
# UNIVERSIDAD EUROPEA DE MADRID

**ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO**

**ÁREA INGENIERÍA INDUSTRIAL**


## GRADO EN INGENIERÍA EN

## SISTEMAS INDUSTRIALES

## TRABAJO FIN DE GRADO

# DISEÑO DE UNA ARQUITECTURA ROS PARA UN ROBOT DE DIRECCIONAMIENTO ACKERMANN


**Alumno: Julio José Girón Mendoza**

**Director: Sergio Bemposta Rosende**



**JUNIO 2024**

**TÍTULO**: DISEÑO DE ARQUITECTURA ROS PARA UN ROBOT DE DIRECCIONAMIENTO ACKERMANN

**AUTOR**: JULIO JOSÉ GIRÓN MENODZA

**DIRECTOR DEL PROYECTO**: SERGIO BEMPOSTA ROSENDE

**FECHA**: 10 de junio de 2024

# RESUMEN

El Trabajo Fin de Grado presentado se centra en desarrollar una arquitectura ROS adaptada a un robot con sistema de direccionamiento Ackermann, explorando su aplicación en un contexto poco común dentro del entorno de ROS. Este proyecto transforma un coche de carreras radiocontrolado de escala 1/10 en una plataforma robótica. Se utilizan microcontroladores externos, como Arduino UNO y Arduino Nano, que se comunican con una Raspberry Pi para gestionar sensores y actuadores. Estos dispositivos capturan datos de orientación y velocidad, permitiendo determinar la posición del robot mediante odometría y otras técnicas de localización, considerando las transformaciones del robot. Además, se implementan técnicas de control PID para optimizar la respuesta del sistema a las instrucciones de control de velocidad. Se emplea el algoritmo SLAM para facilitar la recolección de datos espaciales del entorno, como el escaneo de mapas. El sistema permite el control del robot mediante teleoperación y se desarrollan interfaces gráficas para la visualización y manejo de datos en tiempo real, utilizando las herramientas, mensajería y librerías proporcionadas por el framework ROS. Los resultados validan la aplicabilidad de la arquitectura ROS en robots con direccionamiento Ackermann y sugieren futuras mejoras para optimizar y expandir las capacidades del sistema desarrollado, incluyendo adaptaciones para operar en entornos exteriores con mayor precisión.

**Palabras clave:** ROS, Direccionamiento Ackerman, SLAM, Arduino, Arquitectura Software, Interfaz gráfica.

# ABSTRACT

The thesis presented focuses on developing a ROS architecture adapted to a robot with Ackermann steering, exploring its application in an uncommon context within the ROS environment. This project transforms a 1/10 scale radio-controlled race car into a robotic platform. External microcontrollers, such as Arduino UNO and Arduino Nano, communicate with a Raspberry Pi to manage sensors and actuators. These devices capture orientation and speed data, allowing for the determination of the robot's position through odometry and other localization techniques, considering the robot's transformations. Additionally, PID control techniques are implemented to optimize the system's response to speed control instructions. The SLAM algorithm is used to facilitate the collection of spatial data from the environment, such as map scanning. The system allows for the robot to be controlled via teleoperation, and graphical interfaces are developed for real-time data visualization and management, using the tools, messaging, and libraries provided by the ROS framework. The results validate the applicability of the ROS architecture in robots with Ackermann steering and suggest future improvements to optimize and expand the capabilities of the developed system, including adaptations for operating in outdoor environments with greater precision.

**Key words:** ROS, Ackermann Steering, SLAM, Arduino, Software Architecture, GUI

**AGRADECIMIENTOS**

Este proyecto no habría sido posible sin el apoyo de mi familia, compañeros y amigos.

Estaré eternamente agradecido con mi tutor, Sergio Bemposta, quien me brindó su ayuda y orientación a lo largo del desarrollo de este proyecto.

# Índice

# Índice de Figuras

# Índice de Tablas

# Capítulo 1. INTRODUCCIÓN

## 1.1 Planteamiento del problema

En el diseño de robots, ya sea para robótica educativa e investigación, las personas frecuentemente optan por el framework estándar ROS (Robot Operating System). Este marco de trabajo no solo destaca por la estructura que presenta para el desarrollo de software en robótica, sino que también se beneficia de una gran comunidad que desarrolla constantemente paquetes (programas específicos) para cumplir las distintas funciones de un sistema.

Estos paquetes pueden ser reutilizados en otros proyectos que requieran funcionalidades similares a las actuales. Además, siendo un framework de uso libre, ROS se ha convertido en la herramienta esencial para gestionar los diversos sistemas de un robot.

El portal de wiki ROS describe su framework como *una estructura más de sistema* (Juan Eduardo Rivas, 2021), reconociendo así su papel entre otras estructuras y frameworks ampliamente utilizados en desarrollo e investigación. Sin embargo, su principal objetivo de facilitar la reutilización de código ha llevado a muchos a optar por este marco de trabajo para desarrollar sus prototipos iniciales.

En el ámbito de los robots móviles, ROS proporciona paquetes extensos y documentación detallada sobre modelos de robots específicos que son ampliamente utilizados por sus aplicaciones prácticas. Un ejemplo de esto son los robots diferenciales, que destacan por su agilidad de maniobra y amplia gama de aplicaciones.

Su sencilla construcción y cinemática facilitan un control directo que depende de la velocidad de las ruedas. Al regular esta velocidad, el robot puede girar sobre su propio eje, proporcionando un control preciso, aunque algo rígido (Edisonsasig, 2023). La implementación de estos robots es relativamente sencilla, lo que ha llevado a que exista una abundante documentación sobre ellos.

Sin embargo, esta focalización en el desarrollo de tales arquitecturas ha relegado a un segundo plano otras configuraciones convencionales, como los que cuentan con sistema de direccionamiento Ackermann.

Estos últimos, aunque menos explorados en el contexto de ROS, también podrían beneficiarse significativamente de una integración más profunda con este framework, dada su capacidad para manejar movimientos complejos en entornos estructurados como carreteras y pistas.

Debido a la falta de documentación, es común buscar en internet o en repositorios como GitHub y encontrar muy poca información disponible. En ocasiones, aunque se encuentre información relevante, suelen ser aplicaciones sofisticadas y avanzadas que resultan difíciles de comprender o de utilizar como punto de partida.

## 1.2 Objetivo del proyecto

Ante la problemática previamente mencionada, este proyecto se centra en el desarrollo de una arquitectura ROS para un robot con sistema de direccionamiento Ackermann. El objetivo es proporcionar, a través de un repositorio de GitHub, una arquitectura sencilla que sirva como recurso para quienes estén interesados en diseñar robots que requieran este tipo de sistemas.

La arquitectura por realizar desea mostrar los resultados del desarrollo y adaptación de un coche de carreras radiocontrolado de escala 1/10, donde se utilizarán los parámetros físicos y mecánicos del vehículo seleccionado.

Para dar coherencia y un nombre distintivo a todo el desarrollo realizado, la adaptación y arquitectura diseñada se denominará "Sundybot".

Para adaptar el vehículo, se diseñarán las piezas utilizando programas de diseño asistido por computadora (CAD). El propósito es documentar y mostrar cómo se puede realizar una adaptación sencilla y efectiva a este tipo de vehículos que normalmente no están diseñados para incorporar componentes electrónicos en su estructura.

Se documentarán los esquemas de conexiones realizados con el objetivo de proporcionar una guía sobre la conexión y selección de componentes para robots de este tipo

Para desarrollar la arquitectura mediante ROS, se utilizarán los lenguajes C++ y Python3, Los datos capturados por los sensores serán procesados y emitidos siguiendo la estructura de ROS, utilizando los formatos de mensajes específicos que ofrece el framework. La arquitectura proporcionara los mensajes básicos de navegación que debe emplear un robot móvil.

Por lo tanto, se pretende desarrollar y facilitar configuraciones de software que permitan una visualización de los parámetros calculados y sensorizados por el robot, utilizando las herramientas graficas de ROS.

Se pretende diseñar dos interfaces gráficas especializadas para la gestión y análisis de los datos operativos del vehículo, sin utilizar las herramientas graficas proporcionadas por ROS.

Como complemento a la arquitectura desarrollada, se ha establecido como objetivo el uso de paquetes externos para la Localización y Mapeo Simultáneo (SLAM).

Dado que el proyecto no incluye la navegación autónoma en su arquitectura, se establece como objetivo el control del vehículo mediante teleoperación.

Por último, se desea Alojar la arquitectura completa y todos los paquetes de ROS en un repositorio de GitHub, con las explicaciones necesarias para facilitar su uso. Este objetivo está dirigido a garantizar que los usuarios puedan incorporar estos programas en sus propios proyectos fácilmente, preservando la estructura esencial de ROS.

El proyecto está específicamente diseñado para operar en entornos controlados, y no tiene como objetivo abordar la navegación en espacios abiertos. En consecuencia, no se contempla el estudio ni la integración de sensores de posicionamiento exterior, como el GPS, ni su compatibilidad con ROS.

### 1.2.1 Objetivos específicos

Para complementar los objetivos generales del proyecto, este apartado detalla los objetivos específicos. Estos están planteados para especificar el alcance y limitaciones del proyecto. Basado en los objetivos, se establecen pautas que guiarán todas las fases de desarrollo y ejecución del proyecto.

Los objetivos específicos del proyecto son los siguientes:

- Diseño y desarrollo de una base adecuada para instalar la electrónica necesaria en la adaptación de un coche de carreras radiocontrolado de escala 1/10.
    - El diseño empleado debe adaptarse correctamente al vehículo sin interferir en sus mecanismos de movimiento y direccionamiento
    - El diseño debe asegurar una adecuada adaptación de los componentes electrónicos, cuidando su integridad y garantizando que no se vean afectados por la modificación.
    - El diseño debe contemplar las conexiones eléctricas para proporcionar un espacio correspondiente para el enlace y almacenamiento de estas.
    - El diseño contara con diferentes piezas adaptativas las cuales no serán diseñadas.
- Diseño y desarrollo de esquemas de conexiones eléctricas y de comunicación, se debe cumplir lo siguiente:
    - El diseño debe ser claro y preciso para facilitar la comprensión y la implementación. Cada componente y conexión debe estar claramente identificado y ubicado de manera lógica dentro del esquema.
    - El diseño debe permitir modificaciones o expansiones futuras sin grandes alteraciones. Este debe ser adaptable a posibles mejoras o cambios en los componentes.
    - El diseño debe asegurar que el esquema de conexiones sea compatible con todos los componentes y dispositivos con los que interactuará, respetando las especificaciones técnicas de cada uno.
- Para la configuración y preparación de entornos destinados a la visualización graficas usando ROS, se debe cumplir con lo siguiente:
    - El objetivo es proporcionar una visualización clara y comprensible de los datos del robot y su entorno.
    - La configuración debe ser flexible, permitiendo a los usuarios personalizar la visualización según sus necesidades específicas.
    - Facilitar la depuración y el diagnóstico de problemas en el robot.

- Para desarrollar la arquitectura de ROS se deben cumplir con los siguientes objetivos:
    - La arquitectura deberá contener un diseño inicial que sea sencillo, facilitando la comprensión y el manejo de la información sin recurrir a un número excesivo de nodos y tópicos.
    - Se deberá asegurar que cada programa esté acompañado de documentación detallada a través de comentarios en el código, mejorando así la accesibilidad y la mantenibilidad.
    - La arquitectura debe permitir una comunicación fluida entre nodos, utilizando tópicos, servicios y acciones para facilitar la interoperabilidad y adaptabilidad del sistema.
- Para la incorporación de paquetes externos de complemento:
    - No se detallará el funcionamiento de los paquetes externos a menos que hayan sido modificados. En tal caso, se proporcionará una explicación clara de las alteraciones realizadas y su impacto en la funcionalidad del paquete.
- Para la incorporación y subida de un repositorio de GitHub
    - El repositorio contará con una estructura clara y ordenada, facilitando la navegación y el acceso a los archivos.
    - Incluirá documentación detallada de todos los paquetes externos utilizados, asegurando que los usuarios puedan identificar y gestionar fácilmente las dependencias necesarias para el proyecto.

## 1.3 Justificación y motivación

Una de las mecánicas de direccionamiento más comunes a nivel de sistemas de locomoción en robótica es el direccionamiento Ackermann. Aunque la mecánica ha sido utilizada en vehículos durante mucho tiempo, ha cobrado gran importancia en los últimos años debido al avance en la navegación de coches autónomos. Este interés renovado se debe tanto a la investigación académica como al desarrollo de competiciones patrocinadas por desarrolladores de tecnología, como NVIDIA.

Estos últimos años, aprovechando el avance en hardware gráfico como el Jetson Nano y el Jetson orion, NVIDIA ha propuesto a diversas universidades la organización de competiciones con coches de carrera RC a escala 1/10. El objetivo es demostrar la capacidad de su hardware para aplicaciones reales, como el cálculo de trayectorias dinámicas en tiempo real (nvidia, 2020).

Paralelamente, los últimos avances en inteligencia artificial han impulsado la navegación autónoma en vehículos, permitiendo el reconocimiento de señales de tránsito en tiempo real. Estas innovaciones suponen la importancia y la necesidad de desarrollar sistemas de direccionamiento eficientes y adaptativos.

Con base en estas justificaciones, mi motivación personal desde que comencé a estudiar robótica ha sido automatizar un coche de carreras para que pueda completar

circuitos de manera autónoma. Al buscar información y explorar proyectos existentes, encontré que muchos de ellos eran excesivamente complejos, lo que complicó desarrollar una base en la cual poder partir. Inspirado por compañeros que trabajaban en sistemas de visión y detección de objetos usando Python, reconocí la necesidad de diseñar una arquitectura que pudiera complementar y potenciar sus proyectos.

## 1.4 Estructura del proyecto

En este apartado se explican brevemente los capítulos presentes en el documento. La distribución es la siguiente:

- **CAPÍTULO 2. MARCO TEÓRICO**
  En este capítulo se explican los conceptos técnicos necesarios que se encontraran desarrollados durante todo el proyecto.

- **CAPÍTULO 3. DISEÑO Y ARQUITECTURA HARDWARE**
  En este capítulo se encuentra la distribución y arquitectura hardware adaptada al vehículo, así como los componentes, sus especificaciones y diagramas de conexión al igual que los planos y piezas diseñadas para adaptar el vehículo.

- **CAPÍTULO 4. HERRAMIENTAS DE CONSTRUCCIÓN DE ARQUITECTURA SOFTWARE**
  Este apartado detalla todas las herramientas de código, IDEs (Entornos de Desarrollo Integrados) y lenguajes de programación utilizados en el proyecto.

- **CAPÍTULO 5. INSTALACIÓN Y PREPARACIÓN DE ENTORNO ROS**
  Este capítulo explica cómo preparar una Raspberry Pi 4 y una máquina virtual de Ubuntu en VMware. Se explica instalación de ROS Noetic.

- **CAPÍTULO 6. CONSTRUCCIÓN DE ARQUITECTURA EN ROS**
  Este capítulo detalla el diseño de modelos URDF, junto con el cálculo y la sensorización de los parámetros transmitidos entre nodos de ROS.

- **CAPÍTULO 7. RESULTADOS OBTENIDOS**
  Se contrastan los resultados de los diferentes cálculos y mensajes emitidos durante las pruebas de navegación y configuración de sensores.

- **CAPÍTULO 8. RESULTADOS OBTENIDOS**
  Se explica como acceder y poder clonar los diferentes paquetes desde GitHub.

- **CAPÍTULO 9. CONCLUSIONES**
  Se presentan las futuras líneas de trabajo y se autoevalúa el proyecto con el fin de ver si los objetivos fueron cumplidos.

- **BIBLIOGRAFÍA**
  Se colocan las diferentes fuentes consultadas durante la realización del proyecto en formato APA7.

- **ANEXOS**
  Este apartado contiene los códigos de los diversos nodos y microcontroladores, además de lanzadores. También se proporcionan enlaces a videos de las navegaciones ejecutadas durante la recopilación de resultados y al repositorio en GitHub.

# Capítulo 2. MARCO TEÓRICO

## 2.1 Arquitecturas Software y Hardware

**Arquitectura software**

Como observadores, muchos de nosotros hemos presenciado la construcción de edificios, casas o infraestructuras. Desde el punto de vista de la arquitectura como disciplina que se encarga del diseño y planificación de obras de construcción civil se puede decir que todos estos conceptos hacen referencia a la toma de decisiones antes de ser construidos (Pablo Huet, 2022).

Las arquitecturas a nivel de software comenzaron a introducirse en la década de los 60, adoptando el concepto similar al de la ingeniería civil. Sin embargo, en este contexto, se refiere a la planificación que utiliza modelos, patrones y abstracciones al diseñar productos finales de software con ciertas complejidades (Pablo Huet, 2022).

La importancia de esta arquitectura nace por la capacidad para proporcionar marcos de trabajo que ayuden en la toma de decisiones sobre diseño y desarrollo. Aunque la misma no sea visible, determina la eficiencia de un sistema (Miriam Martínez Canelo, 2024).

Los servicios dentro de este contexto son programas que realizan tareas automatizadas, responden a eventos de hardware o escuchan solicitudes de datos de otros programas. Estos servicios pueden ser solicitados mediante los dos componentes principales de esta arquitectura: el Frontend y el Backend (Computer Hope, 2023).

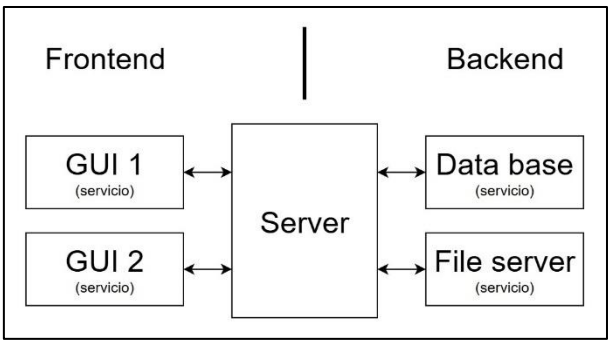



***Figura 1:*** *Diagrama de servicio, fontend y backend de una arquitectura sofware*

En base a esto según las aplicaciones se presentan diferentes estructuras software que pueden dividirse en los siguientes tipos:

- **Monolítica:** Esta estructura presenta una interconexión entre todas los componentes levantando una única unidad. Es especialmente útil para aplicaciones pequeñas debido a su simplicidad. Sin embargo, el problema surge cuando la red se vuelve más compleja (Miriam Martínez Canelo, 2024).

- **Basada en microservicios:** Los sistemas se dividen en servicios muy pequeños que se comunican entre sí. Esta estructura ofrece la ventaja de mantener la estabilidad y facilitar la implementación de actualizaciones a medida que aumenta la complejidad (Miriam Martínez Canelo, 2024).
- **Orientada a servicios:** Las funcionalidades se ofrecen como servicios independientes los cuales se combinan construyendo complejas aplicaciones.

Este tipo de estructura promueve la reutilización de servicios (Miriam Martínez Canelo, 2024).

- **De tres capas:** esta estructura se divide en tres subestructuras o capas: presentación, lógica de negocio y almacenamiento de datos. Esta estructura es utilizada en sistemas web donde la lógica de negocio funciona como intermediario entre el almacenamiento y el usuario (Miriam Martínez Canelo, 2024).

- **Sin servicios:** las instrucciones se ejecutan directa e individualmente desde la nube, esto es ideal para cargas de trabajo eventuales y altamente escalables (Miriam Martínez Canelo, 2024).

Estas estructuras son vistas en todas las aplicaciones software que presentan las aplicaciones que se tienen en nuestro dispositivos inteligente y no inteligente, entre las aplicaciones más comunes se tienen las siguiente:

- Red Social (Arquitectura Monolítica)
- Plataforma de e-commerce (Arquitectura basada en microservicios)
- Sistema bancario (Arquitectura orientada a servicios – SOA):
- Plataforma de aplicación empresarial (Arquitectura de tres capas)

**Arquitectura hardware**

Una vez definida la arquitectura de software, podríamos asumir que la arquitectura de hardware es un concepto similar, lo cual es en gran medida cierto.

Cabe recalcar que la estructura de un computador no es lo mismo que la arquitectura de hardware. Esta constituye la base conceptual y técnica que permite la creación de ordenadores y sistemas informáticos funcionales. Su estructura incluye componentes fundamentales como la CPU, buses, reloj, unidades de entrada y salida, memoria principal y sistema operativo (Isis Sulbarán, 2023).

Estos al igual que la estructura software presentan sus tipos de arquitectura en función de su construcción:

- Von Neumann
- Harvard
- RISC (Reduced Instruction Set Computer)
- CISC (Complex Instruction Set Computer)
- Paralela

La arquitectura de hardware implica la implementación de periféricos de entrada y salida que complementan el funcionamiento del computador. Estos periféricos incluyen dispositivos de entrada como micrófonos, memorias externas, teclados, ratones y mandos, así como dispositivos de salida como altavoces e impresoras. Para conectar estos periféricos, se emplean protocolos de comunicación tanto inalámbricos como por cable, utilizando extensores, hubs y otros medios.

Al diseñar sistemas de hardware, es crucial considerar la compatibilidad entre los sistemas de conexión y los periféricos seleccionados.

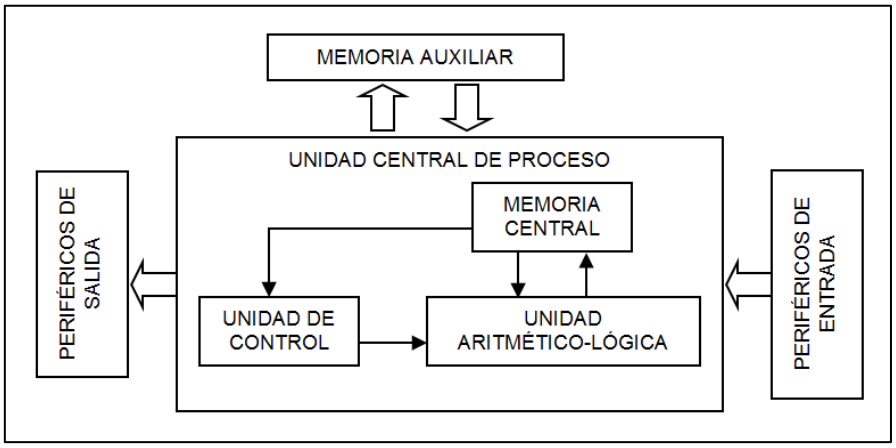



***Figura 2:*** *Arquitectura computacional sencilla, Figura tomada de https://edea.juntadeandalucia.es*

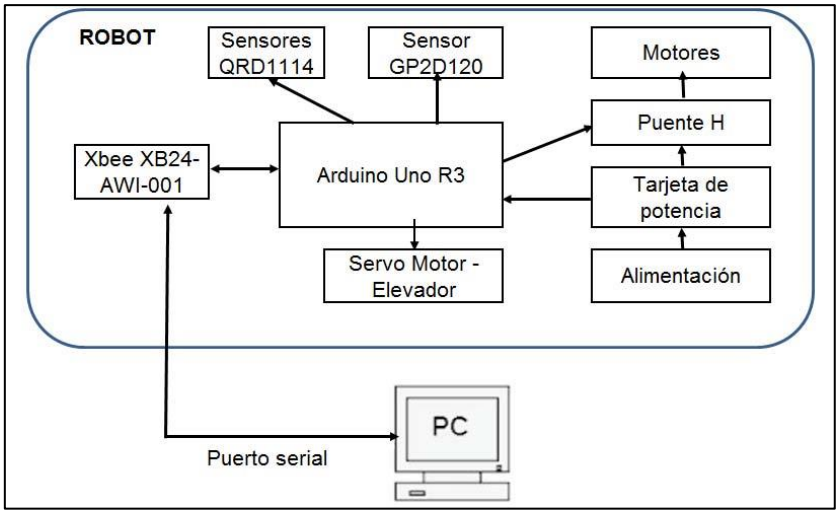



***Figura 3:*** *Arquitectura hardware sencilla, Figura tomada de https://www.researchgate.net*

## 2.2 ¿Qué es ROS?

Al igual que sistemas como Player, YARP, Orocos, Orca, MOOS y Microsoft Robotics Studio, La página oficial de wiki ROS define a ROS (Robotic Operating System) como un meta sistema operativo de código abierto destinado al desarrollo de robots.

Este sistema proporciona todos los servicios esenciales que se esperan de un sistema operativo convencional. Adicionalmente, ROS ofrece herramientas y bibliotecas diseñadas para facilitar la comunicación entre varias computadoras (robots) conforme a su estructura de mensajería específica (Juan Eduardo Rivas, 2021).

ROS no solamente contempla un método de comunicación, los tres métodos que utiliza ROS son los siguientes:

- RPC síncronos sobre servicios
- ***Transmisión asíncrona mediante tópicos***
- Guardado de datos en un servidor de parámetros

Es importante recalcar que ROS no garantiza por sí mismo un funcionamiento en tiempo real; sin embargo, puede adaptarse para operar bajo estas condiciones utilizando

protocolos como pr2_etherCAT, el cual se emplea en el robot de Willow Garage (Juan Eduardo Rivas, 2021).

ROS ha evolucionado a través de varias versiones o distribuciones que han mejorado con el tiempo, aunque su modo de funcionamiento básico se ha mantenido constante. Entre las versiones más recientes se encuentran Kinetic, Melodic y Noetic.


***Figura 4:*** *Distribuciones de ROS, Figura tomada de https://wiki.ros.org/Distributions*

Al ser un meta sistema cada versión de ROS es ejecutable sobre una versión de Ubuntu en específico. Aunque es ejecutable en otros sistemas operativos wiki ROS recomienda instalarlo sobre el sistema Ubuntu ya que es el sistema con soporte oficial (Chris Lalancette, 2023).

**Tópicos**

Los tópicos son canales de comunicación que se utilizan para enviar y recibir mensajes en ROS, funcionando similar a buses a los que se les debe asigna un nombre específico. Esto permite que los nodos accedan y compartan información eficientemente (Tully Foote, 2019).

Cada tópico tiene una tipificación o semántica definida, que puede ser de suscripción o de publicación. Los nodos involucrados en esta comunicación no necesitan conocer con quién están interactuando; simplemente acceden al tópico para publicar o recibir mensajes de manera estructurada. Además, los tópicos operan de forma unidireccional (Tully Foote, 2019).


***Figura 5:*** *Ejemplo tópico unidireccional*

Como se puede ver en la Figura 5, ROS gráficamente toma los tópicos como un cuadrado y los nodos como un rombo. La información es enviada del nodo 1 al nodo 2 mediante el tópico de manera unidireccional.

**Nodos**

Los nodos son programas ejecutables que procesan datos, con la capacidad de enviar y recibir información. Dentro de la estructura de ROS, los nodos son conocidos como suscriptores y publicadores, dependiendo de su función. Un nodo puede actuar exclusivamente como suscriptor o publicador, o puede desempeñar ambas funciones simultáneamente, también puede proveer o usar un servicio (Juan Eduardo Riva, 2021).

Los nodos pueden ser programados en C++ o en Python3, usando una librería cliente para comunicarse con otros nodos (Juan Eduardo Riva, 2021).

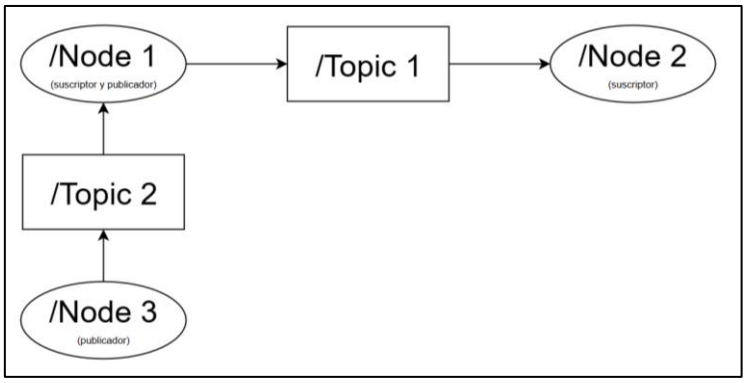



***Figura 6:*** *Ejemplo de semántica de nodos*

Como se muestra en la Figura 6, la disposición gráfica de ROS ilustra la apariencia de un nodo publicador y un suscriptor.

Sin embargo, es importante destacar que, para acceder a los tópicos, los nodos deben realizar peticiones como suscriptores o publicadores. (Tully Foote, 2019).

**Mensajes**
Un mensaje en ROS es un tipo de dato que es publicado por un nodo en un tópico, estos mensajes mantienen una estructura de datos simple el cual se guarda en uno o múltiples campos según el tipo de mensaje que sea.

Los mensajes que se admiten pueden ser primitivos: enteros, flotantes, booleanos, etc. También pueden ser arreglos o múltiples arreglos. Estos también pueden ser mensajes especial: velocidad, orientación, traslación, entre otros (Dorian Kurzaj, 2016).

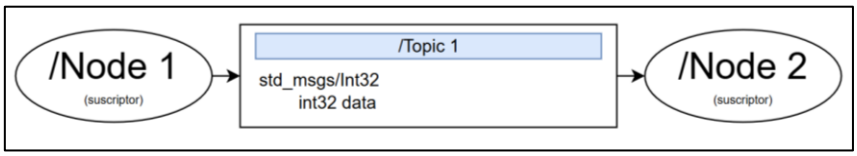



***Figura 7:*** *Ejemplo de tipo de mensaje*

Como se ilustra en la Figura 7, el Nodo 1 publica en el Tópico 1 para enviar un mensaje de tipo entero de 32 bits, al cual el Nodo 2 se suscribe posteriormente para recibir dicho valor.

### 2.2.1 ¿Qué es una arquitectura ROS?

La comunicación de múltiples nodos mediante tópicos representa la arquitectura del sistema robótico diseñado en ROS, sin embargo, más allá de esta estructura, ROS opera con una arquitectura maestra que trabaja sobre la principal, sugiriendo así la existencia de dos arquitecturas distintas dentro de ROS

- **Arquitectura de ROS Máster:** Esta es la principal, la cual se ejecuta permitiendo la comunicación de mensajes levantados por ROS

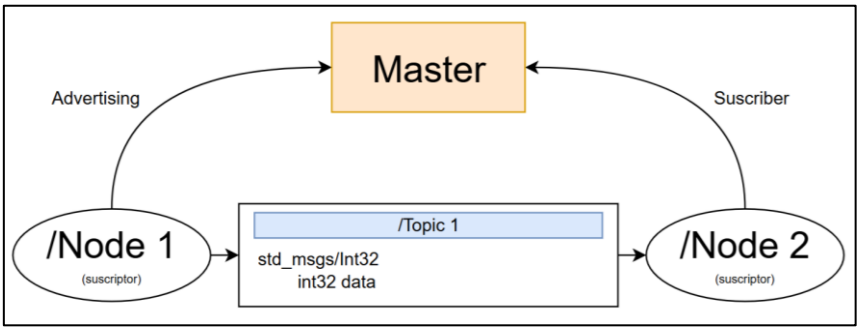



***Figura 8:*** *Arquitectura de ROS Máster*

- **Arquitectura de Comunicación:** Presenta la comunicación de los diferentes nodos mediante tópicos, dichos nodos pueden tener sub tópicos y sub-nodos.

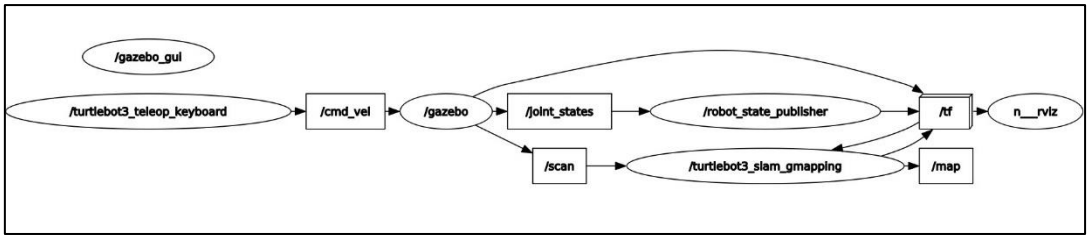



***Figura 9:*** *Arquitectura de SLAM con el Turtlebot*

## 2.3 Direccionamiento Ackermann

El direccionamiento de Ackermann es un mecanismo de control de movimiento de ruedas delanteras el cual permite un giro sueve y preciso con el menor consumo de energía. Este sistema lleva el nombre de Rudolf Ackermann quien fue el creador de este tipo de direccionamiento patentado en 1817 (HR motor, 2023).

La dirección es guiada mediante una barra localizada en el eje delantero del vehículo, esta permite que las ruedas giren en una dirección de ángulo deseada. Este concepto permite que al añadir otra barra se pueda el giro de las dos ruedas delanteras optimizando el deslizamiento de los neumáticos mejorando la estabilidad y el rendimiento del vehículo (HR motor, 2023).

Este tipo de direccionamiento aumenta la tensión en los neumáticos, lo que puede acelerar su desgaste. Sin embargo, al permitir un giro preciso y suave, este mecanismo mejora significativamente la maniobrabilidad y el control del vehículo a altas velocidades (HR motor, 2023).

En robótica móvil los robots con direccionamiento Ackermann se encuentran entre las configuraciones motrices más convencionales, junto con los robots diferenciales, triciclo y omnidireccionales

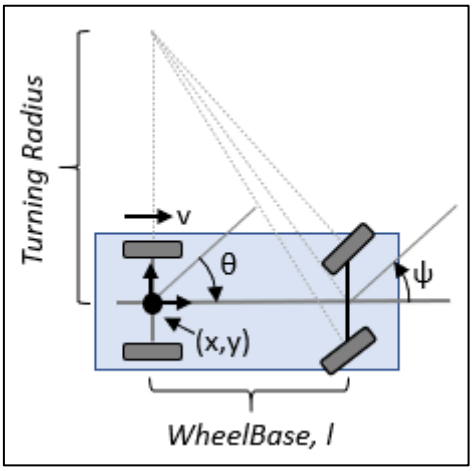



***Figura 10:*** *Giro de Ackermann de un solo ángulo, Figura tomada de https://es.mathworks.com/help/robotics/ref/ackermannkinematics.html*

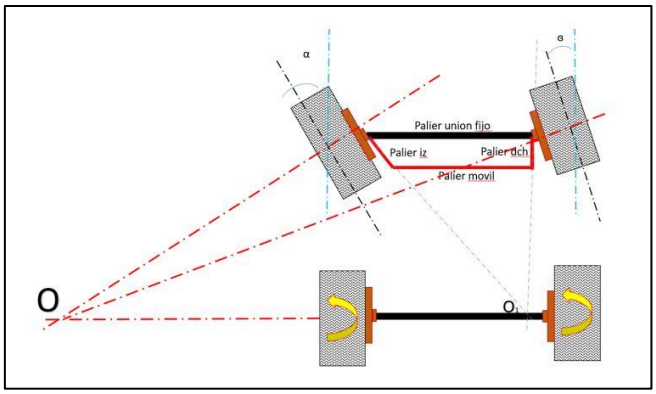



***Figura 11:*** *Giro de Ackermann de un doble ángulo, Figura tomada de https://www.motortime.es/2020/03/tecnica-de-formula-1-la-geometria-de-ackermann/*

## 2.4 Cinemática

La cinemática es la rama de la física que describe el movimiento de un sistema mecánico a través de ecuaciones que relacionan la posición del robot y el movimiento de sus articulaciones (Edisonsasig, 2021).

La cinemática se divide en dos tipos:

- Directa: En este tipo de cinemática, se obtienen los puntos de localización $\boldsymbol{x, y, z}$ conociendo la orientación del robot en un marco de referencia global.

- Inversa: Esta cinemática implica determinar la orientación de los ángulos de cada articulación a partir de los puntos $\boldsymbol{x, y, z}$ en un plano de referencia global.

Al definir la cinemática, es importante diferenciar entre el marco de referencia local (de cuerpo fijo) y el marco de referencia global, ya que las medidas de uno variarán con respecto al otro (Edisonsasig, 2021).

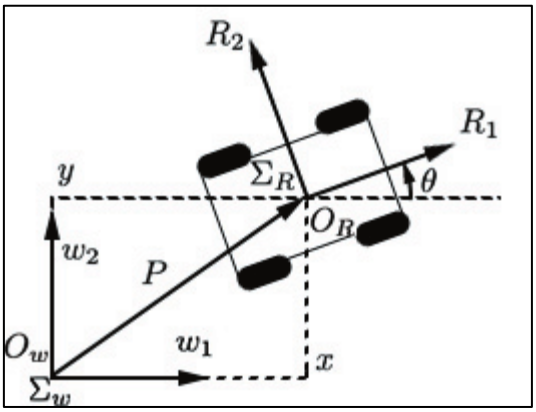



***Figura 12:*** *Marco de referencia global y local de un robot móvil, Figura tomada de https://www.researchgate.net/figure/Figura-1-Marco-inercial-S-w-y-marco-del-robot-S-R-Para-describir-la-postura-de-un_fig1_310491038*

Tal como se puede apreciar en la figura 12. Un robot móvil en un plano presenta tres grados de libertad (GDL):

- Posición en el plano **$(x, y)$**
- Orientación en el plano **$\theta$**

## 2.5 Odometría

La odometría como su nombre dice es la métrica del camino, consistente en un conjunto de posiciones consecutivas calculadas mediante el giro de las ruedas del robot. Estas posiciones se expresan tomando como referencia una pose inicial, considerada el punto (0,0) (ARME, 2022).

