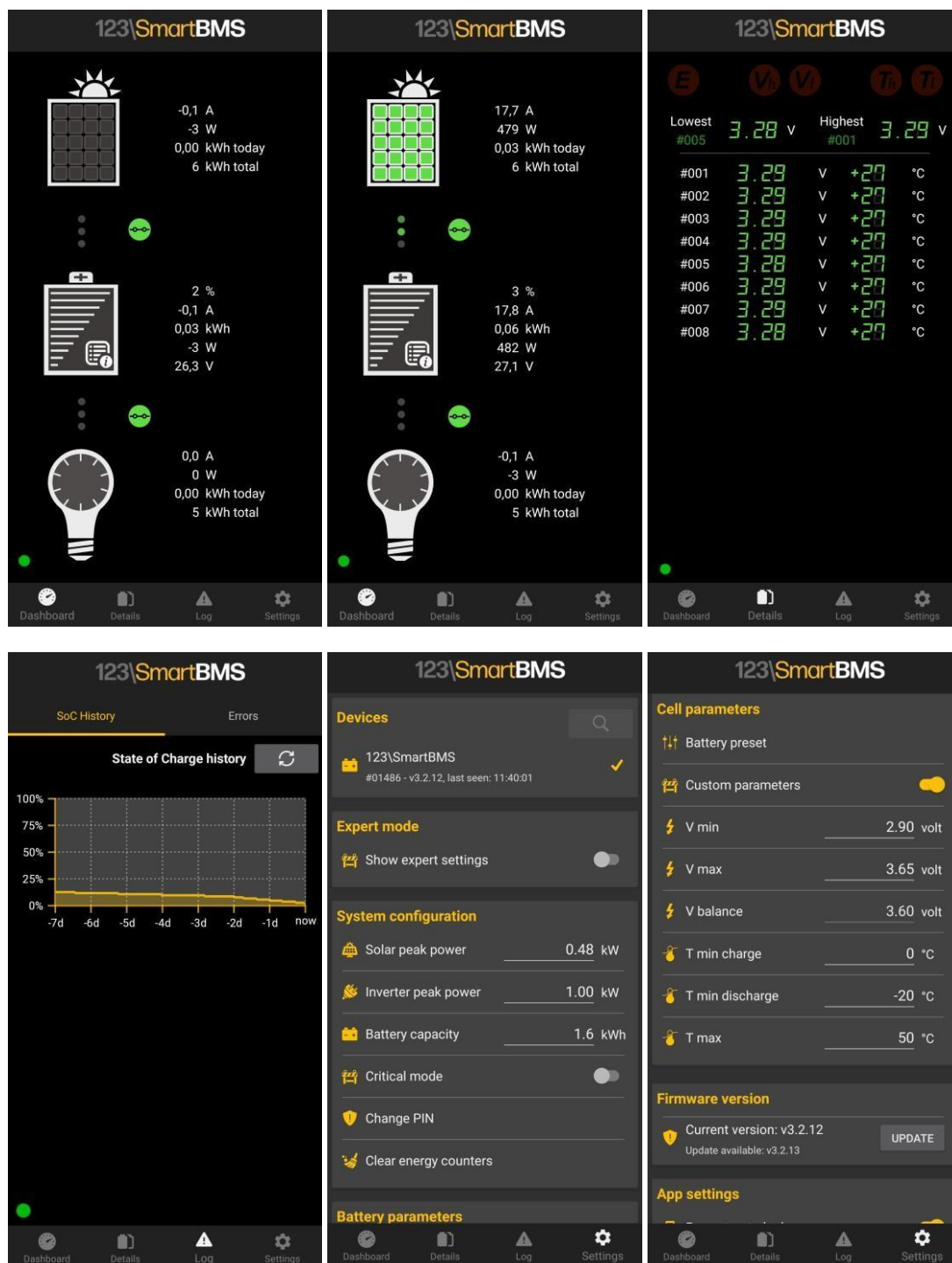


Figura 3.13: Capturas de la aplicación 123SmartBMS.

4

DESARROLLO DEL TRABAJO

En este capítulo se describirá el proceso que se ha seguido hasta conseguir el objetivo de este TFG. Se hará especial hincapié también en los problemas y soluciones presentados a los mismos, puesto que se entiende que forman parte importante del proceso de desarrollo y aprendizaje de este trabajo.

No sólo son ideas rechazadas, sino que también son opciones a las que volver y tener en cuenta para líneas futuras o mejoras.

4.1 Primeros pasos y problemas

Inicialmente, la idea era usar una Raspberry Pi 4 como PC. A esta Raspberry se le instalaría ROS 2 Foxy Fitzroy, una distribución específica. Estaría conectada tanto a las controladoras, como al joystick y al LiDAR.

En primera instancia, en la Raspberry se instaló el sistema operativo Ubuntu Desktop 20.04, sin embargo, su rendimiento era deficiente. Ubuntu Desktop ofrece un software completo, pero lleno de elementos pesados que no van a servir para este robot.

Como la Raspberry cuenta con unos recursos limitados (4GB de RAM), se optó por la instalación de un sistema operativo ligero, en este caso se usó Ubuntu Server 20.04 y se instaló "xfce" como interfaz de escritorio. Esta solución fue un éxito que optimizó el rendimiento.

Se probó el código de un usuario de Github para las controladoras, pero no funcionaba como se esperaba. El robot no obedecía los comandos y se movía de manera errática. Se intentó comprobar la velocidad de transmisión, el código, baterías, ... Pero finalmente se desestimó esta solución.

Se terminó escogiendo el código oficial de la página de RoboClaw para ROS 1 Noetic. Sin embargo, no fue suficiente tampoco para resolver el problema, el robot seguía teniendo problemas de comportamiento. Finalmente, el código que se expondrá es un *fork o bifurcación*¹ realizada por otro usuario, en la que se han realizado modificaciones del código oficial.

El código original está adaptado para ROS1 distribución Melodic que utiliza Python2. Este usuario de Github realizó algunas modificaciones y además lo adaptó a Python3. Este último código ha sido la base para este proyecto como veremos en el apartado 4.2.3.

Veremos a continuación dos secciones, una dedicada a la creación de los nodos y códigos, mucho más centrada en el software; y otra dedicada a la electrónica y a los cambios necesarios en la plataforma para su correcto funcionamiento.

4.2 Modificación y creación de los nodos

En primer lugar, se creó un workspace (espacio de trabajo) de ROS siguiendo los pasos de la guía oficial. De esta forma, nos encontramos el siguiente contenido en la carpeta que conforma el workspace:

¹Un fork o bifurcación es un nuevo repositorio que comparte la configuración de visibilidad y código con el repositorio "ascendente" original.

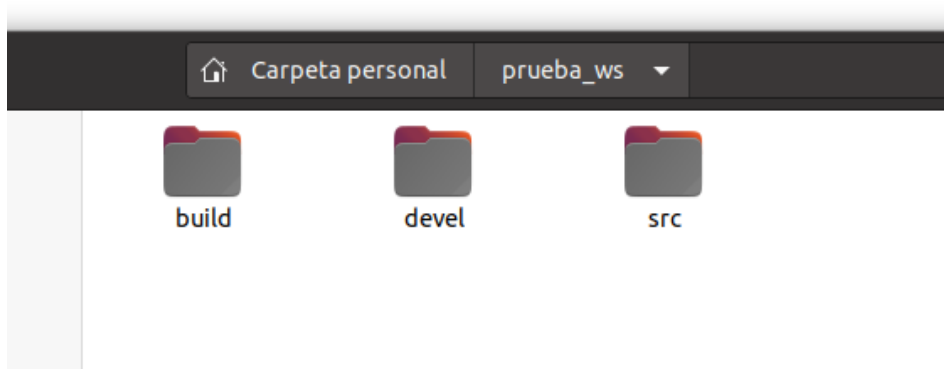


Figura 4.1: Captura de las carpetas del workspace.

- **build:** Almacena archivos generados durante la compilación de paquetes. Contiene binarios y objetos compilados, facilitando la separación entre archivos fuente y compilados, y permitiendo la construcción eficiente de nodos y componentes robóticos.
- **devel:** Guarda los archivos y enlaces simbólicos de los paquetes compilados, permitiendo que los nodos y componentes se ejecuten directamente desde el workspace. También facilita la depuración y el desarrollo al proporcionar acceso a los archivos generados durante la compilación.
- **src:** Reúne los paquetes fuente que contienen el código y los recursos necesarios para construir nodos y componentes robóticos. Facilita la organización y gestión de los paquetes, permitiendo el desarrollo modular y reutilizable en el entorno de ROS. En este caso, es la carpeta más interesante para analizar y nos adentraremos en su contenido

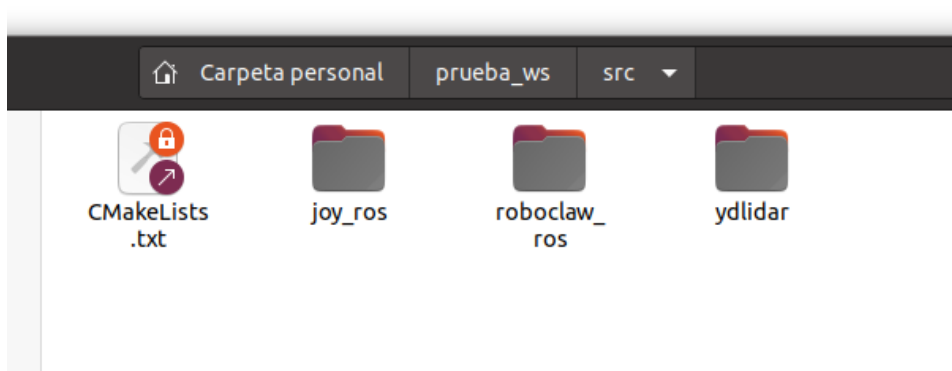


Figura 4.2: Captura de las carpetas de src.

