UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería Arquitectura y Diseño

Programa de Maestría y Doctorado en Ciencias e Ingeniería

Planificación de trayectorias para robot móvil autónomo usando algoritmos inteligentes

T E S I S

que para obtener el grado de

Maestría en Ingeniería

Presenta:

Jorge Galarza Falfan

Director de Tesis:

Dr. Everardo Inzunza González

Ensenada, Baja California, México
Junio, 2024

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño

Planificación de trayectorias para robot móvil autónomo usando algoritmos inteligentes

TESIS

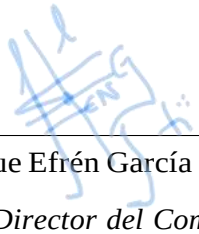
que para obtener el grado de MAESTRO en INGENIERÍA presenta:

Jorge Galarza Falfan

Y aprobada por el siguiente comité:

Dr. Everardo Inzunza González

Director del Comité



Dr. Enrique Efrén García Guerrero

Co-Director del Comité

Dr. Oscar Roberto López Bonilla

Miembro del Comité

Dr. Ulises Jesús Tamayo Pérez

Miembro de Comité

Dr. Oscar Adrián Aguirre Castro

Miembro del Comité

Junio, 2024

RESUMEN de la tesis de **Jorge Galarza Falfan**, presentada como requisito parcial para obtener el grado de MAESTRO EN INGENIERÍA, del programa de Maestría y Doctorado en Ciencias e Ingeniería de la UABC. Ensenada, B. C. México, Junio, 2024.

Planificación de trayectorias para robot móvil autónomo usando algoritmos inteligentes

Resumen aprobado por:

Dr. Everardo Inzunza González
Director de Tesis

Dr. Enrique Efrén García Guerrero
Co-Director de Tesis

Los sistemas robóticos empleados en procesos industriales y de servicio están incorporando cada vez más tecnología de aprendizaje automático. Dicha evolución se justifica porque estos ofrecen una mayor adaptabilidad a los constantes cambios en el entorno del robot, interacción y funcionamiento más rápidos. Este trabajo de tesis presenta el proceso de ensamblaje y desarrollo de un Robot Móvil Autónomo (AMR por sus siglas en inglés), la implementación de planificación de trayectorias por medio del algoritmo de localización adaptativa de Monte Carlo (AMCL por sus siglas en inglés), y visión artificial mediante Redes Neuronales Convolucionales (CNNs por sus siglas en inglés), concretamente ResNet18 y YOLOv3 para la detección de objetos en el entorno. El sistema de visión artificial usando ResNet18 alcanzó una Exactitud del 98,5 %, con una Precisión del 96,91 %, una Recuperación del 98.58 % y un F1-score del 98,5 %. Al usar YOLOv3, se obtuvo una Exactitud del 96 %, con una Precisión del 98.45 %, una Recuperación del 96 % y un F1-score del 95.99 %. Estas métricas demuestran la fiabilidad de los algoritmos inteligentes, el potente hardware integrado, el sistema operativo de código abierto ROS (Robot Operating System por sus siglas en inglés) y los sensores en las aplicaciones robóticas. Los resultados confirman que las tecnologías de aprendizaje automático, especialmente en el contexto de los AMR, ofrecen una opción atractiva en varios campos de la robótica, conocidos por sus importantes usos en movilidad, transporte, limpieza, ensamblaje y muchos escenarios diferentes.

Palabras clave: *robot móvil autónomo; planificación de trayectorias; navegación; visión artificial; aprendizaje profundo; aprendizaje automático; inteligencia artificial.*

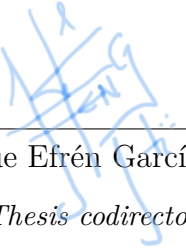
ABSTRACT of the thesis of **Jorge Galarza Falfan**, presented as a partial requirement to obtain the degree of MASTER in ENGENIEERING, of the program of MSc and PhD in Sciences and Engineering of UABC. Ensenada, B. C., Mexico, June 2024.

Path planning for autonomous mobile robot using intelligent algorithms

Abstract approved by:

Dr. Everardo Inzunza González

Thesis director



Dr. Enrique Efrén García Guerrero

Thesis codirector

Robotic systems used in industrial processes and services increasingly adopt machine learning technology. This evolution is justified as it offers greater adaptability to constant changes in the robot environment, faster interaction and more straightforward operation. This thesis work presents the assembly process and development of an Autonomous Mobile Robot (AMR), the implementation of path planning using the Adaptive Monte Carlo Localization (AMCL) algorithm, and artificial vision through Convolutional Neural Networks (CNNs), specifically ResNet18 and YOLOv3 for object detection in the environment. The artificial vision system using ResNet18 achieved an Accuracy of 98.5%, with a Precision of 96.91%, a Recall of 98.58% and an F1-score of 98.5%. YOLOv3 reached an Accuracy of 96%, with a Precision of 98.45%, a Recall of 96% and an F1-score of 95.99%. These metrics demonstrate the reliability of intelligent algorithms, powerful embedded hardware, the open-source operating system ROS (Robot Operating System) and sensors in robotic applications. The results confirm that machine learning technologies, especially in the context of AMRs, offer an attractive option in several fields of robotics, known for their essential uses in mobility, transportation, cleaning, assembly and many different scenarios.

Keywords: *autonomous mobile robot; path planning; navigation; artificial vision; deep learning; machine learning; artificial intelligence.*

A mi familia

La investigación de este trabajo se realizó gracias al apoyo económico de las siguientes entidades:

- Este proyecto de tesis se realizó con recursos propios del CA Instrumentación Electrónica Aplicada a Sistemas de Producción. Principalmente con recurso del proyecto de investigación interna aprobado en la 22a convocatoria interna.
- Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT).

Agradecimientos

Agradezco profundamente al ***Dr. Everardo Inzunza González*** *por su amistad y entusiasmo en el desarrollo del presente trabajo; por su guía y apoyo constante.*

Al ***Dr. Enrique Efrén García Guerrero****, por su dirección y paciencia durante el desarrollo de este proyecto.*

Al ***Dr. Oscar Roberto López Bonilla*** *y* ***Dr. Ulises Jesús Tamayo Pérez****, por siempre ofrecerme su asesoría y por sus interesantes cuestionamientos que ayudaron a delimitar este proyecto.*

Al ***Dr. Oscar Adrián Aguirre Castro****, por su amistad, por el tiempo y esfuerzo dedicados para guiarme en este trabajo.*

A mis padres ***Jorge Galarza López*** *y* ***Victoria Falfan Juárez****, por apoyarme diariamente y por su confianza.*

A mis hermanas ***Victoria*** *y* ***Okzanna****, por ser un ejemplo a seguir.*

A ***CONAHCYT****, por apoyar económicamente este proyecto de maestría.*

A la ***Universidad Autónoma de Baja California*** *y la* ***FIAD****, por darme las herramientas de formación profesional e instalaciones necesarias para realiza este*

trabajo.

A los profesores de posgrado, por su dedicación y paciencia.

A mis compañeros de clase, por su amistad y apoyo.

A todas las personas que de alguna manera me ofrecieron su ayuda.

Ensenada, B. C. México. Jorge Galarza Falfan

Junio, 2024.

Índice general

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

La robótica y específicamente la móvil se compone de una amplia cantidad de otras áreas de estudio como lo son las ingenierías mecánica, eléctrica, electrónica, así como informática, inteligencia artificial (IA) y también en algunos casos ciencias sociales [2]. Debido a ello, a lo largo de las décadas, desde sus inicios a principios del siglo XX [9], los robots móviles se han desarrollado en función de las herramientas disponibles en la época y se han planteado como máquinas programables para tareas puntuales, sin embargo las capacidades cognitivas surgieron décadas después de la mano de investigaciones como la supervisada por Charles Rosen en el Instituto de Investigaciones de Stanford en la década de 1960, donde el robot móvil autónomo "Shakey" [1], mostrado en la Figura 1 fue capaz de navegar mediante el uso de una cámara de televisión y un sistema de control basado en un algoritmo heurístico para realizar correcciones en el movimiento y adaptarse a las características del entorno.

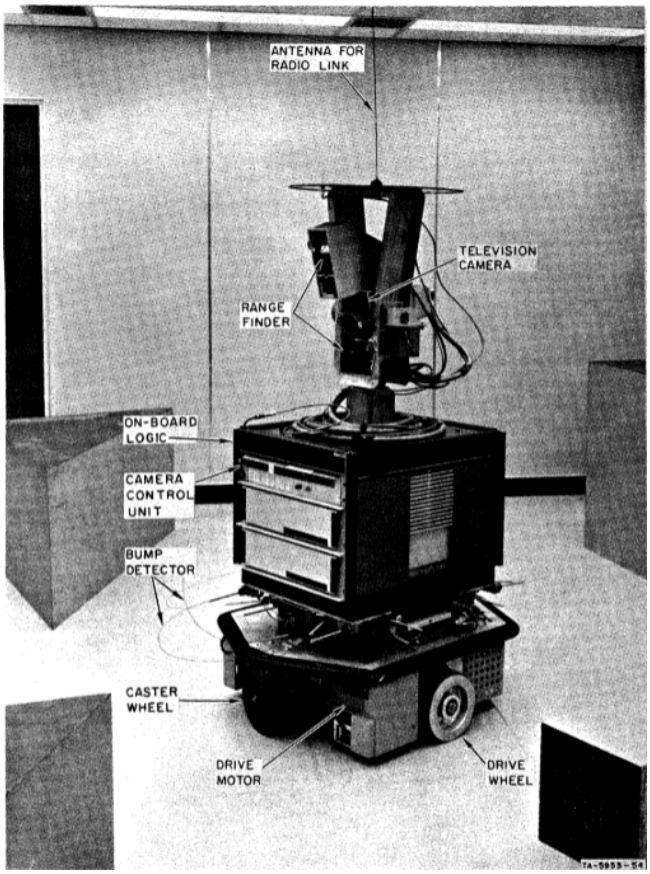


Figura 1: Robot autónomo Shakey introducido en 1968, tomado de [1].

Una importante adición al área de estudio de la robótica móvil es la inteligencia artificial, que desde finales de 1950 [10] se ha establecido como una herramienta que permite introducir capacidades de percepción visual, toma de decisiones, razonamiento, aprendizaje, y uso de lenguaje natural [9]. Es posible afirmar que aquellos robots que cuenten con dichas habilidades tendrán la suficiente capacidad para trabajar conjuntamente con el ser humano y hacer que su implementación sea mucho más comprensible por parte de estos. Su funcionamiento se basa en el uso de algoritmos bio inspirados para crear modelos que utilizan técnicas de probabilidad y estadística para calcular una respuesta del sistema con base en un conjunto de datos previamente procesados y que sirven como ejemplo del tipo de respuesta que debe proporcionar.

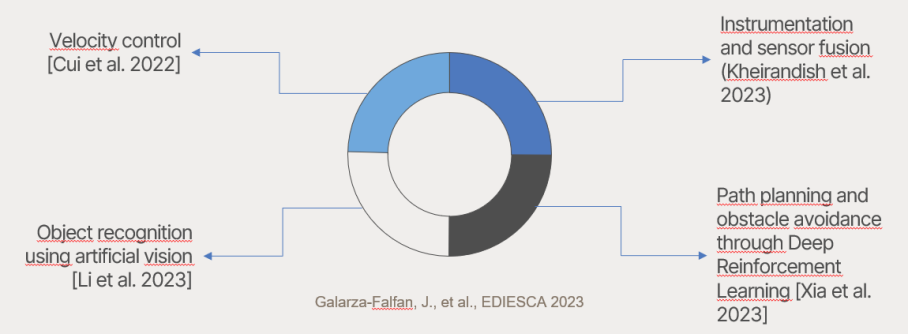


Figura 2: Cuatro áreas de actual desarrollo que relaciona la IA con la robótica móvil.

La Figura 2 ilustra cuatro de las más relevantes áreas de investigación que actualmente se están abordando para mejorar el funcionamiento de robots móviles mediante modelos de Aprendizaje Máquina (ML, por sus siglas en inglés) y Aprendizaje Profundo (DL, por sus siglas en inglés).

El hardware requerido para realizar este tipo de tareas también ha sufrido una evolución que, de manera escalonada ha permitido la implementación de técnicas de inteligencia artificial en robots móviles. A diferencia de los primeros robots autónomos propuestos en el siglo XX, en la actualidad existen sistemas embebidos altamente potentes que permiten realizar cómputo distribuido para procesar la información de todos los sistemas del robot (Figura 2) de manera simultánea. Esto ha ayudado a incrementar la confiabilidad y reducir las dimensiones en los robots móviles actuales.

1.1 Motivación

La principal motivación de esta investigación es la exploración de métodos relacionados a redes neuronales artificiales para solucionar el problema de planificación de trayectorias de robots móviles. Existe evidencia en la literatura reciente acerca del interés de la comunidad científica y también del sector industrial por fortalecer el desarrollo de sistemas autónomos. Esto incluye equipo de automatización industrial y movilidad. Este último es también parte de la motivación de esta investigación; porque

los sistemas de movilidad industriales que se usan actualmente en gran parte de la industria de manufactura son de tipo Vehículo Guiado Automático (AGV, por sus siglas en inglés). Este tipo de vehículo no es autónomo y require de control por parte de un operador, limitando su flexibilidad.

1.2 Planteamiento de problema

Los trabajos actuales en robótica móvil se centran principalmente en aplicar la inteligencia artificial a distintos niveles del sistema robótico en su conjunto [11]. Esto incluye el control velocidad [12], instrumentación, fusión de sensores [13], reconocimiento de objetos, etc. Sin embargo, la planificación de trayectorias es una tarea particular que aumenta la fiabilidad de cualquier robot, simplifica la interacción con un operador humano y reduce los costes de mantenimiento.

A lo largo del tiempo han existido una gran cantidad de metodologías para resolver la planificación de trayectorias [14], incluyendo los planificadores tradicionales basados en modelos como el método de Campos Potenciales Artificiales, método de coste de celdas, método de ventana dinámica, etc. [15]. A estos algoritmos se les conoce como clásicos y se han usado para controlar el movimiento de la mayoría de los primeros robots móviles. Aunque los métodos tradicionales son fiables, requieren recálculos constantes, lo que produce un procesamiento y un cálculo considerablemente elevados. Los primeros experimentos para la solución de la planificación de trayectorias de robots mediante enfoques heurísticos datan de finales de la década de 1960 [16], en el Instituto de Investigacion de Stanford, donde Nilsson et al. desarrollaron "Shakey"; un robot autónomo que navega utilizando el algoritmo inteligente A* [1]. Sin embargo, de acuerdo con las desventajas de los algoritmos tradicionales expuestas con anterioridad, existe una necesidad por explorar métodos más optimizados de la mano de técnicas de cómputo avanzadas como son las redes neuronales artificiales [17]. Porque se conoce

que existen ventajas [18] al emplear dichos algoritmos en los equipos de hardware producidos en la actualidad, que junto a sistemas de software de acceso abierto y open-source se pueden desarrollar soluciones al problema de planificación de trayectorias de manera flexible y escalonada.