La odometría no es un cálculo exacto y siempre presenta un error acumulativo que aumenta con la distancia recorrida. Las odometrías más precisas suelen tener errores de menos de 10 centímetros por cada metro recorrido, mientras que las menos precisas pueden presentar errores mayores a 10 centímetros por cada metro recorrido (ARME, 2022).

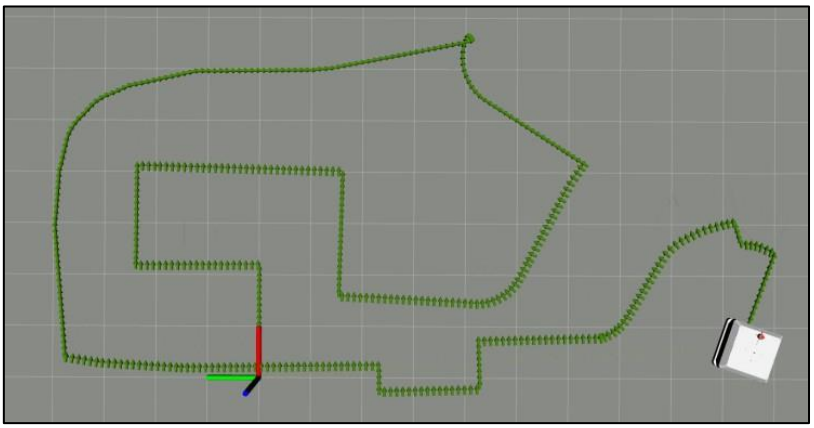


***Figura 13:*** *Odometría del robot Moby de Steering Machines, Figura tomada de https://es.linkedin.com/pulse/qu%C3%A9-es-la-odometr%C3%ADa-arme-rob%C3%B3tica-m%C3%B3vil*

## 2.6 Orientaciones

La orientación de un objeto dentro de un espacio tridimensional o bidimensional se define como las posibles posiciones del objeto sin cambiar su punto de referencia (Wikipedia, 2024). Hacer uso de estas posibles posiciones se le define como rotar el objeto, la cual siempre es en movimiento circular (Wikipedia, 2024).

Para especificar la orientación de un objeto se pueden utilizar diferentes métodos de calculo los cuales presentan diferentes ventajas y desventajas. Estos se pueden representar utilizando bases vectoriales las cuales definen los planos del sistema de referencia local y global (Wikipedia, 2024).

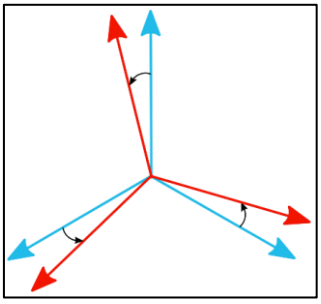


***Figura 14:*** *Orientación de un objeto, Figura tomada de https://es.wikipedia.org/wiki/Orientaci%C3%B3n_(geometr%C3%ADa)#:~:text=Una%20orientaci%C3%B3n%20de%20un%20objeto,un%20punto%20fijo%20de%20referencia.*

### 2.6.1 Ángulos de Euler

Los ángulos de Euler se utilizan para describir la orientación de un sistema de coordenadas fijo en relación con su marco de referencia local. Estos ángulos describen las rotaciones de un objeto en el espacio tridimensional mediante rotaciones sucesivas alrededor de los ejes de coordenadas (Universidad de Sevilla, 2019).

Los ángulos RPY (Roll, Pitch, Yaw) son una aplicación directa del concepto de Leonhard Euler y representan las rotaciones alrededor de los ejes $\boldsymbol{x, y, z}$ variando tanto el sentido de giro como los ejes sobre los que se realizan dichas rotaciones (Universidad de Sevilla, 2019).

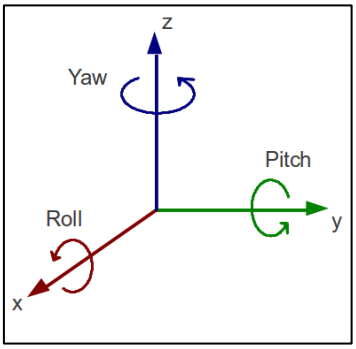



***Figura 15:*** *Orientación de giro de los ángulos RPY, Figura tomada de https://www.researchgate.net/figure/Representation-of-the-yaw-pitch-and-roll-angles_fig4_318959581.*

### 2.6.2 Cuarternión

El cuartenión es la representación de un vector tetradimensional, cuyos cuaterniones representan las coordenadas de dirección y la magnitud del vector. En este caso se presentan mediante los componentes $x, y, z, w$ (Universidad de Sevilla, 2019).

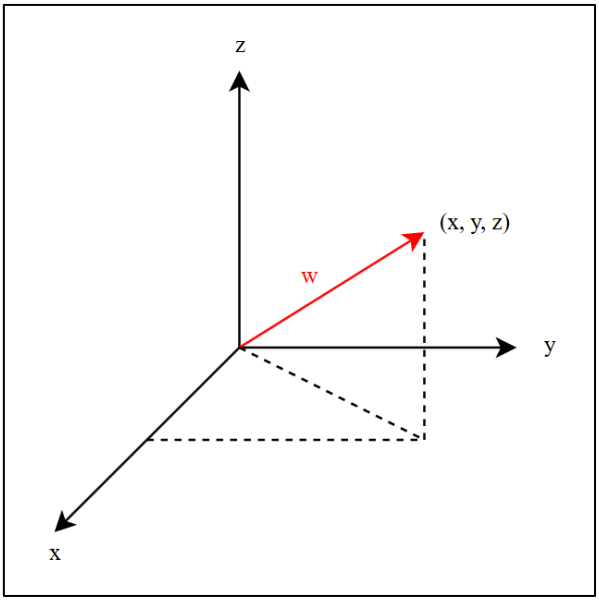



***Figura 16:*** *Orientación cuaternión*

## 2.7 Traslaciones y transformaciones

En una aplicación robótica se pueden utilizar diferentes marcos de referencia locales, estos definen la posición de los diferentes componentes del robo (Base, LIDAR, IMU y ruedas).

A medida que la base principal del robot se desplaza, estas posiciones y orientaciones cambian, un proceso conocido como transformación, que puede ser estática o dinámica (MATLAB & Simulink, n.d.-b).

Ejemplo de transformaciones estáticas:

- **Posición LIDAR y base:** En este caso, el LIDAR siempre se mantiene a la misma distancia que la base principal.

- **Posición IMU y base:** Al igual que el LIDAR, la IMU siempre se mantiene a la misma distancia de la base principal.

Ejemplo de transformaciones dinámicas:

- **Posición base y odometría:** La transformación de la base varia con respecto al marco fijo de odometría.

- **Posición LIDAR y mapa:** La variación del marco LIDAR se produce en relación con el marco fijo de mapa.

- **Giro de una rueda:** La misma gira contantemente sobre el eje base del robot, esta revolución es continua y presenta una transformación contante tanto en traslación como en rotación.

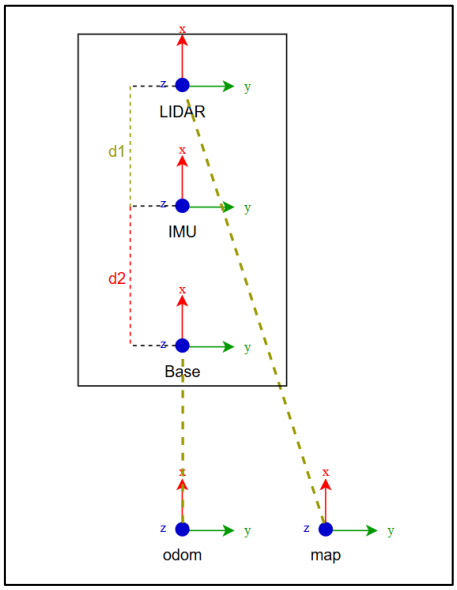



***Figura 17:*** *Traslaciones estáticas y dinámicas*

## 2.8 SLAM

SLAM es un acrónimo de "localización y mapeo simultáneo". Este método es utilizado por robots y vehículos autónomos para crear mapas y localizarse en un entorno de manera simultánea, tanto en tiempo real como de manera diferida (MATLAB & Simulink, n.d.-a).

Este método emplea el uso de LIDARs, tanto tridimensionales como bidimensionales. Además, en este enfoque se pueden utilizar cámaras, denominándose entonces vSLAM (localización y mapeo simultáneo mediante imágenes) (MATLAB & Simulink, n.d.-a).

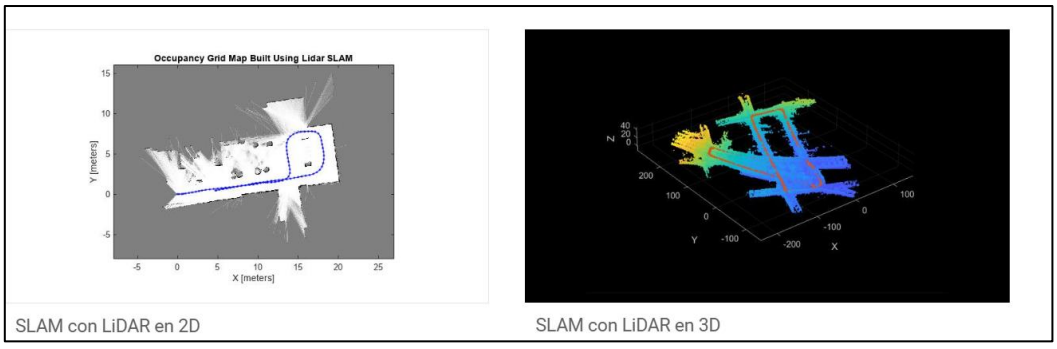



***Figura 18:*** *SLAM 2D y 3D, Figura tomada de https://es.mathworks.com/discovery/slam.html*

# Capítulo 3. DISEÑO Y ARQUITECTURA HARDWARE

## 3.1 Componentes hardware utilizados

### 3.1.1 Controladores

#### Rasberry PI modelo 4

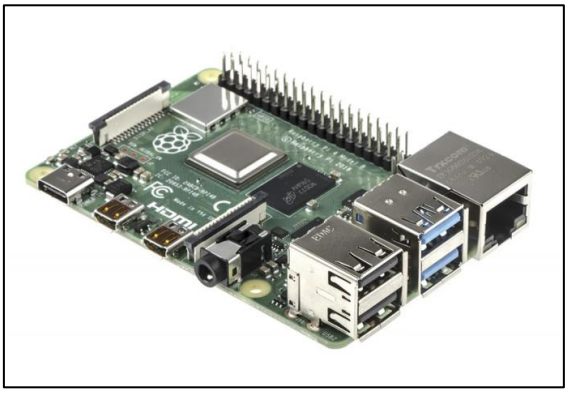


***Figura 19:*** *Raspberry Pi Model B, Figura tomada de https://tienda.esteamedu.es/placas/116-raspberry-pi-4-model-b-4-gb*

El Raspberry Pi 4 Model B es un miniordenador de bajo costo desarrollado por la fundación Raspberry Pi, representando la cuarta generación de la serie. Este dispositivo es ampliamente utilizado para configurar servidores web de baja capacidad, emulación de consolas de videojuegos, proyectos de domótica y aplicaciones de robótica (eSTEAM Education, 2023).

#### Arduino UNO R3

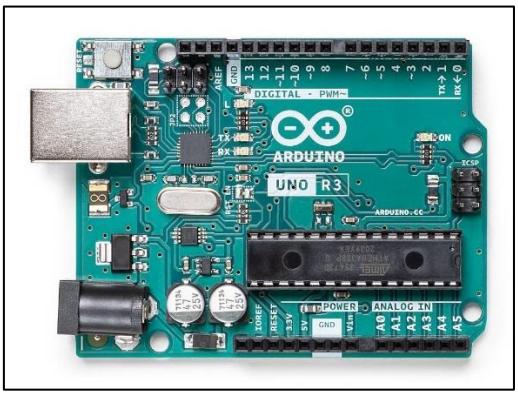



***Figura 20:*** *Arduino UNO R3, Figura tomada de https://www.amazon.es/Arduino-UNO-A000066-microcontrolador-ATmega328/dp/B008GRTSV6*

Arduino Uno es una plataforma de creación y prototipado de electrónica, ampliamente utilizada por inventores y desarrolladores para diseñar proyectos de manera sencilla. Este hardware está basado en ATMEGA 328 y cuenta con su propio lenguaje de programación, basado en C++ (Yúbal Fernández, 2022).

**Arduino Nano**

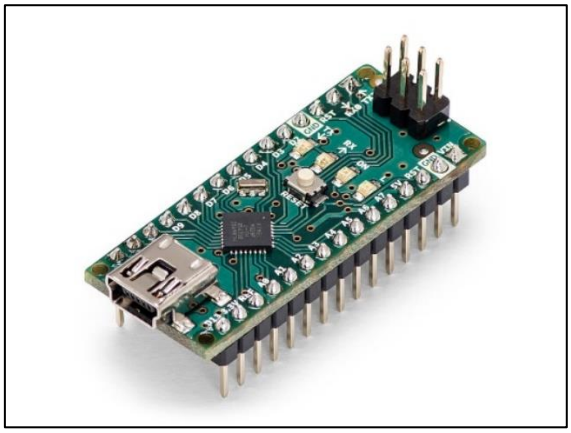


***Figura 21:*** *Arduino Nano, Figura tomada de https://store.arduino.cc/products/arduino-nano*

Arduino Nano, al igual que Arduino Uno, es una plataforma de hardware y software de electrónica, la diferencia está en que es utilizado para proyectos con menor capacidad y donde no se ocupe mucho espacio, está basado en ATMEGA 328 y su programación es en Arduino, basada en C++ (Yúbal Fernández, 2022).

### 3.1.2 Sensores

**LD19**

***Figura 22:*** *LD19 LiDAR, Figura tomada de https://www.robotshop.com/es/products/ld19-d300-lidar-developer-kit-360-degrees-dtof-laser-scanner-support-ros1-ros2-raspberry-pi-jetson-nano*

El sensor LD19 es un sensor LiDAR (Detección de Luz y Rango o Imágenes por Láser y Detección de Rango) desarrollado por la empresa Youyeetoo. Este sensor utiliza un láser infrarrojo central capaz de emitir 4500 pulsos por segundo. Estos pulsos son capturados por una unidad receptora que calcula el tiempo de vuelo del láser para determinar la distancia de la trayectori. El sensor incluye un módulo de procesamiento de datos que entrega la información necesaria. La emisión del láser es bidimensional, permitiendo mediciones en un plano (RobotShop, 2020).

**BNO055**

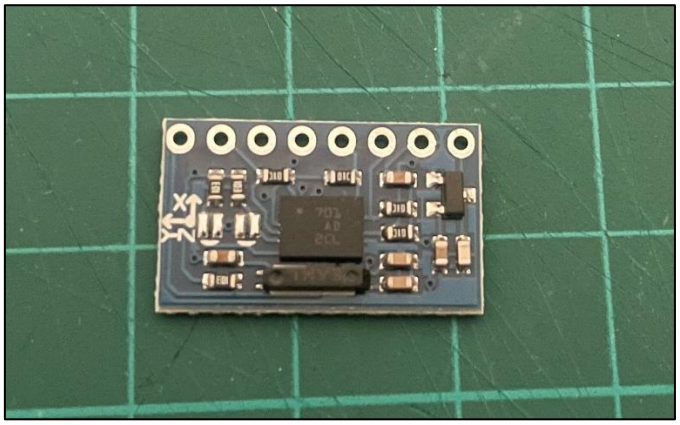


***Figura 23:*** *BNO055*

El sensor BNO055 es un sensor de movimiento inercial (IMU) que ofrece nueve grados de libertad (DOF). Para lograr esta capacidad, está equipado con un magnetómetro, un giroscopio y un acelerómetro, cada uno de ellos operando en tres ejes. La efectividad de este sensor se debe a su microcontrolador ARM Cortex-M0, encargado de realizar todos los cálculos matemáticos necesarios para procesar los datos recogidos por el sensor (Adafruit Industries, 2020).

**SlavaLiPo**

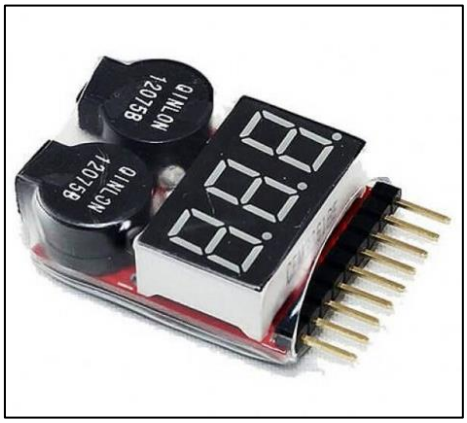


***Figura 24:*** *SalvaLiPo, Figura tomada de https://biltihobby.com/combustible-y-accesorios/14160-salva-lipo-8s-voltimetro-y-alarma.html*

Las salva LiPo son sensores que miden el voltaje de las celdas de una batería LiPo. Esta herramienta funciona para cuidar la vida de este tipo de baterías las cuales deben mantener un voltaje óptimo para evitar su desgaste.

**DC0-25**

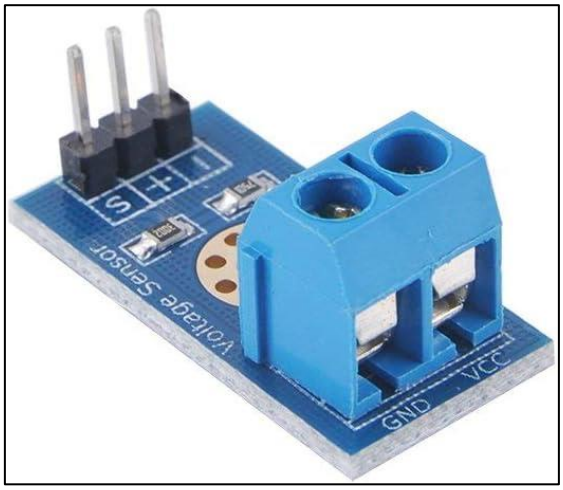



***Figura 25:*** *D0-25V, Figura tomada de https://www.amazon.es/dp/B07RFJYSM4?psc=1&ref=ppx_yo2ov_dt_b_product_details*

Este sensor funciona como un medidor de voltaje a través de un divisor de tensión, lo que le permite leer voltajes superiores a 5V. Esto resulta especialmente útil en plataformas como Arduino, cuyas entradas analógicas tienen un límite máximo de 5V.

### 3.1.3 Actuadores

**Motor YSIDO**


***Figura 26:*** *Motro YSIDO, Figura tomada de https://es.aliexpress.com/item/1005006406801871.html?spm=a2g0o.order_list.order_list_main.4.6590194dDfke9y&gatewayAdapt=glo2esp*

El motor sin escobillas YSIDO, modelo 540, cuenta con un parámetro de 13.5 T, lo que indica su bobinado. Este tipo de motor requiere un controlador de velocidad. Además de su construcción sin escobillas, que reduce la fricción y aumenta la durabilidad, este motor es ideal para aplicaciones donde se requiera un control preciso de velocidad y un mantenimiento mínimo.

Este motor es sensorizado por lo que cuenta con 3 sensores efecto hall para detectar la posición de las bobinas. Este sensor mejora el comportamiento del motor a bajas velocidades.

**ESC**

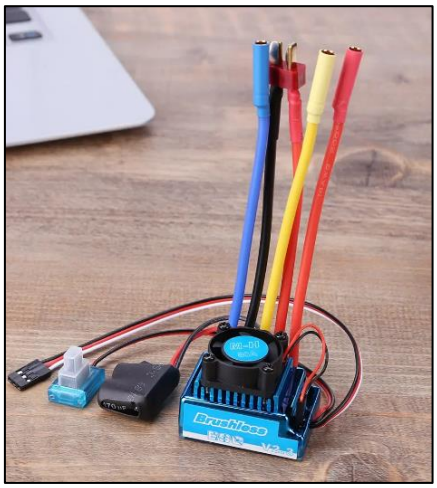


***Figura 27:*** *ESC brushless, Figura tomada de https://es.aliexpress.com/item/1005004271730782.html?spm=a2g0o.order_list.order_list_main.10.6590194dDfke9y&gatewayAdapt=glo2esp*

El controlador de velocidad brushless es utilizado para controlar la secuencia de giro para un motor sin escobillas, esto se ven limitados por los parámetros generales del motor como amperaje máximo y bobinado. La información al mismo es enviada por PWM y su amperaje máximo es de 180 A para baterías Lipo de dos celdas.

**Servo motor**

***Figura 28:*** *Servo motor Carson*

Estos motores pueden alcanzar rangos de movimiento típicamente de 0 a 180 grados, lo que los hace ideales para controlar la dirección en vehículos y en aplicaciones de automatización donde se requiere ajuste preciso de la posición.

### 3.1.4 Complementos

**Gamepad Logitech F710**

***Figura 29:*** *Gamepad Logitech F710*

El controlador F710 con conexión Bluetooth, está equipado con 14 botones de acción tipo todo o nada y dos joysticks incrementales. Adicionalmente, cuenta con dos modos de funcionamiento compatibles con ROS (Robot Operating System), lo que lo hace versátil para su uso tanto en videojuegos como en aplicaciones de robótica.

**Hub USB**


***Figura 30:*** *HubUSB, Figura tomada de https://www.amazon.es/UGREEN*

Extensor de USB 3.0, este funciona para aumentar y colocar los puertos necesarios para conexiones externas requeridas.

### 3.1.5 Adaptadores de Red

#### GL iNet GL-AR300M16


***Figura 31:*** *GL iNet, Figura tomada de https://www.amazon.es/GL-iNet-GL-AR300M16-Ext*

Módulo extensor con 3 modos de funcionamiento: Local Host, Repeater y SIM. Este funciona para generar una red local con conexión o sin conexión a internet, la velocidad de red máxima que ofrece este dispositivo es de 300Mbps.

### 3.1.6 Alimentación

#### Batería LiPo de 2 celdas

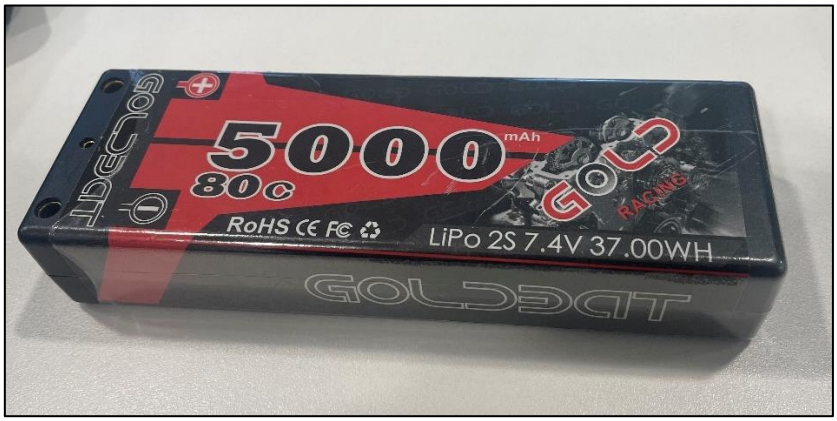



***Figura 32:*** *Batería GOLD*

Las baterías de polímero de litio son comúnmente utilizadas en aplicaciones que requieren altas tasas de carga y descarga. Este proyecto utiliza una batería de la marca GOLD, la cual posee una capacidad de 5000 mAh (miliamperios hora) y puede soportar una tasa de descarga máxima de 80C a 7.4V.

**Power Bank de 22W**

***Figura 33:*** *Power Bank*

Este centro de recarga se utiliza para alimentar toda la electrónica de control y cuenta con dos entradas USB y una entrada tipo C. Este dispositivo es capaz de suministrar una potencia de 22.4W y tiene una capacidad de 10000 mAh (miliamperios hora).

## 3.2 Descripción y configuración físicas y mecánicas del coche RC

El modelo Carson Dirtwarrior 2.0 Brushed es un coche a radio control que cuenta con una serie de características técnicas avanzadas que lo destacan en su categoría:

- Transmisión diferencial delantera y trasera.
- Tracción 4x4 (4WD).
- Amortiguadores de presión de aceite.
- Dirección Ackermann por servo motor.

Para el almacenamiento y adaptación de electrónica, el vehículo cuenta con:

- Compartimento para almacenamiento de baterías de polímero de litio de dos celdas.
- Base para motores de especificaciones 540.
- Espacio para controlador de motor Brushless.
- Adaptación para sistema de dirección por servo motor.

En cuanto a material, el vehículo cuenta con plástico resistente, normalmente utilizado en los coches de competición.

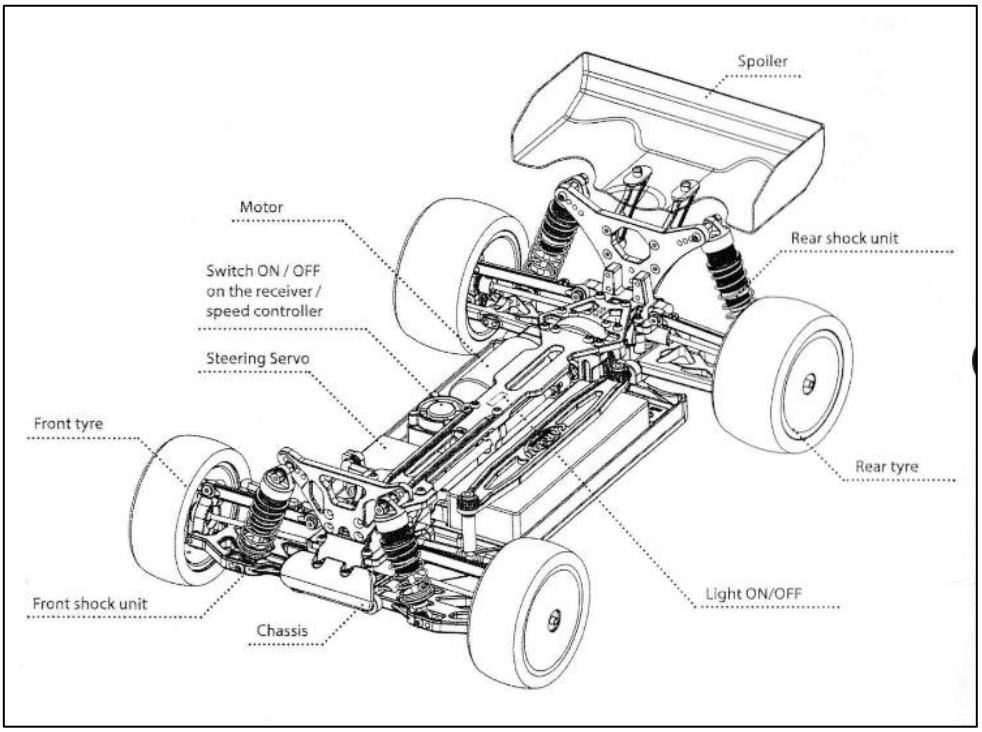



***Figura 34:*** *Chasis Carson Dirtwarrior, Figura tomada del Manual de Carson Dirtwarrior*

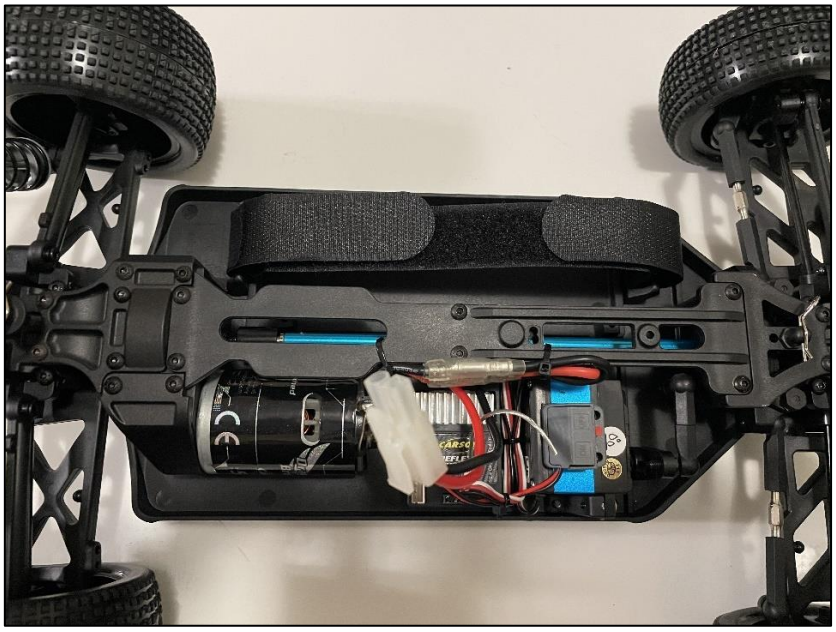


***Figura 35:*** *Vista superior Chasis Carson Dirtwarrior sin tornillos adaptados*

## 3.3 Diseño y adaptación de soporte para colocación de hardware

Antes de diseñar la plataforma, se realiza un análisis detallado de los puntos de anclaje de la base al chasis principal. La base se ubica de manera que no afecte el mecanismo de giro delantero ni causar interferencias con la electrónica instalada en la base principal.

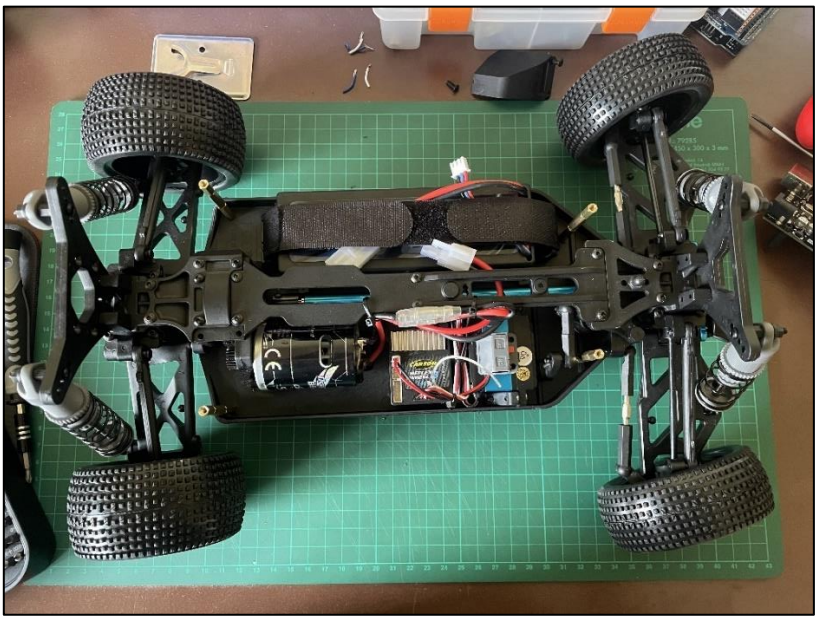


***Figura 36:*** *Vista superior Chasis Carson Dirtwarrior con tornillos adaptados*

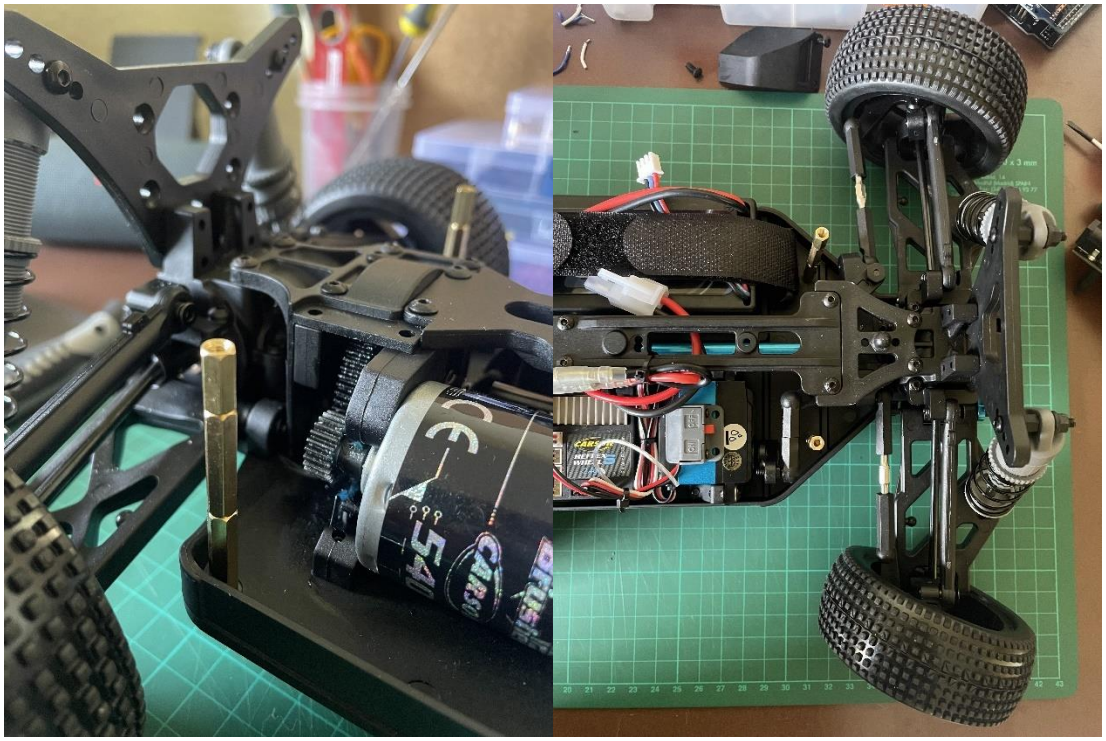


***Figura 37:*** *Colocación de separadores hexagonales de latón*

Como se observa en las figuras 35, 36 y 37, se han instalado separadores hexagonales de latón. Estos se utilizan para montar los componentes electrónicos en la parte superior del dispositivo.

Para determinar la medida exacta de los separadores, se utiliza una pieza de cartón que se perfora y atornilla en su lugar. Tras identificar que el diámetro de la rosca es de aproximadamente 3 mm, se toman las medidas necesarias para establecer las distancias reales entre los separadores.


***Figura 38:*** *Medición de separadores de latón*

Se utiliza SolidWorks para diseñar la base principal y realizar las adaptaciones necesarias a los componentes que lo requieran. El diseño de la base se basa en las medidas obtenidas al calcular la distancia entre los separadores de latón.

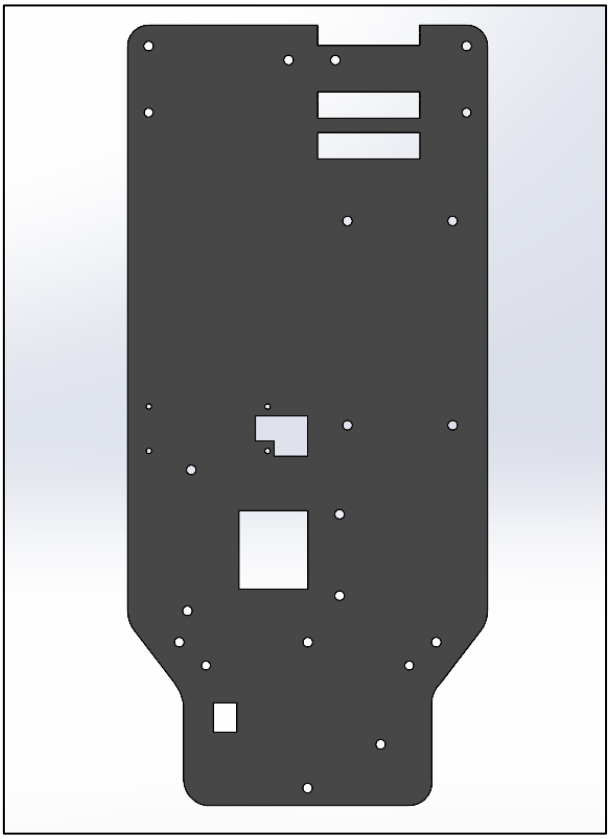


***Figura 39:*** *Diseño base principal*

Para los componentes como el LIDAR LD19 y la Raspberry Pi 4, se diseñan soportes o bases especiales que permiten atornillarlos de manera segura a la base principal. El diseño de estos componentes se muestra en la figura 39.

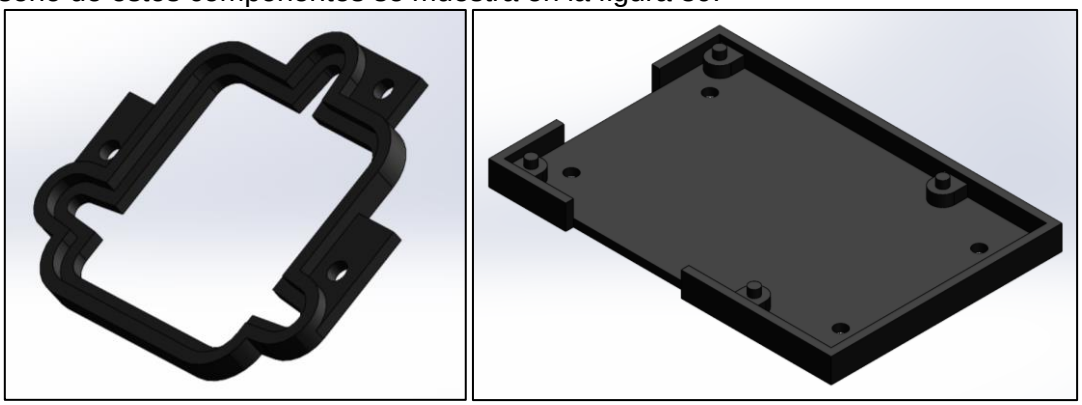


***Figura 40:*** *Diseño base principal*

Una vez diseñadas las piezas necesarias para adaptar el vehículo, se procede a realizar un ensamblaje utilizando componentes 3D de GrabCAD, lo que permite una representación precisa del ensamblaje final previsto.