Dentro de la carpeta `src`, tenemos tres carpetas como vemos en la figura 4.2. En la carpeta `src` se encuentran los códigos para hacer funcionar la plataforma robótica y para cambiar los parámetros de funcionamiento.

Hay elementos comunes a todas las carpetas que veremos a continuación:

- El archivo "CMakeLists.txt" es un archivo esencial que juega un papel crucial en el proceso de compilación y construcción del paquete. CMake es una herramienta de código abierto que se utiliza para gestionar la construcción de software en diversos proyectos, incluidos los paquetes de ROS. El archivo CMakeLists.txt en un paquete de ROS contiene instrucciones para CMake sobre cómo compilar y construir el paquete. Definir e identificar el paquete, declarar las dependencias a usar, configurar la compilación y generar ejecutables son algunas de sus principales tareas.
- El archivo "package.xml" contiene información importante sobre el paquete y sus metadatos. Algunas de sus funciones clave son: Guardar los metadatos del paquete, especificar las dependencias requeridas para compilar correctamente, identificar autores y contribuyentes, definir la licencia para un uso legal y ético del paquete, contener información de compilación y proporcionar información de contacto.
- Los archivos "_init_.py" son archivos especiales en Python que se utilizan para definir un directorio como un paquete. Estos archivos pueden estar vacíos o contener código Python que se ejecutará cuando se importe el paquete. En este proyecto estos archivos estarán vacíos ya que no necesitamos agregar ningún código específico a ese nivel del paquete.

Por otra parte, vamos a ver los diferentes tipos de archivos que se verán en la estructura de carpetas, además de los previamente mencionados:

- Los archivos ".py" son archivos de código fuente en el lenguaje de programación Python. Estos archivos contienen el código que será interpretado y ejecutado por el intérprete de Python.
- Los archivos ".cpp" son archivos de código fuente escritos en el lenguaje de programación C++
- Los archivos ".yaml" es un lenguaje de declaración de datos que facilita la legibilidad y la capacidad de escritura del usuario. Este formato de serialización de datos se encarga de almacenar archivos de configuración y se puede usar junto con todos los lenguajes de programación.
- Los archivos ".launch" son archivos utilizados en ROS para definir y ejecutar lanzamientos de nodos, parámetros y configuraciones. Permiten lanzar varios nodos y componentes en una configuración coordinada. Están escritos en XML.

Dividiremos esta sección en los tres paquetes principales: joy_ros, ydlidar y roboclaw_ros:

- joy_ros: El paquete que transcribe la entrada del joystick.
- ydlidar: El paquete que activa y lee los datos del LiDAR.
- roboclaw_ros: El paquete que maneja las dos RoboClaw.

4.2.1 Joy_ros

En primer lugar, se visualizará en un diagrama el sistema de carpetas de este paquete, y a comentar los archivos que contiene:



Figura 4.3: Diagrama de archivos dentro de joy_ros.

Dentro de este paquete, tenemos que destacar los siguientes archivos:

- joy.config.yaml: Sirve para poder configurar el joystick. Se puede modificar tanto la velocidad base como el eje analógico para controlar esa velocidad en el eje X, en el eje Y, así como la angular. También, si pulsamos un botón, se puede mover el robot en modo turbo, lo que significa que se desplaza el doble de veloz. Por último, se ha añadido un parámetro de precaución, de modo que para enviar comandos hay que mantener pulsado un botón del joystick. De esta forma podemos prevenir que debido a que el joystick se caiga o se pulse una palanca accidentalmente el robot se mueva a una posición no deseada.

```

1 axis_linear:
2   x: 1
3   y: 0
4 scale_linear:
5   x: 0.4
6   y: 0.4
7 scale_linear_turbo:
8   x: 0.8
9   y: 0.8
10 axis_angular:
11   yaw: 3
12 scale_angular:
13   yaw: 0.5
14 scale_angular_turbo:
15   yaw: 1.0
16 enable_button: 4
17 enable_turbo_button: 5
18 steer: 2

```

Figura 4.4: Captura de joy.config.yaml.

- teleop.launch: Se encarga de lanzar los dos nodos principales de este paquete con su configuración. El primer nodo es el nodo "joy", que se encarga de recibir comandos de control desde gamepads, joysticks y otros dispositivos similares. El segundo nodo "teleop_twist_joy" convierte las entradas del joystick en comandos de velocidad y dirección que son enviados al robot.

```

1 <?xml version="1.0"?>
2 <launch>
3   <arg name="joy_config" default="ps3" />
4   <arg name="joy_dev" default="/dev/input/js0" />
5   <arg name="config_filepath" default="$(find joy_ros)/config/joy.config.yaml" />
6   <arg name="joy_topic" default="joy" />
7
8   <remap from="cmd_vel" to="cmd_vel_joy" />
9
10  <node pkg="joy" type="joy_node" name="joy_node">
11    <param name="dev" value="$(arg joy_dev)" />
12    <param name="deadzone" value="0.3" />
13    <param name="autorepeat_rate" value="20" />
14    <remap from="joy" to="$(arg joy_topic)" />
15  </node>
16
17  <node pkg="teleop_twist_joy" name="teleop_twist_joy" type="teleop_node">
18    <rosparm command="load" file="$(arg config_filepath)" />
19    <remap from="joy" to="$(arg joy_topic)" />
20  </node>
21 </launch>

```

Figura 4.5: Captura de teleop.launch.

- teleop_node.cpp y teleop_twist_joy: Ambos archivos de código, ejecutan los paquetes previamente comentados.

Para comprobar que el PC reconocía el joystick y configurar "joy.config.yaml" se ha usado la herramienta "jstest-gtk". Es una herramienta gráfica utilizada para probar y calibrar dispositivos de control de juegos, como joysticks y gamepads, en sistemas basados en Linux. Permite a los

usuarios verificar y ajustar el funcionamiento de estos dispositivos, así como observar la salida de los ejes y botones en tiempo real.

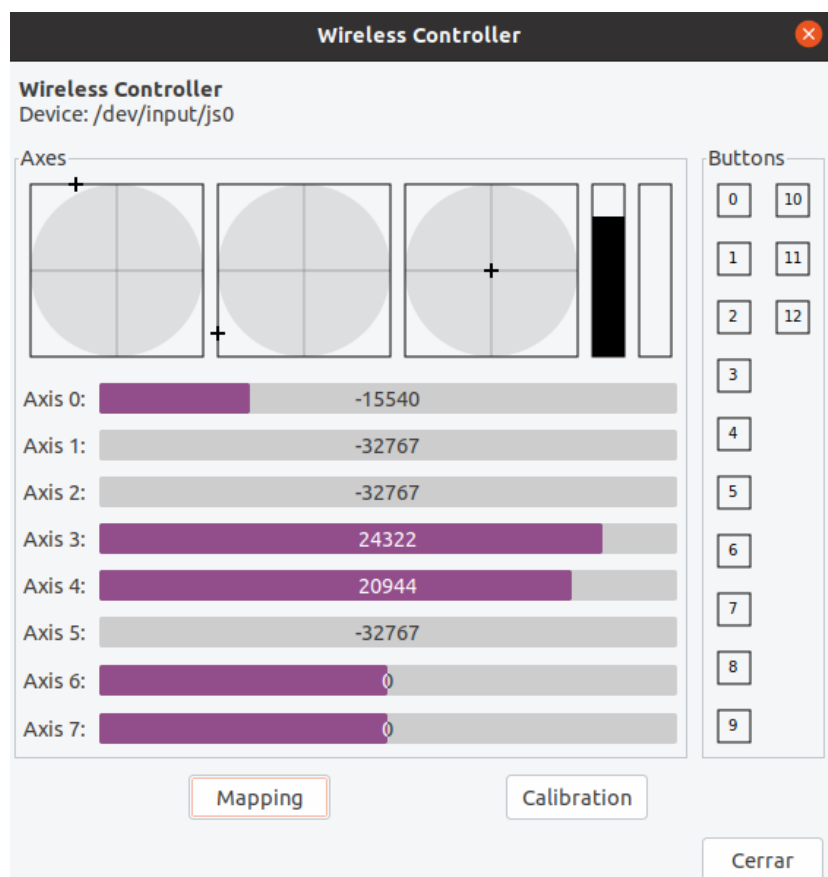


Figura 4.6: Captura del programa Jstest-gtk.

4.2.2 ydlidar

En este paquete se ha cogido el software proporcionado por la empresa creadora del lidar Shenzhen EAI Technology Co.,Ltd.