1.2.1 Descripción del problema

Este proyecto surge debido a la necesidad existente por parte de la industria de estudiar y definir diferentes métodos para controlar el desplazamiento de robots móviles autónomos en ambientes cerrados con aplicaciones específicas.

Como resultado de la revisión bibliográfica, se puede apreciar que los algoritmos basados en aprendizaje máquina están siendo explorados debido a sus claras ventajas. Sin embargo, su implementación no es trivial, y se requiere de una clara delimitación de la aplicación que se le dará al robot móvil, y la creación u obtención de un dataset extenso para contemplar distintos escenarios de la interpretación del entorno, entre otros problemas que deben ser atendidos. Es por ello que el principal problema es la implementación del algoritmo de planificación de trayectorias basado en un algoritmo de optimización, para lo cual se requiere definir todas las variables o características que aporten información útil para modelar el comportamiento del robot móvil y su entorno.

1.2.2 Preguntas de investigación

Una pregunta a responder es ¿cuántas y cuáles son las características o variables mínimas que describen de manera clara la relación entre el estado del vehículo con respecto a su entorno y su tarea a realizar? Lo anterior se plantea como un problema de localización porque se debe hallar una ruta viable de acuerdo con ciertas restricciones. Algunas de las variables que se proponen se describen en la metodología.

Por otro lado, otra pregunta que se puede identificar es: ¿cuál es la contribución que

debe tener cada sensor (porcentaje) del robot móvil para lograr una localización con un error reducido y predecible? La solución de este problema facilitaría la corrección del error.

1.2.3 Delimitación del problema

El proyecto está orientado de manera específica al ensamblado y puesta en marcha de un AMR a nivel prototipo, que sirva como base para la implementación de algoritmos de ML y DL que trabajen con herramientas de hardware y software open-source en sistemas de automatización. Esto como consecuencia del actual interés en la integración de nuevas tecnologías y paradigmas para lograr soluciones más óptimas a distintos problemas y necesidades en el sector industrial.

1.3 Objetivos

1.3.1 Objetivo general

Implementar un algoritmo de planificación de trayectorias en un prototipo de robot móvil autónomo, con capacidad de adaptarse a cambios en las condiciones de un entorno cerrado.

1.3.2 Objetivos específicos

- Desarrollar un prototipo de AMR con la capcidad de implementar un sistema embebido con capacidad de multi procesamiento y diferentes tipos de sensores.
- Implementar un algoritmo de planificación de trayectorias, habilitando los canales de comunicación necesarios.
- Generar un conjunto de imágenes de las clases necesarias para trabajar con algoritmos de inteligencia artificial.

- Entrenar un modelo de DL que pueda ayudar en la tarea de evasión de obstáculos, usando aprendizaje por transferencia.

1.4 Organización de la tesis

El orden de los siguientes capítulos de la presente tesis es el siguiente. Primero, en el Capítulo 1 se hace el planteamiento del problema donde se explica el escenario y contexto de la investigación. Además, también se especifican los límites que definen el trabajo realizado. Después, en el Capítulo 2 se presenta el marco teórico que incluye los fundamentos mas relevantes. En el Capítulo 3 se exploran diferentes autores y sus trabajos de investigación relacionados al tema. Seguido del Capítulo 4, que contiene la metodología utilizada, donde se desglosa el hardware y software seleccionados. En el Capítulo 5 se concentran todos los resultados que se obtuvieron durante la experimentación. Finalmente, en el Capítulo 6, se hace un recuento de las conclusiones a las que llegaron después de analizar los resultados.

Capítulo 2

Marco Teórico

Un AMR es un tipo de robot específico, que pertenece al conjunto de robots móviles debido a características como locomoción, circuito de potencia, y requerimientos de energía. Los robots móviles son todos aquellos robots capaces de navegar en su entorno moviéndose por medio de su propio sistema de locomoción, ya sea éste a base de ruedas o extremidades [2]. La locomoción de un robot se puede interpretar como el tipo de mecanismo que éste utiliza para lograr desplazarse por sus propios medios. Para un robot móvil con ruedas, el movimiento se logra por medio de motores acoplados a algún tipo de transmisión que, a su salida tiene una velocidad y torque adecuados para dicha tarea [9].

2.1 Arquitectura de un AMR

Dependiendo de la configuración de las ruedas del robot, es posible utilizar una sola transmisión para todo el sistema, o una transmisión para cada rueda. Para tener un buen contacto con el terreno, se pueden usar distintos tipos de ruedas, pero el más generalizado son las ruedas de goma. La elección del tipo y tamaño de las ruedas se basa en las características del terreno y la capacidad de estas para evitar deslizamientos indeseados, porque esto puede afectar la capacidad de navegación del robot, al confundir al sistema de odometría que es el encargado de llevar el registro de cada movimiento que

el robot tiene e interpretarlo como desplazamiento dentro de un mapa de navegación [19].

Otra clasificación importante que se puede hacer es el tipo las ruedas del robot y si son motrices o no, esto puede puede ofrecer diferentes capacidades de movimiento. Como se muestra en la Figura 3, un tipo de configuración es el de dirección, que tiene cuatro ruedas de las cuales dos son acopladas a un eje motriz y a un eje que se utiliza para ajustar la dirección. Es muy similar a como funcionan los automóviles, y ofrece cierta simplicidad en el hardware porque normalmente utiliza un solo motor y transmisión para controlar el eje motriz, y el eje de dirección se puede controlar con un servomecanismo. Otra configuración es la diferencial, que es bastante utilizada debido a su simplicidad de programación. Utiliza dos ruedas motrices alineadas en el mismo eje, pero controladas por motores separados, además de una o dos ruedas no motrices que sirven para estabilizar la base del robot y ayudan a distribuir el peso de la plataforma móvil [2].

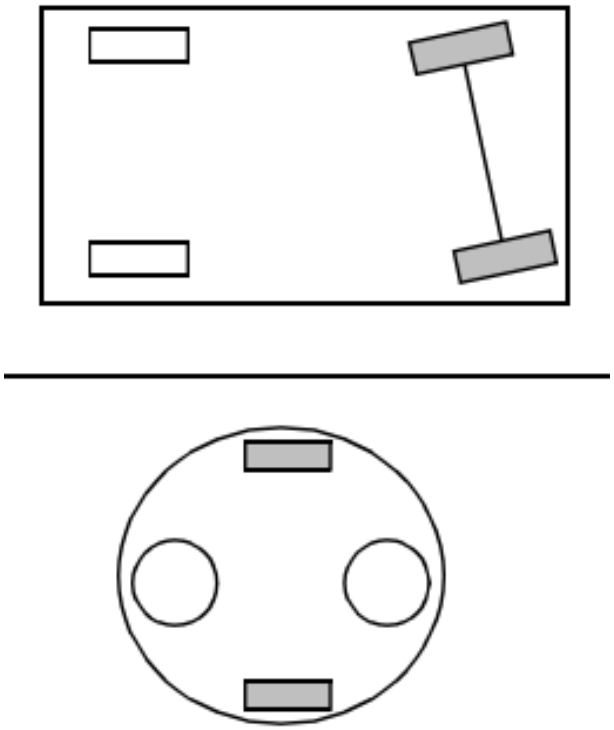


Figura 3: Dos tipos de configuración de ruedas bastante usados. En la parte superior de tipo dirección, y debajo la de tipo diferencial. Tomado de [2].

La capacidad de navegación del robot debe lograrse con un alto grado de exactitud antes de contemplar la planificación y seguimiento de trayectorias de manera autónoma. Este hecho se debe a que el sistema de control del robot debe tener una noción confiable de su estado de posición con respecto al sistema de coordenadas global. De no cumplirse este requerimiento, el robot no tendrá manera de interpretar el entorno real con respecto a la información que tiene del entorno [20].

2.2 Sistema de percepción

Para percibir el entorno, además de un sistema de odometría eficiente, se necesitan otros sensores que proporcionen información sobre diversas magnitudes físicas [21]. De esta manera, se pueden interpretar dichas entradas como sentidos que se habilitan en

el robot móvil. Uno de estos sentidos es la percepción de la velocidad y aceleración con respecto a los ejes del sistema de coordenadas del robot, el de leer la distancia entre el robot móvil y cada elemento a su alrededor [22]. Además, también existe el sentido de visión electrónica, que complementa a los anteriores mencionados, y añade la capacidad de interpretar características adicionales de los elementos del entorno como la morfología de los obstáculos y mejorar la toma de decisiones del robot [23].

2.2.1 LiDAR

El sensor detector de luz y rango (LiDAR, por sus siglas en inglés) es un dispositivo rotativo que posee un sensor de distancia de tipo infrarrojo. Como se observa en la Figura 4, cuando el receptor del sensor detecta el haz de luz emitido, se interpreta la distancia entre el sensor y el elemento más cercano en el campo de lectura del sensor. Al ser rotativo, el sensor puede leer un campo bidimensional alrededor del sensor, la distancia máxima que se puede detectar es una característica que depende del modelo utilizado [24]. Por lo tanto, con este sensor se puede generar un mapa en tiempo real muy similar al de un radar, donde se indique el estado del entorno cercano al sensor [25].

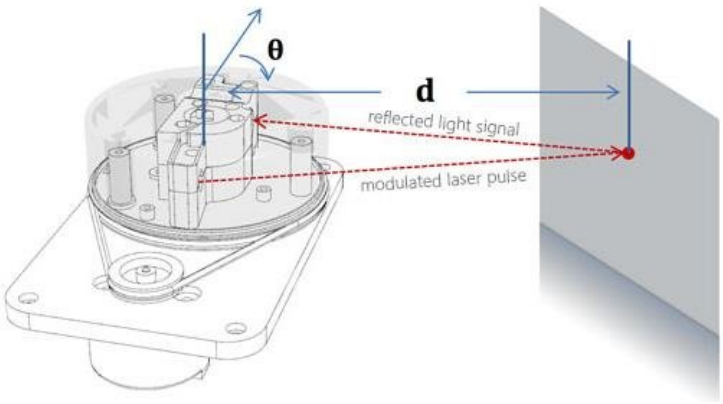


Figura 4: Funcionamiento del sensor RPLiDAR. Tomado de [3]

Al incorporar esta tecnología a un robot móvil, se puede usar para tomar decisiones

en tiempo real cuando se aproxime algún obstáculo dinámico, o un cambio en el mapa que se conoce con anterioridad. Cuando el robot puede usar esta capacidad para cambiar su trayectoria, se le conoce como planificación local, y es fundamental para que el robot se pueda adaptar a diferentes adversidades en su recorrido. El LiDAR es el sensor con mayor contribución en esta tarea [26].

Además de poder interpretar el entorno del robot en tiempo real, un LiDAR se puede usar para mapear un entorno por medio de distintos algoritmos de localización y mapeo simultáneos (SLAM, por sus siglas en inglés) [27]. Uno de ellos es Hector SLAM, que se encarga de interpretar el movimiento realizado por el robot móvil, para hacer un seguimiento del entorno leído dentro del campo de detección del sensor, porque durante el movimiento del robot, el algoritmo se encarga de generar un mapa con la información recabada [28]. Este mapa puede ser almacenado en la memoria del robot, y ser utilizado durante la navegación autónoma. El mapeo por Hector SLAM es especialmente confiable debido a que el algoritmo está leyendo las distancias exactas entre el sensor y el escenario y, como consecuencia la escala y resolución del mapa son las más adecuadas para el robot y el sistema de planificación de trayectorias [29].

2.2.2 Cámara

La cámara es un sensor para visión artificial presente incluso en investigaciones precursoras a los actuales robots autónomos, como el caso de Shakey de Stanford antes mencionado [1].

La cantidad y tipo de cámaras depende de la metodología de reconocimiento de escenario o entorno que se utilice. Por ejemplo, existen cámaras digitales simples o estereoscópicas [30]. Las primeras sirven para detectar objetos cercanos al robot y para realizar segmentación de las áreas importantes dentro del campo de visión [31], mientras que las del segundo tipo pueden hacer las mismas tareas, además de poder detectar profundidad [32]. Ambos tipos son ampliamente empleados actualmente y se

utilizan en sistemas robóticos que disponen de hardware de alto rendimiento mediante el uso de Unidades de Procesamiento Gráfico (GPU, por sus siglas en inglés). debido a que el procesamiento de imágenes digitales en tiempo real es una tarea que tiene una alta exigencia en el procesador [33]. Una cámara sencilla se puede usar para capturar imágenes del entorno del robot, y entrenar una red neuronal para detectar la presencia de obstáculos cercanos [31]. La red neuronal puede tomar un conjunto de imágenes capturadas y dar una respuesta en tiempo real que puede complementar la respuesta de otros sensores como el LiDAR [32].

2.2.3 Sensor inercial

Este sensor aporta información de aceleración lineal en los ejes X, Y y Z, además de velocidad angular en torno a los mismos [34]. Existe una nomenclatura de navegación estandarizada para hacer referencia a cada movimiento. Esta se utiliza para la navegación con distintos vehículos. En un sistema de coordenadas como el mostrado en la Figura 5, es posible definir la nomenclatura como "pitch" para el giro en torno al eje X, "roll" al giro en torno al eje Y, y "yaw" al giro dado en torno al eje Z [4].

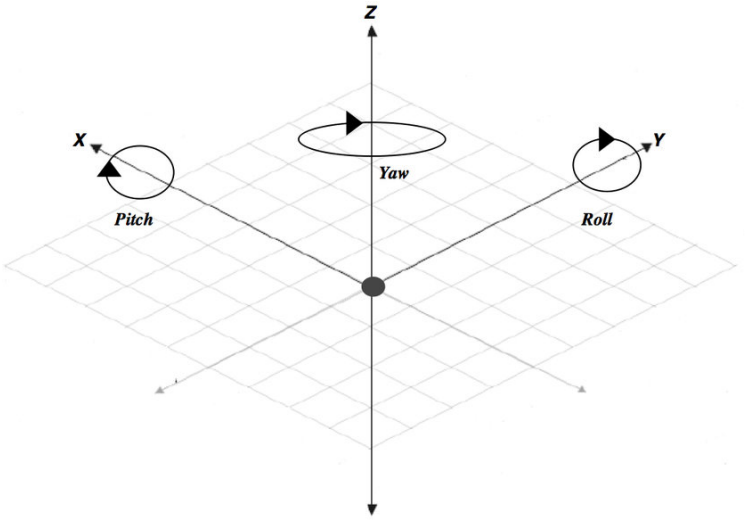


Figura 5: Nomenclatura usada para las velocidades angulares de los ejes del sistema de coordenadas del robot. Tomado de [4]

2.3 Entorno de ROS

El Sistema Operativo Robótico (ROS, por sus siglas en inglés) es ampliamente utilizado para todo tipo de proyectos de robótica y automatización debido a que simplifica la integración de los módulos del sistema de hardware, además de que es una plataforma de acceso y código abierto. ROS se utiliza principalmente en el sistema operativo Ubuntu, y la codificación de algoritmos e instrucciones se hace por medio de Python, C++, o incluso ambos, ofreciendo una amplia adaptabilidad y compatibilidad [35].