Dado que no existe un modelo 3D disponible del vehículo en sí, solo se lleva a cabo el ensamble de las adaptaciones electrónicas finales que se planean instalar.

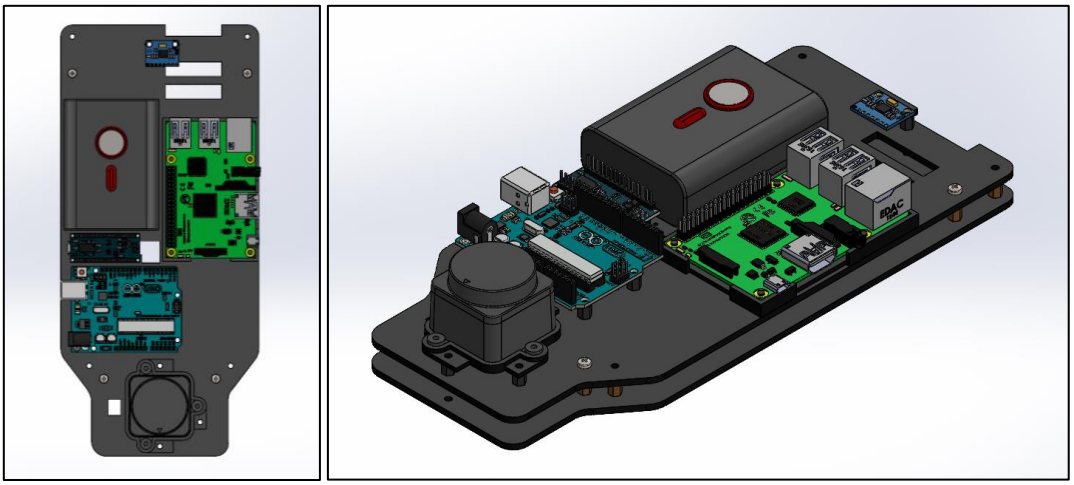


***Figura 41:*** *Ensamblaje en SolidWorks*

Como se observa en la figura 41, el ensamblaje permite estimar la ubicación de los componentes superiores. Se implementa una estructura de doble capa con el objetivo de almacenar adecuadamente los cables de toda la electrónica.

Una vez diseñado, se utilizan las herramientas proporcionadas por el FabLAB de la universidad europea de Madrid para realizar los cortes e impresión 3D de cada uno de los componentes diseñados.

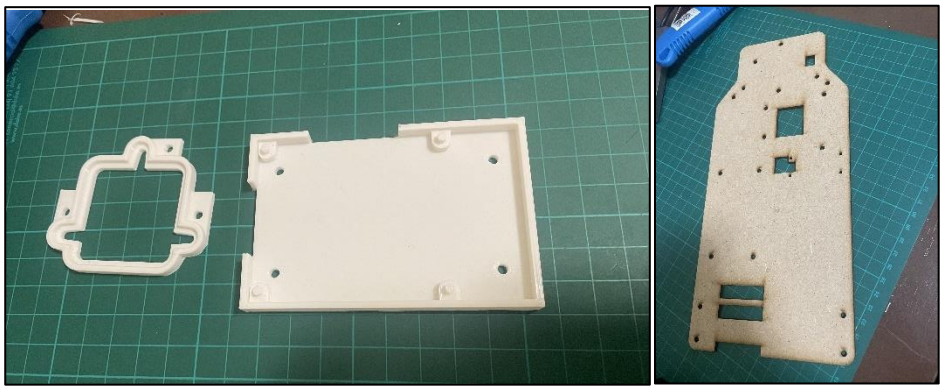


***Figura 42:*** *Diseños realizados*

Las piezas se pintan de color negro antes de realizar el ensamble físico del vehículo. Durante el ensamblaje, se utilizan cables de 90 grados para evitar interferencias con objetos durante la navegación.

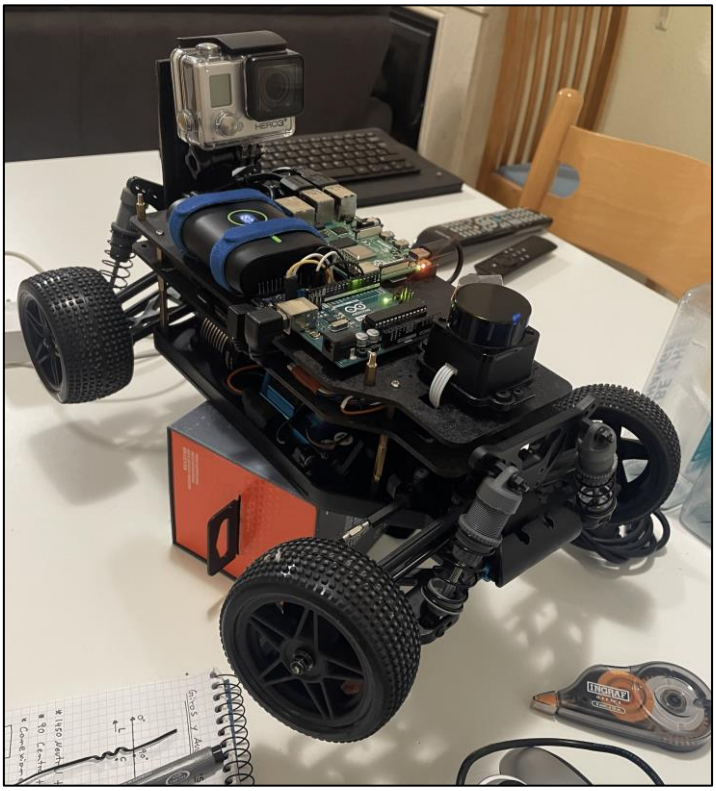


***Figura 43:*** *Ensamblaje base final*

Para completar la adaptación se diseña una pequeña base la cual se ubica en la parte trasera del vehículo, esta sirve para llevar la red local generada por el robot.

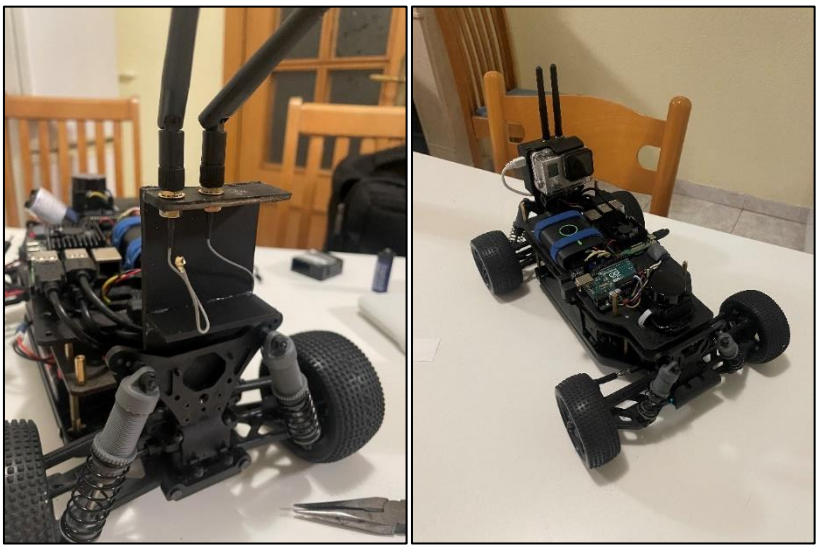


***Figura 44:*** *Ensamblaje final*

## 3.4 Diagramas de conexiones eléctricas

Conociendo cada uno los componentes electrónicos que se colocaran en el proyecto, se realiza un diagrama de conexiones, Este diagrama se dividirá en 3 secciones: diagrama Arduino UNO, diagrama a nano y etapa de potencia.

Para realizar los diagramas se utiliza el software KiCAD, el cual cuenta con las herramientas para dibujar componentes y realizar diagramas.

### Arduino nano

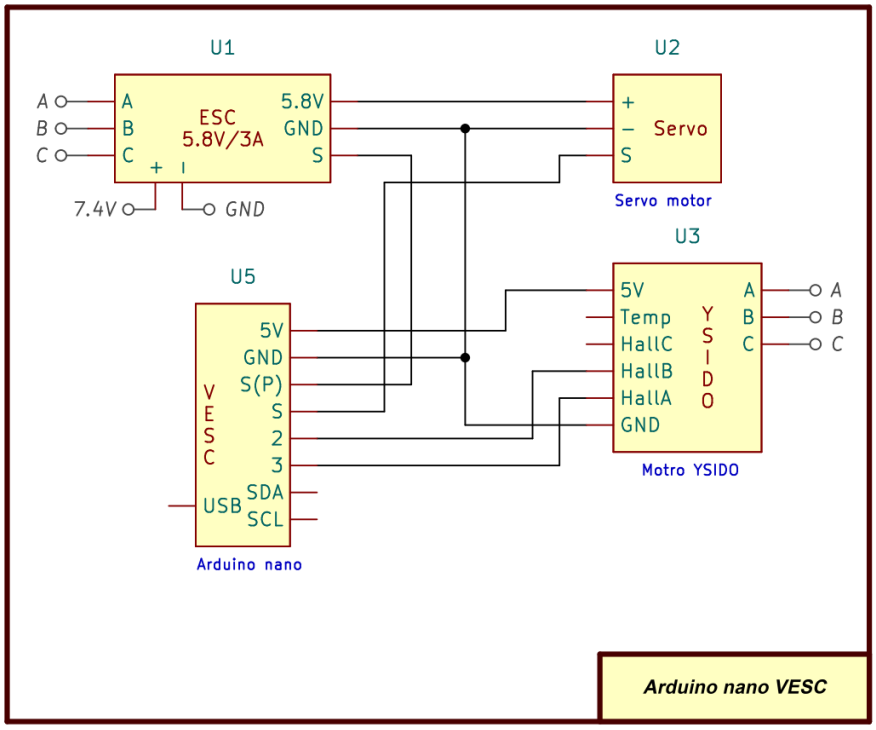



***Figura 45:*** *Esquema Arduino nano VESC*

Como se puede ver en la figura 43, los componentes interconectados son:

- Arduino nano (VESC)
- Motor YSIDO
- Servo motor
- ESC

En este caso, el Arduino Nano no tiene suficiente potencia para operar el servo motor directamente. En los coches de carreras radiocontrolados, los servos suelen ser alimentados por el controlador de velocidad. Por tanto, las conexiones de alimentación del servo motor se derivan de las salidas del controlador.

Para enviar la señal PWM al servo motor, se conecta a la salida marcada con "S" en el diagrama, que representa cualquier salida PWM disponible en el Arduino Nano. Para completar el circuito y asegurar un funcionamiento correcto, se deben unificar las tierras, conectando los cables de tierra del Arduino Nano a los del controlador que alimenta el servo motor.

El controlador de velocidad también opera mediante señales PWM, al igual que el servo motor. Por tanto, la entrada PWM del controlador se conecta a la salida marcada con "P", que representa cualquier salida digital PWM disponible en el Arduino Nano. Dado que las tierras están unificadas, no es necesario realizar una conexión adicional para ellas.

Por último, para conectar el motor YSIDO al controlador, se unen las fases A, B y C del motor con las correspondientes fases A, B y C del controlador. El controlador de velocidad determina automáticamente la secuencia de estas fases, por lo que no es necesario seguir un orden específico al conectarlas.

El mismo tiene 6 pines los cuales se dividen de la siguiente manera:

- 5V
- HALL C
- HALL B
- HALL A
- GND

***Figura 46:*** *Conector sensor efecto Hall*

El sensor de efecto Hall opera a 5V y, dado que tiene una electrónica independiente, se alimenta directamente desde el Arduino Nano. Las salidas de los pines HALL del motor se conectan a los pines de interrupción del Arduino Nano, específicamente al pin digital 2 y al pin digital 3, permitiendo así una lectura de la posición del motor.

**Arduino UNO**

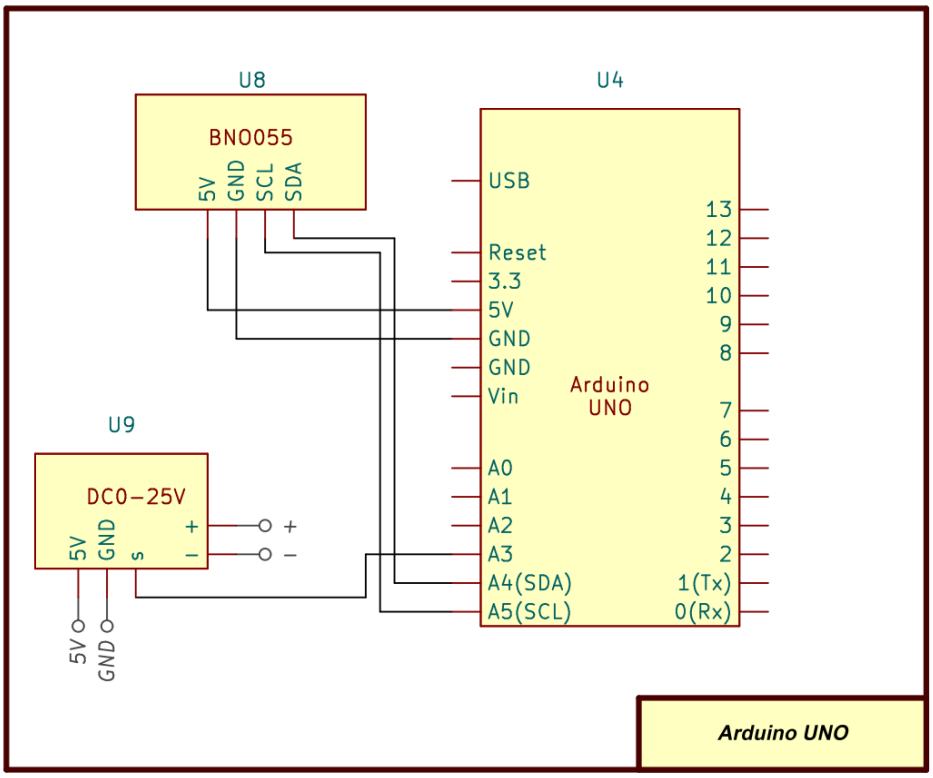



***Figura 47:*** *Esquema Arduino UNO*

Como se puede ver en la figura 45, lo componentes interconectados son:

- BNO055
- DC0-25V
- Arduino UNO

El BNO055 opera mediante el protocolo I2C, por lo que se conecta al Arduino UNO utilizando los pines A4 (SDA) y A5 (SCL). Aunque su voltaje de funcionamiento es de 3.3V, incluye un regulador interno que permite conectarlo también a 5V.

El componente DC0-25V se utiliza para medir el voltaje de una batería LiPo de dos celdas. En el esquema, está indicado que se alimenta con 5V y GND de arduino. Para la lectura del voltaje de entrada, se conecta a la entrada analógica A3 del Arduino. Esto permite monitorear el estado de la batería, asegurando que el sistema funcione dentro de los parámetros de voltaje seguros.

**Etapa de Alimentación**

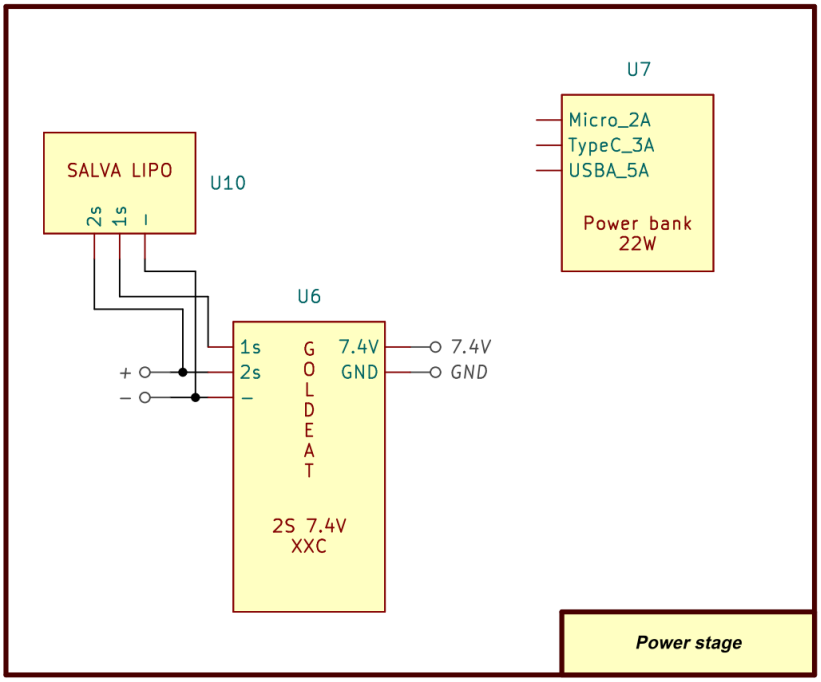



***Figura 48:*** *Esquema Etapa de potencia*

Como se puede ver en la figura 46, los elementos interconectados son:

- Salva LiPo
- Batería LiPo de 2 celdas

El dispositivo de protección para la batería LiPo, conocido como salva LiPo, se conecta directamente a la batería. A su vez, la batería se conecta directamente al controlador de velocidad para suministrarle energía.

Debido a la carga y exgencia que pueden presentar algunos componentes, es importante separar la alimentación en el circuito. Por esta razón, se utiliza la batería LiPo para alimentar la potencia y la powerbank para suministrar energía al circuito de control.

## 3.5 Diagramas de conexiones Hardware

Una vez explicadas las conexiones de los componentes electrónicos, este apartado se centra en cómo están conectados a la Raspberry Pi.

Aunque el diagrama completo se incluye en el anexo 1, aquí se presenta un diagrama simplificado. Este no detalla las conexiones directas de los componentes a los Arduinos, sino que especifica las conexiones directas a la Raspberry Pi.

Además, se muestra cómo se realiza la conexión remota de la Raspberry Pi al ordenador.

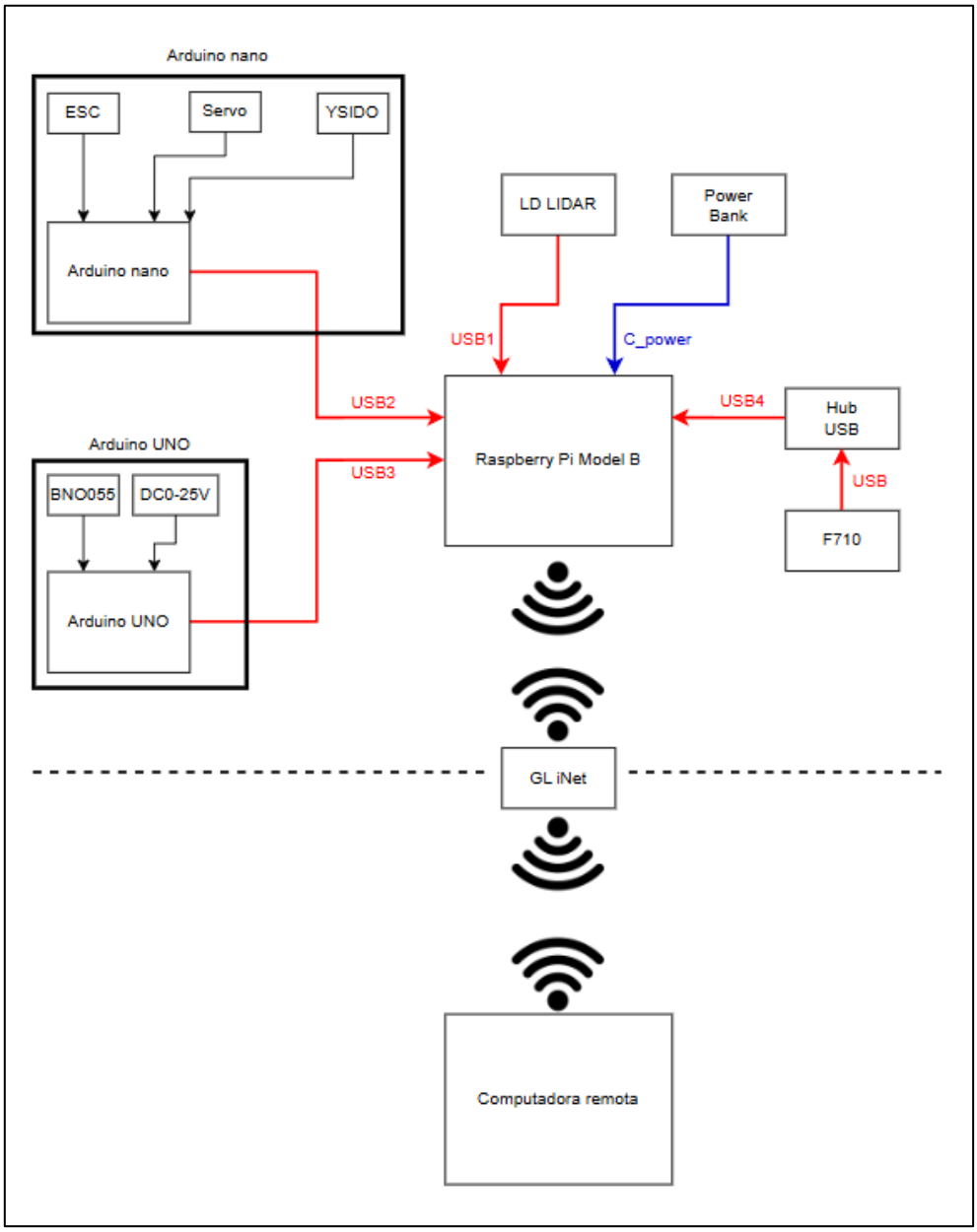



***Figura 49:*** *Esquema de conexión Hardware*

Como se ve en la figura 47, los Arduinos se conectan a través de USB, al igual que el LIDAR y el hub USB. Este último se añade debido a que la Raspberry Pi cuenta con 4 puertos USB, permitiendo así ampliar la capacidad de puertos disponibles para conectar otros dispositivos, como el F710.

El GL iNet facilita la conexión remota entre la Raspberry Pi y el ordenador a través de una red local, la cual no tiene acceso a internet.

Como se mencionó al inicio, para conocer las conexiones específicas de los componentes conectados a los Arduinos, es recomendable consultar el apartado 3.4.

# Capítulo 4. HERRAMIENTAS DE CONSTRUCCIÓN DE ARQUITECURA SOFTWARE

## 4.1 Ubuntu

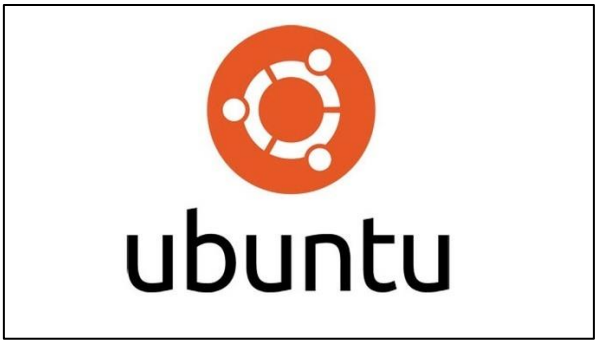



***Figura 50:*** *Logo Ubuntu, Figura tomada de https://computerhoy.com/noticias/software/todo-ubuntu*

Ubuntu es un sistema operativo basado en Linux que ofrece dos versiones principales: una de servidor y otra de escritorio. Desarrollado por la empresa Canonical, Ubuntu se fundamenta en Debian, una de las distribuciones más reconocidas desde 2004 (Betania V., 2023).

Es importante diferenciar entre Linux y Ubuntu; mientras que Linux se refiere al núcleo del sistema operativo, Ubuntu es una distribución que busca simplificar el uso de Linux para los usuarios (Betania V., 2023).

Se ha popularizado especialmente en entornos de desarrollo de servidores debido a su naturaleza de código abierto(Betania V., 2023).

## 4.2 C++

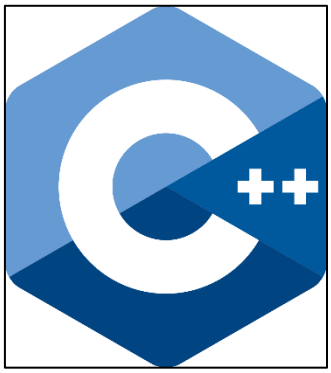


***Figura 51:*** *Logo C++, Figura tomada de https://es.wikipedia.org/wiki/C%2B%2B*

Desarrollado por Danés Bjarne a mediado de los años 80. C++ es una extensión del lenguaje C, a menudo descrito como "C con clases". Este lenguaje fue desarrollado con el objetivo de incorporar la programación orientada a objetos. (Angel Robleadno, 2019).

## 4.3 Python3


***Figura 52:*** *Logo Python, Figura tomada de https://aws.amazon.com/es/what-is/python/*

Python es un lenguaje de programación creado por Guido van Rossum, utilizado ampliamente en el desarrollo web, manipulación de datos y aprendizaje automático (machine learning, ML). Su popularidad se debe a su eficiencia y facilidad de aprendizaje, haciéndolo accesible para programadores de todos los niveles (Amazon server, 2023).

## 4.4 Rviz

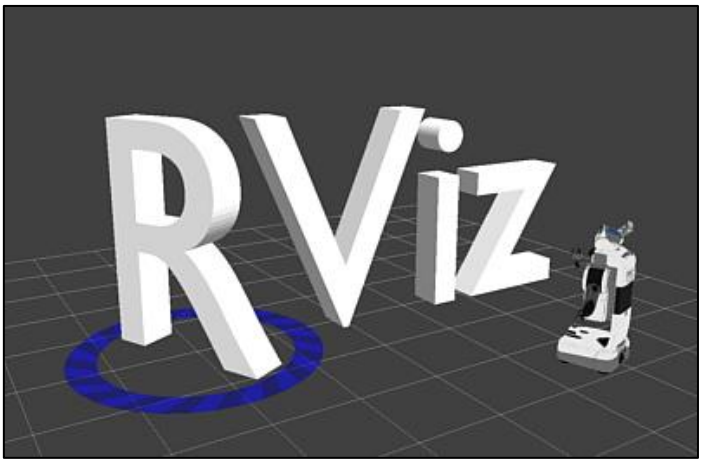



***Figura 53:*** *Logo Rviz, Figura tomada de https://github.com/ros-visualization/rviz*

Rviz, un complemento del framework ROS, es una herramienta que permite visualizar en tiempo real los diferentes parámetros medidos por un robot, tanto en entornos reales como simulados. Rviz muestra gráficamente tópicos específicos asociados a los tipos de mensajes de ROS, lo cual es útil para visualizar las transformaciones de traslación y rotación de las articulaciones u otros parámetros relacionados al robot en un entorno determinado.

# Capítulo 5. INSTALACIÓN Y PREPARACIÓN DE ENTORNO ROS

## 5.1 ROS noetic en Ubuntu desktop 20.04 en VMware

Dado que no se dispone de un ordenador con Linux, se optará por instalar una máquina virtual usando VMware Workstation 17 Player. Como se mencionó en el marco teórico, ROS tiene diferentes versiones que son compatibles con sistemas operativos específicos. En este caso, la versión Noetic es compatible con Ubuntu 20.04.

Para instalar ROS Noetic en Ubuntu 20.04, primero se debe actualizar el sistema. Esto se realiza utilizando los siguientes comandos:

```
$ sudo apt update
$ sudo apt upgrade
```

Una vez que el sistema está actualizado, es importante instalar la versión de ROS que se adecue al propósito de su uso. En este caso, dado que se está utilizando una máquina con interfaz gráfica, se instala la versión **Desktop-Full de ROS**. Esta versión incluye todas las herramientas visuales.

```
$ sudo apt install ros-noetic-desktop-full
```

Una vez instalado, es necesario agregar la configuración de ROS al archivo ".bashrc" para que el sistema lo reconozca automáticamente cada vez que sea necesario ejecutarlo.

```
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Implementado todo esto se instala la dependencia de Python 3 ejecutando los siguientes comandos:

```
$ sudo apt install python3-rosdep python3-rosinstall python3-
rosinstall-generator python3-wstool build-essential
$ sudo apt install python3-rosdep
$ sudo rosdep init
$ rosdep update
```

Para verificar que la instalación se ha realizado correctamente, se utilizan los comandos **`roscore`** y **`rosversion`** `-d`. como se muestra en la Figura 54.

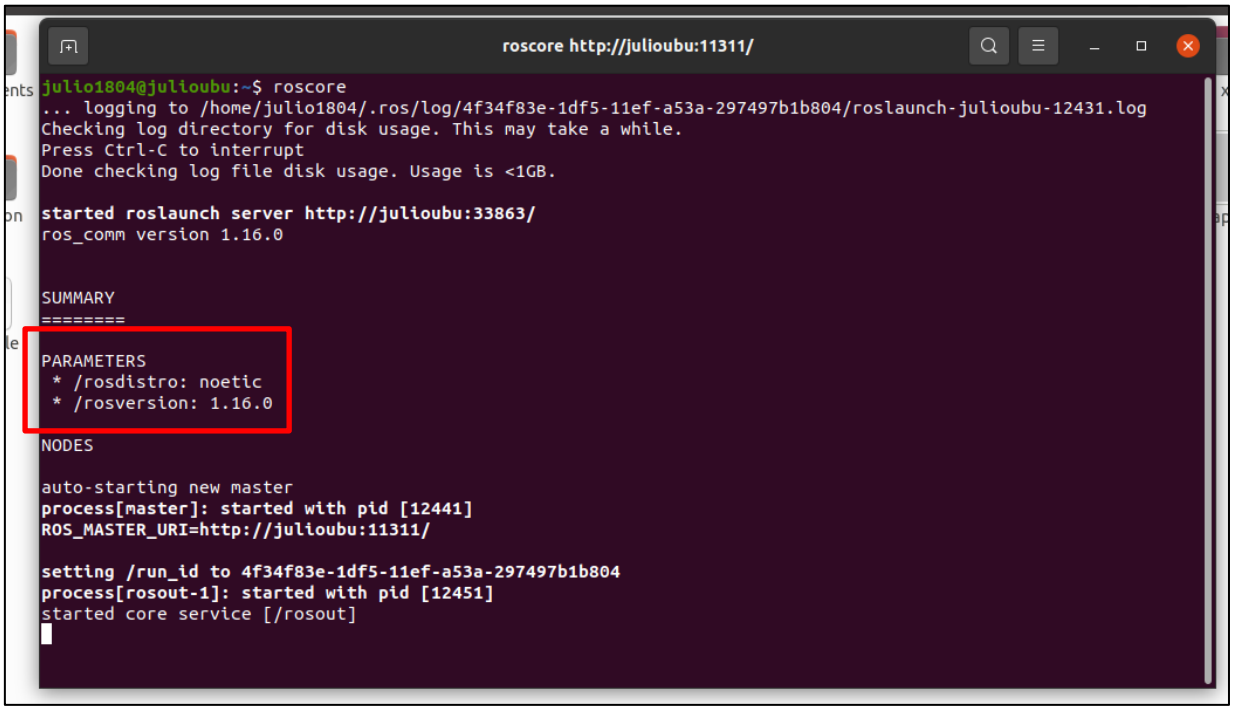



***Figura 54:*** *Instalación de ROS en VMware*

### 5.1.1 Workspace Sundybot en VMware

Para crear el workspace (WS) o entorno de trabajo en ROS Noetic, primero se debe crear un directorio que contenga una carpeta llamada src. Una vez creada esta estructura, se debe compilar el contenido utilizando el comando catkin_make, que es el compilador estándar para ROS Noetic

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

Creada la carpeta es importante añadir la configuración en la .bashrc

```
$ echo "source ~/sundybot/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Dentro del espacio de trabajo creado queda la siguiente estructura:

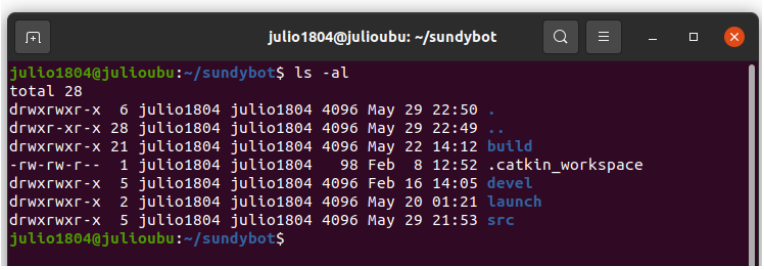



***Figura 55:*** *Estructura de capeta sundybot en VMware*

## 5.2 ROS noetic en Ubuntu server 20.04 en Raspberry Pi

Como se mencionó anteriormente, ROS ofrece distintas configuraciones adaptadas a diversas aplicaciones. En este caso, la Raspberry Pi no lleva instalada una interfaz gráfica, por lo que no es esencial instalar el conjunto completo de herramientas de visualización 3D. Por ello, se opta por ROS-Base, que omite las dependencias y herramientas gráficas.

Se deben seguir los pasos previamente mencionados, con la única variación del paquete a instalar.

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install ros-noetic-ros-base
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt install python3-rosdep python3-rosinstall python3-
rosinstall-generator python3-wstool build-essential
$ sudo apt install python3-rosdep
$ sudo rosdep init
$ rosdep update
```

Para verificar que la instalación se ha realizado correctamente, se utilizan los comandos **`roscore`** y **`rosversion`** `-d`. como se muestra en la Figura 56.

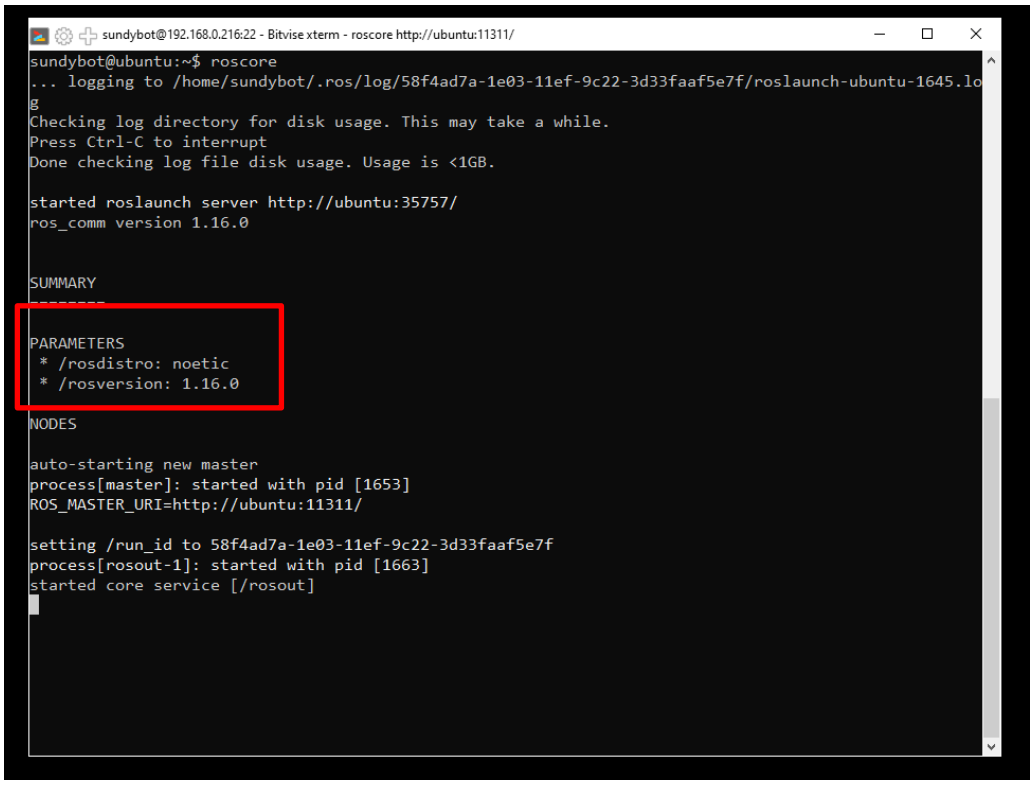



***Figura 56:*** *Instalación de ROS*

### 5.2.1 Workspace Sundybot en Raspberry Pi

Se siguen los pasos utilizados en el apartado 5.1.1 donde se crea un espacio de trabajo en la máquina virtual.