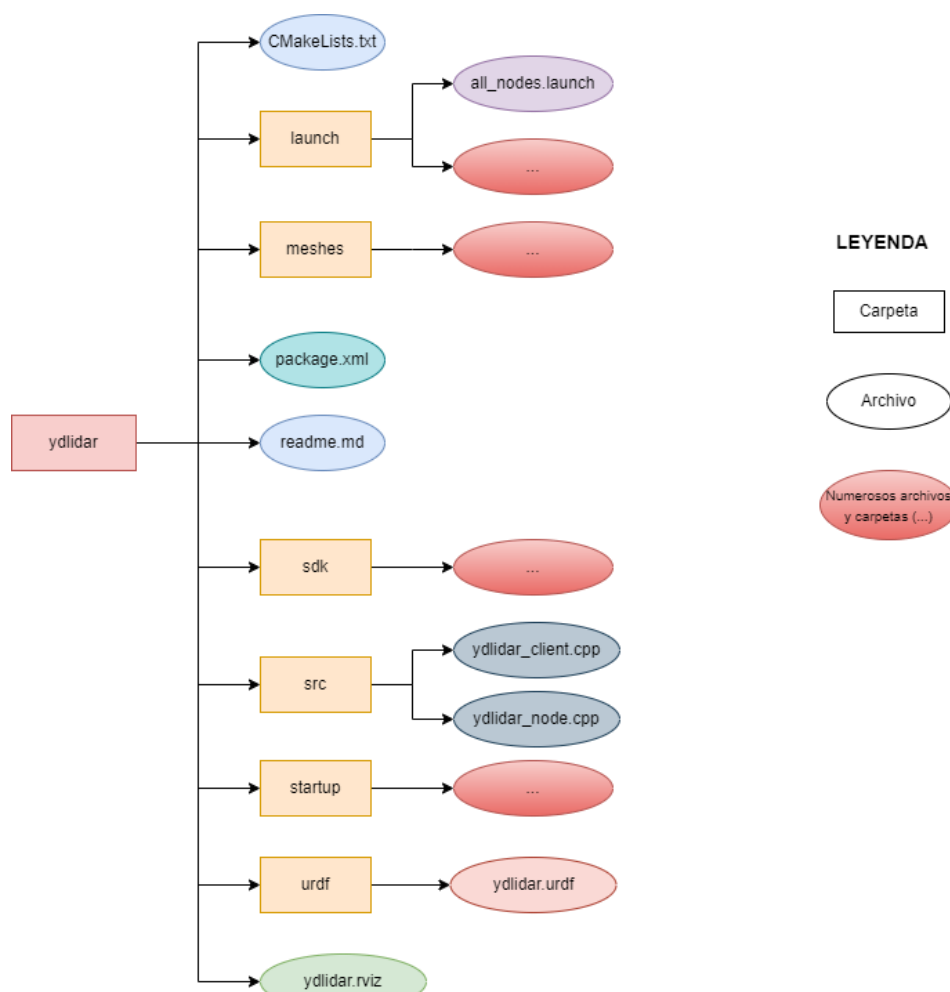


Figura 4.7: Diagrama de archivos dentro de ydlidar.

Como se ha comentado, en este paquete apenas se han realizado modificaciones a los archivos que contiene, sin embargo, se ha integrado con el paquete de SLAM² `hector_slam`, de forma que realiza un mapeo simple del entorno.

Dentro de este paquete, tenemos que destacar los siguientes archivos³:

²SLAM (Simultaneous Localization and Mapping) es una tecnología que permite a un robot o vehículo autónomo crear un mapa de su entorno mientras se localiza en dicho mapa en tiempo real. Utiliza sensores como cámaras y láseres para comprender su posición y entorno

³En este proyecto, al no haber simulación, no se ha contado con los archivos URDF. Un archivo URDF es un formato de archivo en XML utilizado en ROS para describir la geometría, la cinemática y la información visual de un robot en 3D. Se usan para su representación y simulación en herramientas de visualización como RViz o Gazebo.

- `all_nodes.launch`: Este archivo fue creado para poder lanzar el resto de archivos con sus parámetros correspondientes sin tener que lanzarlos individualmente, y de esta forma poder usar `hector_slam` e `ydliar` de forma sencilla . Destacan el nodo "`ydliar`" para que el LiDAR se inicialice, y los nodos del paquete de ROS "`tf`". Este es un paquete que gestiona transformaciones entre sistemas de coordenadas en tiempo real. Permite a los componentes robóticos conocer sus posiciones y orientaciones relativas.
- La carpeta `sdk`: Contiene archivos de testeo para calibrar el LiDAR en caso de que fuera necesario y una breve explicación de cómo realizarlo.
- `ydliar_client.cpp` y `ydliar_node.cpp`: Códigos en C++ para inicializar el LiDAR y asegurar su correcto funcionamiento.
- `ydliar.rviz`: Archivo de configuración utilizado en RViz, la herramienta de visualización 3D en ROS. Define aspectos como la apariencia, la posición relativa y otros detalles visuales del sensor y sus datos en la interfaz de RViz.

4.2.3 Roboclaw_ros

Por último, vamos a ver el paquete con más archivos modificados:

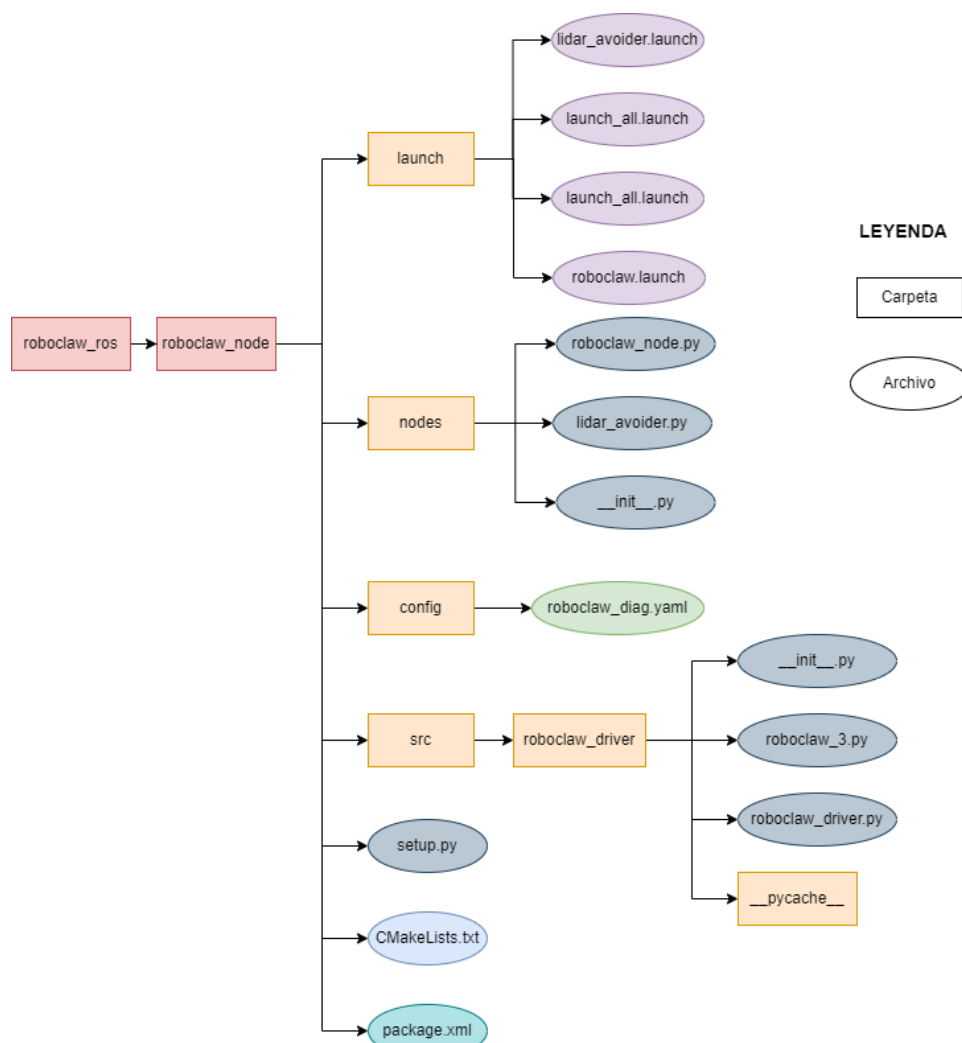


Figura 4.8: Diagrama de archivos dentro de roboclaw_ros.