Existen diversas utilidades de software en ROS que pueden optimizar la comunicación en las etapas de lectura de sensores, procesamiento de señales, algoritmo de control y salida a etapa de potencia. A su vez, otra de sus principales ventajas es la posibilidad de ejecutar tareas de manera paralela, y acceder a las variables del sistema desde cualquier proceso en ejecución [36].

ROS cuenta con herramientas especialmente útiles para el desarrollo de robots móviles, como el paquete de transformación de sistemas de coordenadas "tf", con el cual se pueden definir todos los sistemas de coordenadas que componen al robot [37]. Uno de ellos es el sistema de coordenadas de la "base" del robot, que para un robot tipo diferencial está en el centro del mismo, justo en medio de las dos ruedas. Este sistema de coordenadas es el sistema "local" del robot, y determina el movimiento que puede tener el mismo dentro del mapa. Además, cada sensor montado en el robot debe tener su propio sistema de coordenadas asociado al sistema de coordenadas de la base, donde se indique exactamente su ubicación con respecto a este [38]. Esto ayuda a que el propio sistema operativo pueda interpretar cada señal sensada como perteneciente al mismo robot móvil, y que, por ejemplo, no haya conflictos al leer distancias si dicho sensor no está perfectamente centrado en el sistema de coordenadas de la base.

Un paquete específico para robots móviles es "navigation stack", que contiene diferentes métodos para navegación autónoma mediante algoritmos probabilísticos como el método AMCL, que funciona por medio un filtro de partículas para estimar el

camino más adecuado para llegar a un objetivo [26]. Su implementación requiere del ajuste de todos los sistemas de coordenadas, el uso de los sensores de percepción antes descritos, un sistema de odometría, y un control en la etapa de potencia, para mover el robot físicamente (ibid).

2.3.1 Fundamentos

ROS se compone de paquetes o proyectos que contienen diferentes códigos o librerías. Normalmente, se puede hacer uso de herramientas y utilidades como los paquetes antes mencionados para crear un proyecto base desde el cual partir para más adelante añadir todos los subsistemas más específicos que se requieran. También existen paquetes de código abierto que facilitan la comunicación con sensores, comunicaciones con dispositivos externos, y procesamiento de señales [39].

Como se ha dicho, los códigos se guardan en paquetes que se deben compilar antes de poderse ejecutar, de esta manera se asegura que se tengan instaladas las dependencias que sean necesarias, y que se mantenga actualizado el entorno de desarrollo de ROS y Ubuntu [35].

2.3.2 Nodos y tópicos para un AMR

ROS funciona con base en la codificación de instrucciones en "nodos". Cada tarea que se requiera ejecutar, se hará desde un nodo específico, con un nombre y una serie de requerimientos que pueden ser librerías, variables, datos, entre otros. Los nodos son una manera de organizar las tareas del sistema embebido, porque ayudan en el proceso de desarrollo a detectar problemas, fallas y hallar posibles alternativas cuando se está integrando el proyecto. Puede existir dependencia y comunicación entre nodos, por este motivo, es importante organizarlos adecuadamente, y monitorear su respuesta constantemente.

La manera de comunicar los nodos del sistema es por variables de ROS llamadas

"tópicos", que contienen valores de distinto tipo, desde datos de sensores, instrucciones para otros nodos o para las salidas a la etapa de potencia, resultados de operaciones, procesamiento de señales, comunicaciones con dispositivos externos, etc. Los tópicos pueden contener información en tipos de datos entero, flotante, texto, o conjuntos de datos previamente definidos. Los nodos de ROS crean y publican tópicos a los que se puede acceder desde otros nodos, para usar su información en procesamiento de señales de entrada, y para ejecutar algoritmos de control de navegación, entre [40] otras tareas.

Para poner en marcha un proyecto de ROS, primero se debe correr "roscore", que es un conjunto de paquetes y herramientas de software que permiten ejecutar el entorno base de trabajo de ROS. Después, se debe compilar el proyecto que contenga los códigos y paquetes necesarios en lenguaje Python o C++, y acto seguido, se puede empezar a ejecutar cada programa de manera independiente. Es posible crear un ejecutable donde se indiquen todos los programas que se desean correr simultáneamente, en el que se puede incluir, por ejemplo, los nodos que lean los sensores, el que calcula los movimientos de los sistemas de coordenadas y el que ejecuta el algoritmo de navegación, entre otros. Dicha herramienta son los ejecutables tipo "launch" (ibid).

2.3.3 Comunicaciones

Otra posibilidad que ofrece ROS, es la comunicación con otros dispositivos que también estén ejecutando un entorno de ROS. Un ejemplo de esto es la comunicación serial usando el paquete "rosserial", con el que se puede comunicar la computadora principal, con algún microcontrolador. Esto se puede hacer para acceder a tópicos que el microcontrolador esté publicando, y hacer uso de estos en el propio sistema embebido. Rosserial solamente requiere que se especifiquen los parámetros de comunicación serial como son puerto, tasa de baudios, bit de inicio y paro, etc. De esta manera, ROS puede interpretar el buffer de información recibida [41].

2.3.4 Aprendizaje máquina

El ML, es un conjunto de técnicas empleadas para solucionar problemas complejos para una máquina, emulando las características de un cerebro biológico, el cual requiere de un entenamiento para aprender a tomar decisiones con base en el entendimiento de patrones en la información observada y los resultados esperados [42]. Existen dos tipos principales de entenamiento en ML: el supervisado y el no supervisado.

Las técnicas de aprendizaje supervisado se dividen en clasificación o regresión. La clasificación se utiliza para agrupar datos, mientras que la regresión se utiliza para predecir el comportamiento de variables continuas. Una técnica de aprendizaje supervisado es *k vecino más próximo*, *árboles de decisiones* y *Máquinas de vectores de soporte*. Cualquiera que sea la técnica de aprendizaje supervisado, el éxito de su predicción depende del tipo, cantidad y comportamiento de sus features (ibid).

Para iniciar a desarrollar un sistema de aprendizaje máquina, es necesario saber el tipo de información que se va a manejar, el tipo de decisión que se desea, y la aplicación para la que se requiere esa información. Los siguientes son algunos de los pasos más importantes [43]:

1. Obtención/adquisición de los datos de entrada.
2. Preprocesamiento de datos y señales.
3. Identificar y seleccionar las características (features) de acuerdo con los datos preprocesados.
4. Entrenar distintos modelos con las características seleccionados.
5. Iterar para encontrar el mejor modelo.
6. Integrar el modelo entrenado para satisfacer las necesidades de la aplicación.

2.3.5 Redes Neuronales

Las Redes Neuronales (ANN, por sus siglas en inglés) fueron formuladas por McCulloch y Pitts, quienes identificaron las características que ayudan a un cerebro biológico para resolver problemas difíciles de plantear para un algoritmo convencional, como procesamiento de señales e información. La metodología de ANN requiere un procesamiento distribuido en múltiples capas de neuronas con funciones definidas. A pesar de lo innovadora que fue la propuesta hecha hacia mediados del siglo XX, tuvieron que pasar casi 20 años para que se realizaran aportaciones suficientemente relevantes. Para finales de la década de 1980, se empezaron a realizar publicaciones más frecuentes y aparecieron las primeras revistas científicas especializadas como Neural Networks de INNS, Neural Computation, así como IEEE Transactions on Neural Networks [44]. Durante los siguientes años se realizaron importantes aportaciones en hardware diseñado para aplicaciones con ANNs lo cual muestra el interés comercial por parte de importantes corporaciones. El principio fundamental de esta metodología es organizar una cantidad de unidades de procesamiento (neuronas) en capas. Cada neurona posee una función de transferencia para procesar la información de entrada, que se conecta mediante pesos. Los pesos son parámetros que se asignan a las entradas y se ajustan durante el proceso de aprendizaje. La activación de una neurona es la suma ponderada de sus entradas (ibid). De acuerdo con lo anterior, es posible organizar los elementos de la red de diferentes maneras. Algunas de las variaciones que se pueden hacer para definir diferentes ANNs son:

1. Diseñar un patrón de interconexión de neuronas.

2. Crear o aplicar un algoritmo para la actualización de los pesos.

3. Formular una función de activación para obtener una salida de una neurona de acuerdo con sus entradas y sus pesos.

Un tipo fundamental de ANN es el peceptrón, propuesto por Rosemblat en 1958, que

es un modelo construido principalmente para resolver problemas complejos a base de probabilidades, en contraposición con la lógica booleana empleada tradicionalmente por circuitos y computadoras [5]. Una variante de este modelo son las ANN completamente conectadas también llamadas perceptrón multicapa, mostrado en la Figura 6. Por otro lado, el perceptrón multi capa tiene capas ocultas y más funciones de activación para mejorar las prestaciones del modelo.

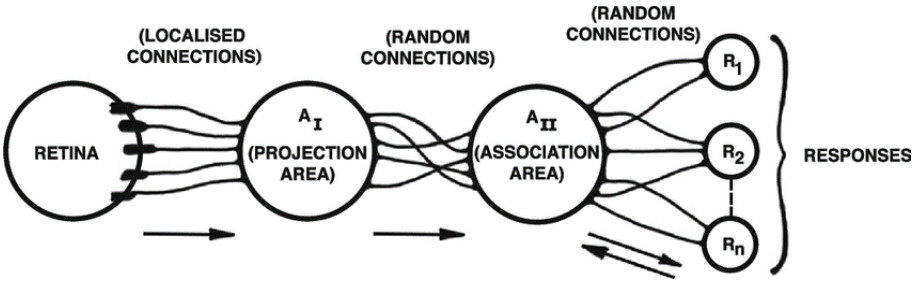


Figura 6: Arquitectura del perceptrón de Rosenblat. Tomado de [5].

Capítulo 3

Estado del arte

En esta sección se explican brevemente los trabajos de los autores más relevantes y las tendencias de investigación relacionadas con algoritmos de aprendizaje aplicados a robótica móvil en los últimos cinco años.

3.1 Trabajos relacionados

Lamini, C. ha publicado cuatro investigaciones sobre planificación de trayectorias de robots móviles utilizando varias metodologías, incluido el Aprendizaje Reforzado (RL, por sus siglas en inglés). Escribió un informe de investigación sobre Q-Learning aplicado a robots móviles autónomos colaborativos. Modificó la forma de actualizar las tablas de recompensa y recompensa esperada, permitiendo la posibilidad de transferir datos de aprendizaje entre múltiples agentes, lo que se denomina "navegación colaborativa"[45]. En los últimos años, Lamini y Benhlima C. han publicado otros tres trabajos sobre aprendizaje colaborativo y lógica difusa aplicada a robótica móvil. Están afiliados a la Faculté des Sciences de Meknès de Marruecos. Lamini tiene 224 citas y un índice h de 4.

Otro autor fundamental es Ruan, X. Ha trabajado en la mejora del Aprendizaje Reforzado Profundo (DRL, por sus siglas en inglés) aplicado a robótica móvil. DRL es una metodología que combina RL con Redes Neuronales para extraer características

esenciales directamente del sistema de percepción del agente o robot, incluyendo cámaras, sensores y más. El RL convencional, por el contrario, requiere la definición de características específicas utilizadas para actualizar las tablas o matrices de aprendizaje [46]. La diferencia crítica entre ambas metodologías es análoga a la diferencia entre Machine Learning y Deep Learning. Ruan también ha trabajado en estrategias de evasión de obstáculos junto con DRL. Utilizó un modelo DRL llamado D3QN, modificó el tamaño del lote de la tasa de aprendizaje e introdujo una función de normalización de capas para procesar las imágenes de la cámara del robot. Estos cambios reducen el tiempo del proceso de aprendizaje, lo que produce a una mejor reacción ante los obstáculos [46].

En 2022, se publicaron dos investigaciones sobre planificación de trayectorias de Vehículos Aéreos no Tripulados (UAV, por sus siglas en inglés) utilizando un algoritmo basado en Q-Learning . Una de ellas propone un planificador de trayectorias en un entorno, que puede estimar la ruta más eficiente desde cualquier posición en el mapa [47, 48, 49]. Es esencial señalar la importancia de encontrar una ruta o camino válido y elegir el de menor costo en términos de tiempo y consumo de energía.

3.2 Tendencias de investigación

Dado el contexto explicado en el Capítulo 1, la integración de las CNNs y RL con la planificación de trayectorias data de principios de la década de 1990 [50].

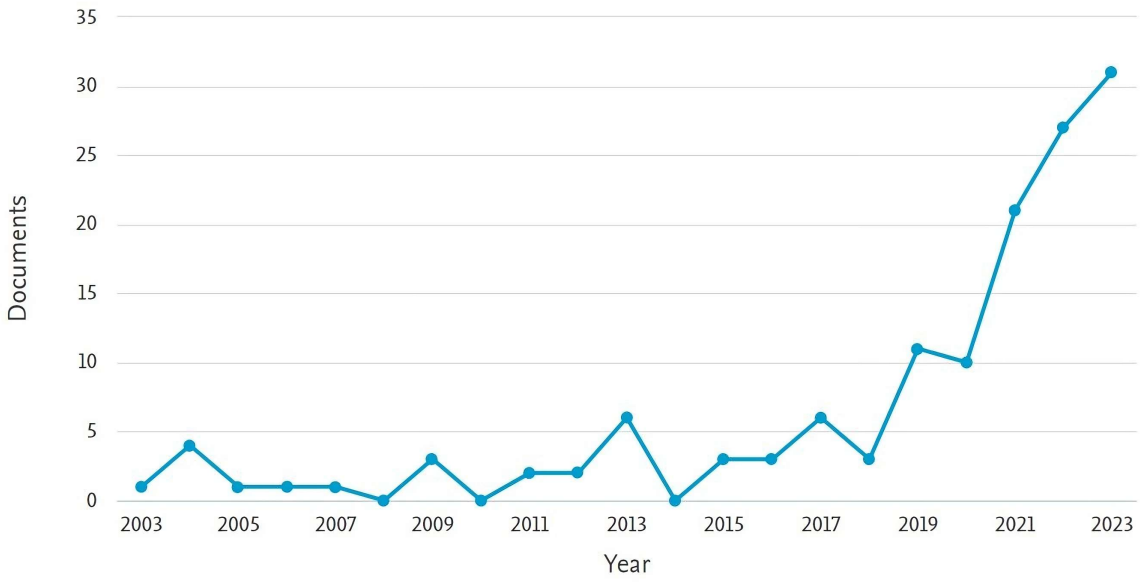


Figura 7: Tendencia de la publicación de documentos por año utilizando las palabras clave: Path Planning AND Autonomous Mobile Robot AND Reinforcement Learning, encontrados en Scopus.