```
$ mkdir -p ~/sundybot/src
$ cd ~/sundybot
$ catkin_make
$ echo "source ~/sundybot/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Queda la siguiente estructura dentro del espacio de trabajo creado

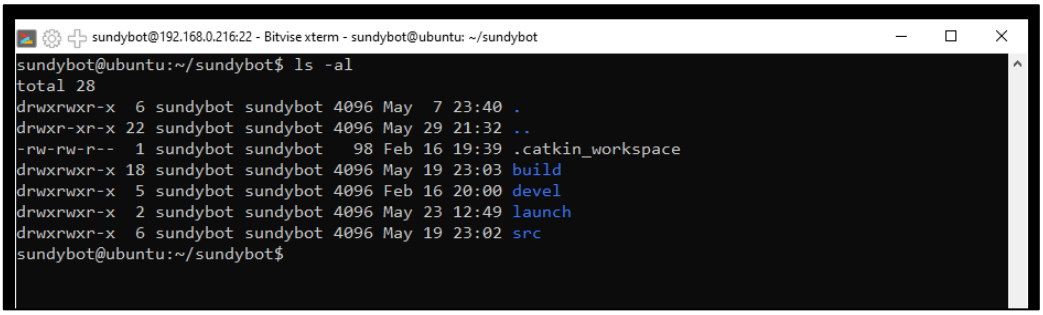



***Figura 57:*** *Instalación de ROS en Raspberry Pi*

## 5.3 configuración de ROS_MASTER y ROS_HOSTNAME

En el capítulo 3, apartado 3.5, se explica que la conectividad entre el robot (Raspberry Pi) y la máquina remota (VMware) se realiza a través de una red local. Para establecer esta conexión, ambas computadoras deben estar preparadas para operar en las siguientes redes:

- **36:** red local del hogar.
- **wifiROS:** red local del laboratorio de robótica de la Universidad Europea de Madrid.
- **sundybot:** red local del robot, GL.iNet

Estas redes representan los entornos de trabajo en los que opera el robot, variando según el tipo de misión a realizar o las mejoras que se necesiten implementar.

Para configurar ROS_MASTER_URI y ROS_HOSTNAME, es necesario determinar cuál ordenador actuará como el nodo maestro de ROS. En este caso, se ha elegido que la máquina virtual sea el nodo maestro. La configuración correspondiente se detalla en la Figura 58.

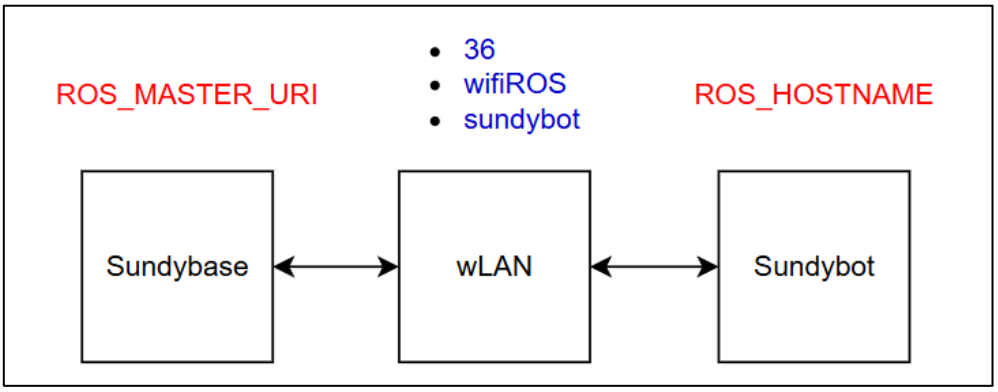



***Figura 58:*** *Comunicación LAN*

Para este caso, la IP de la máquina virtual VMware (Sundybase) en cualquiera de las tres redes será la dirección para ROS_MASTER_URI, y la IP de la Raspberry Pi (Sundybot) será la dirección para ROS_HOSTNAME.

**Configuración en VMware**

Una vez entendido este concepto para establecer la comunicación, se debe abrir la terminal de .bash y colocar las siguientes líneas:

```
#export ROS_MASTER_URI=http://IP_SUNDYBASE:11311
#export ROS_HOSTNAME=IP_SUNDYBASE
```

En este caso, se asigna como ROS_HOSTNAME la IP de Sundybase, ya que, aunque actúe como el nodo maestro, también es un dispositivo que accede a sí mismo para ejecutar programas de ROS.

```
121
122 #WIFI ROS
123 #export ROS_MASTER_URI=http://192.168.8.143:11311
124 #export ROS_HOSTNAME=192.168.8.143
125
126 #WIFI 36
127 #export ROS_MASTER_URI=http://192.168.0.138:11311
128 #export ROS_HOSTNAME=192.168.0.138
129
130 #WIFI sundybot
131 #export ROS_MASTER_URI=http://192.168.8.243:11311
132 #export ROS_HOSTNAME=192.168.8.243
133
```

***Figura 59:*** *.bash VMware*

### Configuración en Raspberry Pi

Se colocan las mismas líneas en el archivo .bash de la Raspberry Pi que en la máquina virtual VMware:

```
#export ROS_MASTER_URI=http://IP_SUNDYBASE:11311
#export ROS_HOSTNAME=IP_SUNDYOT
```

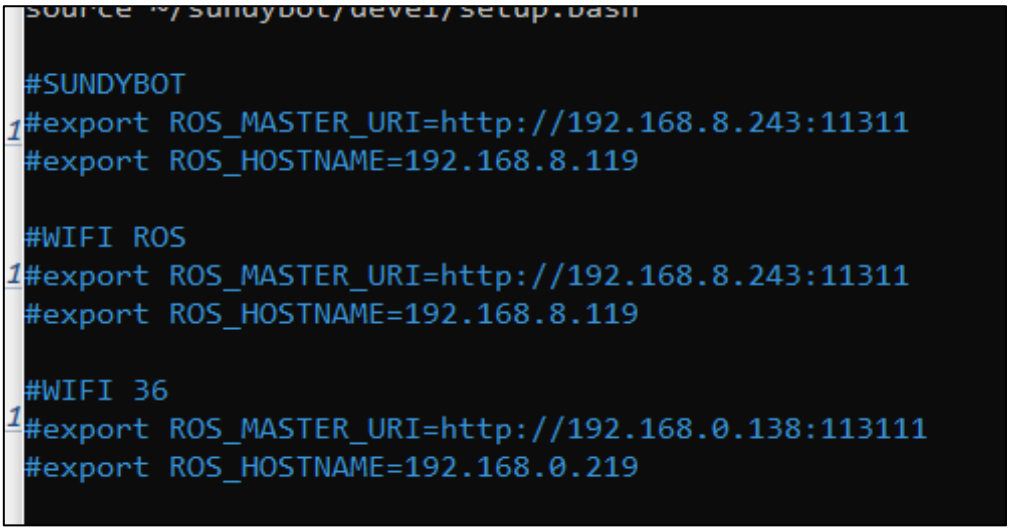



***Figura 60:*** *.bash Raspberry Pi*

Algo importante a considerar una vez realizado este proceso, es que las direcciones IP son dinámicas. Para el caso de la red sundybot, las IP siempre serán las mismas, dado que no es una red que reciba solicitudes de acceso externas a los dispositivos ya configurados. Para las otras redes, es necesario revisar siempre la IP de los dispositivos.

Otro punto importante por considerar es que la red sundybot no tiene conexión LAN a ningún dispositivo, lo cual impide que mantenga una precisión exacta del tiempo. Esta falta de sincronización horaria puede ser problemática al realizar compilaciones con catkin_make, ya que ROS requiere una sincronización adecuada en la red para compilar los paquetes correctamente.

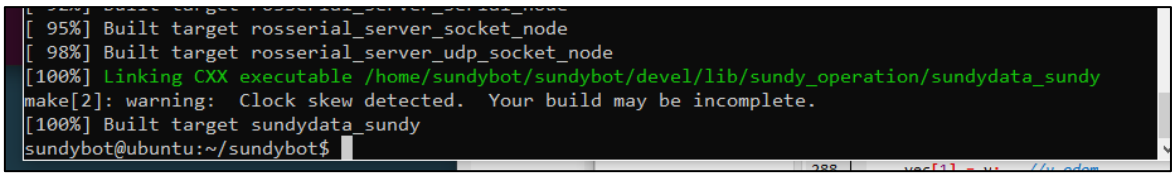



***Figura 61:*** *Fallo de compilación*

# Capítulo 6. CONSTRUCCIÓN DE ARQUITECTURA EN ROS

## 6.1 Configuración de ROS serial

Lo primero que se debe hacer antes de programar la comunicación mediante ROS serial es instalar la librería en el IDE de Arduino y, posteriormente, instalar su paquete correspondiente en el entorno de trabajo de ROS en la Raspberry Pi.

Para instalar ROS serial en el IDE de Arduino, basta con buscar la librería en el gestor de librerías del Arduino IDE.

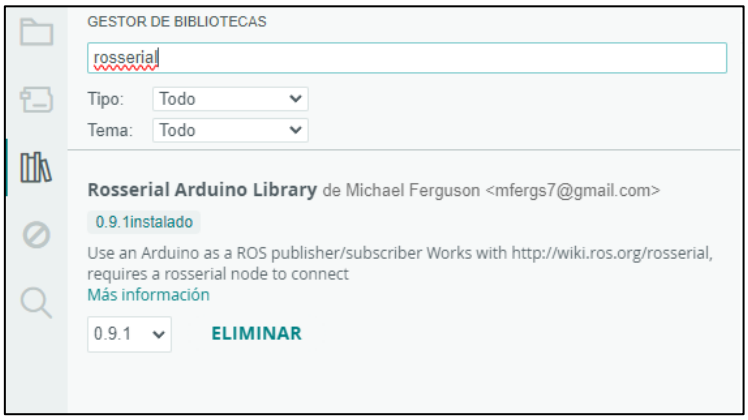



***Figura 62:*** *rosserial en Arduino IDE*

Para instalar ROS serial en el espacio de trabajo de ROS en la Raspberry Pi se deben ejecutar los siguientes comandos:

```
$ sudo apt install ros-noetic-rosserial
$ sudo apt install ros-noetic-rosserial-arduino
$ sudo apt install ros-noetic-rosserial-python
```

## 6.2 Programación de Arduino nano

El Arduino Nano se utiliza como complemento en el sistema de control de velocidad. Al estar conectado al encoder, el Arduino Nano puede calcular la velocidad lineal del vehículo mediante interrupciones. Basándose en este cálculo, el Arduino puede implementar un control PID para ajustar la velocidad.

### 6.2.1 Cálculo de RPM

Para calcular las RPMs del eje de giro del motor se utiliza la siguiente formula:

$$RPM = ({}^{pulsos}/_{seg}) \times \left(\frac{1rev}{IMP \times RDR}\right) \times \left(\frac{60\ seg}{1\ min}\right)$$

Siendo:

- IMP: Pulsos por Vuelta del encoder
- RDR: Relación de reducción

Como se mencionó en el apartado 3.1.3 del capítulo 3, el motor YSIDO está equipado con un encoder que tiene tres sensores de efecto Hall. Debido a las limitaciones en la cantidad de interrupciones que el Arduino nano puede manejar, se utilizan solamente

los sensores Hall B y Hall C. Estos están configurados en Arduino para detectar flancos de tipo CHANGE, lo que implica que reconocen tanto los flancos de subida como de bajada, como se puede ver en la figura 63. A partir de esta configuración se determina que el motor realiza cuatro pulsos por vuelta.

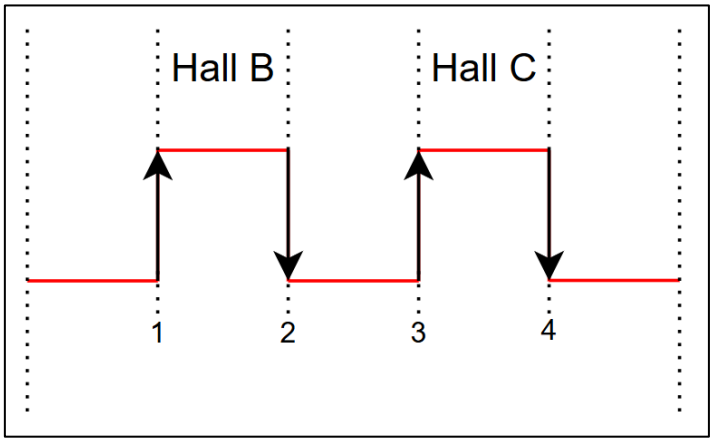



***Figura 63:*** *Detección de flanco CHANGE*

Antes de determinar la expresión final de las RPM del motor, es necesario calcular la RPM del eje de rueda del vehículo. Para ello, se debe tener en cuenta la relación de reducción existente entre el eje del motor y el eje de la rueda.

Para determinar este número experimentalmente, consideramos que se trata de un motor sin escobillas, cuyos rotores siguen el imán del estator. Dado que no están alimentados secuencialmente, los rotores se mueven forzosamente, generando pasos, cada uno correspondiendo a un pequeño giro del motor. Para medir la relación de reducción, se coloca una marca en el eje del motor y otra en la rueda, tal como se muestra en la Figura 64. El objetivo es calcular cuántas vueltas realiza el eje del motor por cada vuelta completa del eje de la rueda.

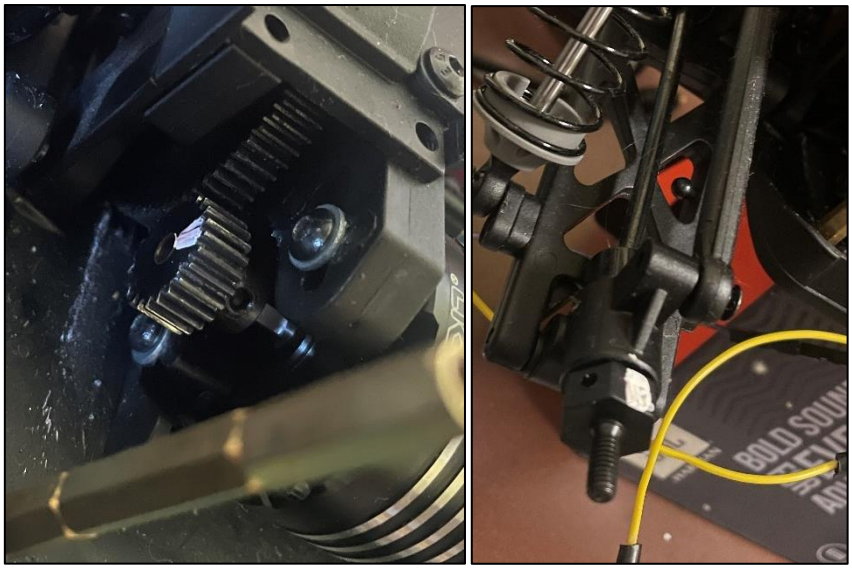


***Figura 64:*** *Configuración de Experimento*

Los resultados de los 3 experimentos realizados se muestran en la tabla 1.

| Experimento | Giro de motor | Giro de rueda |
|---|---|---|
| 1 | 9 | 1 |
| 2 | 9 | 1 |
| 3 | 9 | 1 |

***Tabla 1:*** *Resultados de relación de reducción*

Con estos resultados se determina que la relación de reducción es de 1/9, por cada nueve vueltas del motor es una vuelta de la rueda. Sabiendo los pulsos por vuelta y la relación de reducción, las ecuaciones finales para el calculo de las RPMs son las siguientes:

$$RPM_{motor} = \left({}^{pulsos}/_{seg}\right) \times \left(\frac{1rev}{4\frac{pulsos}{vuelta} \times 1}\right) \times \left(\frac{60\ seg}{1\ min}\right)$$

$$RPM_{rueda} = \left({}^{pulsos}/_{seg}\right) \times \left(\frac{1rev}{4\frac{pulsos}{vuelta} \times 9}\right) \times \left(\frac{60\ seg}{1\ min}\right)$$

Para obtener la velocidad lineal primero se debe calcular la velocidad angular en radianes por segundo

$$w = \frac{2\pi(RPM)}{60}$$

Una vez obtenida la velocidad angular en radianes por segundo, se utiliza la ecuación de velocidad lineal, la cual incluye como parámetro el radio de la rueda. Para este vehículo, el radio de la rueda es de 0.043 metros.

$$v = w \times r \ \rightarrow \ v = \left(\frac{2\pi(RPM)}{60}\right) \times 0.043m$$

Las señales analógicas producidas por sensores como los encoders a menudo generan ruido, el cual puede afectar la precisión de las mediciones.

El filtro exponencial de media móvil (EMA, por sus siglas en inglés) es un tipo específico de media ponderada que permite eliminar o reducir el ruido. En la programación del Arduino Nano, se utiliza la siguiente línea para filtrar la señal del encoder utilizando este filtro:

$$EMA = alpha \times valor + (1 - alpha) \times EMA$$

En este caso, el valor de alpha es 0.5. Este puede introducir un retardo en la señal de entrada, por lo que debe ser elegido adecuadamente para equilibrar la suavidad de la señal y la rapidez de respuesta del filtro.

Todos estos conceptos se aplican es una función de Arduino llamada RPM

```
205 void CalRPM() {
206   //***************CALCULO RPM***************
207   //*****************************************
208   if ((millis() - previusMillis) >= interval) {
209     RPM2 = 100.0 * contador * (60.0 / 36.0); //RPM rueda
210     RPMfilter2 = alpha * RPM2 + (1.0 - alpha) * RPMfilter2;
211     VangularR = 2.0 * PI * RPMfilter2 / 60;
212     VlinealR = VangularR * rRueda;
213     pv = VlinealR;
214     pulXseg = contador * 10;    //pulsos por segundo
215     contador = 0;               //reiniciamos contador
216     previusMillis = millis();   //igualamos al tiempo actual
217   }
218   //*****************************************
219   //*****************************************
220 }
```

El cálculo de los valores se realiza cada 100 ms, es decir, 0.1 segundos. Dado que los pulsos contados representan solo una fracción de un segundo, es necesario multiplicar esta cifra por 10 para obtener una estimación más precisa de los pulsos generados en un segundo completo.

En la línea 209, se realiza el cálculo de las RPM del eje del motor. Al igual que los pulsos contados, el tiempo se ajusta para estandarizarse a 1 segundo. Por lo tanto, se multiplica por 10 para convertir el conteo de 100 ms a 1000 ms, que equivale a un segundo completo. Esto asegura que tanto las RPM como los pulsos se calculen en una base de tiempo estándar.

### 6.2.2 Cálculo de RPM

Conocer la velocidad permite establecer un control en lazo cerrado usando un controlador PID, que estabiliza la velocidad en función de las señales de entrada.

El diagrama de bloques de control para realizar este control PID es el siguiente:

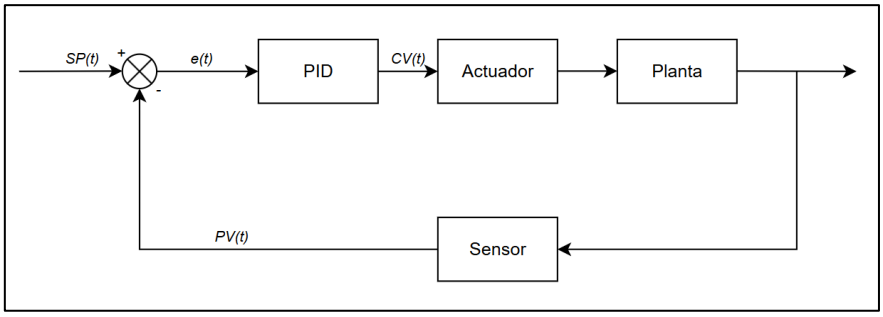



***Figura 65:*** *Diagrama de control*

Este sistema opera con variables continuas, mientras que el Arduino maneja variables discretas. Por lo tanto, es necesario discretizar la fórmula del control PID para adaptarla al funcionamiento del Arduino.

$$PID = \frac{kds^2 + kps + ki}{s}$$

Para discretizar esta función continua se debe añadir un retenedor, el cual aplicándolo y desarrollando los cálculos se presenta de la siguiente forma:

$$cv = cv(n-1) + \left(kp + \frac{kd}{Ts}\right)e(n) + \left(-kp + kiTs - 2\frac{kd}{Ts}\right)e(n-1) + \frac{kd}{Ts}e(n-2)$$

Es importante considerar que los parámetros kp, ki, kd del controlador PID se derivan de un modelo continuo.

Al implementar esta ecuación en Arduino, se debe tener en cuenta la recursividad de los parámetros para adaptarlos adecuadamente al entorno discreto del microcontrolador.

```
void PIDcontroller() {
  //***************CALCULO PID***************
  //*****************************************
  //Error
  error = sp - pv;
  //Ecuación de diferencias
  cv=cv1+(kp+kd/Tm)*error+(-kp+ki*Tm-2*kd/Tm)*error1+(kd/Tm)*error2;
  //recursividad error
  cv1 = cv;
  error2 = error1;
  error1 = error;
}
```

Se ha optado por esta simplificación para facilitar la explicación del concepto de recursividad. Es importante mencionar que el error debe actualizarse continuamente entre las variables de error para mantener la precisión de las diferencias en la ecuación.

Antes de avanzar al siguiente apartado, que presenta los resultados obtenidos, es importante explicar cómo se envían las señales al controlador de velocidad. A través de un programa complementario, que se incluye programa PWM, se logra determinar las señales PWM enviadas por un radiocontrol. Las señales emitidas son leídas por el Arduino y transmitidas al controlador de velocidad para determinar su funcionamiento.

Utilizando este programa, se determina que las señales PWM que regulan el funcionamiento del controlador de velocidad son las siguientes:

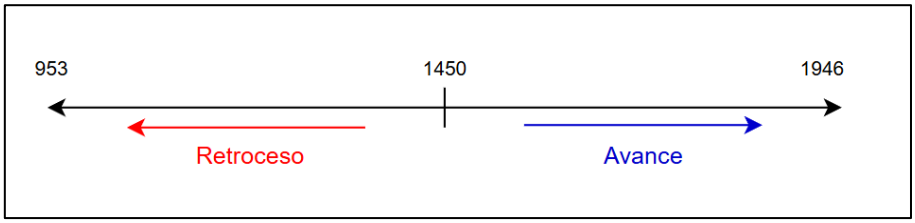



***Figura 66:*** *Diagrama de señal PWM*

Como se muestra en la Figura 66, existen dos rangos de señales que determinan el avance y retroceso del robot. Estos rangos también indican cómo se incrementa y decrece la velocidad según el sentido de movimiento.

Al enviar estas señales PWM, se determinaron los rangos de velocidad máxima que el vehículo alcanza tanto en avance como en retroceso. Estas velocidades se presentan en la tabla siguiente:

| Dato | Avance | Retroceso |
|---|---|---|
| Velocidad ang max (rad/s) | 502.65 | 307.18 |
| Velocidad lineal max (m/s) | 21.61 | 13.71 |
| RPM max | 4800 | 2933.33 |

***Tabla 2:*** *Resultados de relación de reducción*

Esta variable indica el sentido de giro y la velocidad deseada para el robot. Sin embargo, es necesario utilizar el encoder para verificar y determinar el sentido de giro real del motor.

En Arduino se crea un programa que permita saber el estado de los sensores Hall A y B, programa PULSOS. Utilizando este programa se determina que la secuencia para un sentido horario y antihorario del motor es la siguiente:

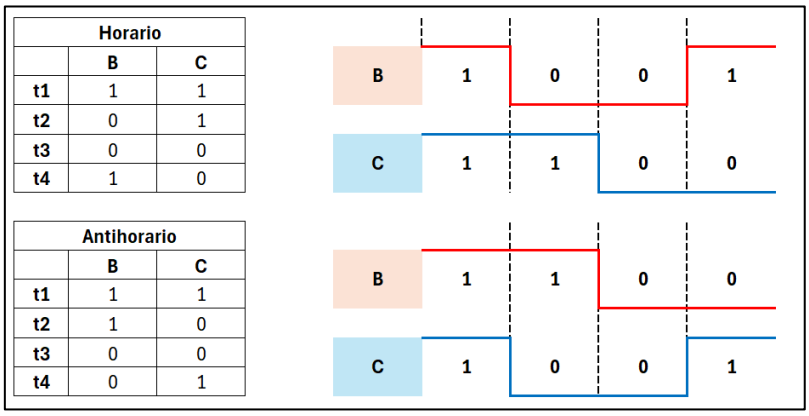



***Figura 67:*** *Secuencia de sentido de giro*

```
void interrupcion2() {
  contador++;
  sB = digitalRead(HALL_B);  // Leer estado del pin 2
  sC = digitalRead(HALL_C);  // Leer estado del pin 3
  if (sB == 0 && sC == 1 && sBa == 0 && sCa == 0) SGiro = VlinealR;
  if (sB == 1 && sC == 1 && sBa == 0 && sCa == 1) SGiro = VlinealR;
  if (sB == 1 && sC == 0 && sBa == 1 && sC == 1) SGiro = VlinealR;
  if (sB == 0 && sC == 0 && sBa == 1 && sCa == 1) sGiro = VlinealR;

  if (sB == 1 && sBa == 1 && sC == 1 && sCa == 0) sGiro = VlinealR * -1.0;
  if (sB == 0 && sBa == 1 && sC == 1 && sCa == 1) sGiro = VlinealR * -1.0;
  if (sB == 0 && sBa == 0 && sC == 0 && sCa == 1) sGiro = VlinealR * -1.0;
  if (sB == 1 && sBa == 0 && sC == 0 && sCa == 1) sGiro = VlinealR * -1.0;
  stateBant = stateB;
  stateCant = stateC;
}
```

Esta versión simplificada, muestra cómo se programa la tabla de valores y secuencias mencionada anteriormente.

Como se observa, dependiendo de la secuencia de giro detectada, el código multiplica la velocidad lineal por -1.0. Esto permite ajustar la velocidad, indicando así el sentido de giro del motor.

Tomando en cuenta todos estos conceptos, el controlador debe ser capaz de regular la velocidad independientemente del sentido de giro que presente. Para ello, se debe comparar la velocidad lineal deseada, que será el set point del controlador, hay que recordar que esta velocidad lleva sentido de giro por lo cual puede ser negativa, los set point no deben ser negativos, para ello se debe comparar esta velocidad y ajustar dicho valor

```
if (VelocidadLineal >= 0) {
    sp = VelocidadLineal;
  }
  if (VelocidadLineal < 0) {
    sp = VelocidadLineal * -1;
  }
```

Considerando la referencia de velocidad lineal, la variable de control calculada por el regulador debe ser mapeada de acuerdo con los valores específicos de PWM. Dado que la función map de Arduino solo maneja números enteros, se ha creado una función alternativa que realiza la misma operación, pero con valores flotantes.

Una vez desarrollada esta función, y conociendo los valores máximos de velocidad alcanzados por el vehículo mostrados en la tabla 2, se utilizó esta nueva función para determinar los valores mapeados correspondientes al vehículo.

Para enviar la señal PWM mapeada, se hace uso de la librería de Arduino servo. Declarando un objeto llamado myESC se escribe la señal en microsegundos.

```
if (VelocidadLineal >= 0) {
    //mapeo para avance
    vel = mapear(cv, 0.0, 20.0, 1450.0, 1946.0);
  } else if (VelocidadLineal < 0) {
    //mapeo para retroceso
    vel = mapear(cv, 0.0, 20.0, 1450.0, 953.0);
  }
myESC.writeMicroseconds(vel);
```

Por último, un parámetro importante a considerar es la saturación del controlador. Ante perturbaciones muy altas, el controlador puede generar una variable de control excesivamente alta, superando los valores máximos de velocidad del motor. Aunque en este caso no se ve afectado gracias a que el controlador satura la señal PWM enviada, como medida de seguridad se limitará la velocidad máxima del controlador a 5 m/s. Si se desea aumentar este límite, simplemente se debe cambiar el valor especificado, el cual no debe exceder los 20 m/s.

```
//saturación de salida
  if (cv > 5.0) {
    cv = 5.0; //valor maximo de 20
  }
```

### 6.2.3 Resultados de RPM y ajuste de control PID

**Resultados de RPM**

Debido a que ciertos parámetros fueron calculados de manera experimental, es importante verificar estos parámetros.

Para comprobar el cálculo de las RPM, se instaló un encoder óptico en el eje de la rueda, como se muestra en la Figura 64. Utilizando un Arduino Mega, se programó la función calculada anteriormente en los pines de interrupción 0 y 1. El pin de señal del sensor óptico se conectó al pin de interrupción 3.

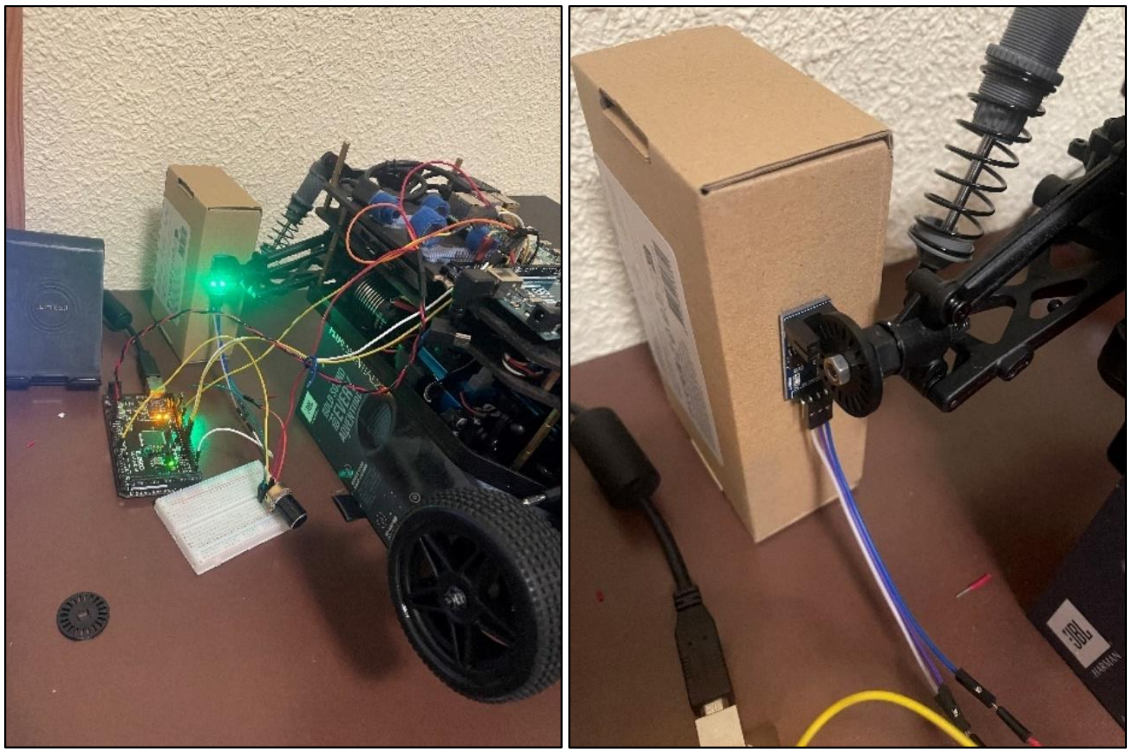


***Figura 68:*** *Comprobación de RPM*

Se realizaron tres pruebas para simular diferentes situaciones: la primera prueba simuló una velocidad constante, la segunda varió ligeramente la velocidad, y la última simuló una situación de cambio constante de velocidad.

Los resultados obtenidos del experimento realizado son los siguientes:

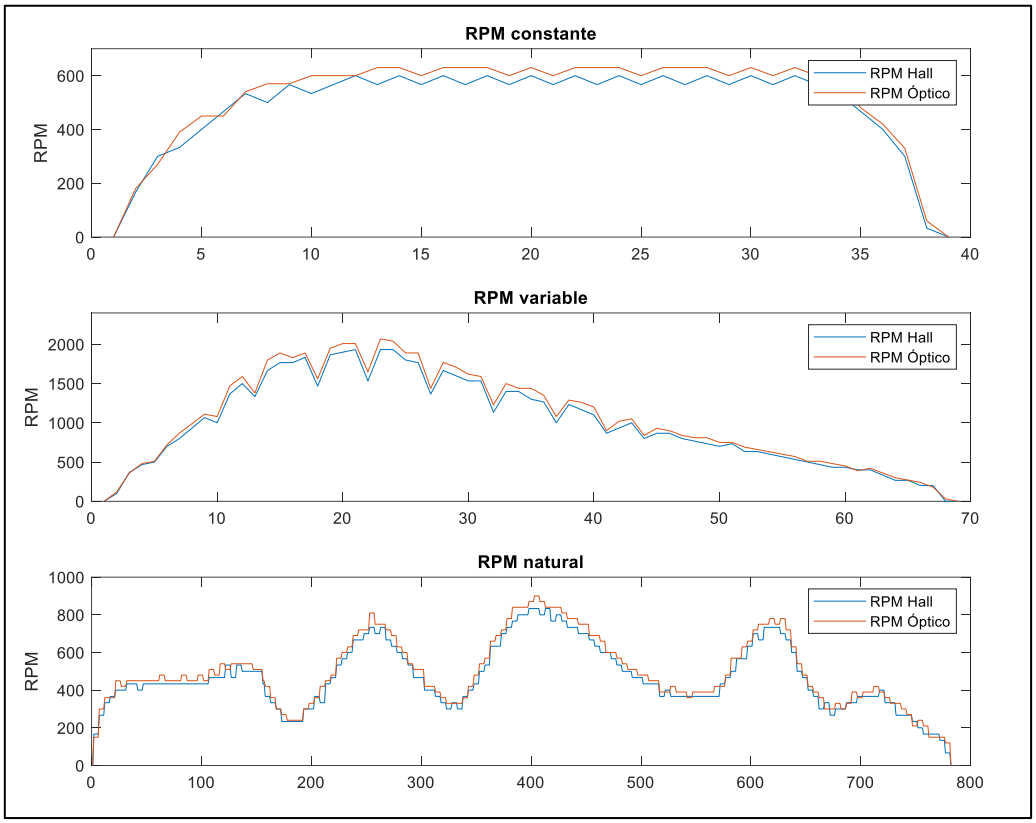



***Figura 69:*** *Graficas RPM Matlab*

Comparando los resultados, se observa que la baja resolución del encoder de efecto Hall afecta la precisión y resolución de la señal.

El encoder óptico registra 20 pulsos por vuelta, mientras que el encoder de efecto Hall registra solamente 4 pulsos por vuelta. La diferencia de 20 pulsos se refleja en que el encoder de efecto Hall siempre registra una medición menor en comparación con el encoder óptico.

**Ajuste regulador PID**

Para determinar los valores kp, pi y kd, que se asignan al regulador, se realizaron diversas pruebas con diferentes configuraciones. Este documento presenta las seis pruebas más significativas.

El primer caso presentado muestra un resultado que no se considera satisfactorio, ya que ilustra el comportamiento de un sistema inestable.

Utilizando un graficador en tiempo real escrito en Python 3 con la librería Matplotlib, se obtuvo la señal que se muestra en la Figura 70.

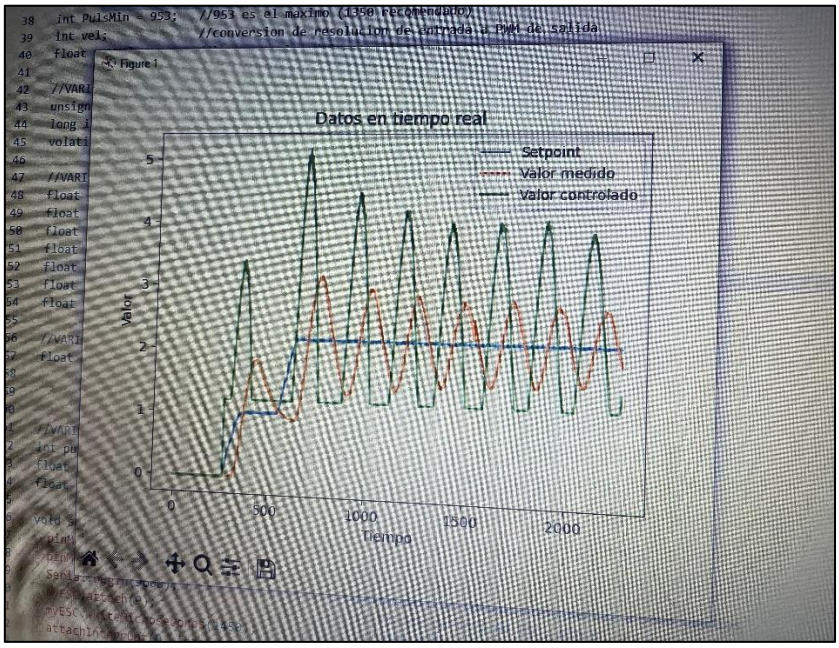



***Figura 70:*** *PID Matplotlib*

Los valores colocados para que este sistema fuese inestable fueron kp=1, ki=2 y kd=1. Con un Alpha de 0.6.

Durante el cálculo de las RPM se mencionó que las señales producidas por los sensores analógicos presentan ruido, en la figura 71, se puede ver como esto influye en el regulador PID.