Este paquete es una modificación del código original proporcionado por Basicmicro, como se ha comentado previamente

Por otra parte, este paquete es el principal para el funcionamiento, podemos ver en este diagrama cómo se encarga de gestionar toda la información proporcionada por el resto de nodos:

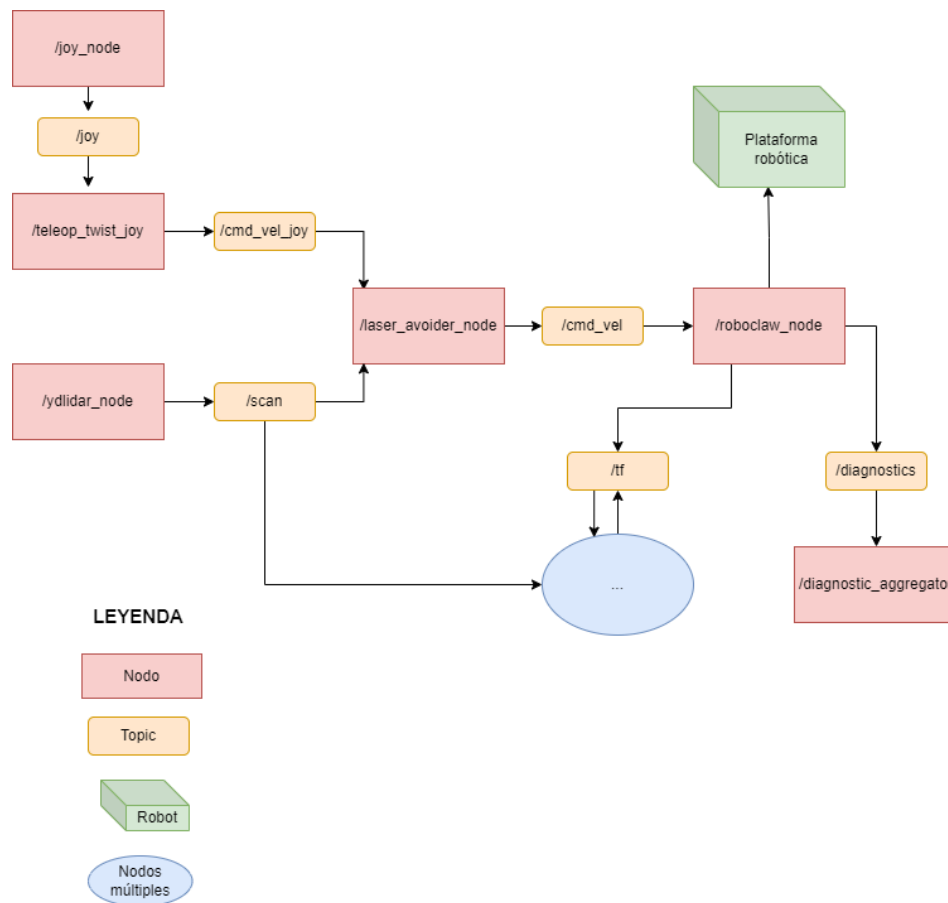


Figura 4.9: Grafo de funcionamiento de roboclaw_ros.

A continuación, se detallarán los principales archivos que participan en este proceso:

- `roboclaw_diag.yaml`: Este nodo se encarga de guardar los parámetros de otro paquete llamado "diagnostic_aggregator". Este paquete incluye un nodo llamado "aggregator_node", que se suscribe a mensajes en el topic `/diagnostics`. Luego, procesa y organiza los datos de diagnóstico y los repubblica en el topic `/diagnostics_agg`. El nodo utiliza complementos "Analyzer" para el procesamiento y categorización de diagnósticos. La configuración y el ajuste de cada agregador de diagnóstico son específicos para cada robot y pueden ser determinados por usuarios o desarrolladores para adaptarse a sus necesidades particulares.

El paquete "diagnostic_aggregator" es útil para gestionar sistemáticamente la información de diagnóstico en un entorno ROS, ayudando a identificar y resolver problemas dentro de los sistemas de robots.

- `roboclaw.launch`: Lanza tanto el script `roboclaw.node.py` como `diagnostic_aggregator` con la configuración de `roboclaw_diag.yaml`. También contiene todos los parámetros necesarios para controlar las RoboClaw:

4. DESARROLLO DEL TRABAJO

```

1 <?xml version="1.0"?>
2 <launch>
3 <!--RoboClaw Motor Controller Parameters-->
4 <arg name="dev1" default="/dev/ttyACM0"/>
5 <arg name="dev2" default="/dev/ttyACM1"/>
6 <arg name="baud" default="115200"/>
7 <arg name="address1" default="128"/>
8 <arg name="address2" default="129"/>
9 <arg name="max_speed" default="0.8"/>
10 <arg name="ticks_per_meter" default="4500"/>
11 <arg name="base_width" default="0.4223"/>
12 <arg name="cmd_frequency" default="20"/>
13
14 <!--RoboClaw Node-->
15 <node pkg="roboclaw_node" type="roboclaw_node.py" name="roboclaw_node" output="screen">
16 <param name="--dev1" value="$(arg dev1)"/>
17 <param name="--dev2" value="$(arg dev2)"/>
18 <param name="--baud" value="$(arg baud)"/>
19 <param name="--address1" value="$(arg address1)"/>
20 <param name="--address2" value="$(arg address2)"/>
21 <param name="--max_speed" value="$(arg max_speed)"/>
22 <param name="--ticks_per_meter" value="$(arg ticks_per_meter)"/>
23 <param name="--base_width" value="$(arg base_width)"/>
24 <param name="--cmd_frequency" value="$(arg cmd_frequency)"/>
25 </node>
26
27
28 <node pkg="diagnostic_aggregator" type="aggregator_node"
29 name="diagnostic_aggregator">
30 <rosparam command="load"
31 file="$(find roboclaw_node)/config/roboclaw_diag.yaml"/>
32 </node>
33
34 </launch>

```

Figura 4.10: Captura de código del archivo roboclaw.launch.

- roboclaw_node.py: Es la pieza principal del paquete. Se encarga de enviar el comando adecuado cada vez que llegue un mensaje por el topic `"/cmd_vel"`. También controla el puerto de comunicación evitando que queden comandos taponados, comprueba que no haya errores de ningún tipo en las RoboClaw al iniciar el nodo, lee los encoders, en general, se encarga del correcto funcionamiento de las RoboClaw a varios niveles.

Es importante comentar que el código original está realizado **para una única RoboClaw**, por lo que hubo que adaptarlo para que leyera las dos a la vez y no hubiera problemas de sincronización.

Además, se añadió la cinemática de la plataforma, extraída de un *paper*. Podemos verlo a continuación:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(l_x + l_y) \\ 1 & 1 & (l_x + l_y) \\ 1 & 1 & -(l_x + l_y) \\ 1 & -1 & (l_x + l_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}.$$

Figura 4.11: Cinemática de la plataforma robótica.

Se multiplicó por el radio para calcular la velocidad lineal de las ruedas, y al integrarla en el código se vería de la siguiente forma:

```

def cmd_vel_callback(self, twist):
    self.last_set_speed_time = rospy.get_rostime()

    linear_x = twist.linear.x
    linear_y = twist.linear.y

    if linear_x > self.MAX_SPEED:
        linear_x = self.MAX_SPEED
    if linear_x < -self.MAX_SPEED:
        linear_x = -self.MAX_SPEED

    if linear_y > self.MAX_SPEED:
        linear_y = self.MAX_SPEED
    if linear_y < -self.MAX_SPEED:
        linear_y = -self.MAX_SPEED

#128 DELANTE
#129 DETRAS
v1r = linear_x + linear_y + twist.angular.z * self.BASE_WIDTH / 2.0 # m/s V2
v1l = linear_x - linear_y - twist.angular.z * self.BASE_WIDTH / 2.0 #V1
v2r = linear_x - linear_y + twist.angular.z * self.BASE_WIDTH / 2.0 # m/s V3
v2l = linear_x + linear_y - twist.angular.z * self.BASE_WIDTH / 2.0 #V4

v1r_ticks = int(v1r * self.TICKS_PER_METER) # ticks/s
v1l_ticks = int(v1l * self.TICKS_PER_METER)
v2r_ticks = int(v2r * self.TICKS_PER_METER) # ticks/s
v2l_ticks = int(v2l * self.TICKS_PER_METER)

rospy.loginfo("v1r_ticks:%8d v1l_ticks: %8d", v1r_ticks, v1l_ticks)
rospy.loginfo("v2r_ticks:%8d v2l_ticks: %8d", v2r_ticks, v2l_ticks)

```

Figura 4.12: Captura de código del archivo roboclaw_node.py.

En el código podemos ver que la velocidad indicada no puede superar la velocidad máxima (MAX_SPEED), a continuación se calcula la velocidad lineal de las cuatro ruedas, y se pasa de m/s a ticks/s para enviarla a las RoboClaw.

- `roboclaw_3.py`: Este paquete ha sido proporcionado por Basicmicro, y contiene la definición de la clase⁴ RoboClaw que se usa en `roboclaw_node.py` con todas las métodos útiles que manejan las controladoras RoboClaw.
- `lidar_avoider.py`: Este código ha sido diseñado para gestionar la evasión de obstáculos. Tiene una función principal que se ejecuta continuamente y está suscrita a dos topics (/scan y /cmd_vel_joy). Cada topic tiene una función de callback⁵. La función "lid_callback" se ejecuta cuando se recibe un mensaje por "/scan" y comprueba si algún valor está por debajo del umbral de seguridad. Envía un booleano⁶ llamado "obstaculo" a "mov_callback", que se ejecuta cuando se recibe un mensaje por "/cmd_vel_joy" (o en su defecto con que esté el botón de seguridad pulsado aunque no se envíen movimientos nuevos) y en función de este booleano, se manda por "/cmd_vel" un valor de velocidad predefinido o el mensaje de "/cmd_vel_joy". Para ilustrar mejor esta explicación mirar el diagrama 4.13.