Haciendo una comparación con el volumen de publicaciones en el campo de investigación *RL aplicado a AMRs* de los últimos diez años como se muestra en la Figura 7, está claro que ha habido un aumento considerable de interés desde 2019 hasta el presente. Podría decirse que la causa principal es la reciente popularidad de los sistemas de DL y la mejora del hardware disponible, incluidos mejores procesadores multinúcleo y utilidades de software con aquitectura abierta que cuentan con librerías para trabajar con ANNs y CNNs aprovechando el cómputo de alto rendimiento basado en GPU. La Tabla 1 describe el hardware que se ha usado en los últimos años.

La Figura 8 muestra las conexiones entre 786 palabras clave en los datos de Scopus. Se limita a las palabras clave con una ocurrencia mínima de 5, lo que permite que el mapa se lea mucho más fácilmente, conteniendo sólo las 40 palabras clave más relevantes y conectadas. Se generó utilizando la herramienta de código abierto VOSviewer [51]. El diagrama indica la proximidad entre las palabras clave en función de su frecuencia en la base de datos. La palabra clave *aprendizaje reforzado* tiene un total de 82 apariciones y está estrechamente relacionada con *evasión de obstáculos*, lo que significa que la

metodología de RL se aplica constantemente en esa tarea específicamente [46, 52, 12]. La segunda palabra clave más relacionada es *robots móviles*, con 63 apariciones. Esto incluye una amplia variedad de tipos de robots móviles, no sólo los AMR. Además, Q-learning es el único algoritmo presente en el diagrama, lo que sugiere que es el algoritmo de RL más abordado en ésta área. Por último, la palabra clave *planificación del movimiento* tiene 59 apariciones, la tercera más conectada. El análisis realizadoindica que el RL tiene un espectro de aplicaciones amplio, aparte de que actualmente se utiliza dicha metodología junto con distintas variantes de CNNs y ANNs para la planificación de trayectorias.

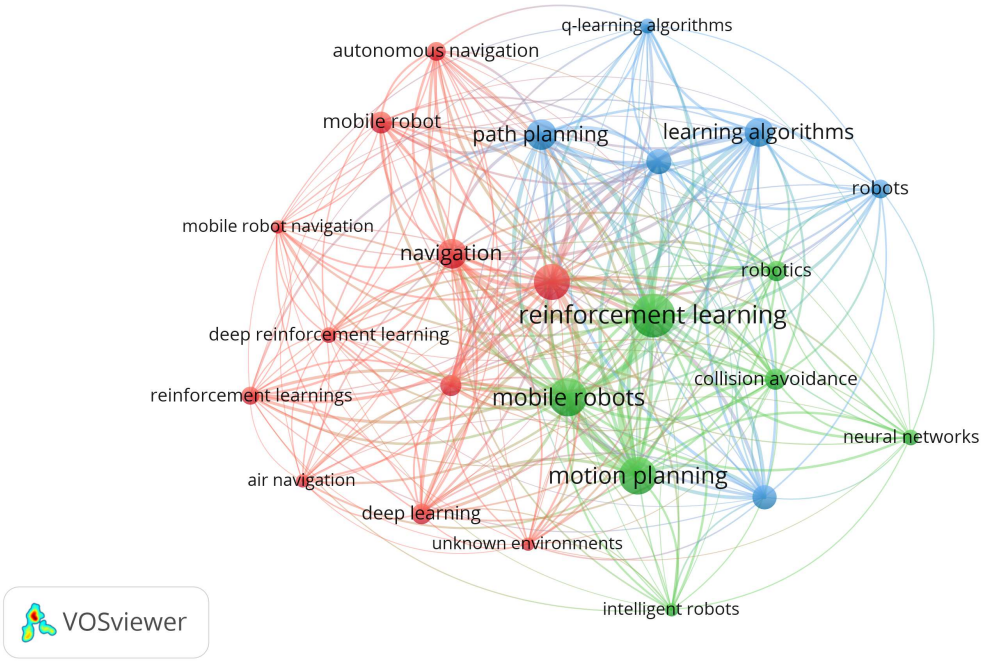


Figura 8: Mapa de VOSviewer de las conexiones entre las 40 palabras clave más relevantes encontradas en la base de datos obtenida en Scopus.

El país de origen de las publicaciones relacionadas más relevantes proporciona una noción de la prioridad que la robótica móvil tiene en dichas regiones, y se muestra en la Figura 9. Los países con un mayor interés en este tipo de tecnología y metodologías están dispuestos a invertir más fondos y esfuerzos para aplicar más algoritmos basados en aprendizaje a aplicaciones industriales reales, respaldados por un

beneficio económico. El país con una cantidad significativamente mayor de documentos en la base de datos Scopus es China. Se trata de un país que ha aportado algunos de los artículos más citados y relevantes de los últimos cinco años. Algunos de los temas de estas investigaciones son las mejoras de los algoritmos de RL que resuelven problemas de navegación global y de navegación en entornos desconocidos.

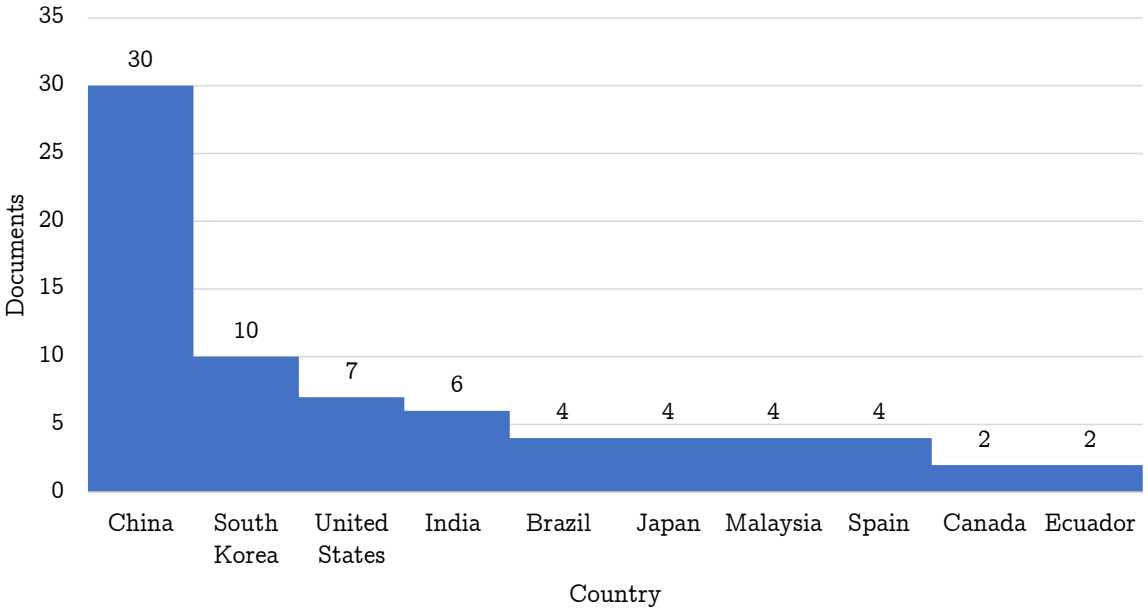


Figura 9: Países que han publicado el mayor número de investigaciones sobre RL aplicada a la planificación de trayectorias de AMRs, adquirido en Scopus.

Según la Figura 10, IEEE Robotics and Automation Letters es la revista que ha publicado más artículos sobre el tema en los últimos cinco años. Algunos de ellos son [53, 52, 54, 55, 56]. Otra revista con importantes contribuciones es Sensors de MDPI. En los últimos años, algunos artículos son [57, 58, 59]. Su principal objetivo es la planificación de trayectorias en entornos más amplios, obteniendo tablas de probabilidad más confiables [60].

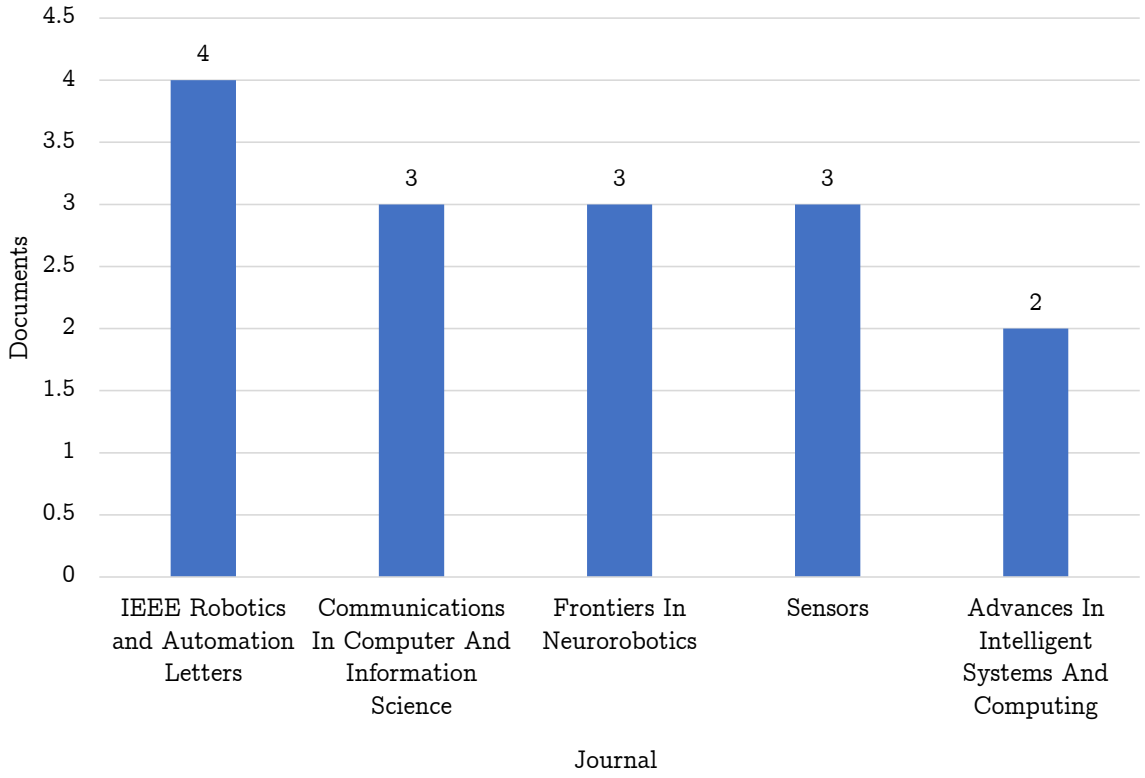


Figura 10: Revistas que han publicado la mayor cantidad de documentos relacionados con el tema, adquiridos de Scopus.

3.3 Comparación de hardware

Esta sección contiene un análisis del hardware que utilizan investigadores para implementar y probar los algoritmos de RL y DL. Según la Tabla 1, la mayoría de las investigaciones recientes aplican procesamiento basado en GPU, siendo las más populares NVIDIA RTX y GTX. Sin embargo, muchas simulaciones se están realizando en CPU, especialmente Intel Core desde i5 a i9. Los artículos que hacen pruebas en entornos y agentes reales; no simulados, utilizan mayoritariamente el sistema embebido Raspberry Pi 3, con comunicación remota a una computadora que ejecuta el algoritmo de aprendizaje. Según la revisión bibliográfica, la mayoría de los experimentos se limitan a simulaciones utilizando software como Gazebo de ROS[61]. Pocos artículos mencionan una implementación real de los algoritmos entrenados [60].

Tabla 1: Descripción del hardware utilizado en diferentes experimentaciones de AMRs con CNNs o RL.

Referencia	**Algoritmo/ modelo**	**Hardware**	**Resultados**	**Año**
Este trabajo	ResNet18 y YOLOv3 ejecutándose en ROS Melodic	NVIDIA Jetson NANO, procesador MPCore Quad-core, 128-core Maxwell 1098 MHz, 4 GB DDR4 RAM	El sistema de visión evita las colisiones con obstáculos dinámicos, alcanzando una exactitud del 98,5 %, una recuperación del 97 % y un F1-score del 98,5 %.	2024
Wang et al. [62]	QEA-Learning	CPU Intel(R) Core(TM) i5-7200, con 2,50 GHz y 2,71 GHz	El algoritmo QEA-Learning reduce la probabilidad de fallo al dar menos peso a la función de recompensa.	2023
Yang et al. [63]	DRL, Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO)	CPU Intel Xeron(R) E5-2650 v4 a 2,20 GHz	SAC obtuvo mejores resultados que PPO al encontrar una mayor recompensa máxima y un mejor equilibrio entre exploración y explotación.	2022
Ruan et al. [46]	DRL, Dobule Q-Network (D3QN)	GPU NVIDIA GTX 3080, sistema operativo para robots (ROS)	La función de pérdida se estabiliza tras 1000 pruebas de entrenamiento	2022
De Carvalho et al. [47]	Q-Learning	16 GB de RAM, Intel Core i7-7700	Tiempo de procesamiento inferior a 2 minutos para mapas de celdas de 20x20x5	2022
[64] Yeom [64]	DRL	Raspberry Pi 3, Quad Core 1.2 GHz, CPU Intel 4-core i5 7500, 3.80 GHz, 32 GB RAM	DRL redujo la longitud del camino en aproximadamente un 20 %, en comparación con el enfoque tradicional de ventana dinámica (DWA)	2021
Hu et al. [65]	Deep Deterministic Policy Gradient (DDPG), Prioritized Experience Replay (PER)	Raspberry Pi 3, GPU NVIDIA GTX 1080, CPU Intel Core i9 con 2,9 GHz	El algoritmo realizó 20 misiones con éxito y requirió 40 minutos de tiempo de entrenamiento	2020
Xiang et al. [66]	DRL*, Soft Actor-Critic (SAC)	GPU NVIDIA GeForce RTX 2070, AMD Ryzen 7 2700X de ocho núcleos a 3,70 GHz, 16 GB de RAM, ROS	El modelo se comparó con un gmapping de navegación tradicional, alcanzando resultados competitivos tras pocas horas de entrenamiento	2019

Capítulo 4

Materiales y métodos

En esta sección se describe la metodología de construcción de la estructura del robot, así como la selección de todos los materiales utilizados. La configuración de los componentes se ha organizado para permitir futuras modificaciones y facilitar el mantenimiento. La fuente de alimentación y las baterías están separadas en dos partes para poder inspeccionar los módulos individualmente y por motivos de seguridad.

Además, el prototipo está construido en vertical, montando los módulos unos encima de otros, como se ve en la Figura 20. Los sensores seleccionados como los encoders, el LiDAR e IMU, se incluyen por su sencillez y fiabilidad. Para aumentar la precisión en la navegación y reducir el error de movimiento, podrían añadirse otros sensores, o integrar versiones mejoradas de los ya presentes.