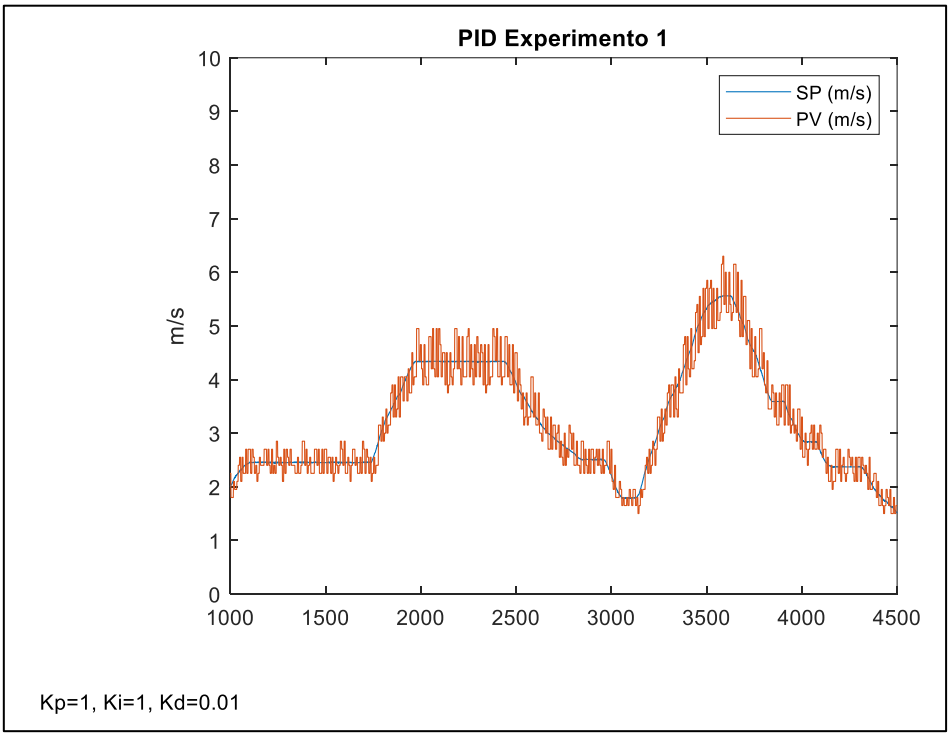



***Figura 71:*** *Experimento 1 Matlab*

Los experimentos a continuación filtran la señal, pero empiezan a presentar cierto retardo en la reacción del controlador.

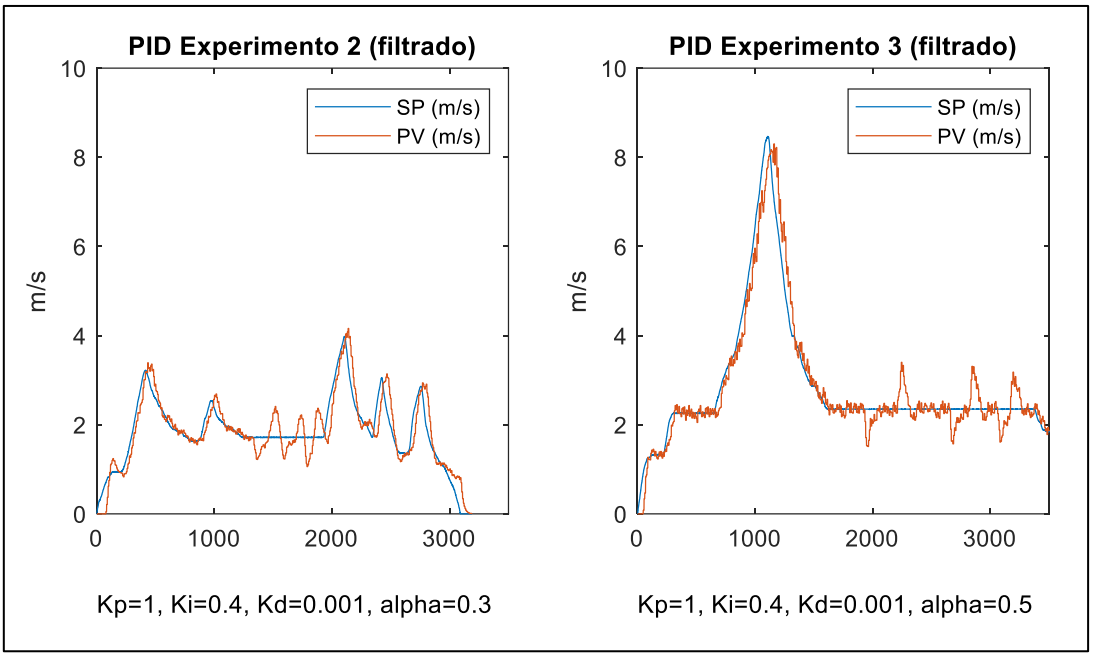



***Figura 72:*** *Experimento 2 y 3 Matlab*

El retardo observado en el experimento 2, mostrado en la Figura 72, se debe al uso del filtro EMA para limpiar la señal. Cuanto más cercano a cero sea el valor del parámetro Alpha, mayor será el retardo en la respuesta del filtro. Sin embargo, existe una compensación: un valor más alto de Alpha puede reducir el retardo, pero incrementará el error en la medición.

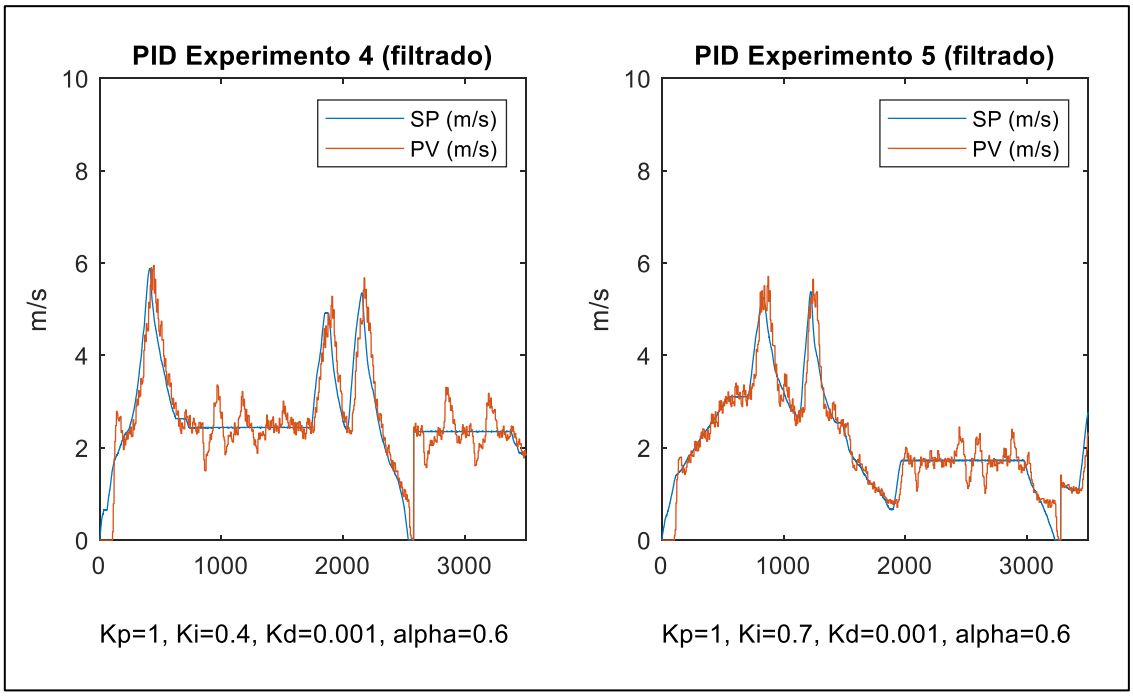



***Figura 73:*** *Experimento 4 y 5 Matlab*

Basándonos en los experimentos anteriores, los experimentos 4 y 5, mostrados en la Figura 72, incluyeron pequeñas variaciones en los valores con el objetivo de determinar si estos cambios menores podrían mejorar o empeorar el comportamiento de la señal.

Dado que estos ajustes mínimos no tuvieron una gran influencia en los resultados finales, los parámetros seleccionados para configurar en el controlador son los siguientes:

```
kp = 1.0;     //ganancia proporcional
ki = 0.5;     //ganancia integral
kd = 0.001;   //ganancia derivativa
Alpha = 0.5;
```

Todos los valores obtenidos para realizar estas pruebas provinieron de un programa en Python 3 que, utilizando la librería SerialCOM, accedió a los datos enviados a través del monitor serial. Se incluye el enlace programa SC.

### 6.2.4 Envio de PWM a servo motor

Antes de explicar cómo se envían los datos al servo motor, es importante considerar su orientación. Según el manual de instrucciones del fabricante, el servo motor debe colocarse horizontalmente, tal como se muestra en la Figura 74.

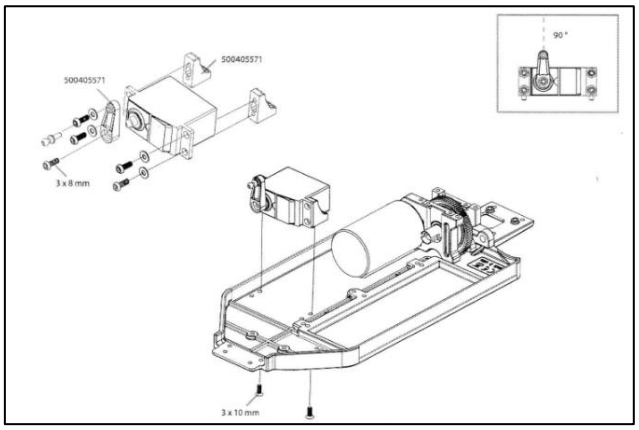



***Figura 74:*** *Ensamble de Servo motor, figura tomada del Manual de Carson Dirtwarrior*

Esto significa que cuando el motor está en 90 grados, indica la dirección central del vehículo. Si está en la posición de 0 grados, girará 45 grados hacia la izquierda, y si está en la posición de 180 grados, girará 45 grados hacia la derecha.

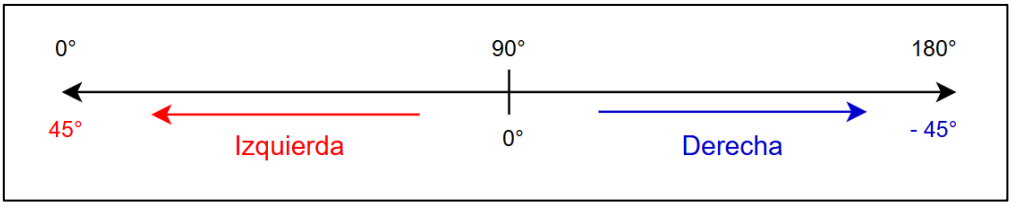



***Figura 75:*** *Sentido de Giro servo motor*

Conociendo la orientación del servo motor, para programarlo en Arduino se utiliza la librería Servo. Se declara un objeto llamado servo1, al cual se le asigna el valor de giro deseado. Este valor se mapea utilizando la función de mapeo para valores flotantes mencionada anteriormente.

```
servo1.write(AnguloGiroAck);

void AckermanAngle() {
  AnguloGiroAck = mapear(AnguloGiro, 0.78, -0.78, 0.0, 180.0);
}
```

El mapeo se realiza conforme a lo explicado e ilustrado en la Figura 71. Los valores de control de giro se proporcionan en radianes, con 0 grados como orientación central. Por tanto, es necesario convertirlos a grados para el servo motor, que opera en un rango de 0 a 180 grados, siendo 90 grados la posición central.

### 6.2.5 Envío y recepción de datos con ROS serial

En la arquitectura de ROS, el Arduino Nano funciona como un nodo más del sistema, el mismo en este caso actúa como un nodo suscriptor y publicador. Se suscribe al tópico de velocidad, que emite mensajes de tipo geometry_msgs/Twist, y publica datos en el tópico ArduinoUno_data, que maneja valores de tipo std_msgs/Float32MultiArray.

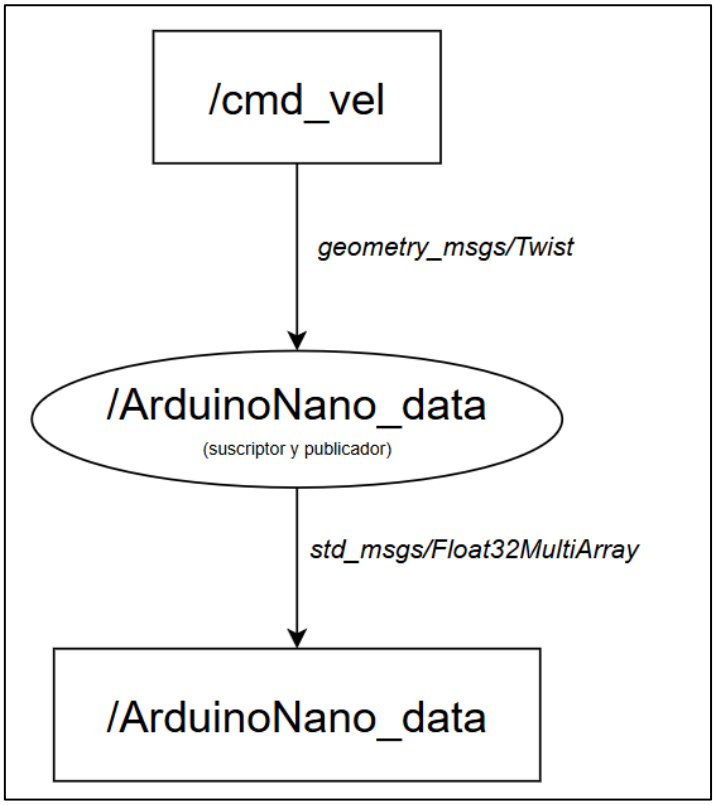



***Figura 76:*** *Comunicación de Arduino nano*

Los mensajes de tipo geometry_msgs/Twist cuentan con 6 datos divididos en dos vectores (velocidad lineal y angular) tal como se muestra en la figura 77.

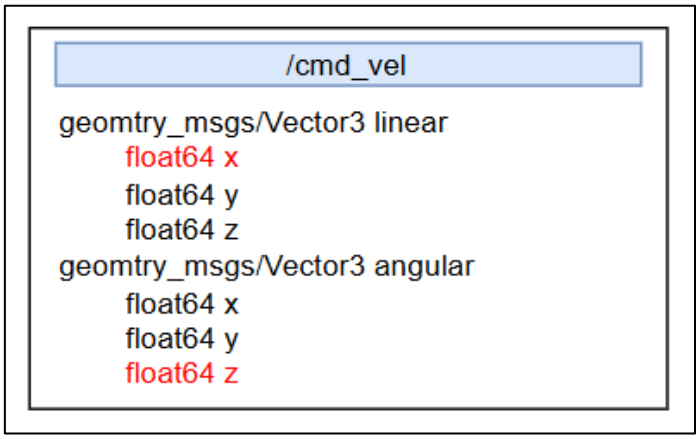



***Figura 77:*** *Estructura mensaje Twist*

En este caso, el Arduino Nano accede a los valores del campo x del vector lineal y del campo z del vector angular, que contienen los valores de velocidad lineal y angular.

Para establecer comunicación, se crea un nodo en ROS utilizando la librería ROS Serial. Luego, se configura la suscripción al tópico de nombre “cmd_vel”. Este suscriptor accede a los campos del mensaje geometry_msgs/Twist, y almacena los valores de linear.x y angular.z en sus variables correspondientes.

```
ros::NodeHandle nh;  //configuracion del nodo
ros::Subscriber<geometry_msgs::Twist> sub_cmd_vel("cmd_vel", cmdVelCallback);

//Callback para suscriptor de cmd_vel
void cmdVelCallback(const geometry_msgs::Twist &vel) {
  VelocidadLineal = vel.linear.x;
  AnguloGiro = vel.angular.z;
}

nh.subscribe(sub_cmd_vel);
```

El Arduino calcula múltiples valores que deben ser publicados para su manipulación. Para enviar estos datos, se utiliza el tipo de mensaje std_msgs/Float32MultiArray.

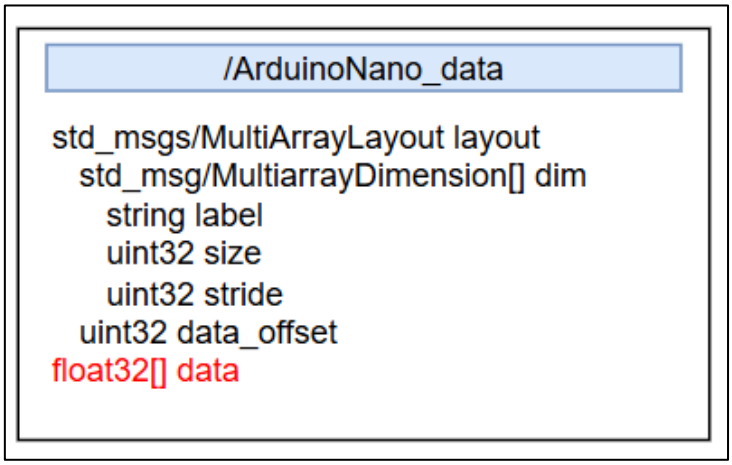



***Figura 78:*** *Estructura mensaje Float32Multiarray*

Este tipo de mensaje tiene varios campos que se llenan para especificar información detallada del vector, como su tamaño, nombre, entre otros detalles, tal como se ve en la figura 78.

Utilizando este tipo de mensajes en Arduino nano, se declara un vector flotante data de 10 elementos, los cuales almacenan los valores de las variables a enviar. Estas variables se transmiten a través de un mensaje std_msgs/Float32MultiArray en ROS.

Para publicar los valores, se utiliza la función de publicación del objeto nodo creado anteriormente, y se le coloca el nombre de “ArduinoNano_data”.

```
ros::NodeHandle nh;  //configuracion del nodo
std_msgs::Float32MultiArray encoder_msg;
ros::Publisher pubPM("ArduinoNano_data", &encoder_msg);

//Variable Array
float data[10] = {};

nh.advertise(pubPM);

  //Envio de Datos************************
  encoder_msg.data_length = 10;  //tamaño del vecto
  data[0] = VelocidadLineal;     //Velocidad lineal introducida
  data[1] = sp;                  //Setpoit
  data[2] = cv;                  //Variable de control
  data[3] = pv;                  //Variable de proceso
  data[4] = RPMfilter2;          //RPM de la Rueda
  data[5] = VangularR;           //Velocidad Angular de la Rueda
  data[6] = VlinealR;            //Velocidad lineal de la Rueda
  data[7] = sentidoGiro;         //Velocidad lineal con sent de Giro
  data[8] = vel;                 //PWM enviadas al motor
  data[9] = AnguloGiro;          //Angulo de Giro
  encoder_msg.data = data;
  //************************************
pubPM.publish(&encoder_msg);
nh.spinOnce();
```

El diagrama final para el Arduino Nano queda de la siguiente manera:

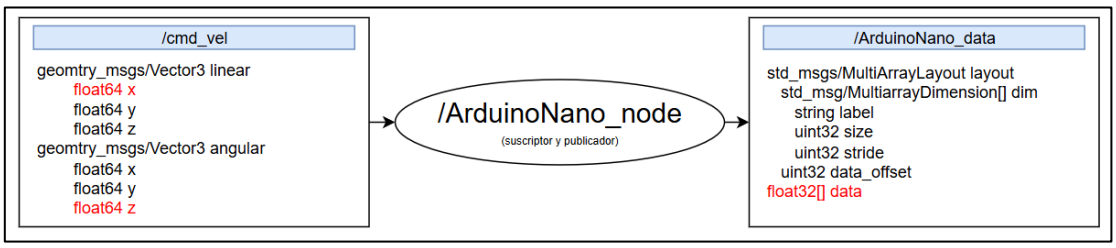



***Figura 79:*** *Comunicación final Arduino Nano*

## 6.3 Programación de Arduino nano

### 6.3.1 Programación sensor D0-25V

El Arduino Uno se utiliza para obtener datos del sensor inercial BNO055 y para leer la señal analógica del sensor D0-25V, que mide el voltaje de la batería Lipo.

Para leer el sensor D0-25V se utiliza el pin analógico 2. Considerando que la resolución de las entradas analógicas del Arduino es de 1023, se lleva a cabo una conversión con la siguiente ecuación:

$$Voltaje = lectura \times \left(\frac{5.0}{1023}\right) \times 5.0$$

La multiplicación por cinco del valor transformado se debe a que el valor máximo que puede medir el sensor es de 25 voltios. Dado que el máximo que puede medir el Arduino es de 5 voltios, al multiplicar este valor por cinco se obtienen los 25 voltios que corresponden al rango máximo del sensor.

Llevado esto al Arduino Uno se utiliza una función que lee el valor de voltaje, lo calcula y transforma seguidamente:

```
void Voltagelecture() {
  // Lee el valor de los pines analógicos A3 y A2
  PBlecture = analogRead(A2);     //voltage en Powerbank
  LIPOlecture = analogRead(A3);   //voltage en LIPO

  // Convierte el valor analógico a voltage
  PBvoltage = PBlecture * (5.0 / 1023.0) * 5.0;
  LIPOvoltage = LIPOlecture * (5.0 / 1023.0) * 5.0;
}
```

En la función mostrada, se lleva a cabo el cálculo para determinar el voltaje de dos tipos de baterías: la batería LiPo, que se utiliza en la etapa de control, y la powerbank, empleada en la etapa de potencia.

### 6.3.2 Programación sensor BNO055

Para poder obtener los valores de manera fácil se hace uso de la librería BNO055 de Adafruit la cual tiene todas las funciones necesarias para manipular los valores.

El sensor BNO055, que integra un microcontrolador ARM Cortex-M0, requiere calibración para determinar la orientación correctamente. Esta calibración se considera completa cuando la variable de calibración alcanza un valor de 3, indicando que el acelerómetro, el giroscopio y el magnetómetro están totalmente calibrados. Este proceso de calibración se realiza cada vez que se enciende el dispositivo.

Para realizar este proceso de calibración una sola vez, se implementa dentro de la función void setup() de Arduino un bucle while. Este bucle continúa hasta que el dispositivo se calibra completamente, momento en el cual se sale del bucle.

```
while (calibration != 3) {
      uint8_t sys, gyro, accel, mag;

      //Verificar el estado de calibración del sensor
      bno.getCalibration(&sys, &gyro, &accel, &mag);
      calibration = mag;
    }
```

Al entrar en el void loop(), se calculan los valores del sensor. Este sensor proporciona la orientación en cuaterniones, pero incluye una función que permite transformarlos en ángulos de Euler expresados en radianes.

Durante la configuración del sensor, se determinó que los valores de orientación presentaban errores de referencia con respecto a los ejes marcados, los cuales se pueden observar en la Figura 23.

```
//***********Calculo de Valores******
//*********************************
    Roll = (euler.z() - 0.044);    //Roll
    Pitch = (euler.y() - 0.047);   //Pitch
    Yaw = euler.x();               //Yaw
    if (Yaw < 0) {
      Yaw = Yaw + 6.28319;
      if (Yaw == 6.28319) {
        Yaw = 0.0;
      }
    }
```

Al revisar el código, se puede observar que los ángulos de Euler calculados por la librería no coinciden con los ejes de giro esperados. Para ello utilizando el valor solamente se ajusta a la variable correspondiente de giro Roll, pitch o Yaw.

Los valores que se restan se deben a que el sensor y la posición en la que están no se encuentran correctamente planas. Esto se determinó utilizando un programa de visualización 3D creado en Python, el cual es adjuntado programa RPY.

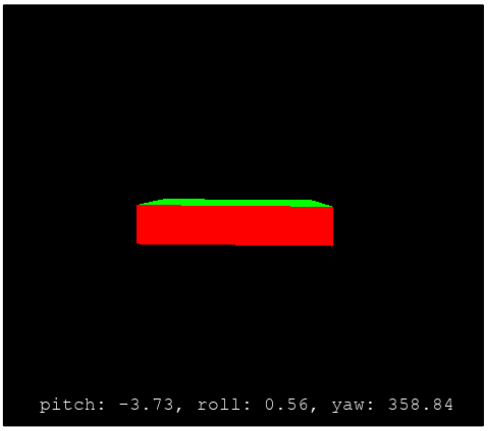



***Figura 80:*** *Visualización 3D del BNO055*

Para evitar errores en el cálculo del Yaw, si se obtiene algún valor negativo, este se convierte en positivo mediante la transformación adecuada del valor.

### 6.3.3 Envío de datos con ROS serial

En la arquitectura de ROS, el Arduino Uno al igual que el Arduino Nano funciona como un nodo más del sistema, Para este caso actúa como publicador de mensaje de tipo std_msgs/Float32MultiArray.

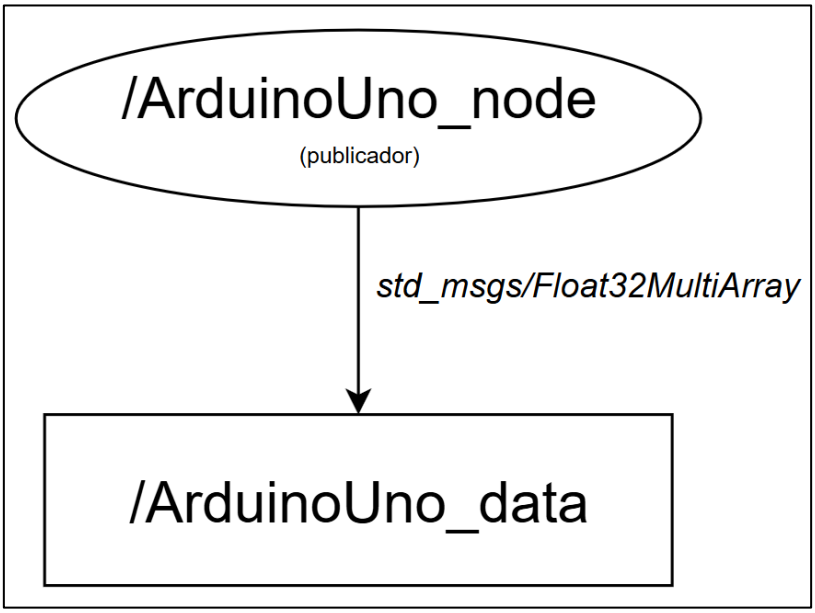



***Figura 81:*** *Comunicación Arduino Uno*

El tópico ArduinoUno_data contiene los datos de Roll, Pitch y Yaw, y además incluye un valor de bandera (flag) que indica el estado de calibración del sensor. Durante la etapa de calibración, el valor del flag será 2. Si el sensor está en óptimo funcionamiento, la variable flag tendrá el valor 3. En caso de que la IMU no esté instalada, el valor del flag será 1.

Para enviar los datos se utiliza el mensaje std_msgs/Float32Multiarray, al igual que se utilizo para enviar los datos del Arduino Uno.

```
//Envio de Datos************************
  Auno_msg.data_length = 5;  //tamaño del vecto
  data[0] = Roll;     //Velocidad lineal introducida
  data[1] = Pitch;                  //Setpoit
  data[2] = Yaw;                  //Variable de control
  data[3] = flag;                  //Variable de proceso
  data[4] = PBvoltage;          //RPM de la Rueda
  data[5] = LIPOvoltage;           //Velocidad Angular de la Rueda
  Auno_msg.data = data;
pub.publish(&Auno_msg);
nh.spinOnce();
```

El código simplificado ilustra los comandos utilizados para enviar los datos obtenidos por el sensor de manera resumida.

No obstante, durante la calibración del sensor es necesario continuar enviando los datos. Por lo tanto, esta función debe ser programada como un bucle que envíe de forma constante la información a ROS. Aquí se presenta cómo quedaría estructurada dicha función:

```
while (calibration != 3) {
      uint8_t sys, gyro, accel, mag;
      flag = 2.0;  //IMU calibrando

      //Verificar el estado de calibración del sensor
      bno.getCalibration(&sys, &gyro, &accel, &mag);
      calibration = mag;
      Voltagelecture();
  //Envio de Datos************************
  Auno_msg.data_length = 5;  //tamaño del vecto
  data[0] = Roll;     //Velocidad lineal introducida
  data[1] = Pitch;                  //Setpoit
  data[2] = Yaw;                  //Variable de control
  data[3] = flag;                  //Variable de proceso
  data[4] = PBvoltage;          //RPM de la Rueda
  data[5] = LIPOvoltage;           //Velocidad Angular de la Rueda
  Auno_msg.data = data;
pub.publish(&Auno_msg);
nh.spinOnce();
}
```

Aunque se trate de un bucle while temporal, es necesario incluir las funciones de publicación de datos y ejecutar el spinOnce() dentro del bucle para mantener actualizada la comunicación con el nodo de ROS.

El diagrama final para el Arduino Uno queda de la siguiente manera:

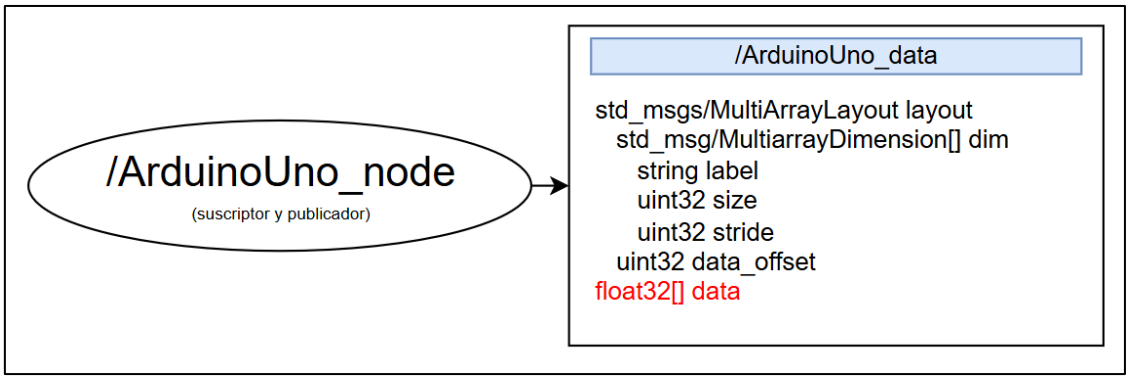



***Figura 82:*** *Comunicación final Arduino Uno*

## 6.4 Programación de nodo maestro

Aunque tanto el Arduino Nano como el Arduino Uno están publicando información en el formato de mensaje std_msgs/Float32MultiArray, en el entorno de ROS es convencional utilizar mensajes específicos para los datos de los sensores, como sensor_msgs/Imu, nav_msgs/Odometry o sensor_msgs/LaserScan.

Estos mensajes tienen formatos específicos que, dentro de la arquitectura de ROS, permiten seguir reglas estandarizadas al momento de publicar datos, facilitando la interoperabilidad y la integración en sistemas más complejos.

Los Arduinos, gracias a la librería ROS Serial, pueden transmitir esta información directamente en la arquitectura de ROS. Sin embargo, el objetivo es que esta arquitectura pueda adaptarse a las necesidades específicas de cada proyecto en el futuro.

Los datos enviados en std_msgs/Float32MultiArray, para esta arquitectura, son gestionados por el nodo Sundy_Collector. Este programa se suscribe a los tópicos ArduinoUno_data y ArduinoNano_data para transformar estos valores en formatos estandarizados de la arquitectura ROS.

Por otra parte, este programa maestro realiza cálculos de odometría y efectúa transformaciones estáticas y dinámicas. Además, emite un tópico adicional denominado Data_total, que se encarga de preparar los datos para su visualización en una interfaz gráfica fuera del entorno de ROS.

La estructura de esta parte de la arquitectura debe presentarse de la siguiente manera:

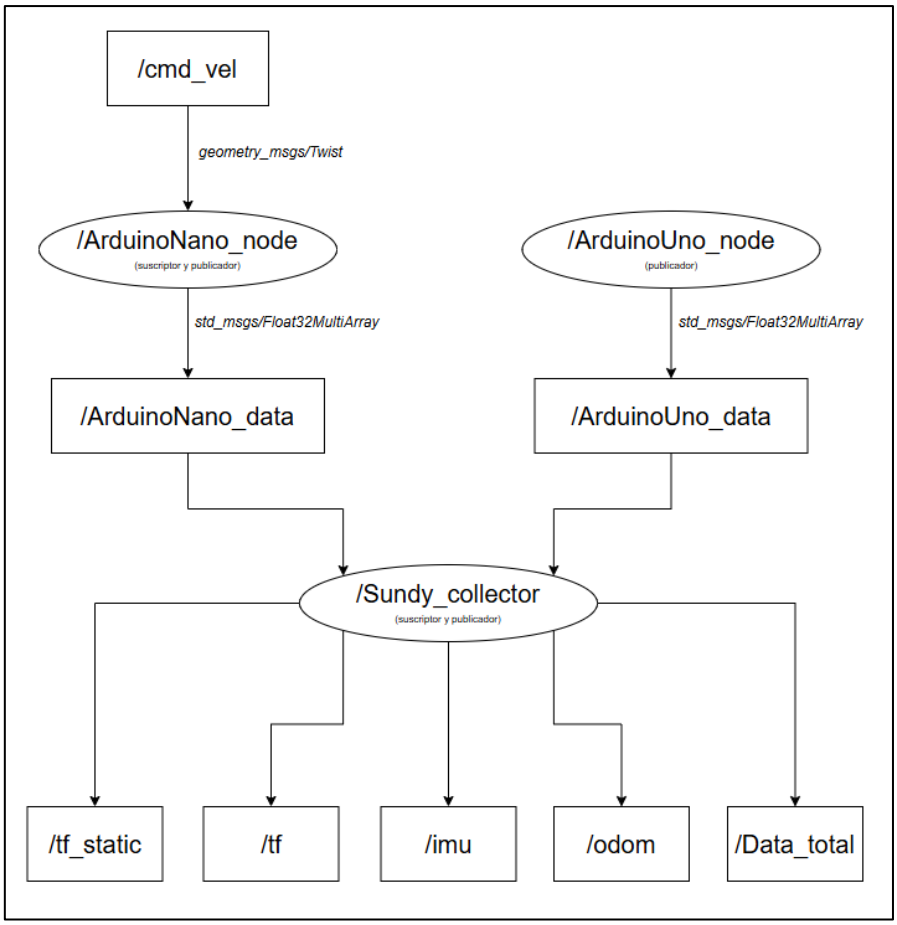



***Figura 83:*** *Arquitectura con Arduino Uno y Nano*

Antes de emitir mensajes, es necesario recibir los datos provenientes de los Arduinos. Recordemos que los Arduinos funcionan como nodos que emiten mensajes del tipo

std_msgs/Float32MultiArray. En este caso, el nodo maestro debe suscribirse a dichos tópicos para procesar la información recibida.

```
//Callback Arduino nano
void Cb1(const std_msgs::Float32MultiArray::ConstPtr& msg) {
  VelocidadLineal = msg->data[0];
  sp = msg->data[1];
  cv = msg->data[2];
  pv = msg->data[3];
  RPMrueda = msg->data[4];
  RPMmotor = RPMrueda * 9;
  VangularR = msg->data[5];
  VlinealR = msg->data[6];
  sentidoGiro = msg->data[7];
  PWM = msg->data[8];
  AnguloGiro = msg->data[9];
}
//Callback de Arduino UNO
void Cb2(const std_msgs::Float32MultiArray::ConstPtr& msg2) {
  RollA = msg2->data[0];
  PitchA = msg2->data[1];
  YawA = msg2->data[2];
  EstadoBNO055 = msg2->data[3];
  PBvoltage = msg2->data[4];
  LIPOvoltage = msg2->data[5];
}

void Cb3(const std_msgs::Int32::ConstPtr& msg3) {
  flag = msg3->data;
}

//Suscriptores
ros::Subscriber sub1 = n.subscribe("ArduinoNano_data", 1000, Cb1);
ros::Subscriber sub2 = n.subscribe("ArduinoUno_data", 1000, Cb2);
ros::Subscriber sub3 = n.subscribe("flag_estado", 1000, Cb3);
```

Se declaran la suscripción a los tópicos de Arduino uno y Arduino nano, y se guardan los valores en las variables respectivamente.

En la estructura mostrada en la Figura 83, no se observa la suscripción al tópico flag_estado. Esto se debe a que no se ha implementado ninguna interfaz gráfica; este tópico publica ciertos valores (estados) que deben considerar las variables manipuladas por el nodo maestro. La variable flag reinicia variables de odometría e IMU.

Cuando se realiza una trayectoria y la misma es completada, es necesario reiniciar las variables para poder ejecutar otras trayectorias. Este flag indica al nodo maestro que dichas variables deben reiniciarse o reiniciarse según la instrucción dada desde la interfaz gráfica. La variable flag realiza las siguientes acciones según su valor:

- **Reiniciar valores de odometría (1):** Reinicia los valores de odometría y ajusta el offset para orientar la IMU según la dirección actual.

- **Reiniciar valores de RPY IMU (2):** Reinicia los valores de orientación Roll, pitch y Yaw de la IMU.

Este nodo maestro, denominado Sundy_Collector, es un programa extenso que realiza la publicación de diferentes tópicos en un orden específico, con el fin de entender de una manera ordenada la programación de este nodo se divide en puntos los diferentes bloques de programación de publicación de tópicos.