⁴Una clase en Python es una estructura de programación que permite definir un conjunto de métodos y atributos que describen un objeto o entidad

⁵Función que se define para manejar eventos relacionados con la recepción de mensajes. Se ejecutan automáticamente cuando un nodo recibe un nuevo mensaje por un topic.

⁶Un booleano es un tipo de dato que puede tener dos valores posibles: True (verdadero) o False (falso)..

4. DESARROLLO DEL TRABAJO

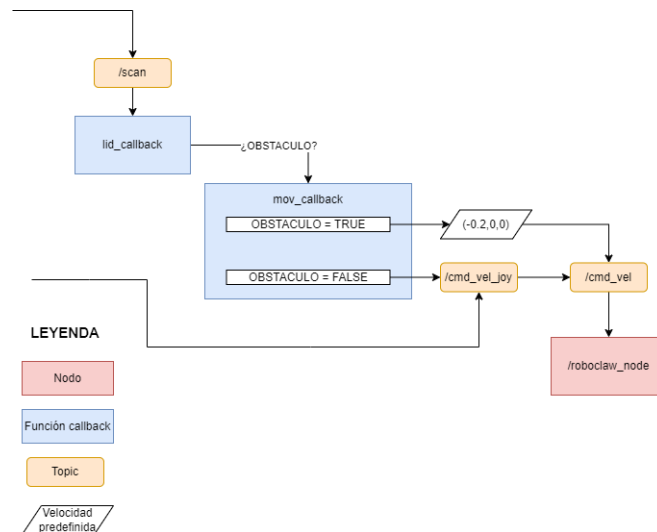


Figura 4.13: Diagrama de funcionamiento de lidar_avoider.

- `lidar_avoider.launch`: El objetivo de este código es parametrizar tanto el umbral de detección como el rango angular de detección, y la respuesta esperada del robot. Como podemos ver, el usuario puede escoger a partir de qué distancia el robot entiende como un obstáculo lo que tiene en frente, entre qué ángulos va a tener en cuenta los valores obtenidos, y qué respuesta va a realizar cuando se detecte un obstáculo. En este caso, considera obstáculo aquello que esté a menos de 1.5 metros del sensor, entre (-30,30) grados y la acción es retroceder a 0.2 m/s hasta dejar de detectar el obstáculo.

```

1 <?xml version="1.0"?>
2 <launch>
3 <!--Lidar Avoider Parameters-->
4 <arg name="range_min" default="-30"/>
5 <arg name="range_max" default="30"/>
6 <arg name="umbral" default="1.5"/>
7 <arg name="lin_x" default="-0.2"/>
8 <arg name="lin_y" default="0"/>
9 <arg name="ang_z" default="0"/>
10
11
12
13 <!--Lidar Avoider Node-->
14 <node pkg="roboclaw_node" type="lidar_avoider.py" name="lidar_avoider_node" output="screen">
15 <param name="range_min" value="$ (arg range_min)"/>
16 <param name="range_max" value="$ (arg range_max)"/>
17 <param name="umbral" value="$ (arg umbral)"/>
18 <param name="lin_x" value="$ (arg lin_x)"/>
19 <param name="lin_y" value="$ (arg lin_y)"/>
20 <param name="ang_z" value="$ (arg ang_z)"/>
21
22 </node>
23
24 </launch>

```

Figura 4.14: Captura de código del archivo `lidar_avoider.launch`.

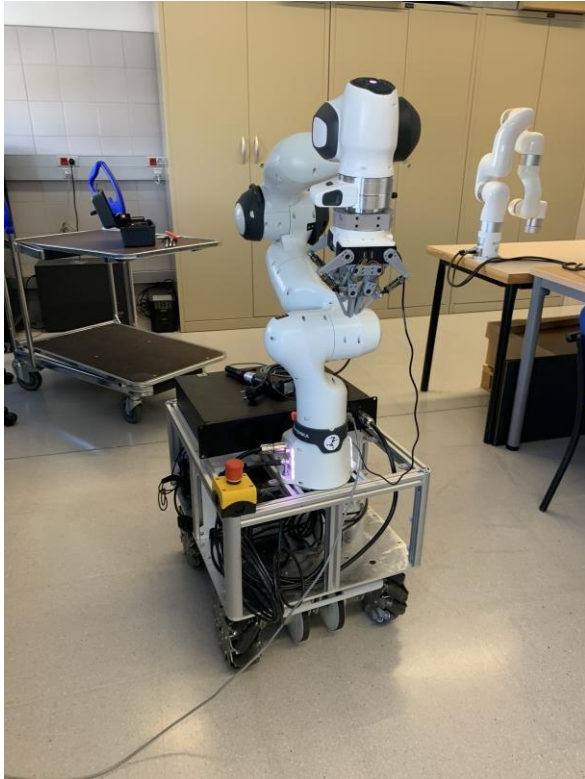
- `launch_all.launch`: Este archivo se ha creado para facilitar el funcionamiento de la plataforma. Como se ha visto, todos los paquetes tienen un archivo `launch`, por lo que se han unificado todos en un archivo. De esta forma, solo se va a tener que ejecutar un comando desde la terminal para poder controlar la plataforma robótica.

Estas serían las implementaciones a nivel de código y software desarrollados, pero para alcanzar este resultado también se han tenido que resolver algunos contratiempos físicos con los que en principio no se contaba.

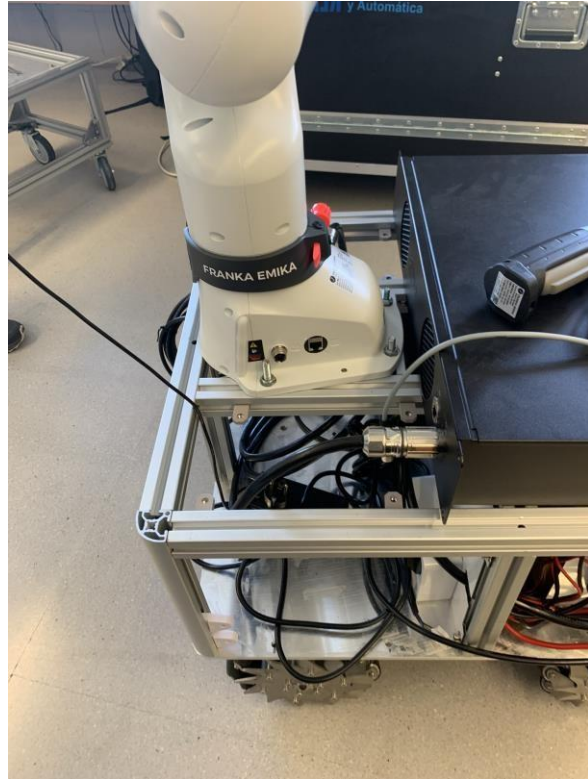
4.3 Desarrollo mecánico y de hardware

El primer desarrollo consistió en crear el chasis:

Se utilizó el diseño inicial, al que se le añadieron las barras con perfil estructural. Se realizó una prueba una vez añadidas las barras de la figura 2.2, y se verificó el planteamiento y se añadieron posteriormente los paneles laterales.



(a) Plataforma con brazo robótico



(b) Plataforma con brazo robótico en detalle

Figura 4.15: Imágenes del brazo robótico colocado en la plataforma

Por otra parte, como se comentó, la idea original era usar la Raspberry previamente descrita en 3.3.3.1. Por lo que no habría mucho problema a la hora de ubicarla debido a su reducido tamaño. El motivo del cambio es que tras numerosas pruebas fallidas y revisiones de código se llegó a la conclusión de que podía existir algún tipo de comunicación errónea entre la Raspberry Pi y las RoboClaw. Un fallo en la conexión de puertos o algún tipo de desconexión.

Para realizar la prueba se configuró el PC analizado en el punto 3.3.3.2, sin embargo, el comportamiento del robot mejoraba, pero no era óptimo, realizaba pequeños cambios de trayectoria y no obedecía los comandos dentro del tiempo de respuesta esperado. Este resultado provocó un estudio de los componentes a nivel electrónico.

4. DESARROLLO DEL TRABAJO

Primero se comprobó el cableado: Varios cables fueron sustituidos debido a su insuficiente diámetro, y el *hub* al que se conectaban las RoboClaw fue eliminado, porque su funcionamiento era errático, provocaba desconexiones.



Figura 4.16: Ubicación del hub desechado.

Tras volver a analizar las RoboClaw observamos que una de las dos no respondía bien. Las originales eran de 15 voltios y se sustituyeron por las actuales de 30 voltios.

Usando la aplicación Basicmicro Motion Studio, se descubrió un parámetro que indicaba un nivel bajo, "Logic battery". Esto hizo comprobar en el manual que hacía falta reforzar la alimentación de las RoboClaw.

El manual indica que bajo cargas pesadas, como es el caso, la alimentación principal puede sufrir caídas de tensión, causando bajones lógicos, teniendo las controladoras un comportamiento errático. Una fuente de alimentación separada de los motores puede remediar estos problemas.