4.1 Selección de hardware

El robot propuesto se basa en el kit de desarrollo JetBot ROS AI de Waveshare. Se mantienen los principales componentes de hardware, sin embargo, tiene notorias modificaciones estructurales, se modifica el módulo de alimentación para permitir la adición de una segunda batería dedicada exclusivamente a suministrar energía a los dos motores y al resto de sensores conectados al microcontrolador.

La Figura 11 ilustra las conexiones y comunicaciones entre los módulos del robot

comenzando por las baterías a la izquierda, el sistema embebido en el centro, y sus sensores de entrada y salidas indicadas por la dirección de las flechas. Esta sección enumera los dispositivos de hardware presentes en el robot organizado como se ve en la Figura 11.

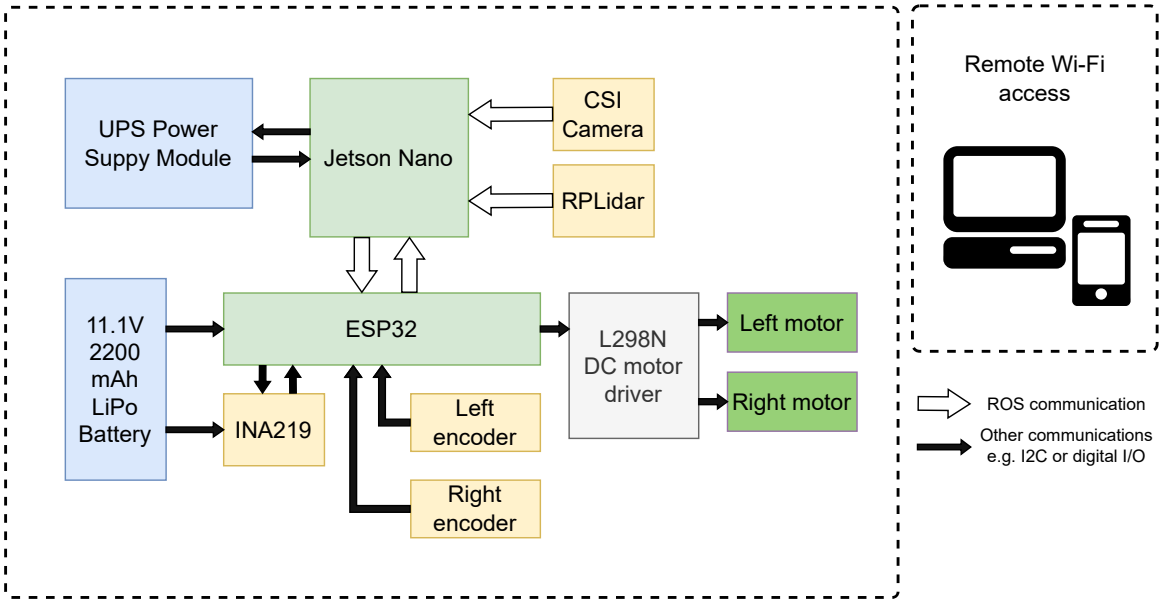


Figura 11: Diagrama a bloques que describe los componentes del robot propuesto y su comunicación remota por WI-FI.

- **Jetson Nano** Este sistema embebido es el dispositivo que funciona como el procesador central que realiza todos los cálculos importantes, el entrenamiento de DL y la comunicación remota a través del sistema operativo ROS. Jetson Nano cuenta con una GPU con 128 CUDA Cores, lo que permite al robot ejecutar el reconocimiento de objetos en tiempo real a través de un modelo de DL. Esa independencia cognitiva es fundamental para alcanzar un correcto nivel de autonomía del robot.

- **ESP32** Cuenta con un microprocesador de doble núcleo y 32 bits, con un rendimiento superior al de otros microcontroladores de bajo costo. El sistema embebido establece una comunicación serie con este dispositivo, ya que lee

continuamente los dos encoders y otros sensores, y proporciona una salida PWM para los dos motores de corriente directa (CD). Otra característica importante del ESP32 es su sencillez de conectividad, ya que dispone de comunicaciones Bluetooth y Wi-Fi, lo que resulta especialmente útil para el control remoto del robot. Sin embargo, la comunicación con Jetson Nano es serial, a través de un puerto USB del sistema embebido.

- **Motores**

  Los motores en el kit de desarrollo JetBot son motores regulares de 12 VDC con escobillas acoplados a una caja reductora con una relación de engranajes de 1:30. Esto proporciona suficiente torque para que el robot se mueva con suavidad en la mayoría de los entornos interiores. La locomoción del robot es diferencial y requiere dos ruedas controlables y dos ruedas de pivote no controlables. Ese diseño proporciona mayor estabilidad al robot que una sola rueda giratoria. Sin embargo, la fricción añadida por el contacto entre las ruedas y la superficie del terreno, podría dificultar el movimiento en determinadas situaciones.

- **Sensores**

  Cada motor de corriente directa tiene un encoder de 2 canales montado directamente en el eje. Dadas las propiedades de la caja de cambios descrita anteriormente, el encoder se utiliza para leer la posición y la velocidad del robot inferida del movimiento local de las ruedas izquierda y derecha. Cada encoder proporciona 330 pulsos por revolución. Al disponer de dos canales separados para leer los pulsos, es posible seguir la dirección del movimiento local de las dos ruedas. Sin embargo, el sensor más apropiado para estimar la posición del robot es el LiDAR. En esta experimentación se utiliza el modelo RPLiDAR del fabricante Slamtec (Figura 4), que tiene un rango máximo de 16 metros. La lectura se hace

con una frecuencia de muestreo de 4000 muestras por segundo.

Como se abordón en el Capítulo 3, se trata de un dispositivo rotativo que emite un haz de luz infrarroja leído por un sensor para estimar la distancia entre el sensor y su entorno.

- **Cámara**

  La cámara incluida en el kit prototipo es una cámara CSI IMX219-160 para Jetson Nano. Tiene 8 megapixeles de resolución y un ángulo de visión de 160°.

- **Baterías**

  Como ya se ha explicado brevemente, el módulo de baterías se ha modificado para aumentar el nivel de autonomía del robot. El módulo de baterías original del kit JetBot ROS AI utiliza tres baterías 18650 recargables de 3,7 V, 9900 mAh en serie que aportan energía a todo el hardware, incluyendo el sistema embebido junto con el microcontrolador y la etapa de potencia. No obstante, trabajar con una sola batería podría causar posibles dificultades, sobre todo porque limita la autonomía del robot, es decir, su capacidad para mantenerse alimentado durante un largo periodo de tiempo. Según la información del fabricante, cualquier manipulación incorrecta de la batería puede provocar un fallo en el sistema embebido. Por lo tanto, es necesario garantizar la seguridad de los módulos en la Figura 11, además de que se debe mejorar la autonomía del prototipo para evitar tener una autonomía limitada.

  El módulo de alimentación ininterrumpida (UPS) proporciona energía a Jetson Nano. Incluye cuatro baterías 18650 y sus respectivos circuitos de protección y monitoreo. La comunicación entre el módulo de batería UPS y Jetson Nano se realiza a través del protocolo I2C. La segunda batería es una batería LiPo de 12 V y 2200 mAh. La cual es adecuada para diferentes robots autónomos. Esta

batería ayuda a aislar el sistema de control del resto del hardware. Además de la seguridad, es una solución fiable a posibles dificultades de comunicación, y alarga el tiempo de trabajo del sistema embebido, que se comunica constantemente de forma remota.

4.2 Selección de software

Esta sección se centra en el método propuesto para hacer que el robot interprete e interactúe con su entorno a través de comandos desde el sistema embebido. Las complejas tareas implicadas, como la visión artificial, reconocimiento de objetos, mapeado SLAM y el control del movimiento, son ejecutadas por nodos de ROS independientes. Esto facilita la modificación y depuración de cada tarea. Además, algunas de ellas son necesarias en situaciones específicas, por lo que el robot puede trabajar con los módulos estrictamente necesarios para evitar un gasto indeseado de energía y tiempo.

4.2.1 Algoritmo de planificaci[on de trayectorias]

En el presente trabajo, se opta por utilizar el algoritmo de localización AMCL, que opera mediante un filtro de partículas empleando una configuración predefinida para el procedimiento de muestreo y asignando pesos a las partículas individuales. Cuando se le proporciona un mapa conocido, el algoritmo AMCL estima la pose del robot a medida que lo recorre y recoge datos del entorno a través de sensores de rango y odometría. Cada partícula del algoritmo representa una posible posición del robot, reflejando así la distribución de estados factibles. AMCL requiere menos memoria que otros métodos, ya que su uso se correlaciona directamente con el número de partículas y se mantiene constante independientemente de la ampliación del tamaño del mapa. En AMCL, el principal factor que influye en la eficacia de la localización es la cantidad de partículas.

En este trabajo se establece el número mínimo de partículas en 500 y un máximo de 5000, que se adapta logrando un equilibrio favorable entre la precisión y velocidad en la navegación.

Además, los valores por defecto para otros parámetros de control se adoptan del paquete de navegación ROS [67]. Debido a la configuración de navegación que favorece el mapa construido sobre los datos del LiDAR, el AMR propuesto percibe los objetos dentro del mapa. En consecuencia, el planificador de trayectorias genera una ruta más larga libre de colisiones para evadir los obstáculos.

4.2.2 Control de velocidad

La Figura 12 representa el control adicional para ajustar la señal de salida de velocidad del AMR propuesto que hace que su base se mueva. Dicho control es diferente del movimiento deseado o predicho, en cuanto a que éste contempla variables físicas que podrían interferir con el objetivo, como un aumento imprevisto de la fricción con el terreno, estado de carga de las baterías que puede causar un aumento o disminución de velocidad considerable que se ve reflejado como una dificultad para estabilizar la orientación del robot.

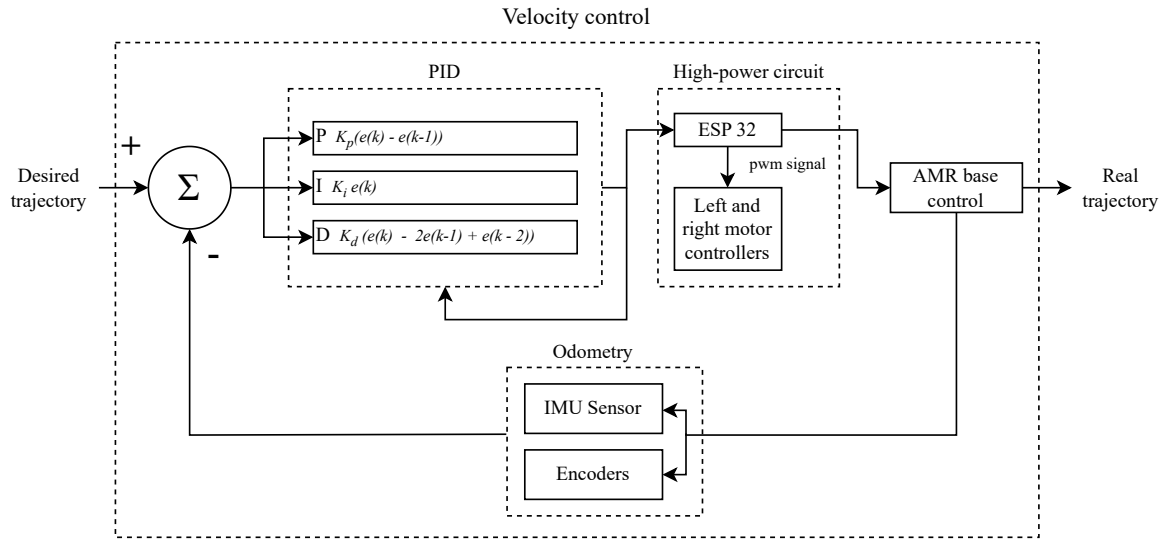


Figura 12: Control utilizado para ajustar la señal de velocidad de salida en función de la interacción del AMR con el entorno. Basado en [6].

4.2.3 Control de la base del AMR

El sistema de ROS se ejecuta en Jetson Nano y es el controlador central, en el cual se toman todas las decisiones importantes y puede acceder a todas las variables (tópicos de ROS). Dado que el microcontrolador ESP32 que interactúa con las entradas y salidas es un módulo independiente, debe comunicarse con Jetson Nano a través del puerto serie utilizando el nodo *rosserial*.

La Figura 13 muestra la ubicación del nodo *rosserial* y sus tópicos. Estos tópicos se publican y, por tanto, se leen desde el sistema embebido. El nodo *teleop* genera el comando de velocidad durante la teleoperación, que es el control remoto del robot a través de Wi-Fi para mover la base durante el mapeo y configuración. El tópico básico que controla el movimiento del robot es *cmd_vel*, un tópico genérico de ROS que contiene un comando de giro para el robot en el sistema de coordenadas global. Incluye comandos de movimiento para los ejes globales X, Y y Z, y rotación alrededor de cada uno de ellos, resultando un total de 6 ejes (tema abordado en el Capítulo 2). Dado que JetBot es un robot de tipo diferencial que se mueve sobre una superficie

plana en espacios interiores, no se mueve a lo largo del eje Z, y gira sólo alrededor del eje Z y no X ni Y. El comando de velocidad global se lee en el microcontrolador y se procesa para generar una señal PWM enviada a la etapa de potencia que son los controladores de los motores, causando en última instancia un movimiento del chasis del robot. Cuando el sistema de planificación se está ejecutando, el *cmd_vel* se genera a partir de su nodo ROS llamado *amcl*.