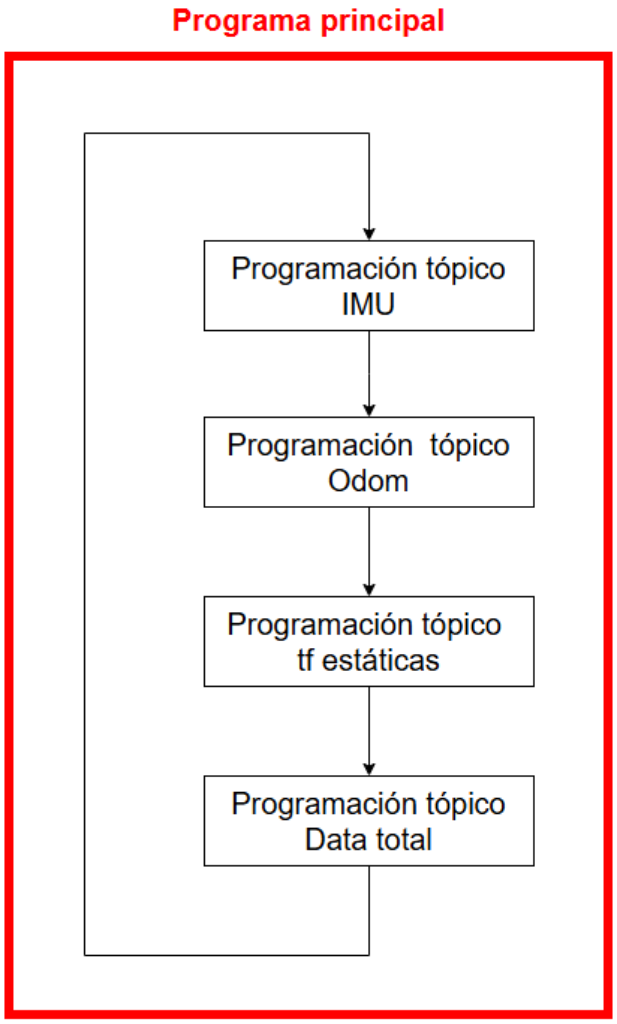



***Figura 84:*** *Estructura secuencia del nodo maestro*

### 6.4.1 Programación de tópico imu

Los datos del sensor BNO055 son enviados por el Arduino Uno como roll, pitch y yaw. Dependiendo del valor del indicador flag, se calcula un offset para estos datos o se resetea dicho offset.

```
//Callback Arduino nano
if (flag == 1) {
  offset();
}
if (flag == 2) {
  RollOFF = 0.0;   //Roll
  PitchOFF = 0.0;  //Pitch
  YawOFF = 0.0;    //Yaw
}
Roll = RollA - RollOFF;
Pitch = PitchA - PitchOFF;
Yaw = YawA - YawOFF;
if (Yaw < 0) {
  Yaw = Yaw + 6.28319;
  if (Yaw == 6.28319) {
    Yaw = 0.0;
  }
}

//Función que muestra offset
void offset() {
  //reseteamos variables
  RollOFF = 0.0;   //Roll
  PitchOFF = 0.0;  //Pitch
  YawOFF = 0.0;    //Yaw
  for (int8_t i = 0; i < 20; i++) {

    //Guardar valores
    RollOFF = RollOFF + RollA;     //Roll
    PitchOFF = PitchOFF + PitchA;  //Pitch
    YawOFF = YawOFF + YawA;        //Yaw
  }
  RollOFF = RollOFF / 20.0;    //Roll
  PitchOFF = PitchOFF / 20.0;  //Pitch
  YawOFF = YawOFF / 20.0;      //Yaw
}
```

La función de offset mide 20 valores y luego calcula la media de estos. El valor medio obtenido se resta del valor real del ángulo proporcionado por el Arduino, que se recibe a través del tópico ArduinoUno_data.

Para emitir estos datos, es necesario convertir los ángulos de orientación en roll, pitch y yaw a cuaterniones. Esto se realiza utilizando las librerías disponibles en ROS, que facilitan la conversión de estos formatos.

```
tf::Quaternion q = tf::createQuaternionFromRPY(Roll, Pitch, Yaw);
sensor_msgs::Imu imu_msg; // Crea un mensaje de IMU
imu_msg.header.stamp = ros::Time::now();
imu_msg.header.frame_id = "imu_sundy";
quaternionTFToMsg(q, imu_msg.orientation);
pub.publish(imu_msg);
```

Una vez los datos transformados se crea un mensaje de tipo IMU, en el cual se define el nombre del marco de colocación del sensor.

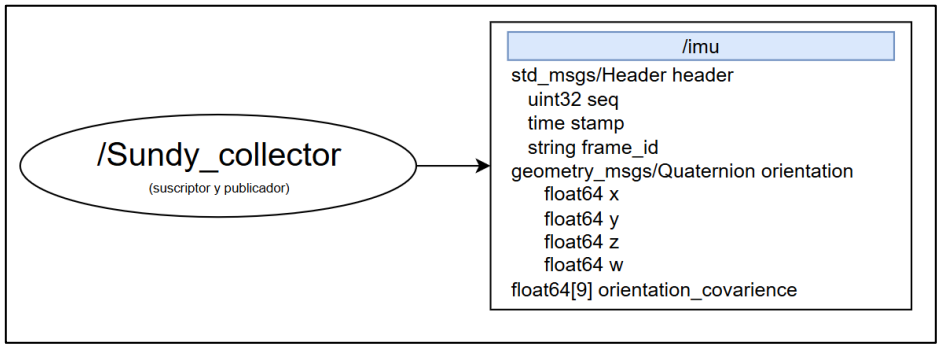



***Figura 85:*** *Comunicación tópico odom*

Los mensajes del tipo sensor_msgs/Imu incluyen campos para la orientación, la aceleración y las mediciones del giroscopio. En este caso, solo se proporciona la orientación, por lo que los campos de aceleración y giroscopio no se utilizan y se dejan sin datos.

### 6.4.2 Programación de tópico odom

Para el tópico de odometría, se utilizan las ecuaciones cinemáticas que describen el movimiento de un vehículo con dirección tipo Ackermann. Estas ecuaciones se describen de la siguiente manera:

**Velocidad**

$$\dot{x} = v\cos\theta \rightarrow m/s$$
$$\dot{y} = v\,sen\theta \rightarrow m/s$$
$$\dot{\theta} = \frac{v\tan\varphi}{L} \rightarrow \frac{rad}{s}$$

**Posición**

$$x = \dot{x} \times t \rightarrow m$$
$$y = \dot{y} \times t \rightarrow m$$
$$\theta = \dot{\theta} \times t \rightarrow rad$$

**Actualización de posición**

$$x' = x' + x \rightarrow m$$
$$y' = y' + y \rightarrow m$$
$$\theta' = \theta' + \theta \rightarrow rad$$

Las ecuaciones dan como resultado la estimación de la posición del robot en base al movimiento de las ruedas donde:

- $\theta = angulo\ de\ orientacion$
- $\varphi = angulo\ de\ giro\ ackermann$
- $v = velocidad\ del\ robot$

El ángulo de orientación, al ser estimado, puede acumular un error respecto a la posición final que pueda tener el robot. Para corregir este error de deriva, se utiliza la orientación del ángulo Yaw proporcionada por el Arduino Uno, lo que permite tener una odometría más precisa mediante la fusión de datos.

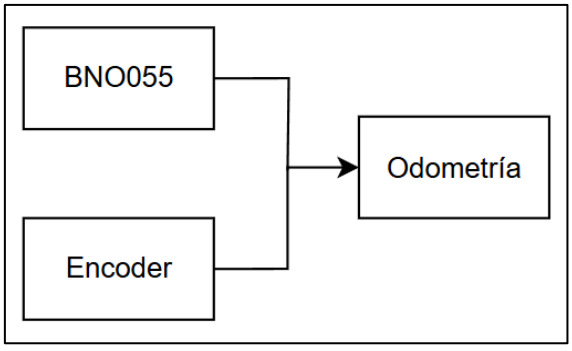



***Figura 86:*** *Fusión de sensores*

Como se observa en las ecuaciones cinemáticas, es necesario multiplicar por el tiempo para obtener la posición en función del tiempo. Para ello, dentro del programa maestro se crea una función similar a millis() como la utilizada en los programas de Arduino. Aunque esta práctica no es la más ortodoxa, permite entender de manera sencilla cómo se ejecutan y actualizan los tiempos dentro del programa.

```
long long milis() {
  auto ahora = std::chrono::system_clock::now();
  auto milisegundos =
std::chrono::time_point_cast<std::chrono::milliseconds>(ahora);
  auto valor = milisegundos.time_since_epoch();
  return valor.count();
}
```

```
long long tiempoActual = milis();
    if(tiempoActual - tiempoAnterior >= 100){
      thetat=(sentidoGiro*tan(AnguloGiro))/L;
      theta=theta+(thetat*tms);
      thetaF=Yaw;
      yt=sentidoGiro*sin(thetaF);
      xt=sentidoGiro*cos(thetaF);
      y=y+(yt*tms);
      x=x+(xt*tms);
      tiempoAnterior=tiempoActual;
}
```

Para enviar los mensajes de odometría, se utiliza la librería de ROS. A diferencia de los mensajes creados anteriormente, este proceso no requiere la creación manual de un publicador porque la librería ya incluye la declaración de uno integrado.

```
nav_msgs::Odometry odom;
odom.header.stamp = ros::Time::now();
odom.header.frame_id = "odom";

odom.pose.pose.position.x = x;
odom.pose.pose.position.y = y;
odom.pose.pose.position.z = 0.0;
odom.pose.pose.orientation.x=0.0;
odom.pose.pose.orientation.y=0.0;
odom.pose.pose.orientation.z=sin(thetaF/2.0);
odom.pose.pose.orientation.w=cos(thetaF/2.0);

odom.child_frame_id = "base_link";
odom.twist.twist.linear.x = sentidoGiro;
odom.twist.twist.linear.y = 0.0;
odom.twist.twist.angular.z = theta;
odom_pub.publish(odom);
```

### 6.4.3 Programación de tópico tf

Como se mencionó en el apartado anterior, dado que se han definido los valores de orientación y posición para la odometría, es necesario ahora declarar las líneas que permiten las transformaciones del robot con respecto a un punto base. Esto implica especificar cómo se transforma la posición y orientación del robot en relación con un marco de referencia fijo, facilitando así la navegación y el seguimiento de la trayectoria del robot.

```
transformStamped.header.stamp = ros::Time::now();
transformStamped.header.frame_id = "odom";
transformStamped.child_frame_id = "base_link";
transformStamped.transform.translation.x = x;
transformStamped.transform.translation.y = y;
transformStamped.transform.translation.z = 0.0;
transformStamped.transform.rotation.x = 0.0;
transformStamped.transform.rotation.y = 0.0;
transformStamped.transform.rotation.z = sin(thetaF/2.0);
transformStamped.transform.rotation.w = cos(thetaF/2.0);
br.sendTransform(transformStamped);
```

Los marcos de referencia establecidos incluyen el marco odom, que se utiliza como punto de origen para la navegación por odometría. El base_link, por su parte, representa la base de las ruedas traseras del robot.

### 6.4.4 Programación de tópico tf_static

El robot está compuesto por diferentes componentes, cada uno con su propio marco de referencia. Como se mencionó anteriormente, el marco base del robot varía en relación con el marco de odometría local.

Durante la navegación por lidar, este tiene un marco que varía respecto a un marco de navegación global. Esto significa que cada componente dentro del robot tiene un marco de referencia definitivo, que debe declararse en relación con otro marco. Por ejemplo, el marco del lidar siempre se encuentra a una distancia específica del marco base del robot, por lo que es necesario declarar este marco fijo utilizando este tópico.

```
static_transform.header.stamp = ros::Time::now();
static_transform.header.frame_id = "base_link";
static_transform.child_frame_id = "lidar_link";
static_transform.transform.translation.x = 0.27;
static_transform.transform.translation.y = 0.0;
static_transform.transform.translation.z = 0.125;
static_transform.transform.rotation.x = 0.0;
static_transform.transform.rotation.y = 0.0;
static_transform.transform.rotation.z = 0.0;
static_transform.transform.rotation.w = 1.0;
static_broadcaster.sendTransform(static_transform);
```

En este caso, se establecen los marcos de referencia padre e hijo y se especifican las distancias relativas entre ellos.

### 6.4.5 Programación de tópico Data_total

Para la publicación de información en este tópico, se utiliza el formato de mensaje std_msgs/Float32MultiArray. Mediante la librería std::vector, se colocan los parámetros del vector que se desea enviar.

```
std_msgs::Float32MultiArray msg;
vec.resize(20);
msg.layout.dim.push_back(std_msgs::MultiArrayDimension());
msg.layout.dim[0].size = vec.size();
msg.layout.dim[0].stride = 1;
msg.layout.dim[0].label = std::string("vectorSundy");
msg.data = vec;

temperaspy = readCPUTemperature();
cpuraspy = readCPUUsage();

vec[0] = x;   //x odom
vec[1] = y;   //y odom
vec[2] = 0.0;   //z odom
vec[3] = Roll;
vec[4] = Pitch;
vec[5] = Yaw;
vec[6] = EstadoBNO055;
vec[7] = sp;
vec[8] = cv;
vec[9] = pv;
vec[10] = RPMmotor;
vec[11] = PWM;
vec[12] = RPMrueda;
vec[13] = sentidoGiro;
vec[14] = PBvoltage;
vec[15] = LIPOvoltage;
vec[16] = temperaspy;
vec[17] = cpuraspy;
vec[18] = 19.0;
pub2.publish(msg);
```

Dentro de los parámetros enviados, que pueden ser manipulados por cualquier nodo, se incluyen la temperatura en grados de la Raspberry Pi y la carga de la CPU expresada en porcentaje. Para obtener estos valores, se desarrollan dos funciones que acceden al registro del sistema y extraen esta información.

## 6.5 Preparación del sensor LD19

Los sensores como los lidar suelen incluir paquetes desarrollados por los fabricantes, que contienen todo lo necesario para ser lanzados directamente con una instalación simple.

En el caso del LD19, el fabricante proporciona instrucciones para la instalación del paquete mediante la clonación de un repositorio de GitHub, tal como se incluye en su manual de instrucciones.

```
$ git clone https://github.com/ldrobotSensorTeam/ldlidar_stl_ros.git
```

Sin embargo, es necesario realizar ajustes a estos paquetes, ya que pueden incluir elementos que no son necesarios. Un ajuste que siempre se debe realizar es la corrección de los marcos de referencia. Es importante recordar que el robot tiene sus propias referencias, las cuales han sido determinadas en el nodo maestro.

Para realizar los ajustes de los marcos de referencia se entra al programa principal de publicación y se verifica que el nombre del tópico sea lidar_link.

```
nh_private.param("frame_id", setting.frame_id, std::string("lidar_link"));
```

Para los datos del LiDAR, también existe un tipo de mensaje específico, y la mayoría de los fabricantes de LiDAR suelen nombrar este tópico scan. Por lo tanto, para este caso se mantendrá este nombre a fin de garantizar la compatibilidad con futuros paquetes de ROS.

Otro ajuste importante es configurar el nombre del nodo y eliminar los marcos de referencia establecidos por el fabricante para su simulación. Para realizar estas modificaciones, es esencial revisar el archivo lanzador del proyecto y eliminar las líneas correspondientes.

```
<!-- ldlidar message subscriber node -->
 <!-- node name="ListenLD19" pkg="ldlidar_stl_ros"
type="ldlidar_stl_ros_listen_node" output="screen">
  <param name="topic_name" value="scan"/>
 </node -->
 <!-- publisher tf transform, parents frame is base_link, child frame is
base_laser -->
 <!-- args="x y z yaw pitch roll parents_frame_id child_frame_id period_in_ms"-->
 <node name="base_to_laser" pkg="tf" type="static_transform_publisher"  args="0.0
0.0 0.18 0 0.0 0.0 base_link base_laser 50" if="$(arg fix_to_base_link)"/>
</launch>
```

Para cambiar el nombre del nodo se reemplaza en la siguiente línea del lanzador

```
<node name= "Sundy_LD19" pkg="ldlidar_stl_ros" type="ldlidar_stl_ros_node"
output="screen" >
```

La estructura quedaría de la siguiente manera tal como se ve en la figura 87.

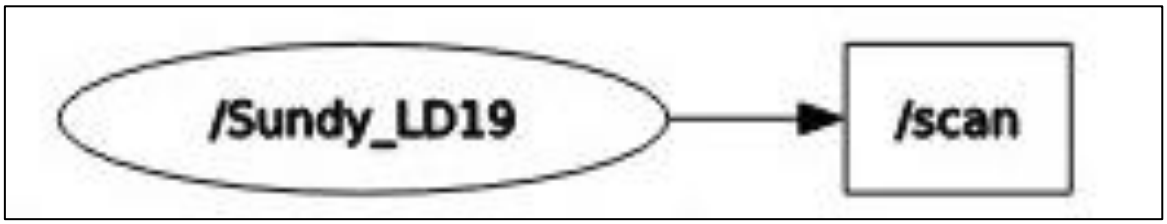



***Figura 87:*** *Fusión de sensores*

## 6.6 Preparación del nodo teleoperación

Este nodo se crea para interpretar la información del mando F710. Utiliza los datos provenientes del tópico joy, que publica información sobre cada joystick y botón del F710. Esta información se transforma en comandos de velocidad que se envían al robot. El tópico joy no es un tópico nativo que se incluya con la instalación estándar de ROS. En cambio, es parte de un paquete que se puede obtener clonando el siguiente repositorio:

```
$ git clone https://github.com/ros-drivers/joystick_drivers.git
```

La estructura que se sigue es la siguiente:

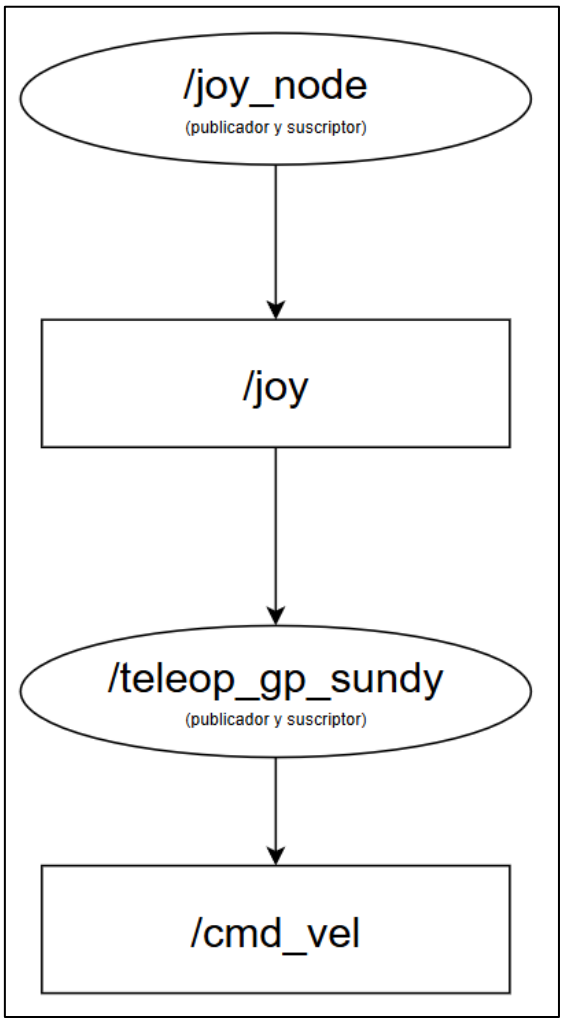



***Figura 88:*** *Fusión de sensores*

El programa de teleoperación se suscribe al tópico joy, que envía sus mensajes en dos vectores: uno para los botones y otro para los joysticks. La información del joystick derecho se encuentra en el campo 3 del vector de joysticks, mientras que la información del joystick izquierdo está en el campo 0 del mismo vector.

```
void joyCallback(const sensor_msgs::Joy::ConstPtr& JS) {
  giro = JS->axes[0];
  avance = JS->axes[3];
}
```

Al igual que en otros programas, se crea una función de mapeo para valores flotantes. Esto permite mapear adecuadamente los datos de entrada almacenados en las variables de giro y avance. Dicha función es usada en el bucle principal.

```
geometry_msgs::Twist msg;

if (avance == -0.0) {
  msg.linear.x = 0.0;
  msg.angular.z = map(giro, -1.0, 1.0, -0.78, 0.78);
} else {
  msg.angular.z = map(giro, -1.0, 1.0, -0.78, 0.78);
  if (avance >= 0) {
    msg.linear.x = map(avance, 0.0, 1.0, 0.0, inputMax);
  }
  if (avance < 0) {
    msg.linear.x = map(avance, -1.0, 0.0, inputMin, 0.0);
  }
}
cmd_vel_pub.publish(msg);  //publicamos mensaje
```

Como se observa en las líneas de código extraídas, la velocidad máxima y mínima se regulan mediante una variable previamente declarada.

```
float inputMin = -3.0;
float inputMax = 5.0;
```

En este caso, la velocidad del vehículo está limitada a un máximo de 5 m/s y un mínimo de -3 m/s.

## 6.7 Lanzadores para Raspberry Pi 4

Como se mencionó en el marco teórico de este proyecto, cada nodo en ROS funciona como un programa independiente, lo que implica que para ejecutar todos los nodos actualmente declarados se necesitaran aproximadamente seis terminales. Sin embargo, ROS facilita la ejecución de múltiples nodos de manera simultánea utilizando archivos .launch, que están escritos en formato XML. Estos archivos permiten iniciar varios nodos con parámetros específicos de manera organizada.

Los lanzadores creados y disponibles para esta arquitectura son los siguiente:

```
roslaunch sundybot/launch/sundybot_fake.launch
roslaunch sundybot/launch/sundybot_gp_teleoperation.launch
roslaunch sundybot/launch/sundybot_with_lidar.launch
```

### 6.7.1 Lanzador sundybot_fake

Este lanzador inicia el nodo maestro, así como la comunicación en ROS serial con los Arduino Uno y Nano.

```
<launch>
  <!-- Lanzar el nodo serial_node.py de rosserial_python -->
  <node name="ArduinoNano_node" pkg="rosserial_python" type="serial_node.py"
output="screen">
    <param name="port" value="/dev/ttyUSB0" />
  </node>

  <!-- Lanzar el nodo serial_node.py de rosserial_python -->
  <node name="ArduinoUno_node" pkg="rosserial_python" type="serial_node.py"
output="screen">
    <param name="port" value="/dev/ttyACM0" />
  </node>
  <!-- Lanzar el nodo serial_node.py de rosserial_python -->
  <node name="Sundy_collector" pkg="sundy_operation" type="sundydata_sundy"
output="screen" />
</launch>
```

Los nodos Arduino Uno y Arduino Nano utilizan el mismo paquete y el mismo nodo de ejecución. Esta es una forma eficaz de comunicar múltiples Arduinos utilizando la comunicación ROS serial.

### 6.7.2 Lanzador sundybot_gp_teleoperation

Este lanzador inicia el nodo de teleoperación creado para el control de velocidad y dirección, así como el nodo joy_node para la comunicación con el mando F710.

```
<launch>
  <!-- Lanzar el nodo joy_node con el dispositivo específico -->
  <node name="joy_node" pkg="joy" type="joy_node" output="screen">
    <param name="dev_name" value="Logitech Logitech Cordless RumblePad 2" />
  </node>

  <!-- Lanzar el nodo teleop_gp_sundy -->
  <node name="teleop_gp_sundy" pkg="sundy_operation" type="teleop_gp_sundy"
output="screen" />
</launch>
```

Para la ejecución del joy_node, se configuran los parámetros específicos del mando F710.

### 6.7.3 Lanzador sundybot_with_lidar

Este lanzador inicia los mismos nodos que sundybot_fake, pero además incluye el paquete del LiDAR. En el lanzador se incluye el código configurado en el apartado dedicado a la preparación del sensor LD19.

```
<launch>
<arg name="topic_name" default="scan"/>
<arg name="port_name" default="/dev/ttyUSB1"/>
<arg name="port_baudrate" default="230400"/>
  <!-- Lanzar el nodo serial_node.py de rosserial_python -->
  <node name="ArduinoNano_node" pkg="rosserial_python" type="serial_node.py"
output="screen">
    <param name="port" value="/dev/ttyUSB0" />
  </node>

  <!-- Lanzar el nodo serial_node.py de rosserial_python -->
  <node name="ArduinoUno_node" pkg="rosserial_python" type="serial_node.py"
output="screen">
    <param name="port" value="/dev/ttyACM0" />
  </node>
  <!-- Lanzar el nodo serial_node.py de rosserial_python -->
  <node name="Sundy_collector" pkg="sundy_operation" type="sundydata_sundy"
output="screen" />

<!-- ldldiar message publisher node -->
 <node name="Sundy_LD19" pkg="ldlidar_stl_ros" type="ldlidar_stl_ros_node"
output="screen" >
  <param name="product_name" value="LDLiDAR_LD19"/>
  <param name="topic_name" value="$(arg topic_name)"/>
  <param name="port_name" value ="$(arg port_name)"/>
  <param name="port_baudrate" value ="$(arg port_baudrate)"/>
  <!-- Set laser scan directon: -->
  <!--    1. Set counterclockwise, example: <param name="laser_scan_dir" type="bool"
value="true"/> -->
  <!--    2. Set clockwise,        example: <param name="laser_scan_dir" type="bool"
value="false"/> -->
  <param name="laser_scan_dir" type="bool" value="true"/>
  <!-- Angle crop setting, Mask data within the set angle range -->
  <!--    1. Enable angle crop fuction: -->
  <!--       1.1. enable angle crop,  example: <param name="enable_angle_crop_func"
type="bool" value="true"/> -->
  <!--       1.2. disable angle crop, example: <param name="enable_angle_crop_func"
type="bool" value="false"/> -->
  <param name="enable_angle_crop_func" type="bool" value="false"/>
  <!--    2. Angle cropping interval setting, The distance and intensity data within
the set angle range will be set to 0 -->
  <!--       angle >= "angle_crop_min" and angle <= "angle_crop_max", unit is degress
-->
  <param name="angle_crop_min" type="double" value="135.0"/>
  <param name="angle_crop_max" type="double" value="225.0"/>
 </node>
</launch>
```

## 6.8 Modelado URDF

Los modelos URDF, o "Unified Robot Description Format", son un formato unificado de descripción de robots en ROS, escritos en formato de archivo XML. Este formato permite describir y representar modelos reales de robots, modelando la estructura de los enlaces (links) y articulaciones (joints). Esto facilita la comprensión visual del comportamiento de los parámetros del robot, crucial para la simulación, visualización y planificación del movimiento en entornos de desarrollo de robots.

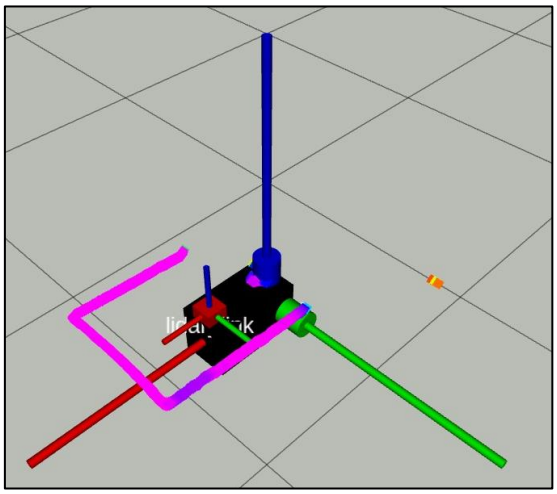


***Figura 89:*** *Modelo URDF*

El modelo URDF del robot presentado en la figura 84, describe los límites máximos del robot sin tomar en cuenta las ruedas, de igual manera se coloca un cubo rojo el cual representa la posición del LIDAR.

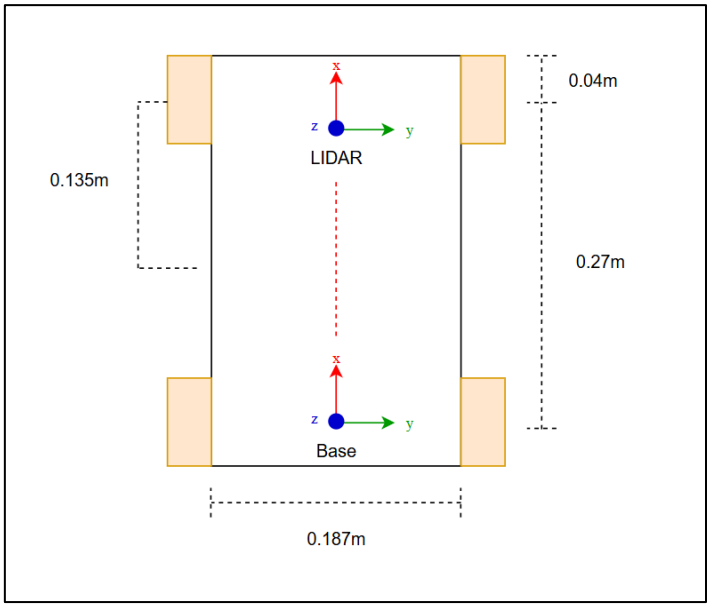



***Figura 90:*** *Esquema para modelo URDF*

Además del modelo URDF del robot, se describe un modelo URDF vacío llamado odom, ubicado en el punto 0,0,0. Este modelo representa la posición inicial desde la cual se empieza a contar la distancia de odometría como se puede ver en la figura 89.

## 6.9 Configuración de entorno Rviz

En Rviz, se configuran los parámetros que se desean visualizar, incluyendo los siguientes:

- **Global Options:** Aquí se definen parámetros generales como el color del fondo del entorno de simulación del robot.
- **Robot Model:** Carga el modelo URDF del robot para visualización.
- **Grid:** Muestra una cuadrícula que ayuda a visualizar y orientar el espacio en el que se encuentra el robot.
- **Axes:** Exhibe ejes para indicar la orientación espacial de los componentes del robot.
- **Odometry:** Muestra la información de odometría del robot, que puede incluir la trayectoria y la posición actual.
- **LaserScan:** Visualiza los datos provenientes del escáner láser, útiles para detectar objetos y estructuras alrededor del robot.
- **TF:** Muestra las transformaciones entre diferentes marcos de referencia utilizados en la simulación y operación del robot.

En el apartado de Lanzadores para Raspberry Pi 4, se mencionó que existen dos tipos de lanzadores específicos: uno que inicia el paquete del LD19 (sundybot_with_lidar) y otro que no inicia este nodo (sundybot_fake). Debido a esta distinción, existen dos entornos de visualización en Rviz, denominados sundyRviz.rviz y sundyRviz_with_lidar.rviz, respectivamente.

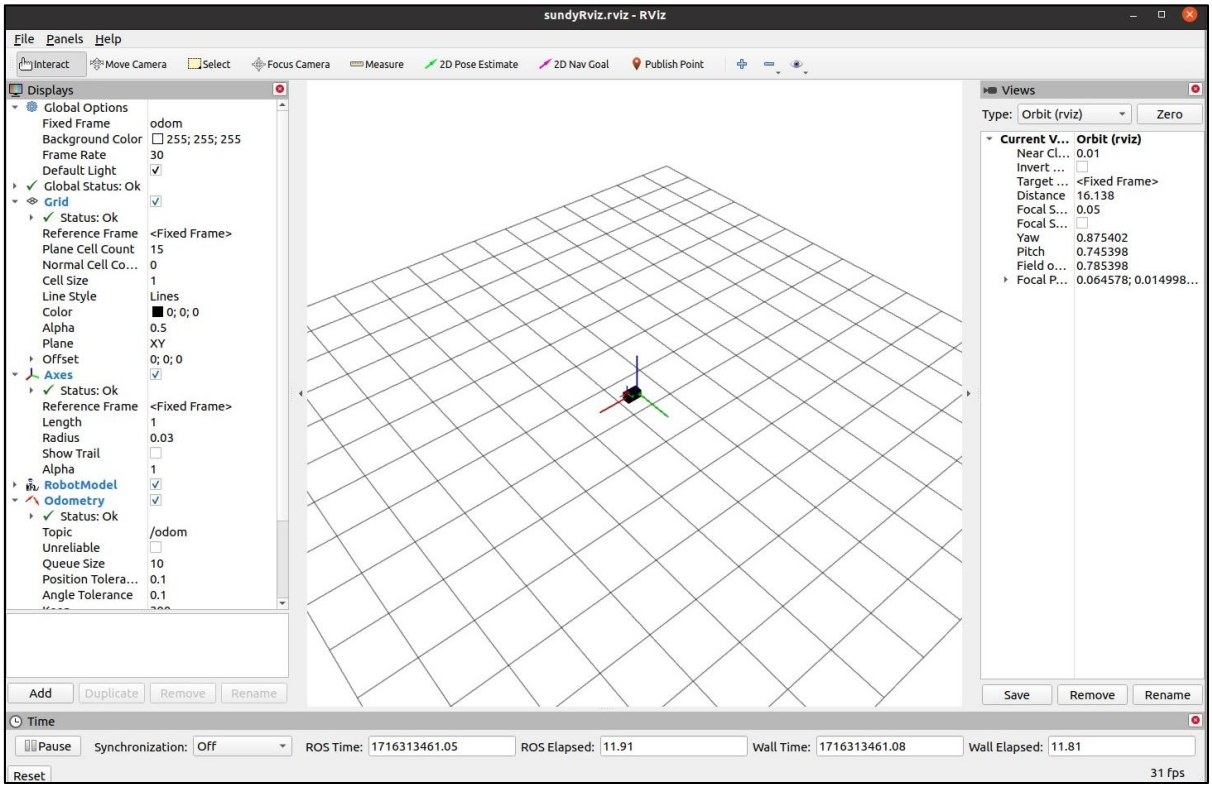



***Figura 91:*** *Entorno Rviz sin información de LIDAR*

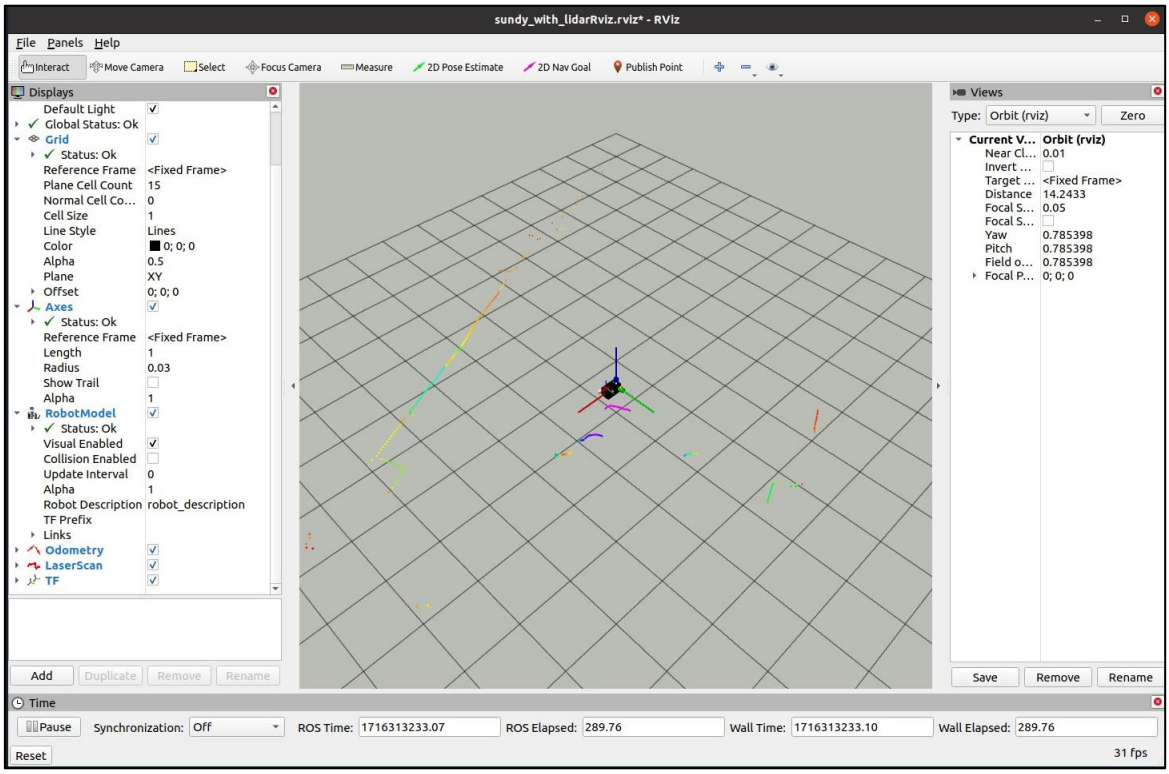



***Figura 92:*** *Entorno Rviz con información de LIDAR*

En las figuras 91 y 92, se muestran las diferentes configuraciones visuales según los nodos y tópicos activados.

Los archivos de formato .rviz contienen líneas de argumentos estructurados que luego son visualizados en una interfaz gráfica. Es importante mencionar que Rviz no funciona como un programa de escritorio convencional; más bien, opera como un nodo más dentro del entorno ROS, que carga y utiliza los argumentos y parámetros almacenados en el archivo .rviz.

## 6.10 Lanzadores para PC remota

Al igual que se simplifica la activación de múltiples nodos usando lanzadores, estos archivos también pueden iniciar modelos URDF así como nodos de Rviz. Estos últimos pueden configurarse previamente y la configuración puede ser cargada como un argumento dentro del lanzador.