La conexión original se realizó de la siguiente manera:

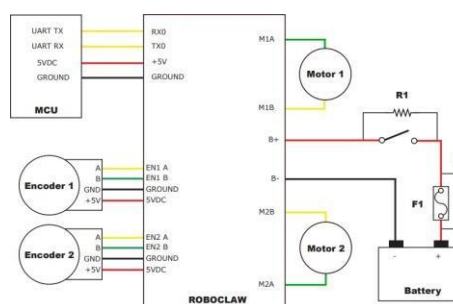


Figura 4.17: Diagrama de conexión de las RoboClaw.

A la que tras consultar el manual se le añadió el conexionado propuesto en la siguiente imagen:

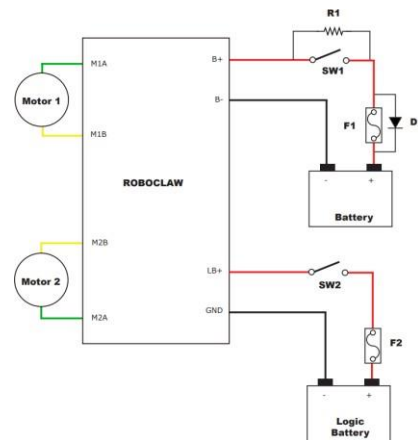


Figura 4.18: Diagrama de conexión de la Logic Battery de las RoboClaw.

Todos los puertos se pueden ver en la siguiente imagen real de una RoboClaw:

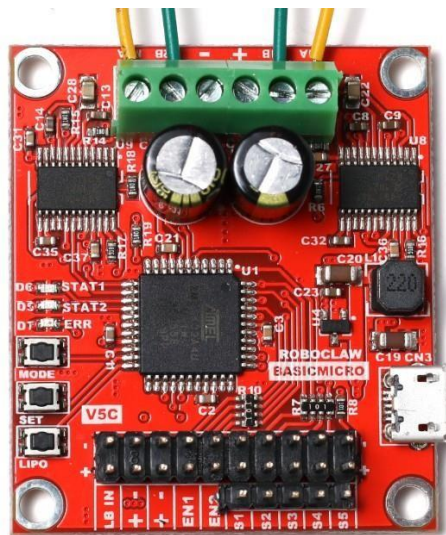


Figura 4.19: Imagen real de una controladora RoboClaw.

Tras estos cambios se consiguió el comportamiento esperado.

Sin embargo, este trabajo reveló un problema de diseño, la ubicación de las RoboClaw. Como se puede ver, su ubicación no permite realizar su correcta manipulación. Cuando se cambiaron las dos y se añadió el nuevo cableado, fue una tarea difícil. Por lo que se propondrá en las líneas de mejora una reubicación de las RoboClaw. Se puede ver la problemática en las siguientes imágenes:



(a) Ubicación de la RoboClaw (b) Ubicación de la RoboClaw en detalle

Figura 4.20: Ubicación de las controladoras en la plataforma

También, hablando de diseño, se añadieron varios enchufes internos para poder añadirle más funcionalidades a la plataforma. De esta forma se puede colocar en el PC en el interior y que no haya problemas tanto para cargar, como para conectar cualquier otra herramienta.



Figura 4.21: Imagen de los enchufes interiores.

Y también se sacó a un panel exterior el cargador de forma que cada vez que se quisiera cargar el robot no hiciera falta pasar el cable por dentro, y se añadieron algunas rejillas de ventilación. Se pueden ver en la siguiente imagen:



Figura 4.22: Panel lateral con el enchufe del cargador y las rejillas de ventilación.

5

RESULTADOS Y CONCLUSIONES

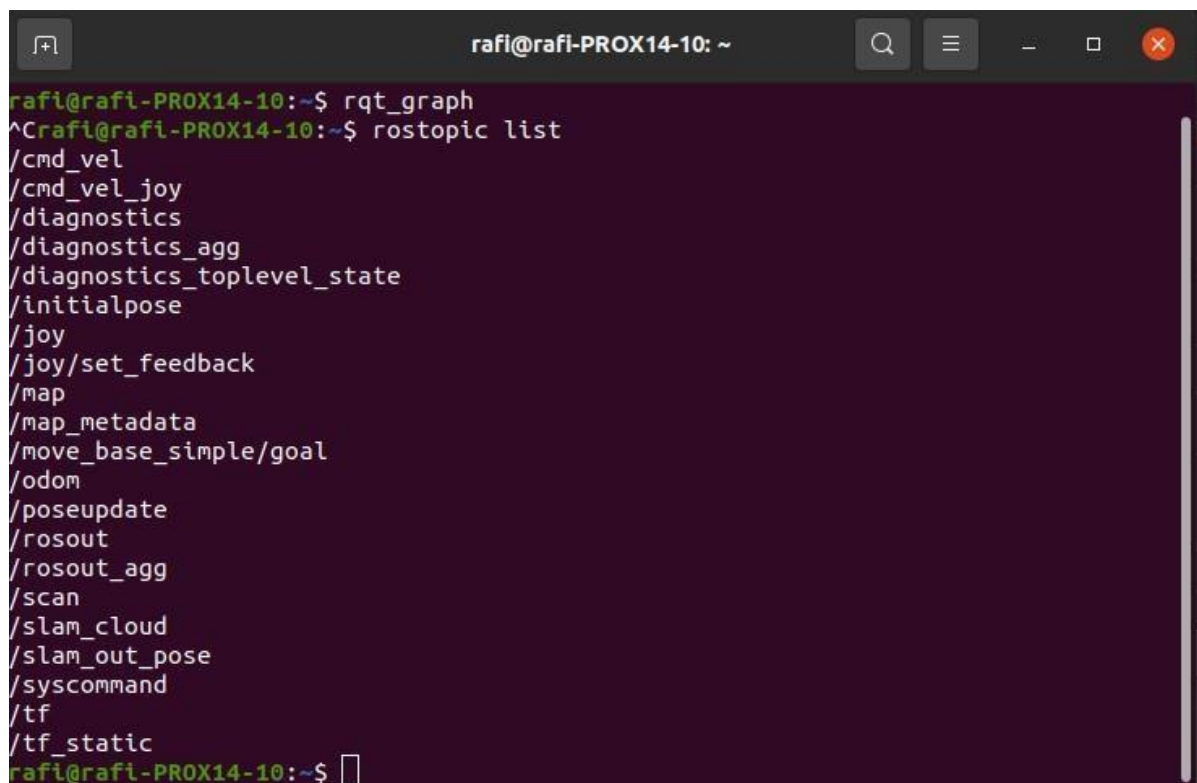
En este capítulo se exponen las conclusiones y resultados realizados una vez terminada la etapa de desarrollo.

5. RESULTADOS Y CONCLUSIONES

Es complejo mostrar resultados de un robot y cuya finalidad es ser usado sin entorno de simulación. Sin embargo, se mostrarán algunos topics y capturas de programas para comprobar el robot en funcionamiento.

Para verificar esto, se han recurrido a distintas herramientas como `rqt_graph`, `rqt_plot`, y el comando `rostopic list` para mostrar el contenido de los topics.

De esta forma, con el robot funcionando podemos ver todos los topics que se encuentran en uso escribiendo en la terminal `rostopic list`:

A terminal window titled 'rafi@rafi-PROX14-10: ~' with standard window controls. The terminal shows the command 'rostopic list' being executed, which outputs a list of active ROS topics. The topics listed are: /cmd_vel, /cmd_vel_joy, /diagnostics, /diagnostics_agg, /diagnostics_toplevel_state, /initialpose, /joy, /joy/set_feedback, /map, /map_metadata, /move_base_simple/goal, /odom, /poseupdate, /rosout, /rosout_agg, /scan, /slam_cloud, /slam_out_pose, /syscommand, /tf, and /tf_static. The prompt 'rafi@rafi-PROX14-10:~\$' is visible at the bottom.

```
rafi@rafi-PROX14-10:~$ rqt_graph
^Crafi@rafi-PROX14-10:~$ rostopic list
/cmd_vel
/cmd_vel_joy
/diagnostics
/diagnostics_agg
/diagnostics_toplevel_state
/initialpose
/joy
/joy/set_feedback
/map
/map_metadata
/move_base_simple/goal
/odom
/poseupdate
/rosout
/rosout_agg
/scan
/slam_cloud
/slam_out_pose
/syscommand
/tf
/tf_static
rafi@rafi-PROX14-10:~$
```

Figura 5.1: Nodos activos con la plataforma en funcionamiento.