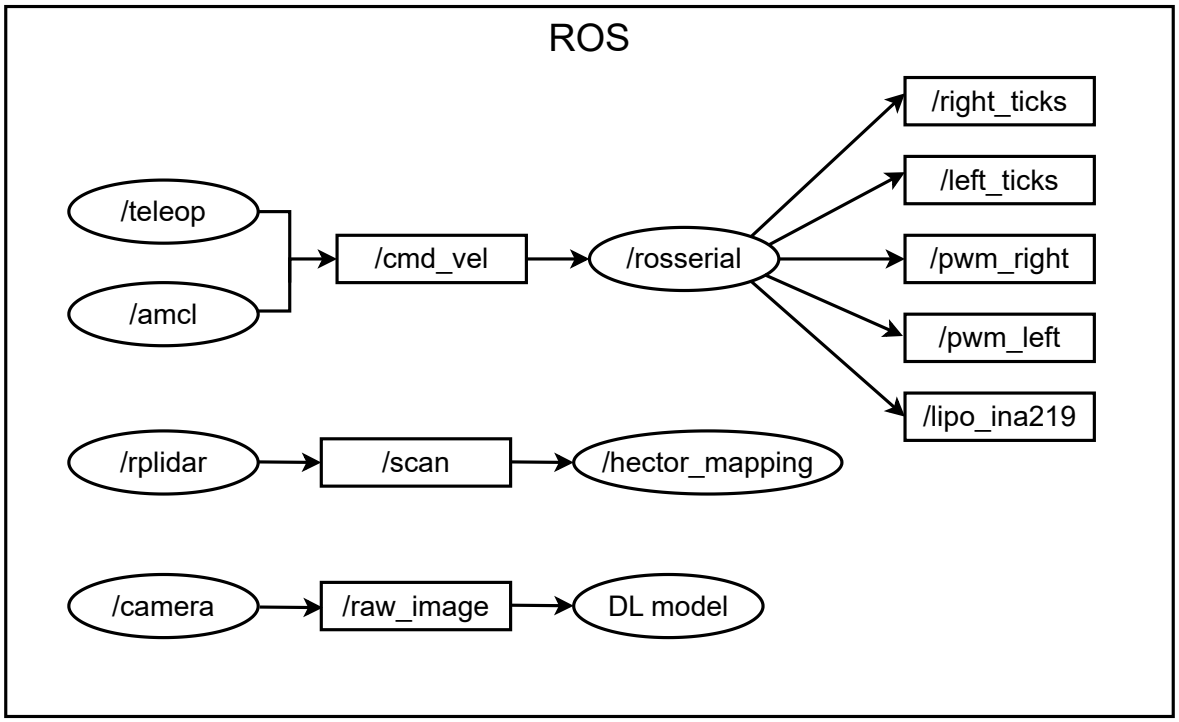


Figura 13: Nodos ROS (óvalos), y sus respectivos tópicos (rectángulos)

.

En la Figura 13 muestra una representación gráfica de los nodos ROS que se ejecutan cuando el robot se mueve de forma autónoma. Las señales PWM *pwm_derecha* y *pwm_izquierda* son el resultado del procesamiento del comando de movimiento por parte del microcontrolador, eso requiere convertirlo en términos del sistema de coordenadas global (en los ejes X, Y y giros alrededor de Z) al sistema de coordenadas local del robot (movimiento en el eje X y giro alrededor de Z). Esa conversión requiere un algoritmo para equilibrar las variaciones de velocidad de los dos motores. Las señales

de los codificadores se leen directamente en las entradas digitales de interrupción del microcontrolador ESP32. Esos ticks se acumulan en variables locales y se publican a través de rosserial. Cada encoder tiene dos canales, por lo que los contadores también pueden disminuir para añadir la posibilidad de seguir la posición exacta de cada rueda diferencial, esto en el sistema de odometría. El seguimiento de la batería LiPo se realiza leyendo el sensor de corriente INA219 a través del protocolo I2C. Luego, esos datos se comparten con Jetson Nano como un tópico de ROS.

4.2.4 Mapeo del entorno

En la Figura 13 se incluye el nodo RPLidar, que funciona mediante comunicación serie entre el dispositivo y Jetson Nano. El sensor escanea el entorno, y el nodo *hector_mapping* genera un mapa basado en el movimiento del motor con un punto de partida como referencia. Para crear un mapa se ejecuta el algoritmo de mapeo mientras el robot se mueve utilizando el nodo textit/teleop. El paquete ROS del algoritmo Hector SLAM permite generar un mapa en formatos *yaml* y *geotiff*. Es importante mencionar que el proceso de mapeo puede ser monitoreado utilizando la herramienta *ROS Rviz*. La velocidad del robot es de aproximadamente 0,05 m/s durante el proceso de mapeo.

4.2.5 Visión artificial

El modelo de DL propuesto para integrar la visión artificial en el robot es YOLO V3, y su arquitectura se muestra en la Figura 14. Este modelo es una CNN para la detección de objetos y fue seleccionada por su buen desempeño y compatibilidad con Jetson Nano.

	Type	Size	Output
	Convolutional	3 x 3	256 x 256
	Convolutional	3 x 3 / 2	128 x 128
1x	Convolutional	1 x 1	
	Convolutional	3 x 3	
	Residual		128 x 128
	Convolutional	3 x 3 / 2	64 x 64
2x	Convolutional	1 x 1	
	Convolutional	3 x 3	
	Residual		64 x 64
	Convolutional	3 x 3 / 2	32 x 32
8x	Convolutional	1 x 1	
	Convolutional	3 x 3	
	Residual		32 x 32
	Convolutional	3 x 3 / 2	16 x 16
8x	Convolutional	1 x 1	
	Convolutional	3 x 3	
	Residual		16 x 16
	Convolutional	3 x 3 / 2	8 x 8
4x	Convolutional	1 x 1	
	Convolutional	3 x 3	
	Residual		8 x 8
	Avgpool	Global	
	Connected	1000	
	Softmax		

Figura 14: Estructura de YOLO V3, también conocido como Darknet-53, tomado de [7].

.

Tras instalar y configurar un modelo entrenado con sus respectivos pesos, el algoritmo puede reconocer objetos capturados desde la cámara integrada del robot. A continuación, puede aplicarse para reconocer posibles obstáculos en la trayectoria cercana del robot. Dado el número de clases cubiertas en el entrenamiento de YOLO V3, el robot puede interpretar objetos específicos e ignorar deliberadamente algunos de ellos. En el tercio inferior de la Figura 13, YOLO V3 se representa como un nodo de ROS que trabaja simultáneamente con el resto del sistema de control.

La Figura 15 representa la arquitectura de la CNN ResNet18, que se utiliza para la percepción de escenas, seleccionada por su rendimiento en pruebas previas de visión artificial en tiempo real en el sistema embebido. Además, este modelo DL está

disponible en la versión del firmware del robot propuesto JetBot ROS AI [68].

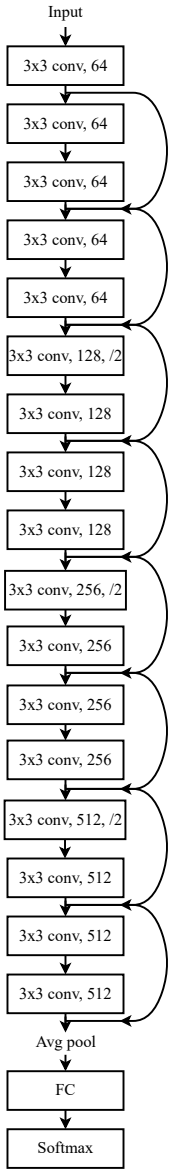


Figura 15: Arquitectura de ResNet18 CNN, tomada de [8].

Capítulo 5

Resultados experimentales

Los resultados que se muestran a continuación se dividen en hardware y software porque hay información relevante para ambos grupos; por lo tanto, separarlos ayuda a definir las diversas funciones del rendimiento del robot.

5.1 Desarrollo de dispositivo

La Figura 16 muestra la vista inferior del prototipo, y también se indican sus dimensiones y la ubicación de las cuatro ruedas. Las características mecánicas del mismo están dadas por el fabricante del prototipo comercial. La configuración diferencial sugiere que el centro del chasis es el origen del robot, que debe tomarse en cuenta para los cálculos de localización. Dicho centro es también el origen del sistema de coordenadas del robot, por lo que el eje Z *atraviesa* ortogonalmente la base del robot en su centro, y el eje X longitudinalmente (véase la Sección 5.1.1).

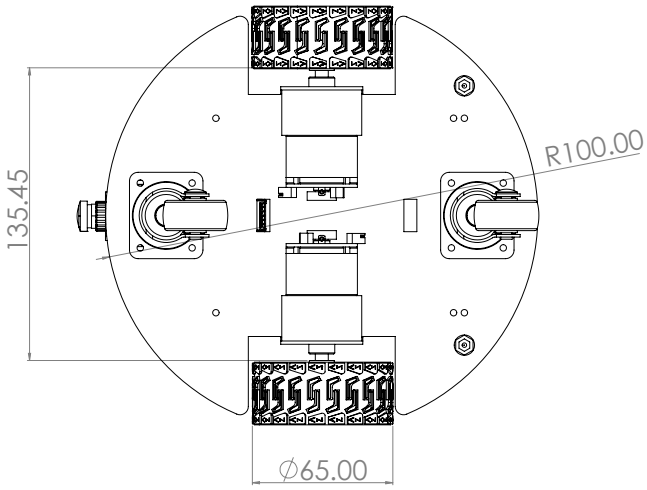


Figura 16: Vista inferior del robot (las unidades están en milímetros)

.

5.1.1 Modelo cinemático

La Figura 17 muestra el sistema de coordenadas global, también denominado inercial, y se representa como X_I y Y_I. Consiste en los ejes que establecen el entorno o el mapa en el que opera el robot. Dado que el robot móvil diferencial se mueve en un espacio bidimensional, la forma correcta de analizar su movimiento es con una matriz de rotación bidimensional que define todas las características de la velocidad lineal $v(t)$ y la velocidad angular $\omega(t)$ del robot, mostradas en la Figura 18.

El sistema de coordenadas local (X_R, Y_R) se define para analizar el movimiento producido por las dos ruedas del robot en relación con el punto central del chasis P, que se considera como un punto de referencia de posición en el marco de coordenadas global (X_I, Y_I), ilustrado en la Figura 17. Con este otro sistema de coordenadas, es posible separar el movimiento de cada rueda producido por su velocidad angular $\dot{\phi}_r$ y $\dot{\phi}_l$ y relacionarlos con el movimiento del robot en el sistema de coordenadas global (X_I, Y_I).

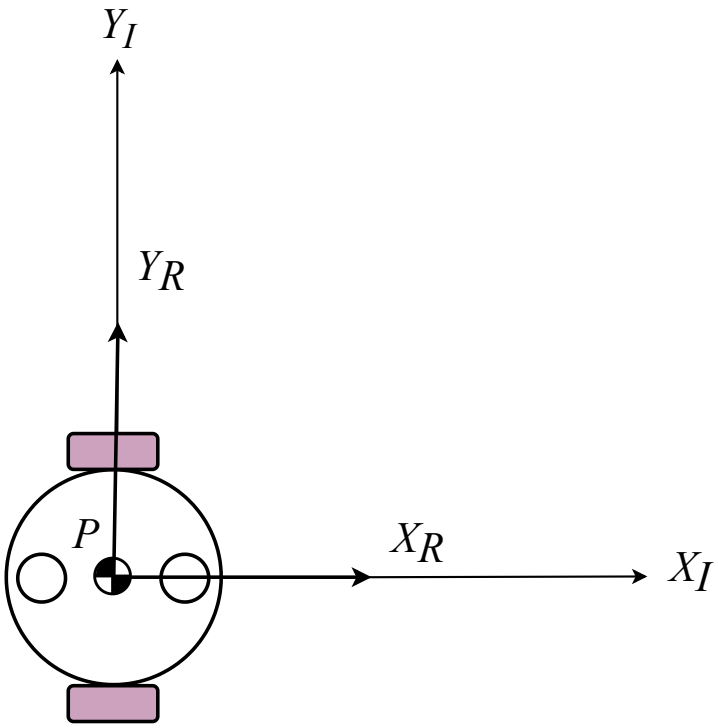


Figura 17: Representación gráfica de los sistemas de coordenadas globales y locales del robot móvil.

Suponiendo que el robot comienza a moverse desde la posición representada en la Figura 17, con orientación θ medida en sentido antihorario desde el eje X_I, la matriz de rotación se define como se presenta en la Ec. 1.

$$R(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

donde $R(\theta)$ es la matriz de rotación.

Considerando que el robot móvil interactúa con el entorno a través de sus dos ruedas motrices, su trayectoria se describe mediante un modelo cinemático que relaciona el vector de movimiento $\dot{\xi}_R$ del robot en el sistema de coordenadas local con el vector de movimiento $\dot{\xi}_I$ en el sistema de coordenadas global (Ec. 2).

$$\dot{\xi}_I = R(\theta)\dot{\xi}_R. \tag{2}$$

Donde:

$\dot{\xi}_I$ es la velocidad del robot con respecto al sistema de coordenadas global (X_I, Y_I), y

$\dot{\xi}_R$ es la velocidad del robot con respecto al sistema de coordenadas local (X_R, Y_R).

El modelo cinemático del robot móvil de configuración diferencial está definido por una matriz de rotación multiplicada por el vector de velocidad local del robot, que incluye la velocidad lineal $v(t)$ y la velocidad angular $\omega(t)$.

La incapacidad del robot para moverse a lo largo del eje Y_R se debe a que sus ruedas están orientadas perpendicularmente a este, como se ve en la Figura 17. Esto da lugar a que el componente y del vector velocidad en la Ec. 3 se anule.

$$\dot{\xi}_I = R(\theta)\dot{\xi}_R = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ 0 \\ \omega(t) \end{bmatrix}. \tag{3}$$

Donde:

$v(t)$ es la velocidad lineal en el eje local X_R del robot, y

$\omega(t)$ es la velocidad angular en torno al eje local Z del robot.

La velocidad lineal $v(t)$ y la velocidad angular $\omega(t)$ se expresa en relación con las medidas específicas del robot y las velocidades angulares de las ruedas $\dot{\phi}_r$ y $\dot{\phi}_l$, como se ve en la Ec. 4 y Ec. 5.

$$v(t) = \frac{r}{2}\left(\dot{\phi}_r + \dot{\phi}_l\right). \tag{4}$$

$$\omega(t) = \frac{r}{l}\left(\dot{\phi}_r - \dot{\phi}_l\right). \tag{5}$$

Donde:

r es el radio de las ruedas izquierda y derecha del robot,

l es la distancia entre ruedas,

$\dot{\phi}_r$ es la velocidad angular de la rueda derecha, y

$\dot{\phi}_l$ es la velocidad angular de la rueda izquierda.

La Ec. 6 resulta de resolver el producto en la Ec. 3, y sustituir $v(t)$ y $\omega(t)$ con la Ec. 4 y la Ec. 5. Esa ecuación ayuda a relacionar el movimiento del robot en el sistema de coordenadas global (el entorno) y el movimiento de las dos ruedas.

$$\dot{\xi}_I = \begin{bmatrix} \frac{r(\dot{\phi}_r+\dot{\phi}_l)}{2}\cos\theta \\ \frac{r(\dot{\phi}_r+\dot{\phi}_l)}{2}\sin\theta \\ \frac{r(\dot{\phi}_r-\dot{\phi}_l)}{l} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}. \tag{6}$$

Donde:

$\dot{x}$ es la velocidad lineal en el eje global X_I,

$\dot{y}$ es la velocidad lineal en el eje global Y_I, y

$\dot{\theta}$ es la velocidad angular en torno al eje global Z_I.

La Figura 18 muestra la dirección de las velocidades lineal y angular $v(t)$ y $\omega(t)$, y los parámetros del robot r, l, y θ descritos anteriormente [2].