Los lanzadores programados para ser lanzados desde el ordenador remoto son los siguientes:

```
roslaunch sundy_model_pkg sundy_robot.launch
roslaunch sundy_model_pkg sundy_robot_with_lidar.launch
```

### 6.10.1 Lanzador sundy_robot

Este lanzador inicia el modelo URDF del robot y el modelo center_point. Además, levanta el nodo Rviz con una configuración de entorno que no incluye la información del LIDAR.

```
<launch>
       <arg name="model" />
       <param name= "robot_description" textfile="$(find
sundy_model_pkg)/urdf/sundy_robot_lidar.urdf" />
       <param name= "pointcenter_description" textfile="$(find
sundy_model_pkg)/urdf/centerpoint.urdf" />

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find
sundy_model_pkg)/sundyRviz.rviz" required="true" />
</launch>
```

### 6.10.2 Lanzador sundy_robot_with_lidar

Al igual que el lanzador anterior, este inicia los modelos y carga el entorno de Rviz, que esta vez incluye la información proporcionada por el LIDAR.

```
<launch>
       <arg name="model" />
       <param name= "robot_description" textfile="$(find
sundy_model_pkg)/urdf/sundy_robot_lidar.urdf" />
       <param name= "pointcenter_description" textfile="$(find
sundy_model_pkg)/urdf/centerpoint.urdf" />

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find
sundy_model_pkg)/sundyRviz.rviz" required="true" />
</launch>
```

## 6.11 Paquete HectorSLAM

El paquete HectorSLAM, desarrollado y mejorado por la comunidad de ROS, está diseñado para abordar las limitaciones asociadas con el mapeo de áreas extensas. Este paquete tiene su propia arquitectura y publica tópicos necesarios para la creación de mapas.

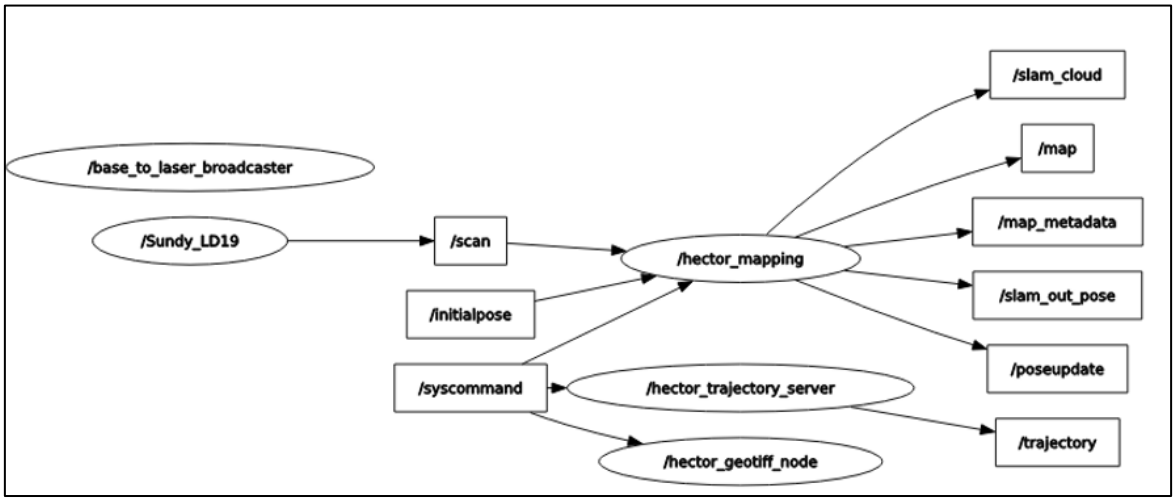



***Figura 93:*** *Estructura paquete HectorSlam*

Además, incluye una configuración para Rviz que se puede cargar directamente desde el lanzador incluido en el paquete.

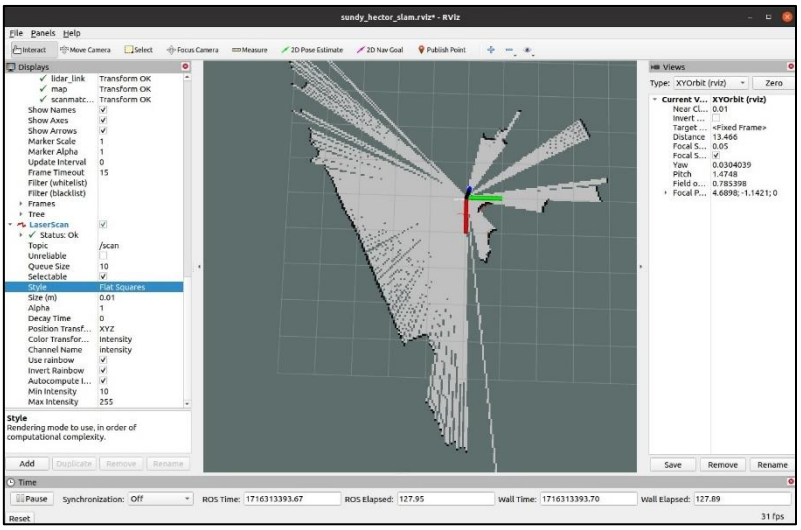



***Figura 94:*** *Entorno Rviz de HectorSlam*

Dado que es un paquete externo, similar al paquete del LD19, es necesario adaptar ciertos parámetros como los nombres de los tópicos y los marcos de referencia. Para realizar estas adaptaciones, se modifican las siguientes líneas en el archivo de lanzamiento del paquete de mapeo dentro de HectorSLAM.

```
<arg name="base_frame" default="lidar_link"/>
<arg name="odom_frame" default="lidar_link"/>
<node name="base_to_laser_broadcaster" pkg="tf"
type="static_transform_publisher"  args="0.0 0.0 0.18 0 0.0 0.0
base_link lidar_link 100"/>
```

El lanzador comúnmente utilizado dentro de los múltiples lanzadores de HectorSLAM es tutorial.launch. Para mantener la consistencia en la nomenclatura de los archivos, este lanzador se clona y se renombra para seguir el convenio de nombres de los archivos anteriores.

```
roslaunch sundy_model_pkg sundy_robot_hectorslam.launch
```

```
<launch>
  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>
  <param name="/use_sim_time" value="false"/>
  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find sundy_model_pkg)/sundy_hector_slam.rviz"/>
  <include file="$(find
hector_mapping)/launch/mapping_default_prueba.launch"/>
  <include file="$(find
hector_geotiff_launch)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  </include>
</launch>
```

## 6.12 Diseño de interfaces graficas

El nodo maestro sundy collector, publica en el topico data_total los siguientes datos: x, y, z, Roll, Pitch, Yaw, EstadoBNO055, sp, cv, pv, RPMmotor, PWM, RPMrueda, sentidoGiro, PBvoltaje, LIPOvoltage, temperaspy, cpuraspy, temperatura externa.

Las interfaces gráficas se suscriben a este tópico para realizar tareas como almacenar y guardar la información, o graficarla e interactuar con ella en tiempo real.

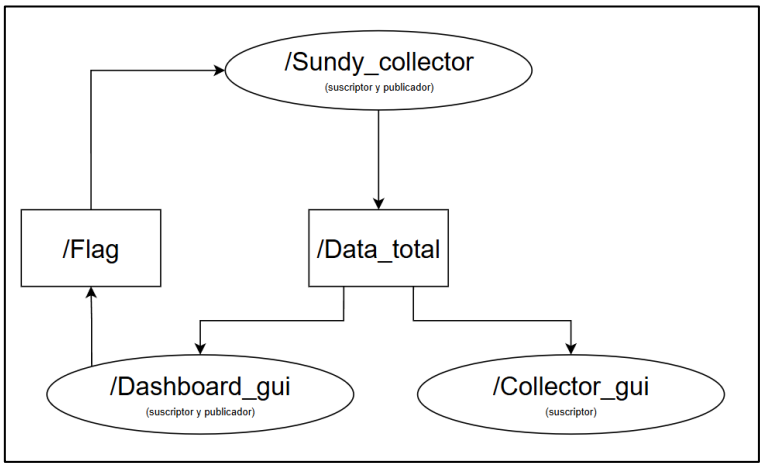



***Figura 95:*** *Comunicación GUI*

### 6.12.1 Interfaz gráfica Dashboard (panel de control)

Esta interfaz permite al usuario visualizar rápidamente los valores enviados por el tópico Data_total. Además de graficar y mostrar los valores en tiempo real, ofrece la posibilidad de reiniciar y ajustar los valores de odometría y la IMU.

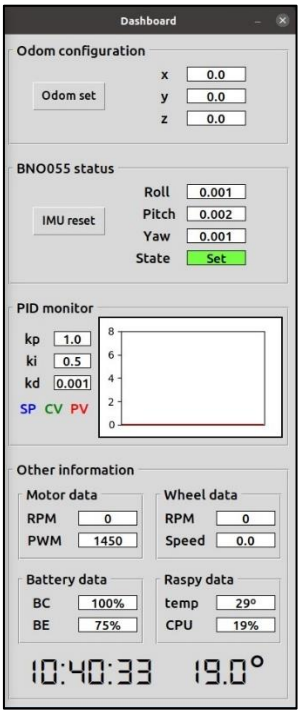



***Figura 96:*** *Interfaz gráfica Dashboard*

Esta interfaz gráfica ha sido programada en python utilizando las bibliotecas Tkinter y Matplotlib. Estas bibliotecas generan un hilo para cada proceso, y junto con el hilo que levanta ROS, es necesario garantizar el cierre completo de todos los hilos al terminar.

Normalmente, un proceso o nodo de ROS abierto en un terminal se cierra utilizando Control+C. Sin embargo, para asegurar el cierre de todos los hilos, se recomienda a los usuarios cerrar la aplicación utilizando la "X" en la interfaz gráfica. Al hacerlo, se ejecuta una acción que cierra todos los hilos adecuadamente.

```
def on_close():
    global root
    # Detener ROS
    rospy.signal_shutdown('Window Closed')

    # Cerrar la ventana de Tkinter
    root.destroy()
    plt.close('all')  # Asegurarse de cerrar todas las figuras de Matplotlib
    plt.close(fig)
    print("NOTE: Interface and ROS nodes closed correctly :)")
```

```
julio1804@julioubu:~$ rosrun sundy_gui_pkg dashboard.py
NOTE: To close this NODE and free the terminal close the window directly!!!
NOTE: Interface and ROS nodes closed correctly :)
julio1804@julioubu:~$
```

***Figura 97:*** *Cierre de interfaz gráfica Dashboard*

### 6.12.2 Interfaz gráfica Dashboard (panel de control)

Esta interfaz permite guardar los valores publicados en el tópico Data_total en un archivo CSV para su posterior manipulación.

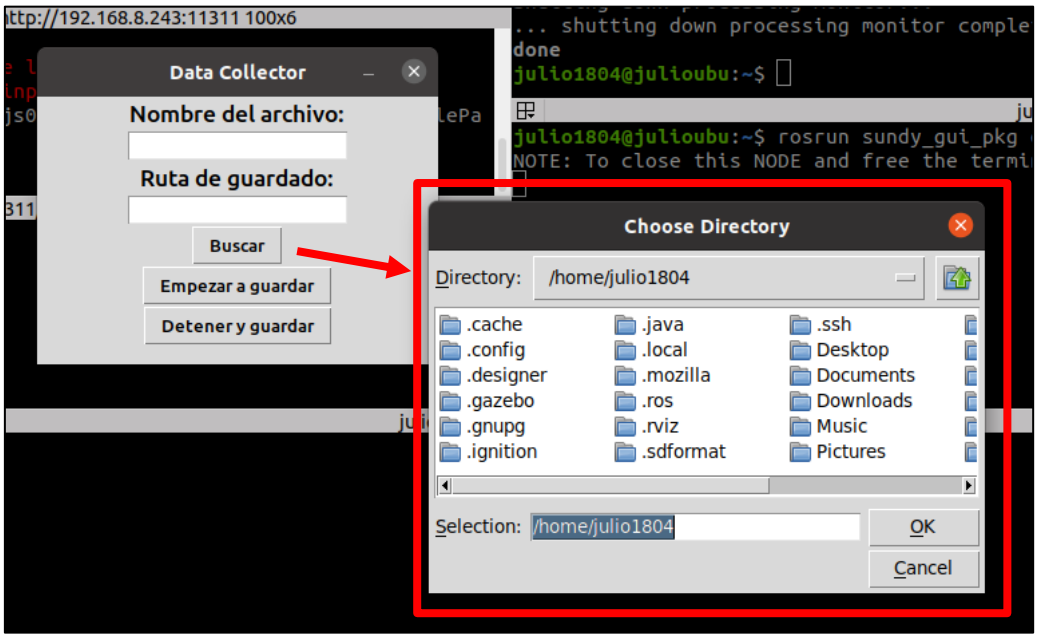



***Figura 98:*** *Interfaz gráfica Collector*

Como se muestra en la Figura 98, la interfaz permite al usuario nombrar el archivo y seleccionar la ruta donde desea guardarlo. Los tres botones disponibles facilitan iniciar la guardado de información, detener la escritura del archivo y cerrar el archivo guardado.

Durante el proceso de guardado, se muestra un mensaje en el terminal indicando que los datos se están guardando, así como un mensaje final que confirma que los datos han sido guardados correctamente.

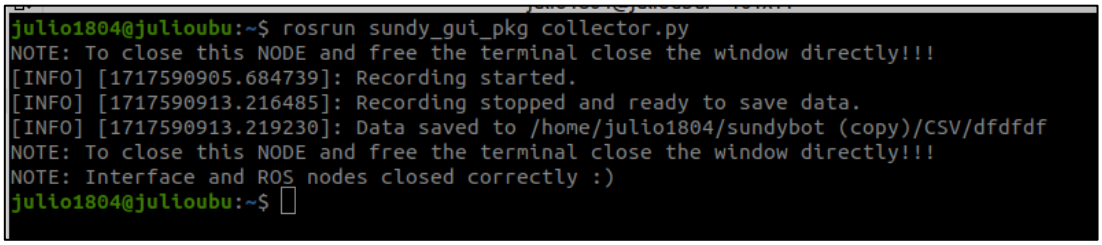



***Figura 99:*** *Mensajes Interfaz gráfica Collector*

En la figura 99, se puede apreciar los diferentes mensajes mostrados en el terminal según los procesos realizados por la interfaz.

Los datos son guardados en el siguiente orden:

- x, y, z (odometría)
- Roll, Pitch, Yaw, EstadoBNO055 (BNO055)
- sp, cv, pv (control PID)
- RPMmotor, PWM, RPMrueda, sentidoGiro (Información de rudas y motor)
- PBvoltaje, LIPOvoltage (voltaje de baterias)
- temperaspy, cpuraspy (información Raspberry Pi)

## 6.13 Arquitectura final

La arquitectura final depende de los nodos y tópicos activos. Para este apartado en específico, se creará un enunciado que pone en contexto los nodos lanzados y cómo varía la arquitectura de mensajes final para dichas situaciones.

**Primera situación:**
Para esta situación inicial, supongamos que la persona desea simplemente monitorear el recorrido por odometría que realiza el robot en un área amplia y que su robot no está equipado con un LIDAR. Los nodos lanzados para realizar esta tarea son los siguientes:

```
En la Raspberry Pi
roslaunch sundybot sundy_operation sundybot_fake.launch
roslaunch sundybot sundy_operation sundybot_gp_teleoperation.launch

En la PC remota
roslaunch sundy_model_pkg sundy_robot.launch
rosrun sundy_gui_pkg dashboard.py
```

Por lo tanto, la arquitectura de mensajería en ROS es la mostrada en el anexo 2

**Segunda situación:**
Para esta segunda situación, supongamos que la persona desea monitorear el recorrido por odometría que realiza el robot en un área amplia y que su robot está equipado con un LIDAR. Los nodos lanzados para realizar esta tarea son los siguientes:

```
En la Raspberry Pi
roslaunch sundybot sundy_operation sundybot_with_lidar.launch
roslaunch sundybot sundy_operation sundybot_gp_teleoperation.launch

En la PC remota
roslaunch sundy_model_pkg sundy_robot_with_lidar.launch
rosrun sundy_gui_pkg dashboard.py
```

Por lo tanto, la arquitectura de mensajería en ROS es la mostrada en el anexo 3

**Tercera situación:**
Para esta tercera situación, supongamos que la persona desea monitorear el recorrido por odometría que realiza el robot y, además, desea guardar los valores de dicho recorrido en un área amplia, pero su robot no está equipado con un LIDAR. Los nodos lanzados para realizar esta tarea son los siguientes:

```
En la Raspberry Pi
roslaunch sundybot sundy_operation sundybot_fake.launch
roslaunch sundybot sundy_operation sundybot_gp_teleoperation.launch

En la PC remota
roslaunch sundy_model_pkg sundy_robot.launch
rosrun sundy_gui_pkg dashboard.py
rosrun sundy_gui_pkg collector.py
```

Por lo tanto, la arquitectura de mensajería en ROS es la mostrada en el anexo 4

**Cuarta situación:**
Para esta cuarta situación, supongamos que la persona desea monitorear el recorrido por odometría que realiza el robot y, además, desea guardar los valores de dicho recorrido en un área amplia, pero su robot está equipado con un LIDAR. Los nodos lanzados para realizar esta tarea son los siguientes:

```
En la Raspberry Pi
roslaunch sundybot sundy_operation sundybot_with_lidar.launch
roslaunch sundybot sundy_operation sundybot_gp_teleoperation.launch

En la PC remota
roslaunch sundy_model_pkg sundy_robot_with_lidar.launch
rosrun sundy_gui_pkg dashboard.py
rosrun sundy_gui_pkg collector.py
```

Por lo tanto, la arquitectura de mensajería en ROS es la mostrada en el anexo 5

**Quinta situación:**
Para esta quinta situación, supongamos que la persona desea escanear un mapa, lo cual implica que su robot está equipado con un sensor LIDAR. Los nodos lanzados para realizar esta tarea son los siguientes:

```
En la Raspberry Pi
roslaunch sundybot sundy_operation sundybot_with_lidar.launch
roslaunch sundybot sundy_operation sundybot_gp_teleoperation.launch

En la PC remota
roslaunch sundy_model_pkg sundy_robot_with_lidar.launch
roslaunch sundy_model_pkg sundy_robot_hectorslam.launch
rosrun sundy_gui_pkg dashboard.py
```

Por lo tanto, la arquitectura de mensajería en ROS es la mostrada en el anexo 6

# Capítulo 7. RESULTADOS OBTENIDOS

## 7.1 Rutas realizadas

Para probar el funcionamiento de la arquitectura diseñada, se realizan tres rutas: dos de ellas involucraron el escaneo de mapas y la última se centra en observar la trayectoria de navegación por odometría.

Es importante tener en cuenta que esta propuesta no implica realizar una ruta autónoma, sino teleoperar el recorrido con el objetivo de completar la ruta y analizar sus resultados.

Utilizando el programa de recolección de datos, se guardan los datos para posteriormente mostrarlos en una gráfica. Las rutas se grabaron utilizando una GoPro Hero +3 con el objetivo de corroborar los datos obtenidos. Para observar el comportamiento en tiempo real, se proporciona un enlace de YouTube para cada ruta realizada.

Los resultados obtenidos que se desean comparar incluyen la odometría, el control PID, así como la velocidad real frente a la velocidad deseada.

### 7.1.1 Ruta 1

Para esta ruta se hace un ida y vuelta, donde el robot realiza un retroceso para llegar en dirección contraria a su punto de inicio. La ruta propuesta es la mostrada en la figura 100.

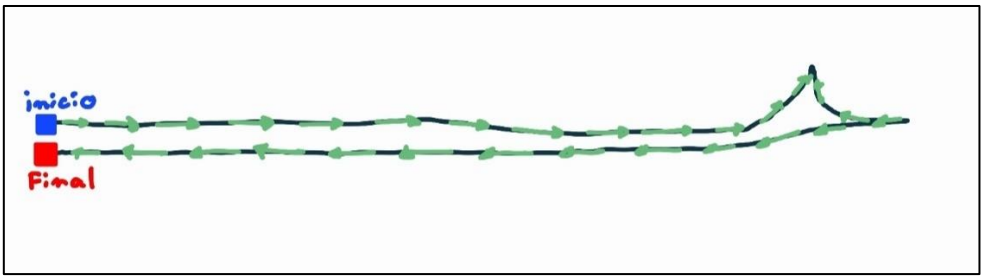



***Figura 100:*** *Propuesta Ruta 1*

La trayectoria realizada se puede ver mediante el enlace: https://youtu.be/DetUCj2KYac

Los resultados obtenidos se muestran en las figuras 101, 102, 103.

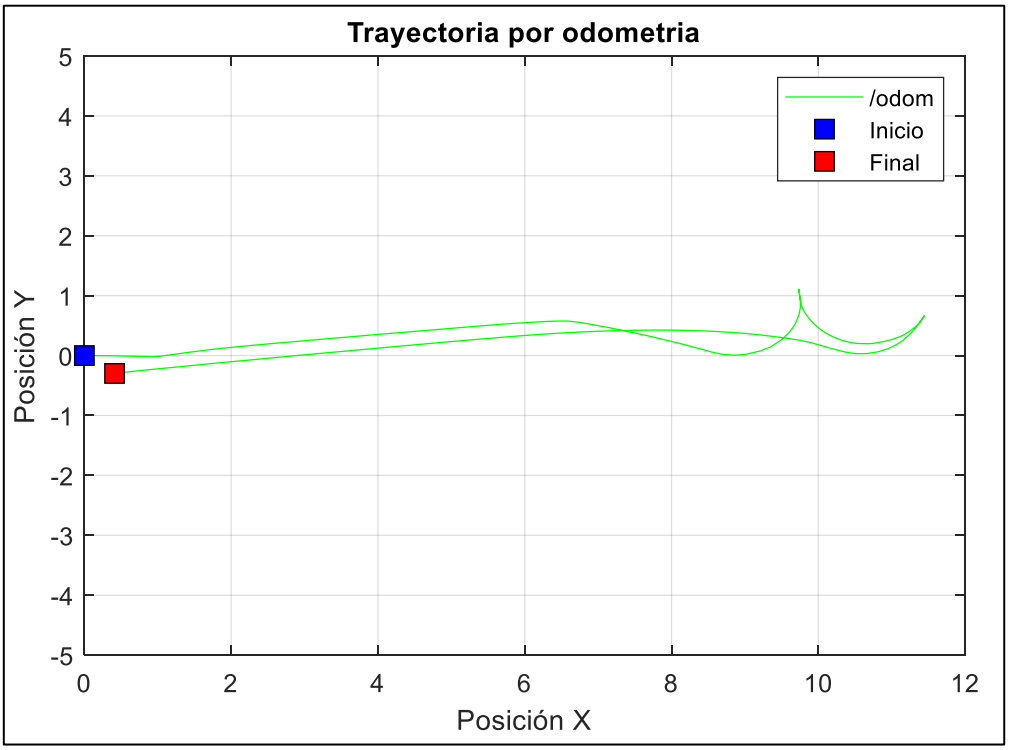



***Figura 101:*** *Resultado odometría Ruta 1*

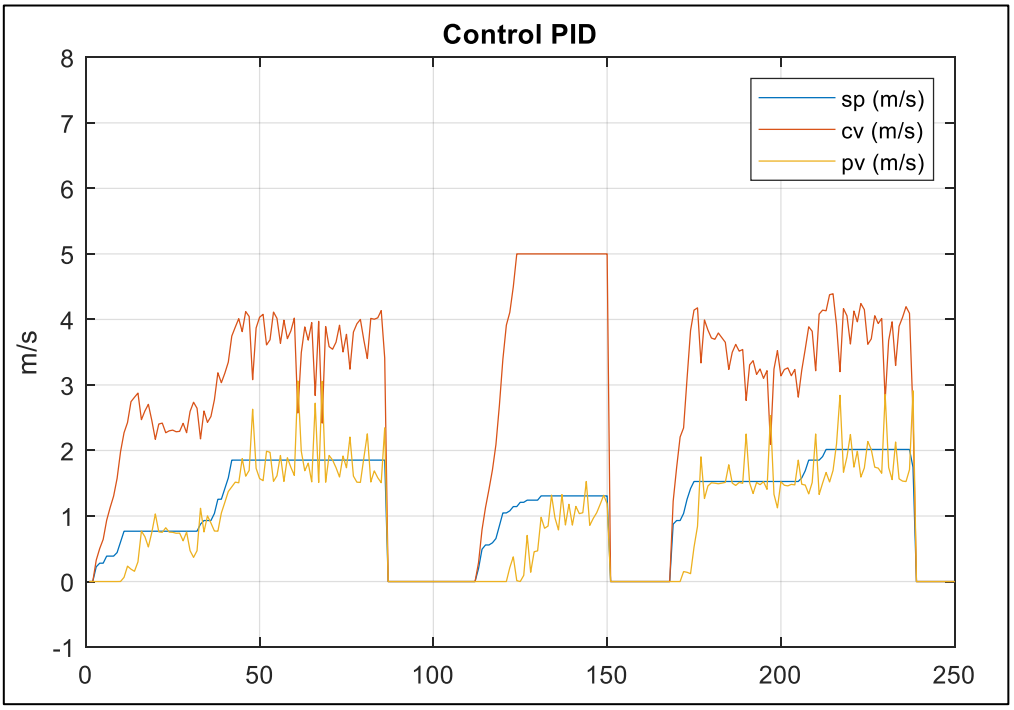



***Figura 102:*** *Resultado control PID Ruta 1*

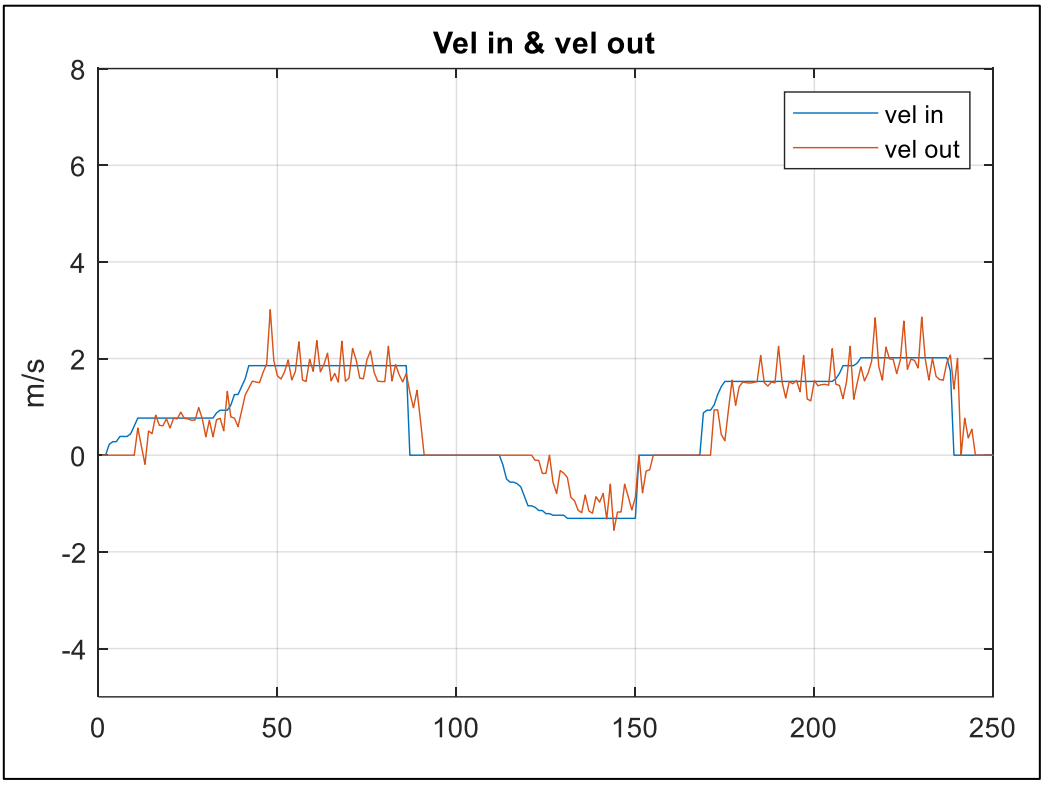



***Figura 103:*** *Resultado velocidad Ruta 1*

### 7.1.2 Ruta 2

Para esta ruta, se realiza una trayectoria circular en la que el robot no retrocede, pero llega en dirección contraria a su punto de inicio. La ruta propuesta se muestra en la Figura 104.

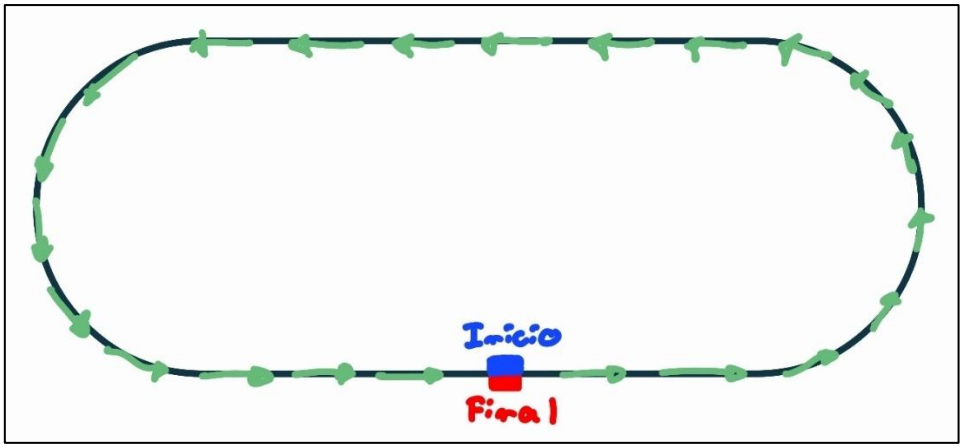



***Figura 104:*** *Propuesta Ruta 2*

La trayectoria realizada se puede ver mediante el enlace:
https://youtu.be/9vmA39pETA4

Los resultados obtenidos se muestran en las figuras 105, 106, 107.

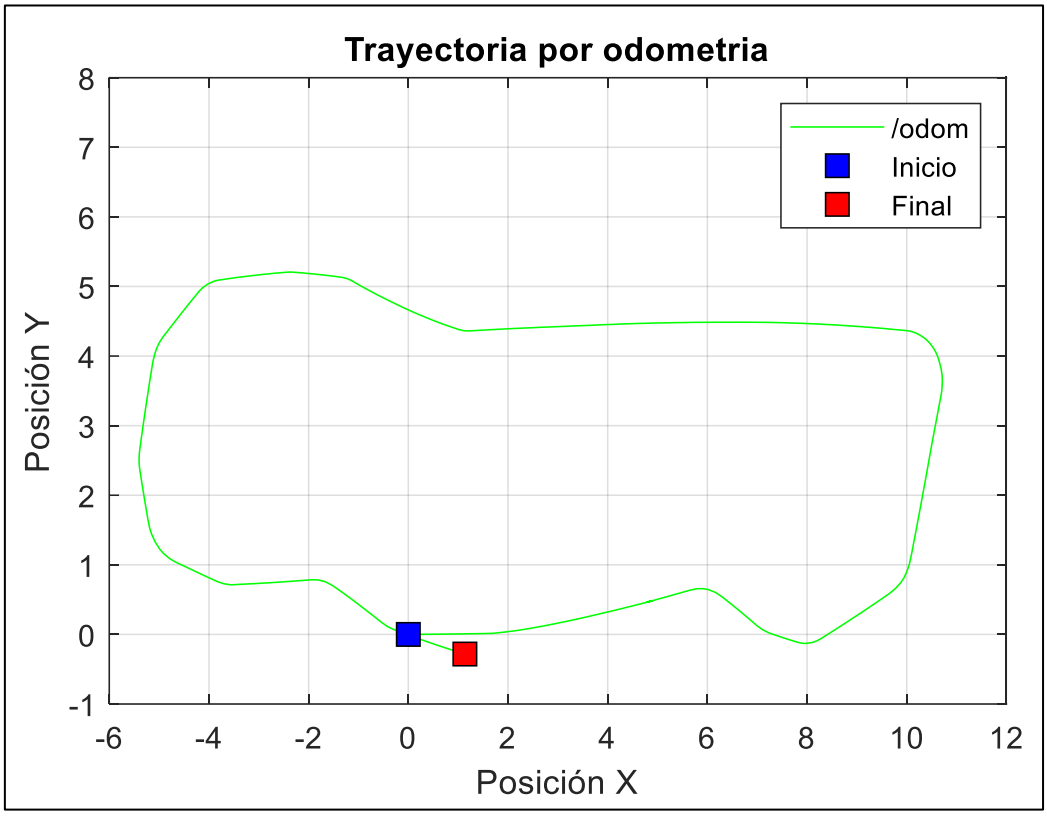



***Figura 105:*** *Resultado odometría Ruta 2*

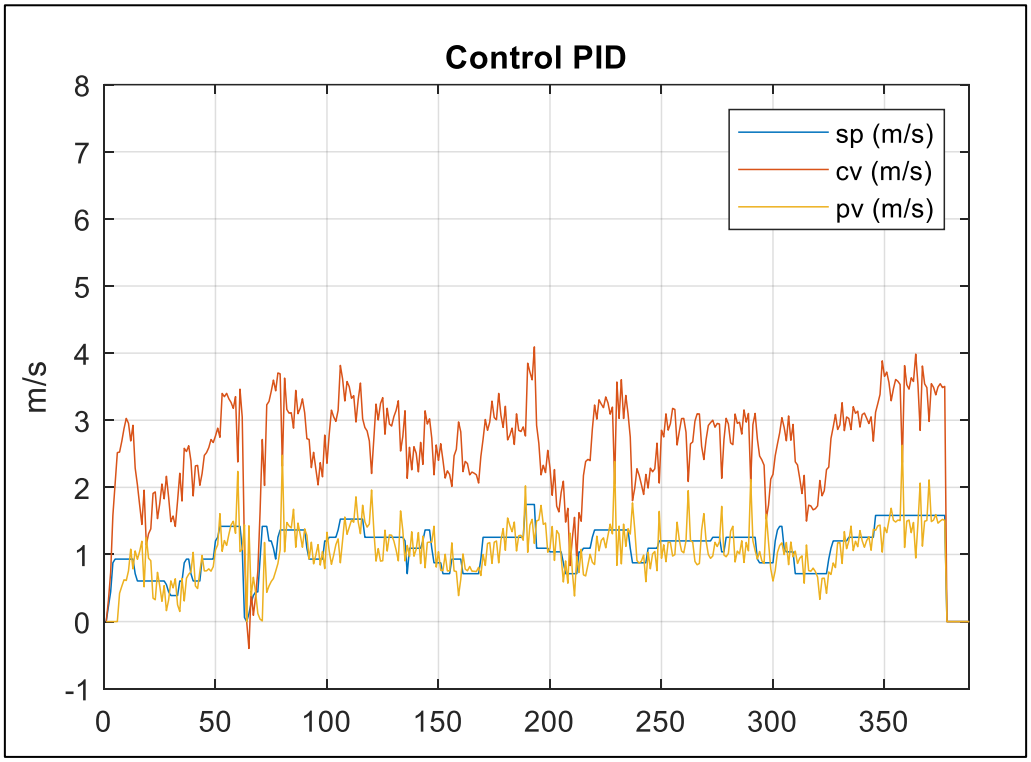



***Figura 106:*** *Resultado control PID Ruta 2*

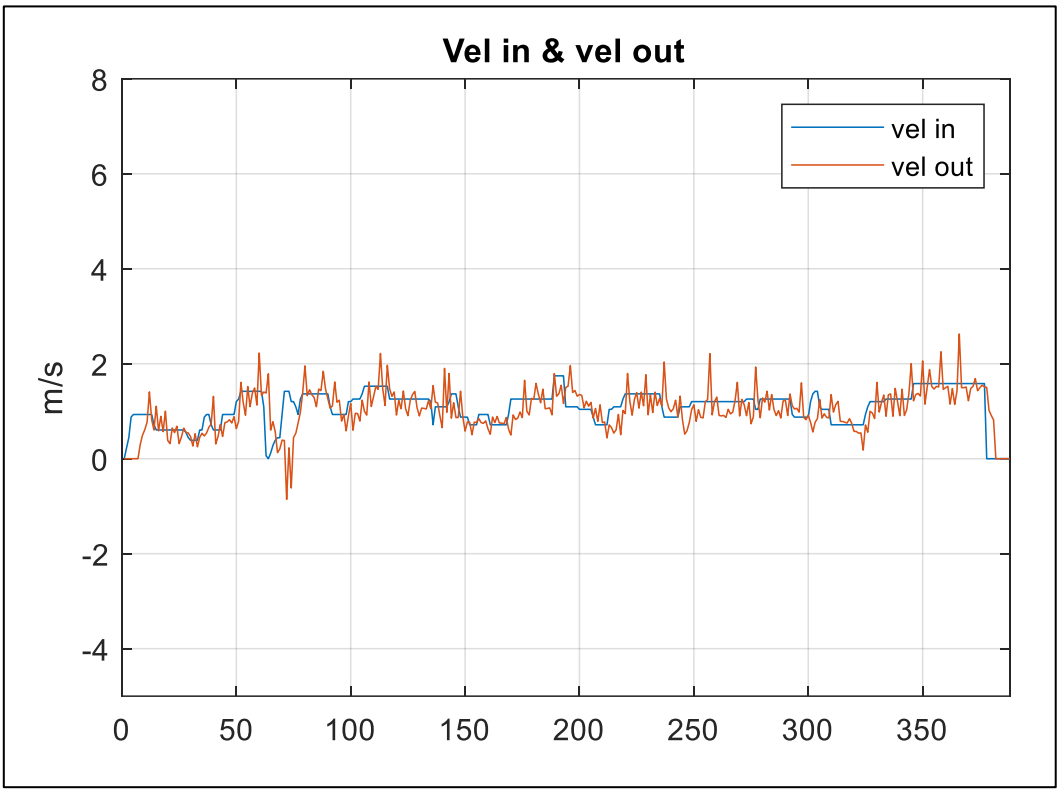



***Figura 107:*** *Resultado velocidad Ruta 2*

### 7.1.3 Ruta 3

Para esta ruta, se realiza una trayectoria lineal que busca mapear un pasillo en la planta inferior del edificio C de la Universidad Europea. Dado que la ruta es lineal y el mapeo se realiza utilizando un LIDAR. Por lo tanto, se presenta el resultado del mapa escaneado en tiempo real. El pasillo propuesto para escanear es el que se ve en la figura 108.