También se puede observar usando el topic `"/diagnostics_agg"` el estado de los dispositivos conectados, por ejemplo, podemos comprobar que las RoboClaw tienen un funcionamiento correcto.

```
^Craft@rafi-PROX14-10:~$ rostopic echo /diagnostics_agg
header:
  seq: 118
  stamp:
    secs: 1693397918
    nsecs: 594626839
  frame_id: ''
status:
-
  level: 0
  name: "/Roboclaw"
  message: "OK"
  hardware_id: ''
  values:
    -
      key: "roboclaw_node: Vitals"
      value: "Normal"
-
  level: 0
  name: "/Roboclaw/roboclaw_node: Vitals"
  message: "Normal"
  hardware_id: "Roboclaw"
  values:
    -
      key: "Main Batt V:"
      value: "26.7"
    -
      key: "Logic Batt V:"
      value: "4.4"
    -
      key: "Temp1 C:"
      value: "32.8"
    -
      key: "Temp2 C:"
      value: "32.8"
-
  level: 0
  name: "/Other"
  message: "OK"
  hardware_id: ''
  values:
    -
      key: "joy_node: Joystick Driver Status"
      value: "OK"
-
  level: 0
  name: "/Other/joy_node: Joystick Driver Status"
  message: "OK"
  hardware_id: "none"
  values:
    -
      key: "topic"
      value: "/joy"
    -
      key: "device"
      value: "/dev/input/js0"
    -
      key: "device name"
      value: ''
    -
      key: "dead zone"
      value: "0.3"
    -
      key: "autorepeat rate (Hz)"
      value: "20"
    -
      key: "coalesce interval (s)"
      value: "0.001"
    -
      key: "recent joystick event rate (Hz)"
      value: "0"
    -
      key: "recent publication rate (Hz)"
      value: "19.8637"
    -
      key: "subscribers"
      value: "1"
    -
      key: "default trig val"
      value: "False"
    -
      key: "sticky buttons"
      value: "False"
-
  key: "Temp1 C:"
  value: "32.8"
-
  key: "Temp2 C:"
  value: "32.8"
```

Figura 5.2: Estado de los dispositivos conectados

5. RESULTADOS Y CONCLUSIONES

Y usando `rqt_graph`, podemos observar todos los nodos y topics que se encuentran activos durante el proceso.

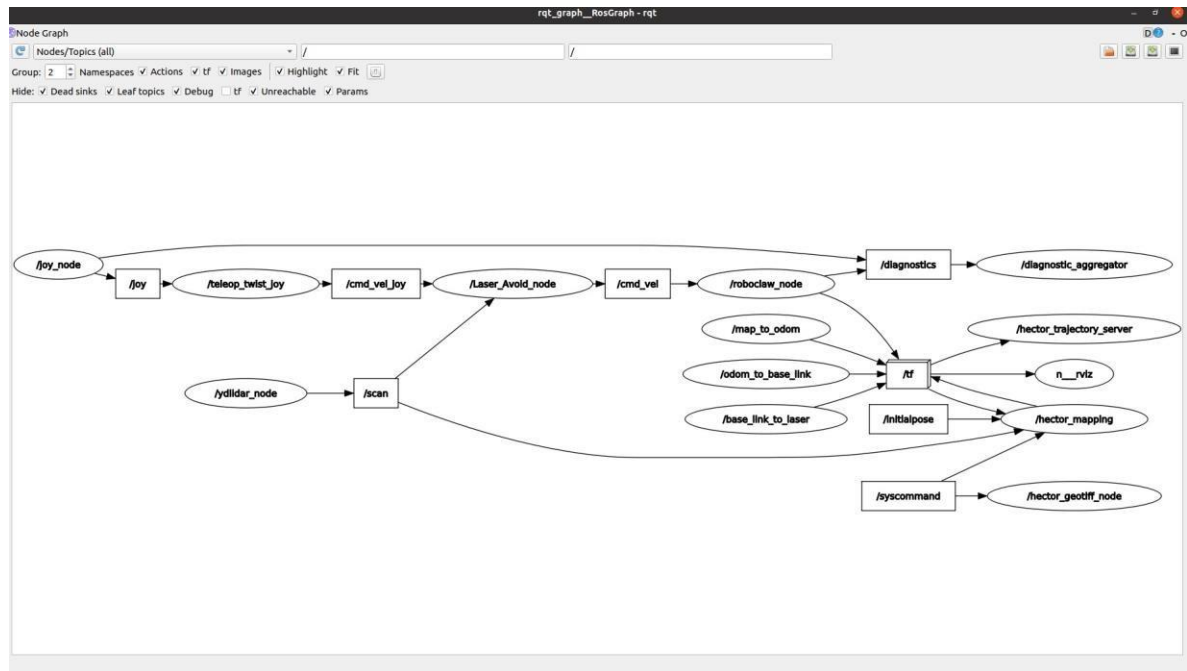


Figura 5.3: Grafo de funcionamiento de `roboclaw_ros`.

Además, como se comentó en el apartado del paquete "ydlidar" 4.2.2, a este paquete se le añadió el paquete "hector_slam". Realmente, esto no era un requerimiento del proyecto, pero se ha realizado un pequeño avance en esta línea debido a que era muy interesante que el robot pudiera mapear. Desde luego, hay que dedicarle más tiempo, ajustar mejor los parámetros y personalizar el visualizador RViz y el paquete en general, pero se han obtenido unos resultados interesantes.

En primer lugar se mostrará la imagen extraída del LiDAR, y la segunda imagen será el mapeo de la habitación con el LiDAR tras pasearlo muy brevemente. Cabe destacar que no se llegó a pasear por toda la sala sino que simplemente se quería hacer una prueba en unos pocos metros cuadrados.

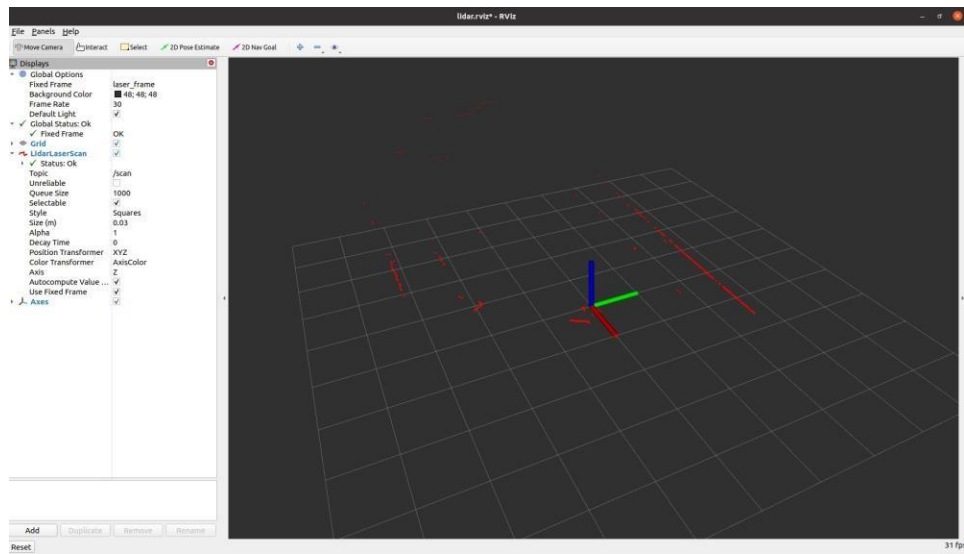


Figura 5.4: Medidas del LiDAR en RViz.

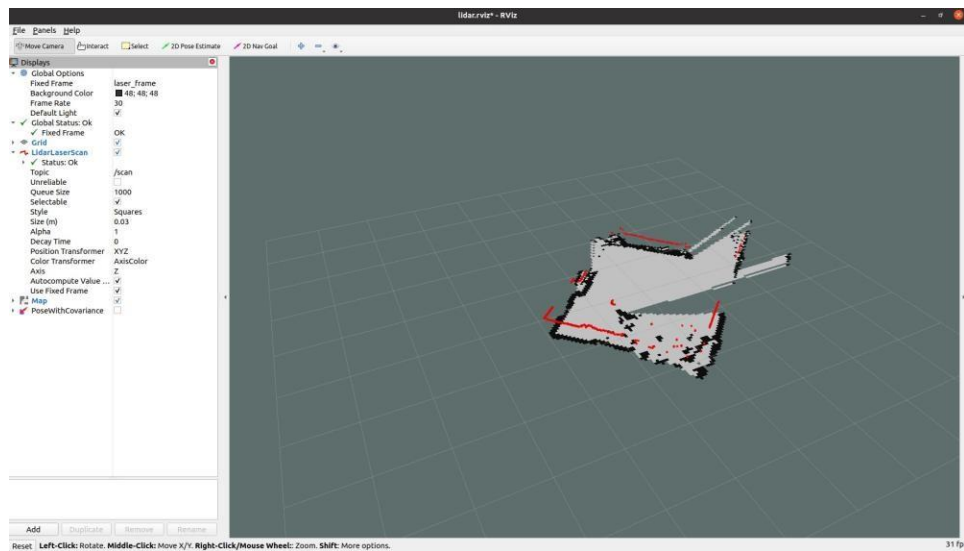


Figura 5.5: Mapeo con hector_slam en RViz.

Para ver el mapa, como se puede ver a la izquierda de las imágenes, se ha activado la visualización del topic /map.

5. RESULTADOS Y CONCLUSIONES

Usando "rqt_plot" podemos observar también la velocidad que se envía a través del topic "cmd_vel".