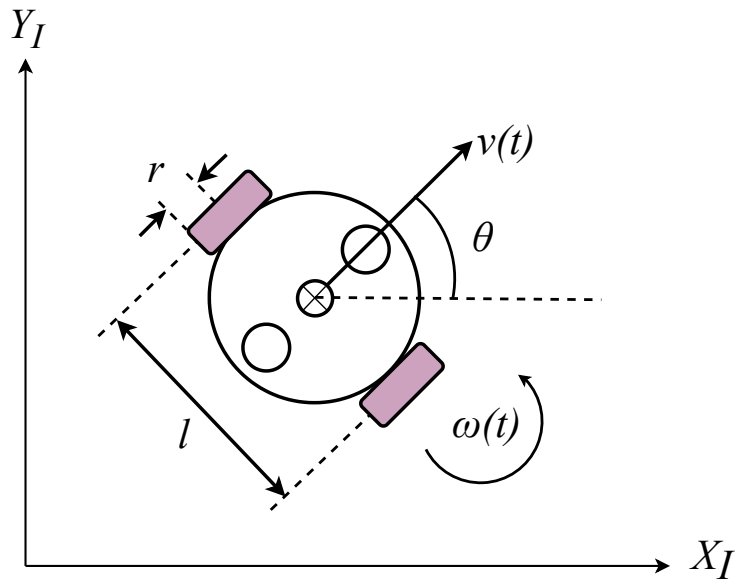


Figura 18: Representación gráfica de las variables del modelo cinemático.

5.2 Implementación del hardware

Se realizaron pruebas a los sensores para conocer sus propiedades y familiarizarse con el tipo de datos que se involucran durante las comunicaciones entre el sistema embebido y el microcontrolador. Ejemplo de ello son los encoders izquierdo y derecho, que generan pulsos para cada rueda durante el giro de estas, y como se observa en la Figura 19, los canales del encoder están desfasados para interpretar el sentido de giro de las ruedas. También con esta medición se pudo observar la aceleración del motor con base en la frecuencia de los pulsos leídos. El encoder tiene una resolución de 330 pulsos por revolución, información con la cual se interpreta la odometría del robot móvil.

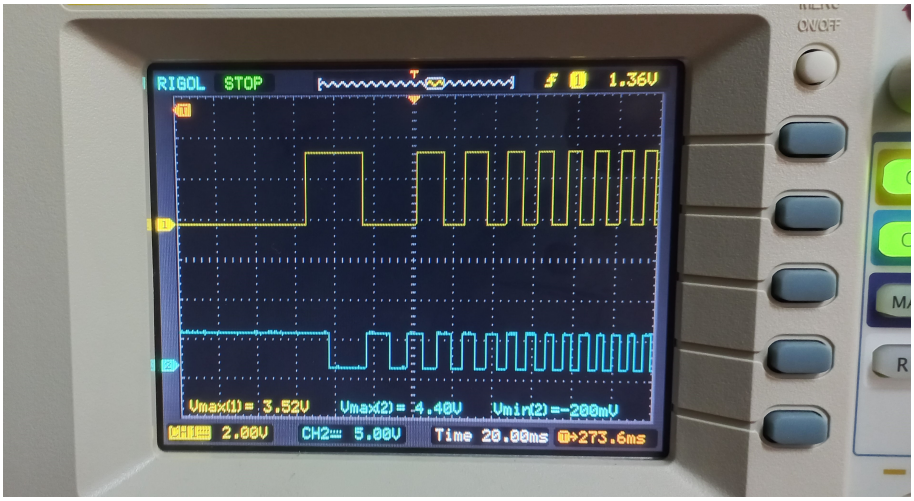
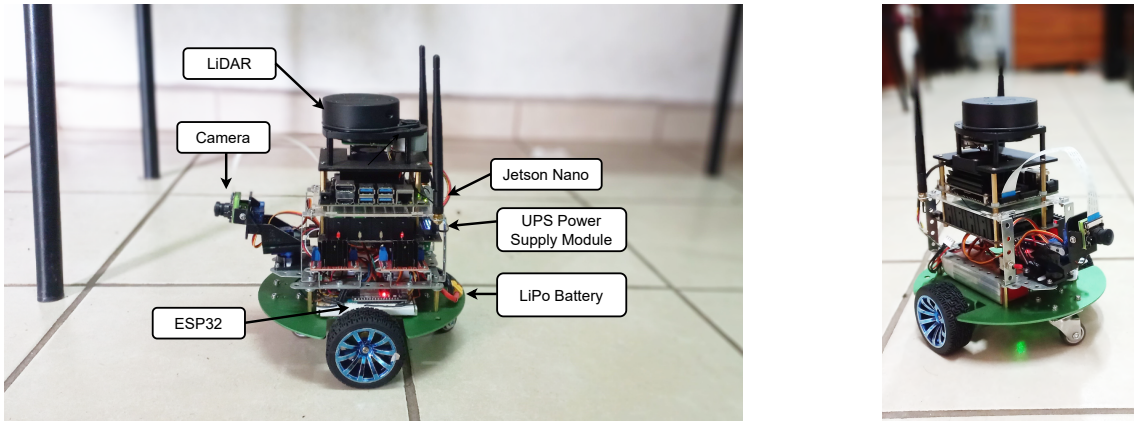


Figura 19: Lectura de los dos canales de un encoder por medio de un osciloscopio. El canal A corresponde al color amarillo, mientras que el canal B al color azul.

La Figura 20a es una vista lateral del robot en el laboratorio donde tuvo lugar la experimentación, y la Figura 20b es una vista frontal que muestra algunas de las modificaciones realizadas en el prototipo base. Dentro de esa figura, empezando por la parte inferior se encuentran el sistema de locomoción, los dispositivos de control y el LiDAR en la parte superior. En la parte delantera se encuentra la cámara digital. La disposición actual de los dispositivos de hardware es el resultado de múltiples ajustes durante el proceso de desarrollo y es tolerante a nuevas mejoras. Una de las mejoras es la actualización del módulo de baterías, que originalmente contaba con un módulo de alimentación de tres baterías. Se sustituyó por un módulo de cuatro baterías para el sistema embebido y los sensores, y una batería LiPo adicional de 12 V para el sistema de locomoción con los motores de corriente directa. Esa modificación del sistema de alimentación se debe a la importancia de la autonomía de las baterías. Otra modificación es la adición de un microcontrolador ESP32 para generar las señales PWM de la etapa de potencia. Eso simplifica la adición de múltiples sensores, como los codificadores del motor y el sensor del monitor de corriente de la batería LiPo.



(a) Vista lateral.

(b) Vista frontal.

Figura 20: Robot en su entorno.

El gráfico de descarga de los módulos de baterías UPS y LiPo de la Figura 21 indica que el tiempo máximo de autonomía del sistema embebido es de aproximadamente 4 horas. La tensión mínima antes de que el sistema se apague es de 7,2 V. La batería LiPo que alimenta los motores se descarga a un ritmo similar. Aunque la batería no está totalmente descargada a 11,48 V, el AMR en incapaz de moverse correctamente, por lo que ese punto se toma como el valor mínimo.

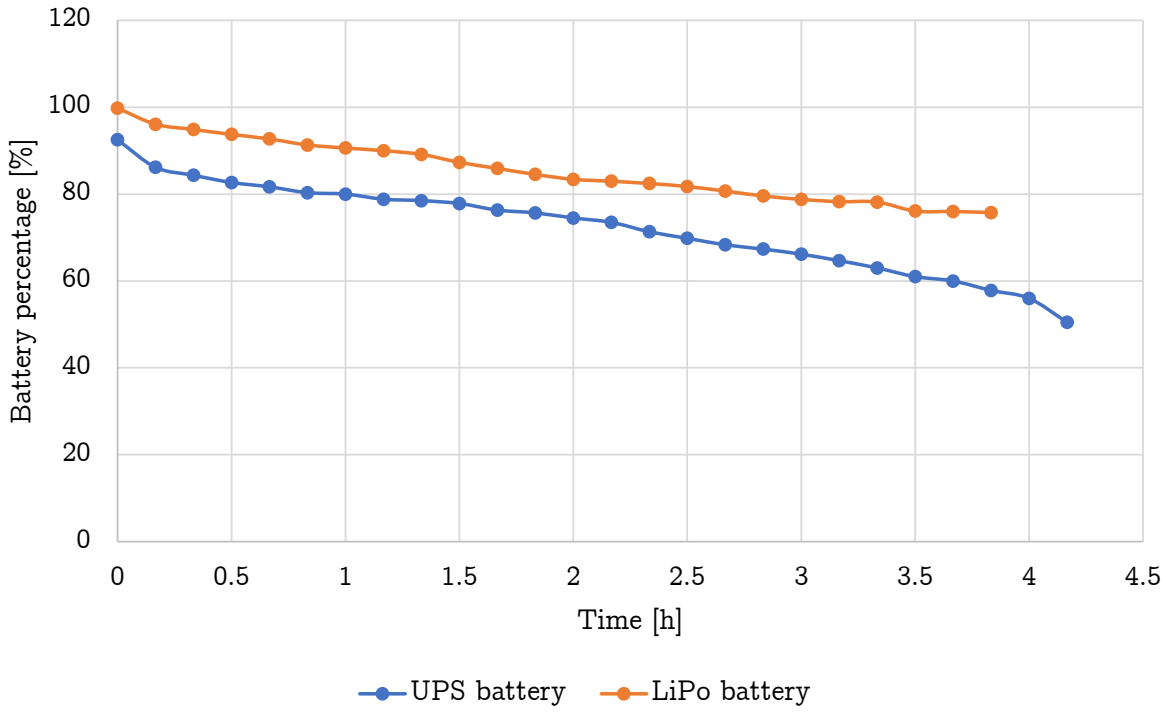


Figura 21: Autonomía de las dos baterías.

5.3 Software

Los resultados del software son la adquisición y comunicación de datos ROS, el mapeado SLAM, la visión artificial y el control de movimiento.

5.3.1 Estabilización de movimiento

La exactitud en el movimiento del robot depende en gran medida de dos factores, el primero es la programación o algoritmo que se encargue de generar la señal de control de la etapa de potencia, y el segundo es el conjunto de características mecánicas que se involucran para producir un movimiento fluido y controlable. De acuerdo a la experimentación hecha, una de estas características, que inicialmente tuvo un impacto negativo es la variación en la velocidad que alcanzan los dos motores que componen el sistema de locomoción. Dicha variación está asociada a la tolerancia aceptada por parte

del fabricante, provocando que el motor izquierdo alcance una velocidad ligeramente mayor que alcanza hasta un 12 % adicional a la velocidad de la rueda derecha. La Figura 22 muestra gráficamente la diferencia en el rendimiento de ambos motores, y se puede observar que es mayormente consistente al incrementar el PWM que se envía desde el microcontrolador.

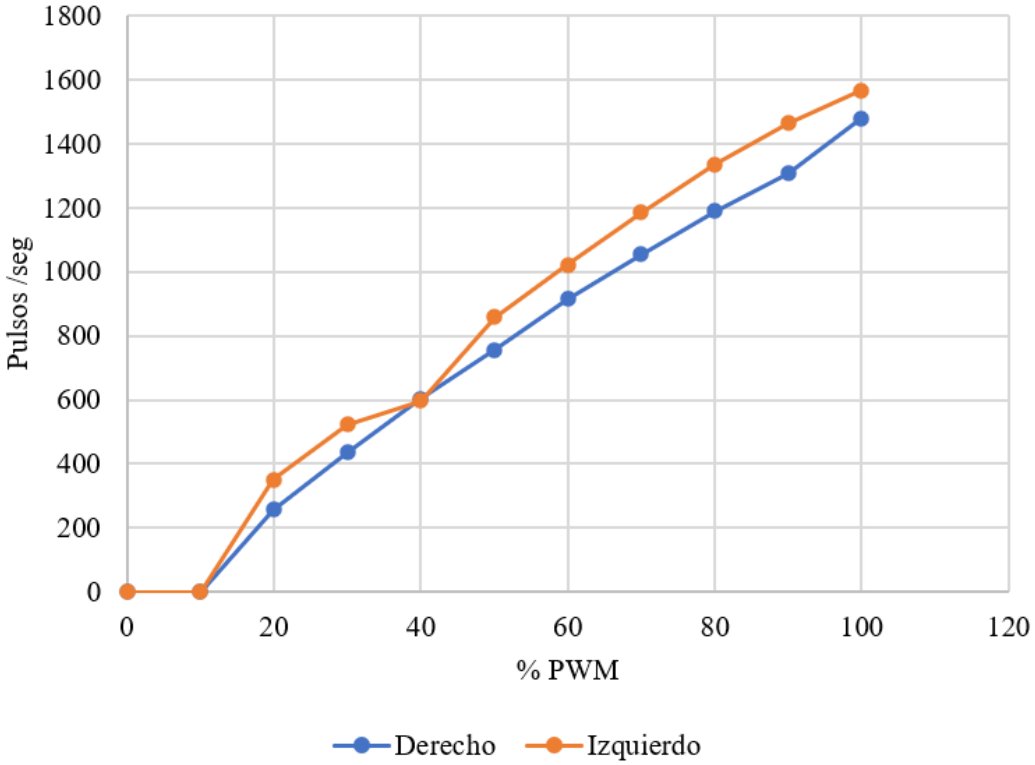


Figura 22: Desempeño de ambos motores del sistema de locomoción en función a una señal de PWM ascendente.

La solución propuesta para este problema consiste en compensar la diferencia mencionada reduciendo la velocidad de la rueda izquierda. Esto se hace calculando la diferencia o error por medio de la siguiente ecuación lineal obtendia por medio de una regresión lineal de los datos mostrados en la Figura 22.

$$Lm_{err} = 0{,}5133x + 81{,}311 \tag{7}$$

Donde:

Lm_{err}: es el error de velocidad del motor izquierdo.

x: es la señal de PWM que actúa como variable independiente.

Como se obseva en la ecuación 7, la estimación del error depende del comando de PWM, y al caracterizarlo, se compensa esa velocidad al momento de enviar el comando de PWM de la rueda izquierda. Para lograrlo, solamente se codificó la ecuación 7 en el microcontrolador, y se utilizó para obtener los nuevos comandos de velocidad. En la Figura 23, se observa claramente el ajuste para estabilizar ambas ruedas, *limitando* el motor de la rueda que alcanza una mayor velocidad.