***Figura 108:*** *Propuesta Ruta 2*

La trayectoria realizada se puede ver mediante el enlace: https://youtu.be/lnf8ti_hqxs

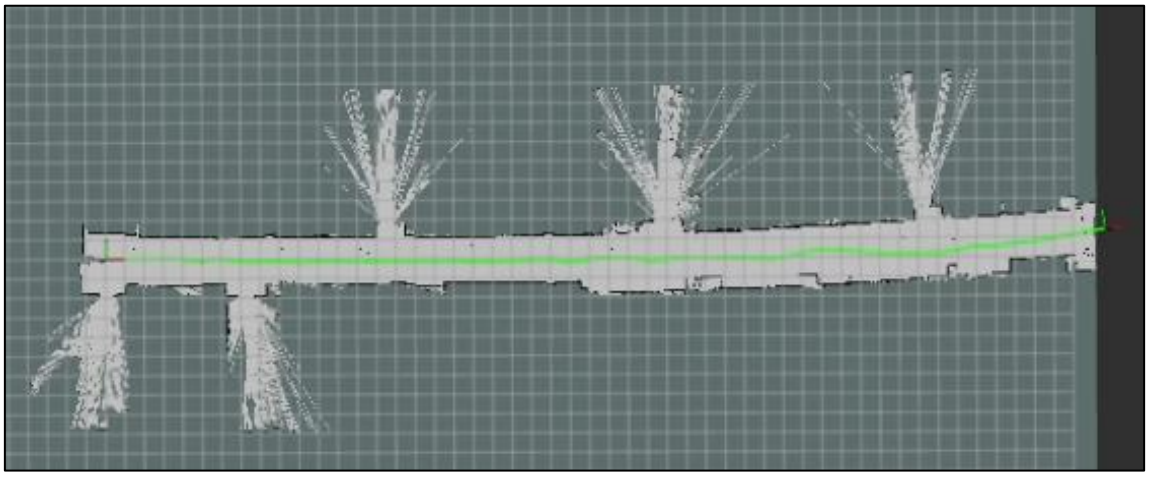


***Figura 109:*** *Resultado Mapa en Rviz Ruta 3*

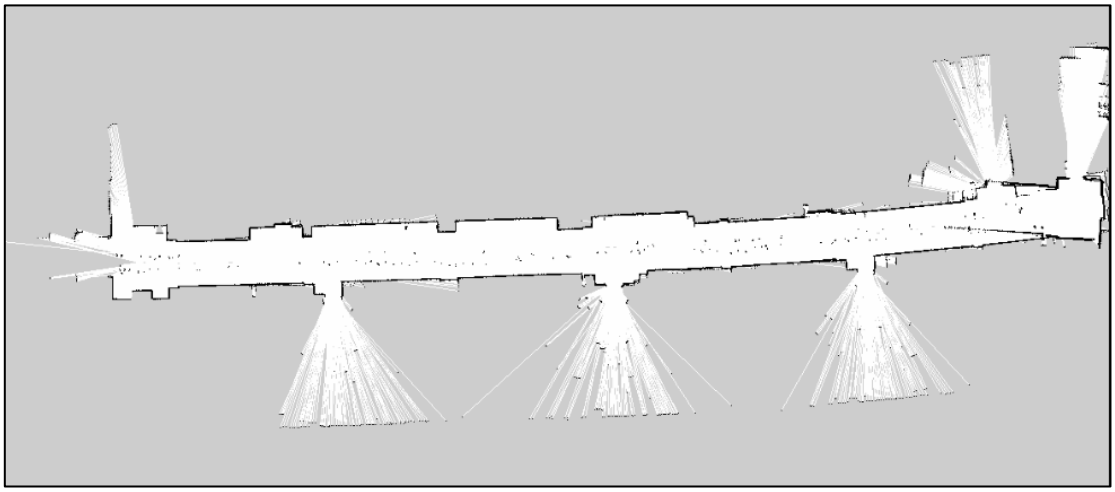


***Figura 110:*** *Resultado Mapa en Ruta 3*

## 7.2 Contraste de resultados obtenidos

Las rutas mostradas anteriormente son el resultado y comprobación de que los cálculos y la estructura implementada funcionan de manera óptima. Sin embargo, durante la recolección de datos se pudieron observar los siguientes problemas.

### 7.2.1 Fallo de Odometría a bajas velocidades

Como se mencionó en el marco teórico de este proyecto, los motores brushless enfrentan problemas al operar a bajas velocidades. Para abordar esto, incluyen un sensor de efecto Hall que permite detectar la posición de la bobina y verificar si ha girado. Sin embargo, en este proyecto, se utiliza dicho sensor como un encoder para detectar los pulsos del motor.

El motor, al girar a bajas velocidades, se enfrenta con la dinámica del vehículo. A estas velocidades, el motor no tiene la fuerza necesaria para mover el vehículo, por lo que, aunque el motor gire debido al diferencial, el vehículo no se mueve, generando un falso pulso de giro. Este falso pulso afecta la medición final de la odometría, resultando en errores de 10 cm y, en algunos casos, sin generar ningún error. En la figura 111, se puede ver gráficamente este tipo de error

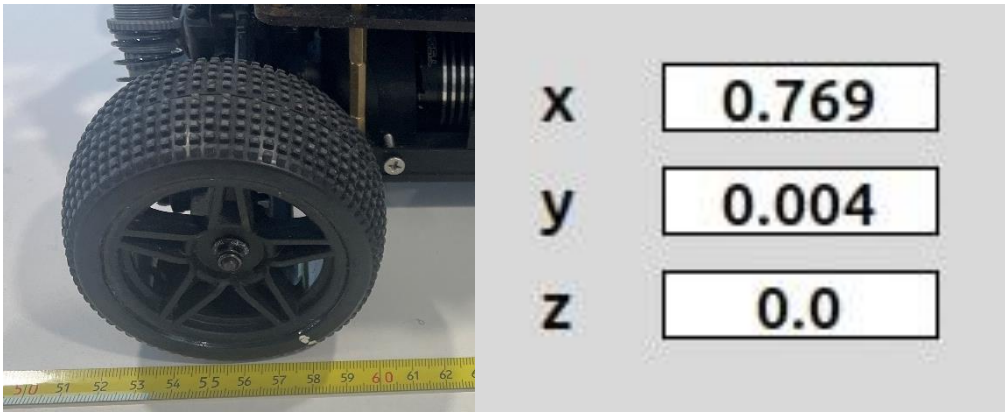



***Figura 111:*** *Error odometría*

En este caso, se realiza una trayectoria lineal a una velocidad de 1 m/s. Al arrancar, el motor genera suficiente fuerza para vencer la dinámica del vehículo, pero también produce múltiples pulsos falsos. Estos contribuyen a un cálculo de odometría que presenta un error específico.

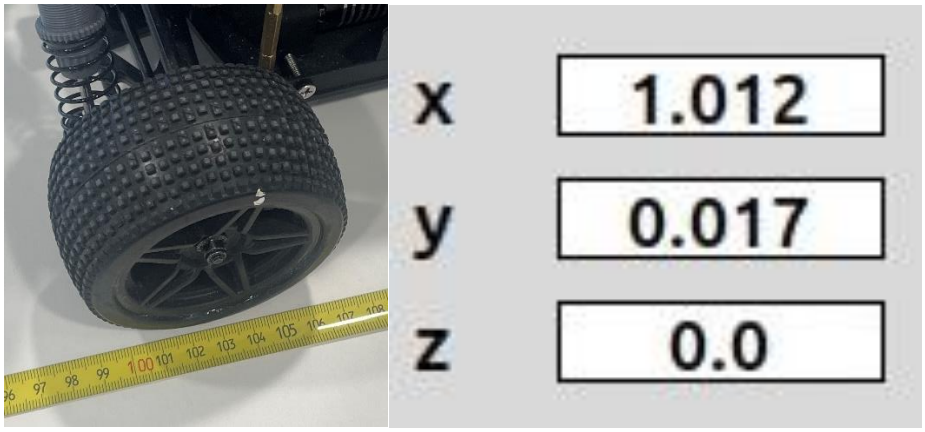



***Figura 112:*** *Corrección odometría*

En este caso se realiza una trayectoria lineal a 2m/s. Al arrancar el motor genera menos pulsos falsos por lo que el error es menora al anterior.

### 7.2.2 Atascamiento y perdida de tracción

Durante una de las pruebas realizadas, el robot se atascó y no pudo avanzar; sin embargo, al tener tracción diferencial, las ruedas continuaron girando.

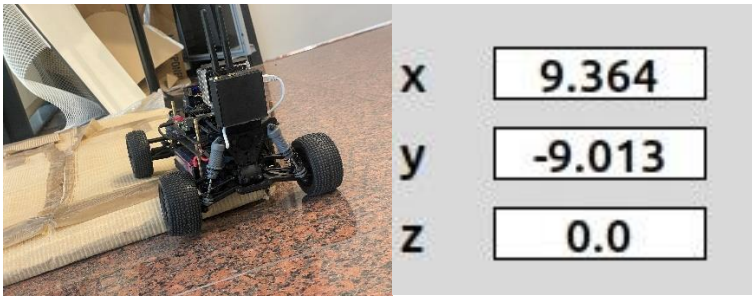



***Figura 113:*** *Corrección odometría*

Este incidente provocó que el motor siguiera generando pulsos, incrementando erróneamente la odometría registrada. Este tipo de errores se debe a que la medición no se realiza directamente en el eje de giro de las ruedas, sino que se calcula utilizando la relación de reducción a partir de un encoder colocado en el motor.

# Capítulo 8. REPOSITORIO DE GITHUB

Un objetivo del proyecto es compartir todo el desarrollo realizado a través de un repositorio en GitHub. Este repositorio, que se encuentra en constante actualización, tiene como objetivo proporcionar una guía concisa sobre los fundamentos del proyecto y ofrecer todos los archivos necesarios para que las personas puedan acceder a ellos y descargarlos.

Dentro del archivo README en GitHub se proporciona una breve guía sobre los archivos incluidos en el proyecto y cómo descargarlos y ejecutarlos.

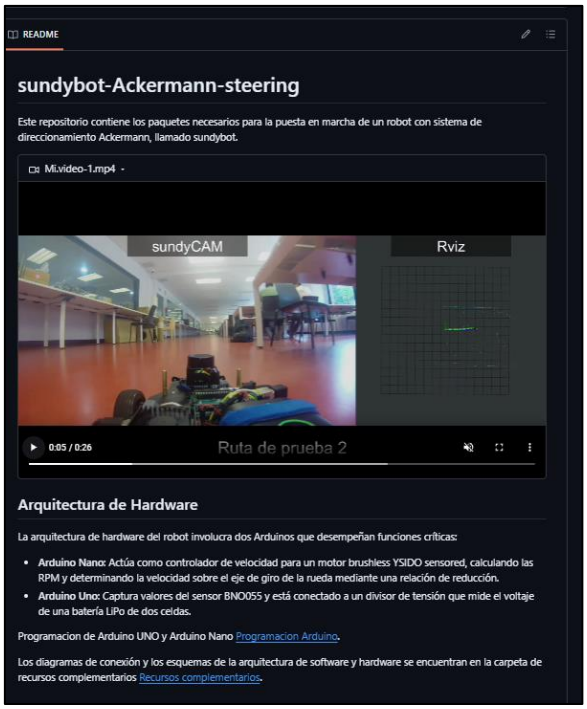



***Figura 114:*** *README*

Además, este repositorio en GitHub contiene material complementario, como diagramas de conexión, imágenes del vehículo, programas adicionales, entre otros recursos útiles.

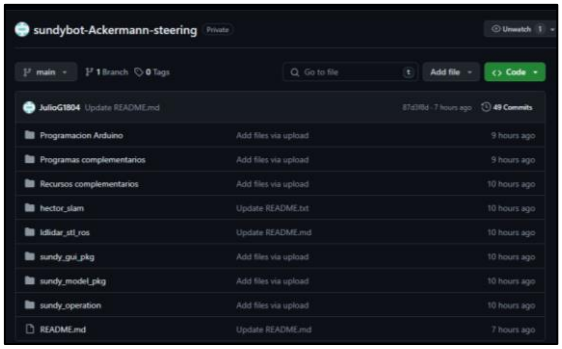



***Figura 115:*** *Carpetas GitHub*

Se puede acceder al repositorio utilizando el siguiente enlace: https://github.com/JulioG1804/sundybot-Ackermann-steering

# Capítulo 9. CONCLUSIONES

## 9.1 Trabajos a futuro

Tras concluir la elaboración del proyecto, se identifican varias líneas de mejora que se abordan en este apartado con el objetivo de desarrollar una mejora continua para los objetivos establecidos en este proyecto.

**Mejoras a futuro**

La adaptación del vehículo en términos de diseño presenta varias deficiencias que no fueron consideradas inicialmente. Estos problemas limitan la capacidad de exploración experimental del proyecto. Al no contar con sistemas adecuados para proteger la electrónica, es complicado utilizar este robot en ambientes externos sin que se vea afectado por factores como el polvo y la lluvia.

Aunque el vehículo incluye una sección para almacenar los cables de la electrónica, estos terminan en conectores que no son óptimos. Estas conexiones podrían ser polarizadas y aseguradas para evitar desconexiones accidentales.

La suspensión del coche actualmente soporta justo el peso del vehículo, lo cual podría limitar la incorporación de electrónica adicional en el futuro.

Las limitaciones mecánicas actuales impiden la adaptación de encoders en los ejes de las ruedas, los cuales podrían considerarse y añadirse en el futuro.

Este robot cuenta con un sistema de control simple, que podría ser mejorado mediante algoritmos implementados en nodos de ROS. Estos algoritmos permitirían tener en cuenta más parámetros y perturbaciones, mejorando así el control de este tipo de vehículos.

Al mejorar los sistemas de control se puede mejorar la comunicación de la arquitectura utilizando mensajes específicos los cuales puedan contribuir a un fácil acceso a los datos comunicados.

**Futuras líneas de trabajo**

Este proyecto ofrece diversas líneas de trabajo y oportunidades de mejora, tales como el perfeccionamiento del sistema de control PID, mejoras físicas y mecánicas para minimizar los errores en el giro de los motores, y la optimización de la comunicación de mensajes entre los dispositivos externos y el ordenador principal.

Este proyecto se centra en proporcionar una arquitectura comprensible; por lo tanto, una línea de trabajo futura es desarrollar programas de navegación autónoma para este tipo de mecanismos. Estos robots tienen aplicaciones específicas, una de las cuales radica en el cálculo y mejora de trayectorias. Con la navegación autónoma, este tipo de proyectos permite realizar pruebas sobre mejoras de trayectorias usando un dispositivo a escala.

Las arquitecturas Ackermann se suelen aplicar en entornos externos donde se requiere una alta eficiencia energética para trayectorias de largo alcance, manteniendo una

velocidad estable y ofreciendo maniobrabilidad completa. Dicho esto, este robot podría utilizarse en las siguientes aplicaciones, que representan futuras líneas de trabajo:

- **Robots de entrega:** Por su maniobrabilidad pueden ser utilizados en la entrega de paquetes o alimentos en espacios urbanos controlados.

- **Vehículos de exploración:** Útiles para realizar misiones de exploración en entornos como desiertos o terrenos no planos.

- **Robots de asistencia en aeropuertos:** Podrían utilizarse para transportar equipaje o guiar a pasajeros en grandes aeropuertos.

- **Robots de limpieza de calles:** Por sus sistemas de dirección pueden realizar barrido en calles.

- **Robots agrícolas:** Pueden ser utilizados en la agricultura para realizar tareas como la fumigación o el monitoreo de cultivos.

- **Robots en parques temáticos:** Para transportar visitantes o realizar tareas de mantenimiento.

- **Vehículos de emergencia robotizados:** Podrían ser utilizados para operaciones de rescate o para transportar equipos de emergencia en áreas afectadas por desastres.

- **Robots en construcción:** Pueden ser útiles en sitios de construcción para transportar materiales pesados o realizar inspecciones.

- **Robots para mapeo y topografía:** Pueden realizar levantamientos topográficos o mapeo de áreas extensas con alta precisión en la dirección.

Mencionando estas aplicaciones que llevan al robot a operar en entornos exteriores, en futuros desarrollos se podría trabajar en paquetes que permitan que el robot realice tareas en dichos entornos, utilizando tecnología GPS para mejorar su navegación y funcionalidad.

## 9.2 Conclusiones del trabajo

Con el desarrollo constante de la robótica, es común desarrollar prototipos y estudiar su comportamiento en tareas específicas que pueden variar en complejidad según su aplicación y sector. El objetivo principal de este trabajo de fin de grado era robotizar un vehículo radiocontrolado para desarrollar una arquitectura en ROS. Esta arquitectura podría utilizarse para desarrollar aplicaciones para robots con este tipo de funcionamiento y continuar evolucionando a través de la comunidad de ROS.

A través de la adaptación y creación del robot denominado Sundybot, donde culla arquitectura de ROS lleva el mismo nombre, diseñada para facilitar su uso en robots con sistema de direccionamiento Ackermann. Esta arquitectura abarca tres aspectos principales: teleoperación, publicación de información y escaneo de mapas mediante algoritmos SLAM, además del desarrollo de interfaces gráficas y recursos de apoyo que permiten visualizar y manipular los datos proporcionados por el robot durante su navegación.

Cada componente ha sido diseñado para funcionar tanto de manera integrada como de forma independiente, permitiendo a quienes acceden al proyecto en el repositorio de GitHub extraer y utilizar cada programa individualmente o en conjunto.

Es importante destacar que esta arquitectura representa un paso inicial para comprender la estructura más básica que puede implementar un robot con direccionamiento Ackermann. Por lo tanto, esta arquitectura puede desarrollarse y mejorarse aún más, abordando los aspectos más débiles que puedan presentar fallos al momento de integrarla en otros proyectos.

Es importante destacar que esta arquitectura representa un paso inicial para comprender la estructura más básica que puede implementar un robot con direccionamiento Ackermann. Por lo tanto, esta arquitectura puede desarrollarse y mejorarse aún más, abordando los aspectos más débiles que puedan presentar fallos al momento de integrarla en otros proyectos.

Finalmente, las pruebas realizadas demuestran que el proyecto desarrollado puede ser un recurso valioso que puede ayudar a muchas personas en el desarrollo de sus propios robots.

## 9.3 Conclusiones personales

El desarrollo de este proyecto supuso un reto personal para mí, pero al mismo tiempo ha sido una experiencia que me ha ayudado a desarrollar habilidades de investigación y análisis crítico. Dado que los sistemas de direccionamiento Ackermann no están académicamente desarrollados, era común encontrar prototipos muy adaptados y complejos que resultaban difíciles de entender. Buscar información relevante representó un desafío considerable, y aplicarla supuso un reto adicional que, al superarlo, generó una gran satisfacción personal.

Diseñar un robot no es una tarea sencilla, ya que implica considerar una amplia variedad de variables y parámetros. A lo largo del desarrollo de este proyecto, dediqué muchas horas a reflexionar sobre cómo aplicar ciertos elementos, dedicando una gran parte del tiempo al diseño, tanto de software como de hardware.

El uso de herramientas computacionales para el desarrollo de circuitos y para el diseño de modelos en 3D es crucial cuando se aborda un proyecto que no ha sido creado anteriormente. Estas herramientas poseen un potencial enorme, que se hace evidente y adquiere valor en el momento en que se utilizan en desarrollos como este.

Al utilizar la herramienta de ROS, me enfrenté a numerosas dudas sobre el funcionamiento de ciertos aspectos de este framework. El portal wiki de ROS se convirtió en una guía esencial para la programación de la arquitectura ROS, y los foros resultaron ser de gran ayuda para integrar herramientas externas al framework.

En definitiva, este trabajo final no solo ha sido un hito para culminar mi carrera universitaria, sino también una experiencia formativa que me ha hecho apreciar lo impresionante que es la evolución tecnológica.

# BIBLIOGRAFÍA

Adafruit Industries. (2020). *Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055*. Adafruit. Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055

Amazon server. (2023). *¿Qué es Pyhton*. Amazon.Com. https://aws.amazon.com/es/what-is/python/

Angel Robleadno. (2019, July 22). *Origen de C++*. OpenWebinars. https://openwebinars.net/blog/que-es-cpp/

ARME. (2022, September 19). *¿Qué es la odometría?* Linkedin. https://es.linkedin.com/pulse/qu%C3%A9-es-la-odometr%C3%ADa-arme-rob%C3%B3tica-m%C3%B3vil

Betania V. (2023, September 26). *¿Qué es Ubuntu? Una guía rápida para principiantes*. Hostinger Tutoriales. https://www.hostinger.es/tutoriales/que-es-ubuntu

Chris Lalancette. (2023, June 27). *Distributions*. Wiki ROS.

Computer Hope. (2023, December 26). *Service*. Computer Hope.

Dorian Kurzaj. (2016, August 26). *Messages*. Wiki ROS. https://wiki.ros.org/Messages

Edisonsasig. (2021, April 13). *La cinemática en la robótica*. Roboticoss. https://www.scribbr.es/citar/generador/folders/3VYxarcCDjnOQlb6T02Dyb/lists/5QZyVbxh3RjMALyEDaTyYx/

Edisonsasig. (2023, July 14). *Modelo cinemático y simulación de un robot móvil diferencial*. Roboticoss. https://roboticoss.com/modelo-cinematico-y-simulacion-con-python-robot-movil-diferencial/

eSTEAM Education. (2023). *Raspberry Pi 4 Model B*. ESTEAM Education. https://tienda.esteamedu.es/placas/116-raspberry-pi-4-model-b-4-gb.html#:~:text=La%20Raspberry%20Pi%204%20Model,ordenador%20de%20muy%20bajo%20coste.

HR motor. (2023, October 20). *¿Qué es la dirección Ackermann?* HR Motor.

Isis Sulbarán. (2023, November 27). *¿QUÉ ES ARQUITECTURA DE COMPUTADORAS?* Tiffin University.

Juan Eduardo Riva. (2021, August 13). *Comprendiendo Nodos ROS*. Wiki ROS. https://wiki.ros.org/es/ROS/Tutoriales/ComprendiendoNodosROS

Juan Eduardo Rivas. (2021, August 18). *Introdroduccion*. Wiki ROS. https://wiki.ros.org/es/ROS/Introduccion

MATLAB & Simulink. (n.d.-a). *Introducción a SLAM*. MathWorks España. Retrieved May 20, 2024, from https://es.mathworks.com/discovery/slam.html

MATLAB & Simulink. (n.d.-b). *Transformaciones de coordenadas en robótica*. MathWorls España. Retrieved May 20, 2024, from https://es.mathworks.com/help/robotics/coordinate-system-transformations.html?category=coordinate-system-transformations&s_tid=CRUX_topnav

Miriam Martínez Canelo. (2024, January 16). *Qué es la Arquitectura de Software: fundamentos clave*. Profile. https://profile.es/blog/que-es-la-arquitectura-de-software/

nvidia. (2020, February 20). *A 1/10th-Scale Race Car for Autonomous Driving Research*. Nvidia. https://resources.nvidia.com/gtcd-2020/gtc2020s22002

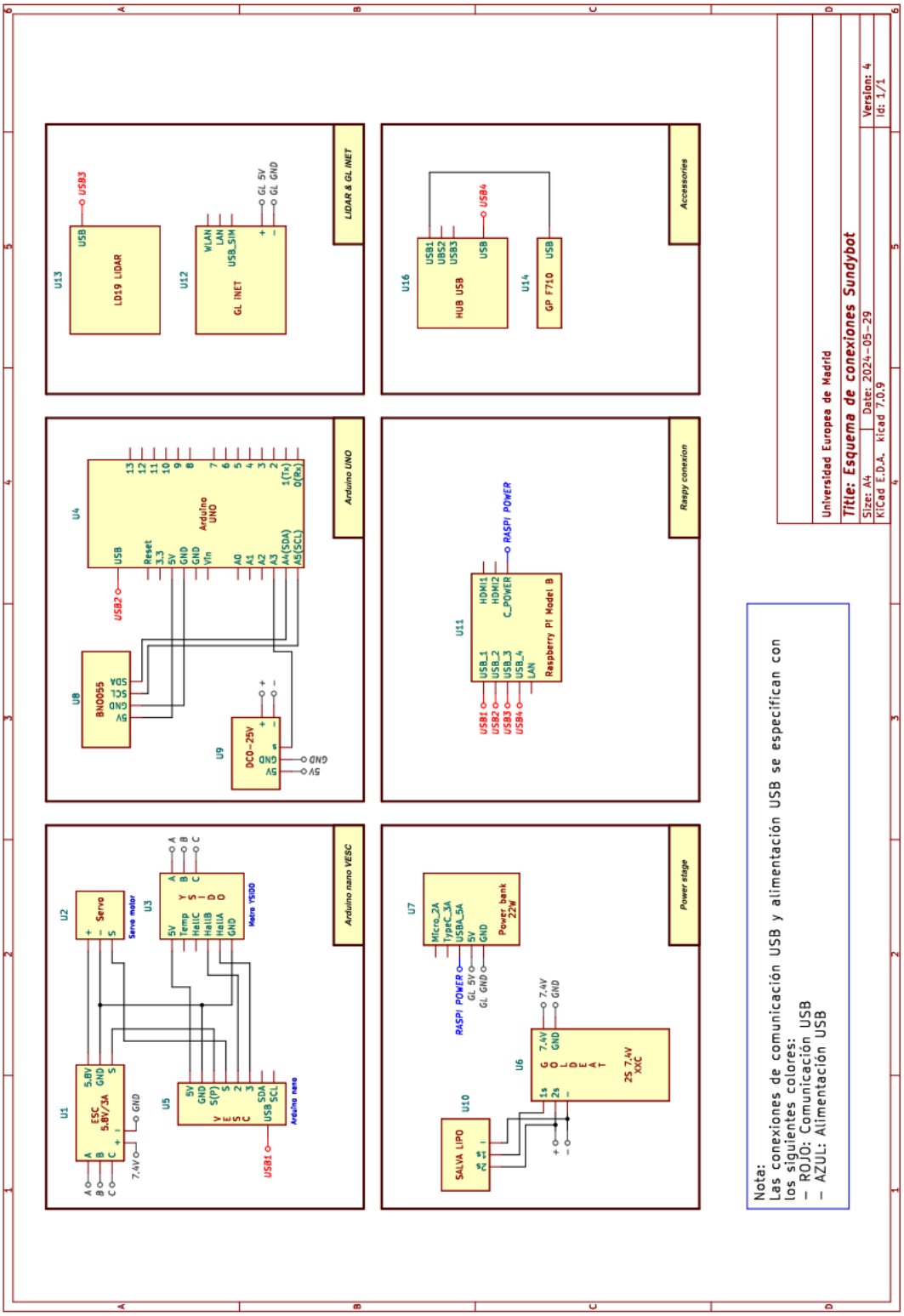
Pablo Huet. (2022, August 24). *Arquitectura de software: Qué es y qué tipos existen*. OpenWebinars. https://openwebinars.net/blog/arquitectura-de-software-que-es-y-que-tipos-existen/

RobotShop. (2020). *LD19 D300 LiDAR Developer*. RobotShop. https://www.robotshop.com/es/products/ld19-d300-lidar-developer-kit-360-degrees-dtof-laser-scanner-support-ros1-ros2-raspberry-pi-jetson-nano

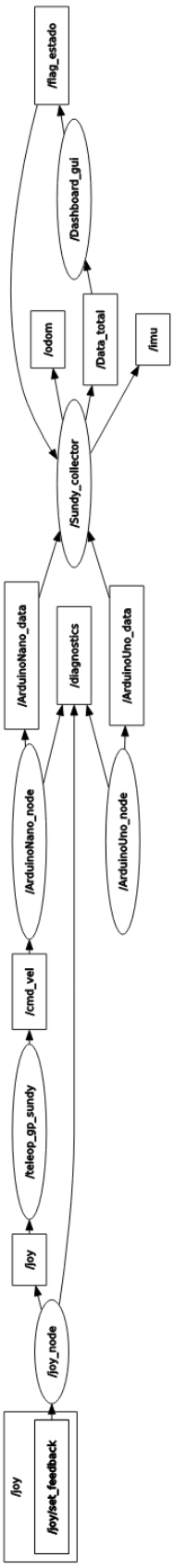
Tully Foote. (2019, February 20). *Topics*. Wiki ROS. https://wiki.ros.org/Topics
Universidad de Sevilla. (2019). *Capítulo 2. Desarrollo teórico*.
Wikipedia. (2024, January 24). *Orientación (geometría)*. Wikipedia.
Yúbal Fernández. (2022, September 23). *Que es Arduino, como funciona y que puedes hacer con uno*. Xatka. https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno

# ANEXOS

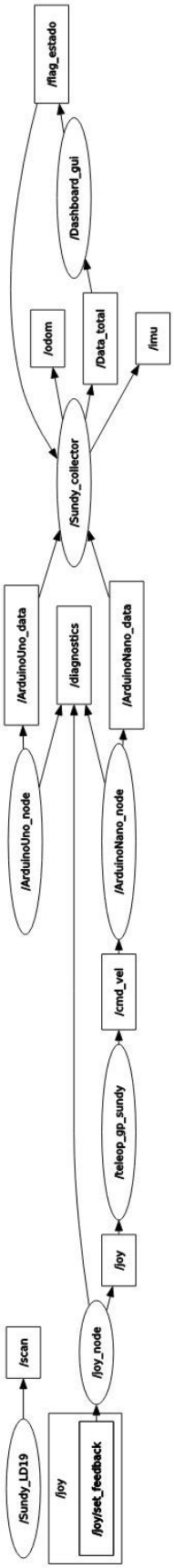
## Anexo 1



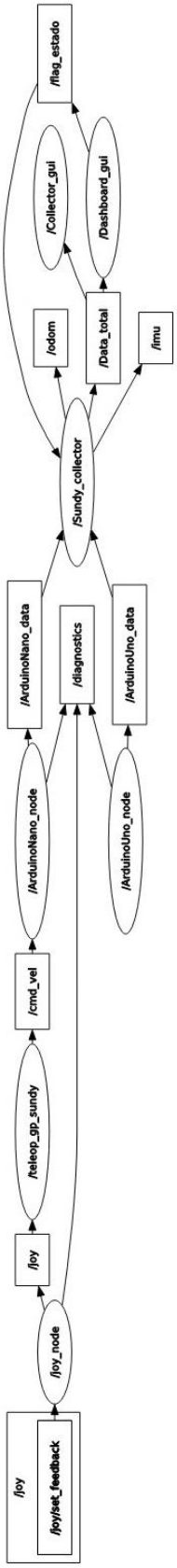
## Anexo 2


/flag_estado
/Dashboard_gui
/odom
/Data_total
/imu
/Sundy_collector
/ArduinoNano_data
/diagnostics
/ArduinoUno_data
/ArduinoNano_node
/ArduinoUno_node
/cmd_vel
/teleop_gp_sundy
/joy
/joy_node
/joy
/joy/set_feedback

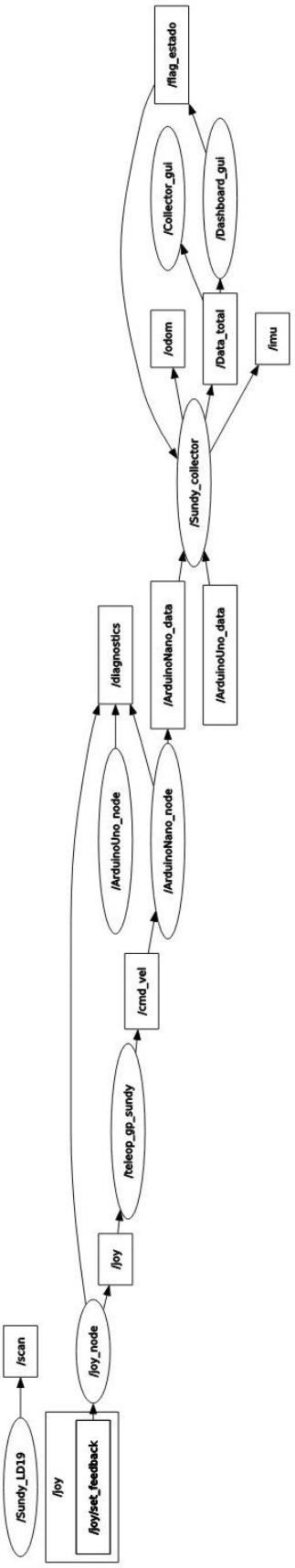
## Anexo 3


/flag_estado
/Dashboard_gui
/odom
/Data_total
/imu
/Sundy_collector
/ArduinoUno_data
/diagnostics
/ArduinoNano_data
/ArduinoUno_node
/ArduinoNano_node
/cmd_vel
/teleop_gp_sundy
/joy
/scan
/joy_node
/Sundy_LD19
/joy
/joy/set_feedback

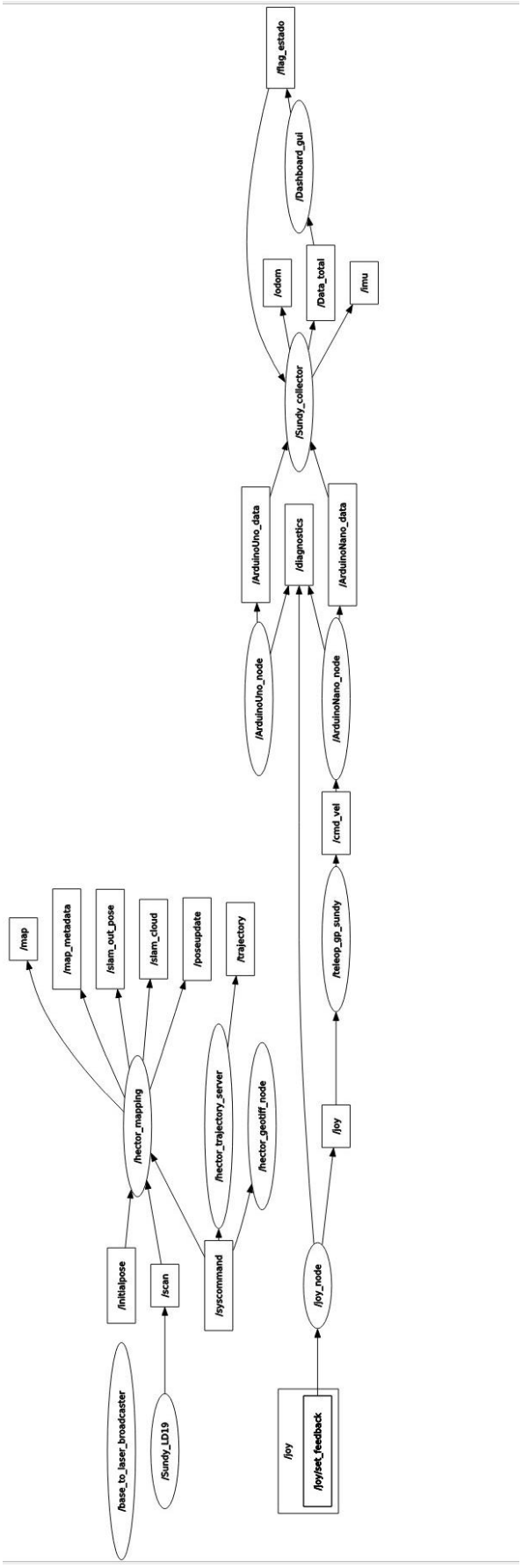
## Anexo 4


/flag_estado
/Collector_gui
/Dashboard_gui
/odom
/Data_total
/imu
/Sundy_collector
/ArduinoNano_data
/diagnostics
/ArduinoUno_data
/ArduinoNano_node
/ArduinoUno_node
/cmd_vel
/teleop_gp_sundy
/joy
/joy_node
/joy
/joy/set_feedback

## Anexo 5


/flag_estado
/Collector_gui
/Dashboard_gui
/odom
/Data_total
/imu
/Sundy_collector
/diagnostics
/ArduinoNano_data
/ArduinoUno_data
/ArduinoUno_node
/ArduinoNano_node
/cmd_vel
/teleop_gp_sundy
/joy
/scan
/joy_node
/Sundy_LD19
/joy
/joyset_feedback

## Anexo 6


/flag_estado
/Dashboard_gui
/odom
/Data_total
/imu
/Sundy_collector
/ArduinoUno_data
/diagnostics
/ArduinoNano_data
/ArduinoUno_node
/ArduinoNano_node
/cmd_vel
/map
/map_metadata
/slam_out_pose
/slam_cloud
/poseupdate
/trajectory
/teleop_gp_sundy
/hector_mapping
/hector_trajectory_server
/hector_geotiff_node
/joy
/initialpose
/scan
/syscommand
/joy_node
/base_to_laser_broadcaster
/Sundy_LD19
/joy
/joyset_feedback