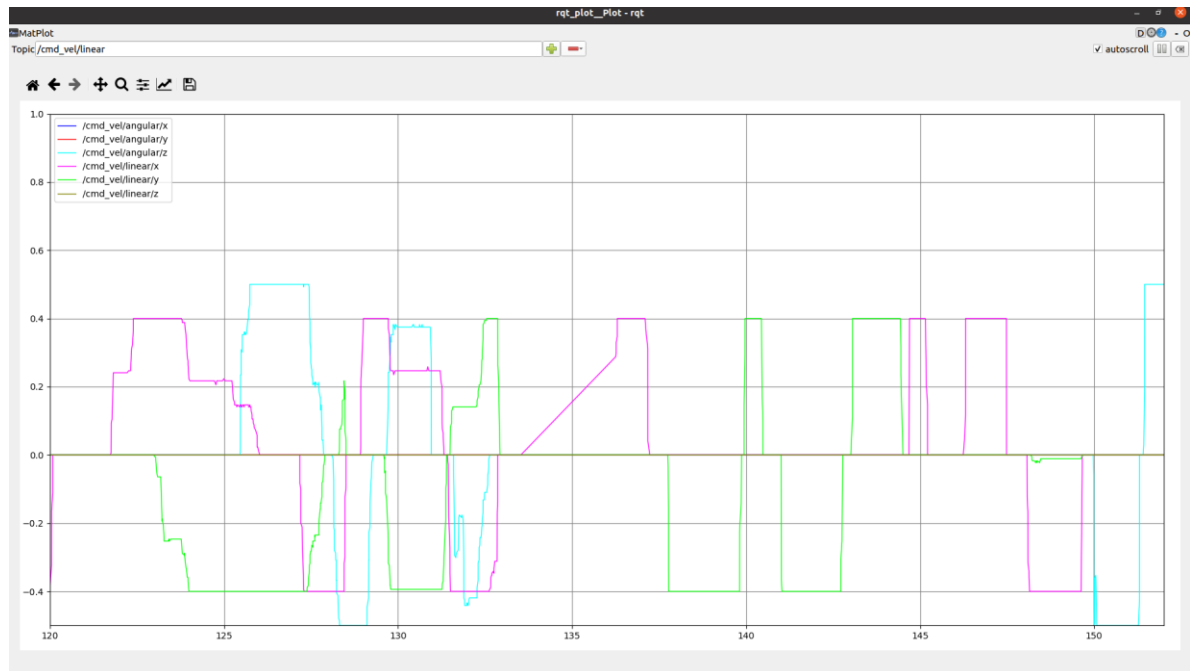


Figura 5.6: Salida de /cmd_vel con rqt_plot.

Como podemos observar, salen en la leyenda las velocidades angulares y lineales para los tres ejes, sin embargo nuestro robot solo puede desplazarse en el plano X-Y y rotar con respecto al eje Z, por eso son estos tres los comandos que aparecen.

Podemos observar cómo el robot no supera la velocidad de 1 m/s marcada en los requerimientos.

También, el chasis quedó terminado como se requería:



Figura 5.7: Robot funcional.

Además, aunque el PC esté encima del robot, esto es únicamente para las pruebas, como ha quedado espacio en el interior del robot, se puede configurar desde fuera y guardarlo dentro del armazón para manejar la plataforma.

Por último, en la plataforma Github del departamento se ha dejado el código junto a un manual de uso, para facilitar la labor a los próximos compañeros.

En conclusión, ha sido un trabajo complejo porque ha combinado problemas de software con problemas electrónicos y mecánicos y a veces ha sido difícil distinguir la causa de un comportamiento errático. Se han cumplido los requerimientos marcados en 2.1, de forma que no sólo continua el proyecto del robot de apoyo, si no que el departamento también tiene una plataforma pesada manejable con muchas posibilidades de investigación.



LÍNEAS DE TRABAJO FUTURAS

Como se ha ido comentando a lo largo de esta memoria, hay varios elementos a mejorar y ampliar. El objetivo de esta sección es dejar propuestas que signifiquen un aporte al trabajo realizado.

En primer lugar, sería una excelente idea añadir localización y mapeo al robot. Si bien el paquete funciona, sería positivo dedicar más tiempo e investigación a calibrar todo correctamente para poder aprovechar al máximo este servicio. Sería necesario tanto completar el desarrollo de la odometría como ajusta los parámetros del paquete.

En segunda lugar, cambiar el PC utilizado por algún tipo de PC compacto y ligero tipo NUC¹, y añadir el launch principal al código de boot, de forma que nada más encenderse el ordenador el usuario no tenga que escribir ningún comando y simplemente pueda mover el robot con el joystick.

También sería interesante añadir más funcionalidades, en especial en el joystick con todos los botones que tiene. Una muy útil podría ser que cuando se pulse un botón no se activa la evasión de obstáculos. Esto vendría bien en entornos estrechos como un pasillo. También una vez que realizara un mapa correctamente que simplemente presionando un botón se guardara ese mapa en el ordenador. O por ejemplo que con los botones *r1* y *l1* se pudiera aumentar y disminuir la velocidad máxima del robot dentro de unos límites.

Por otra parte, sería interesante reorganizar algunos elementos de la plataforma, especialmente las RoboClaw, cuyo acceso es complejo.

Lo interesante de este proyecto es el gran abanico de posibilidades que permite para adaptarse a cualquier necesidad o requerimiento específico.

¹Una NUC (Next Unit of Computing) es una computadora compacta y altamente integrada.

REFERENCIAS

- [1] Taheri, H., Qiao, B., & Ghaeminezhad, N. (2015). Kinematic model of a four mecanum wheeled mobile robot. *International journal of computer applications*, 113(3), 6-9.
- [2] Shah, A. (2019). *Hector mapping in ROS using YDLidar*
X4.<https://archit0994.wixsite.com/architshah/post/manage-your-blog-from-your-live-site>
- [3] BASICMICRO. (2015). *User Manual - Basicmicro*
https://downloads.basicmicro.com/docs/roboclaw_user_manual.pdf
- [4] Sonyccd. (s. f.). *GitHub - sonyccd/roboclaw_ros: Ros for Roboclaw*. GitHub.
https://github.com/sonyccd/roboclaw_ros
- [5] DoanNguyenTrong. (s. f.). *GitHub - DoanNguyenTrong/roboclaw_ros: Ros for Roboclaw*. GitHub.
https://github.com/DoanNguyenTrong/roboclaw_ros
- [6] Eaibot. (s. f.). *GitHub - EAIBOT/Ydlidar: The driver of Ydlidar for ROS on the Linux !* GitHub.
<https://github.com/EAIBOT/ydlidar>
- [7] Newman, J. [Articulated Robotics]. (2022, 4 de noviembre). ROS Tutorial (ROS1) - ROS Noetic 2H30 [Crash Course] [Video]. Youtube. <https://www.youtube.com/watch?v=F5XlNiCKbrY>
- [8] Renard, E. [Robotics Back-End]. (2022, 24 de febrero). Easy wireless control for your homemade robot! [Video]. Youtube. <https://www.youtube.com/watch?v=wfdJAYTMTdk>
- [9] Flores, D. (2023). Diseño e implementación de un sistema de control de energía para un robot móvil con manipulador industrial [Manuscrito no publicado]. Escuela Técnica Superior de Ingeniería de Telecomunicación. Universidad de Málaga.
- [10] Serrano, A. (2021). Control de una plataforma omnidireccional para un manipulador móvil [Manuscrito no publicado]. Departamento de Ingeniería de Sistemas y Automática. Escuela de Ingenierías Industriales. Universidad de Málaga.
- [11] *Documentation - ROS Wiki*. (s. f.). <http://wiki.ros.org/>
- [12] 123electric. (2023). Products - 123electric. <https://123electric.eu/products/>
- [13] Hernández, G. (2023). Garimi: Así es el robot humanoide para cuidar a los adultos mayores a falta de suficientes trabajadores. . . Xataka México. <https://www.xataka.com.mx/robotica-e-ia/garimi-asi-robot-humanoide-para-cuidar-a-adultos-mayores-a-falta-suficientes-trabajadores-salud>

<https://revistaderobots.com/robots-y-robotica/robot-nao-caracteristicas-y-precio/BASICMICRO>.
User manual - basicmicro, 2015.

- [15] Su, E. (2019). Nadine, el robot humanoide capaz de cuidar de personas con demencia. The Objective. <https://theobjective.com/tecnologia/2016-03-08/nadine-el-robot-humanoide-capaz-de-cuidar-de-personas-con-demencia/>
- [16] Caballar, R. D. (2023). What is the Uncanny Valley? IEEE Spectrum. <https://spectrum.ieee.org/what-is-the-uncanny-valley>
- [17] Raspberry Pi. (s.f.). Raspberry Pi Documentation. <https://www.raspberrypi.com/documentation/>
- [18] DIY Baterías LiFePO4. (2021). ¿Qué es LIFEP04? - DIY baterías LIFEP04 . DIY Baterías LiFePO4. <https://www.bateriaslifepo4.com/que-es-el-lifepo4/>
- [19] Global World Logistic, <https://shop.gwl.eu>, Ing. Andel Jiri. (s. f.). Battery and solar distributor in EU | [shop.GWL.eu](https://shop.gwl.eu/). <https://shop.gwl.eu/>