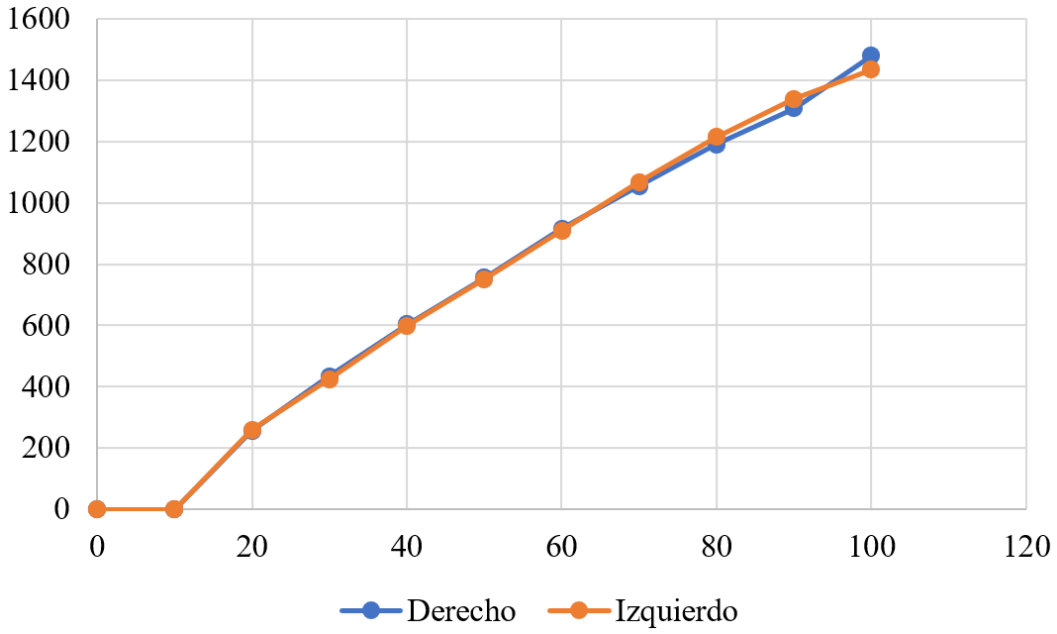


Figura 23: Lectura de velocidad de los motores después de hacer la compensación

5.3.2 Implementación de ROS

Las tareas ejecutadas en ROS comprenden de manera clara al analizar el archivo de ejecución del sistema completo que se debe correr cada vez que el robot trabaje. Es un archivo XML con extensión *.launch*, y entre otras cosas, ayuda a organizar el orden en el que se ejecutan los nodos y también a introducir parámetros globales importantes. El archivo de ejecución está basado en el proyecto encontrado en [69].

Se inicia por especificar la configuración de transformaciones de sistemas de coordenadas. Esto se relaciona con lo explicado en el Capítulo 2, relacionado a los sistemas de coordenadas, e incluye las coordenadas del LiDAR (laser), sensor IMU, base o chasis del robot (base_link), y la huella del robot (base_footprint). También se establece la relación entre el sistema de coordenadas del mapa y el de la odometría, estableciéndose ambos en el origen global.

```
<?xml version="1.0"?>
<launch>
  <!-- Configuración de transformaciones ... Configuración de las relaciones
  entre los sistemas de coordenadas-->
  <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser"
  args="0 0 0.175 3.141 0 0 base_link laser 30" />
  <node pkg="tf" type="static_transform_publisher" name="imu_broadcaster"
  args="0 -0.05 0.045 0 0 0 base_link imu 30" />
  <node pkg="tf" type="static_transform_publisher" name="base_link_broadcaster"
args="0 0 0.048 0 0 0 base_footprint base_link 30" />

  <node pkg="tf" type="static_transform_publisher" name="map_to_odom"
  args="0 0 0 0 0 0 map odom 30" />
```

En la siguiente sección, se cargan parámetros de navegación del paquete ROS Navigation Stack.

```
  <rosparam file="$(find mobile_robot_autonomous_navigation)/config/
  controllers.yaml" command="load"/>
  <rosparam file="$(find mobile_robot_autonomous_navigation)/config/
  joint_limits.yaml" command="load"/>
```

La comunicación entre el sistema embebido NVIDIA Jetson Nano y el microcontrolador ESP32 es de tipo serial, y en la siguiente sección se muestran los

prámetros para entablar dicha comunicación. Esta se compone de la publicación de los pulsos *right_ticks* y *left_ticks* y la suscripción al comando de velocidad *cmd_vel*. Dicha información se visualiza también en la Figura 24, y ha servido para detectar fallos de comunicación, y para confirmar que las lecturas de sensores como los encoders y el sensor de corriente para el monitoreo de la batería LiPo. También sirvió para evitar un conflicto con los puertos USB de NVIDIA Jetson Nano, porque el sensor LiDAR hace uso de un puerto de este tipo, así como el microcontrolador ESP32. De no ser conectado y configurado adecuadamente, puede haber confusión entre ambos.

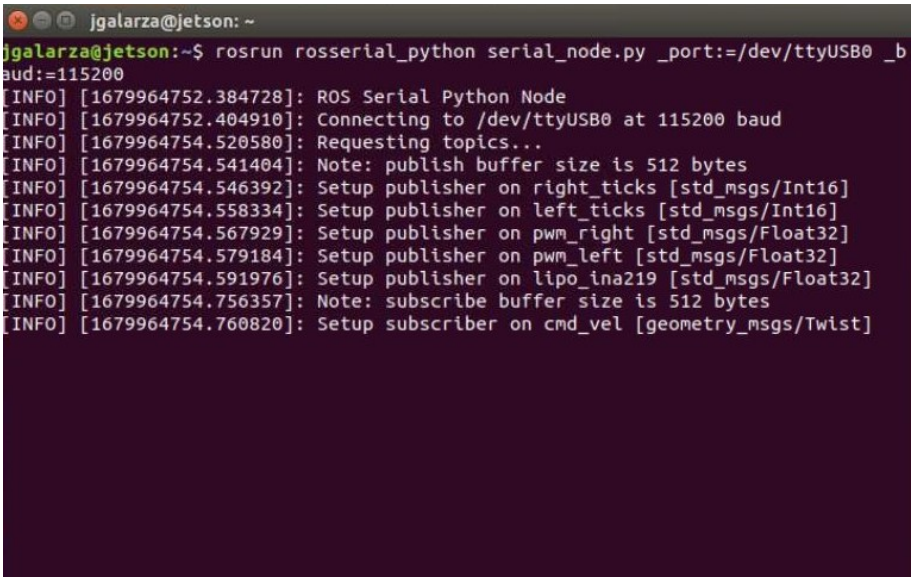


Figura 24: Comunicación serial entre Nvidia Jetson Nano y ESP32. Se observan los tópicos involucrados y el tipo de dato.

```
<!-- Publicación de pulsos de las ruedas desde ESP32 -->
<node pkg="rosserial_python" type="serial_node.py" name="serial_node">
  <param name="port" value="/dev/ttyUSB0"/>
  <param name="baud" value="115200"/>
</node>
<!-- Publicación de pulsos -->
<!-- Subscribe: /right_ticks, /left_ticks, /initial_2d -->
<!-- Publish: /odom_data_euler, /odom_data_quat -->
<node pkg="localization_data_pub" type="ekf_odom_pub" name="ekf_odom_pub">
</node>
<node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false" output="screen"
      args="
          /mobile_robot/joints_update
          /mobile_robot/mobile_base_controller" >
</node>
```

El sensor IMU se utiliza junto con la información de los encoders, porque más adelante se debe calcular la odometría a partir de ambos sensores. Una herramienta empleada para ello es el paquete *mpu_6050_driver*, sirve para obtener la información de los seis ejes del sensor directamente.

```
<!-- Lectura de sensor IMU MPU6050 -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
  type="robot_state_publisher" >
  </node>
  <node name="imu_node" pkg="mpu_6050_driver" type="imu_node.py" >
  </node>
  <node name="imu_filter_node_for_orientation" pkg="imu_complementary_filter"
  type="complementary_filter_node" >
  </node>
  <node name="rpy_tf" pkg="mpu_6050_driver" type="tf_broadcaster_imu.py" >
  </node>
```

El filtro toma toda la información para la localización, y publica un tópico llamado *robot_data_quat*, que es el utilizado por el algoritmo de planificación de trayectorias.

```
  <!-- Filtro Kalman para obtener localización en nodo robot_pose_ekf-->
  <!-- Subscribe: /odom, /imu_data, /vo -->
  <!-- Publish: /robot_pose_ekf/odom_combined -->
  <remap from="odom" to="odom_data_quat" />
  <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
    <param name="output_frame" value="odom"/>
    <param name="base_footprint_frame" value="base_footprint"/>
    <param name="freq" value="30.0"/>
    <param name="sensor_timeout" value="1.0"/>
    <param name="odom_used" value="true"/>
    <param name="imu_used" value="true"/>
    <param name="vo_used" value="false"/>
    <param name="gps_used" value="false"/>
    <param name="debug" value="false"/>
    <param name="self_diagnose" value="false"/>
  </node>
```

También es necesario configurar la interfaz de ROS *Rviz*, en esta se debe proyectar

el mapa, los sistemas de coordenadas, y la localización del robot. Además, cuenta con una interfaz que ayuda a publicar tanto el origen como el destino deseado, esto incluye posición en el eje X y Y, y la orientación θ en ambos casos. Para capturar los clicks en la interfaz, y hacer la publicación, se usa un script en Python.

```
  <!-- Pulbicación de posición de inicio y destino -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d
  /home/jgalarza/catkin_ws/src/Autonomous_Mobile_Robot/
mobile_robot_autonomous_navigation/navigation_data_pub/maps/jetbot2.rviz">
  </node>
  <node pkg="localization_data_pub" type="rviz_click_to_2d"
  name="rviz_click_to_2d">
  </node>
```

El mapa de navegación también se debe cargar para visualizarlo en *Rviz*. Esto se hace por medio de un archivo YAML que contiene la dirección del mapa que es una imagen en formato PGM.

```
 <!-- Archivo de mapa -->
  <arg name="map_file" default=
  "/home/jgalarza/catkin_ws/maps/ciencias/lab_cajas2.yaml"/>
  <!-- Map Server -->
  <!-- Publish: /map, /map_metadata -->
  <node pkg="map_server" name="map_server" type="map_server"
  args="$(arg map_file)" />
  <include file="$(find rplidar_ros)/launch/rplidar.launch" />
  <include file="$(find
  mobile_robot_autonomous_navigation)/launch/amcl.launch" />
  <include file="$(find
  mobile_robot_autonomous_navigation)/launch/move_base.launch" />
```

También se ejecutan otros archivos *launch*, uno para configurar el nodo del sensor

LiDAR y otro para el algoritmo de planificación de trayectorias AMCL.

```
<include file="$(find rplidar_ros)/launch/rplidar.launch" />
<include file="$(find mobile_robot_autonomous_navigation)/launch/
amcl.launch" />
<include file="\$(find mobile_robot_autonomous_navigation)/launch/
move_base.launch" />
```

Finalmente, se optó por habilitar un control por joystick que se usa para controlar el robot de manera remota, principalmente para el proceso de mapeo, y para posicionar el robot de manera libre, con el nodo *joy_node*. Esta utilidad es especialmente útil cuando se están ajustando parámetros del robot, como la velocidad de trabajo. Esta velocidad es crucial para lograr que el control del planificador de trayectorias sea estable y no oscile demasiado al moverse. El joystick permite hacer pruebas de velocidad en la superficie donde se quiera navegar.

```
<!-- Nodo para control por joystick -->
 <node pkg="joy" type="joy_node" name="joy_node">
 </node>
<node name="teleop_twist_joy_node" pkg="teleop_twist_joy"
type="teleop_node" output="screen"/>
</launch>
```

5.3.3 Navegación autónoma

El algoritmo de planificación de trayectorias sigue la secuencia del diagrama en la Figura 25, éste inicia por la lectura de los sensores para calcular la odometría, que también requiere las transformaciones de los sistemas de coordenadas y la información del LiDAR para obtener la localización en tiempo real. El filtro Kalman se utiliza para obtener la localización global del robot, a través de un tópico que contiene la posición en los ejes X, Y, y Z, además de la orientación θ.

El siguiente proceso es el uso del algoritmo AMCL del paquete ROS Navigation Stack que, que requiere de la localización global, además del mapa de navegación. También usa parámetros para calcular mapas de coste de celdas global y local, que están asociados a los métodos de planificación de trayectorias probabilísticos, por lo que el algoritmo obtendrá una estrategia de movimiento que se podrá seguir más adelante. Es importante mencionar que se deben introducir parámetros adicionales para contemplar tolerancias en el funcionamiento del algoritmo, como la distancia de seguridad entre el robot y los obstáculos o paredes, velocidad de trabajo mínima y máxima, además de las dimensiones que delimitan el área que ocupa el robot móvil, principalmente.

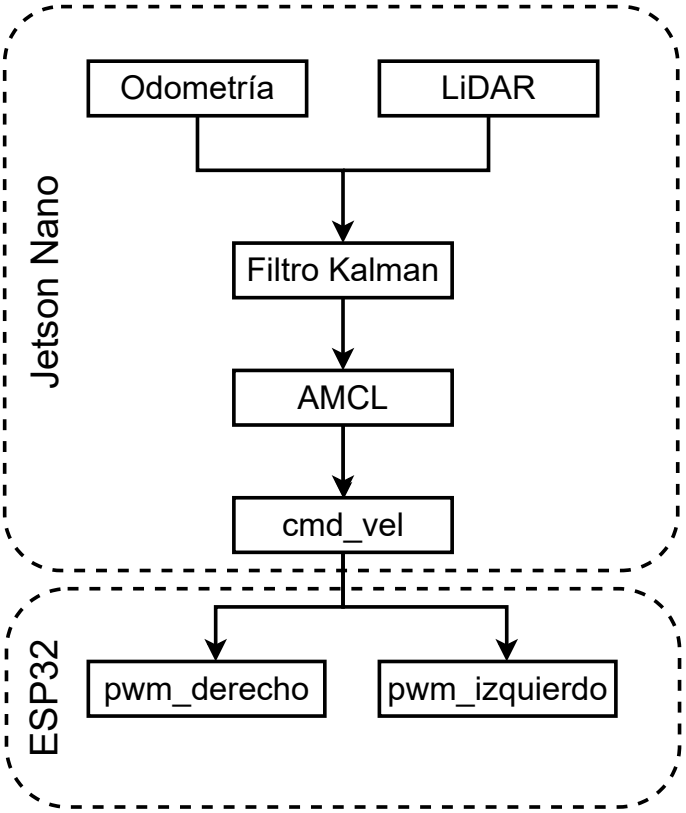


Figura 25: Secuencia del algoritmo de planificación de tayectorias.

En la Figura 25, se muestra que el algoritmo de planificación se ejecuta en NVIDIA Jetson Nano, mientras que el control de la etapa de potencia se hace en el microcontrolador ESP32, que recibe el tópico de ROS *cmd_vel*.

La Figura 26 es una vista desde la interfaz ROS Rviz del mapa utilizado para la experimentación. Se trata de un cuadrado de 1,2 m^2 con tres obstáculos rectangulares fijos donde el robot puede navegar. Es necesaria una distancia de seguridad de 0,15 m entre el robot y los obstáculos. Esto se debe a dos razones: para unir el software y el entorno real con una tolerancia medible y para evitar que el robot navegue demasiado cerca de los obstáculos. En esa misma figura, el eje en el centro del mapa representa el AMR, donde el eje rojo está orientado de la misma forma que el eje local X_R del robot (mostrado en la Figura 17), lo que significa que el segmento rojo del eje apunta hacia donde está orientado el AMR.

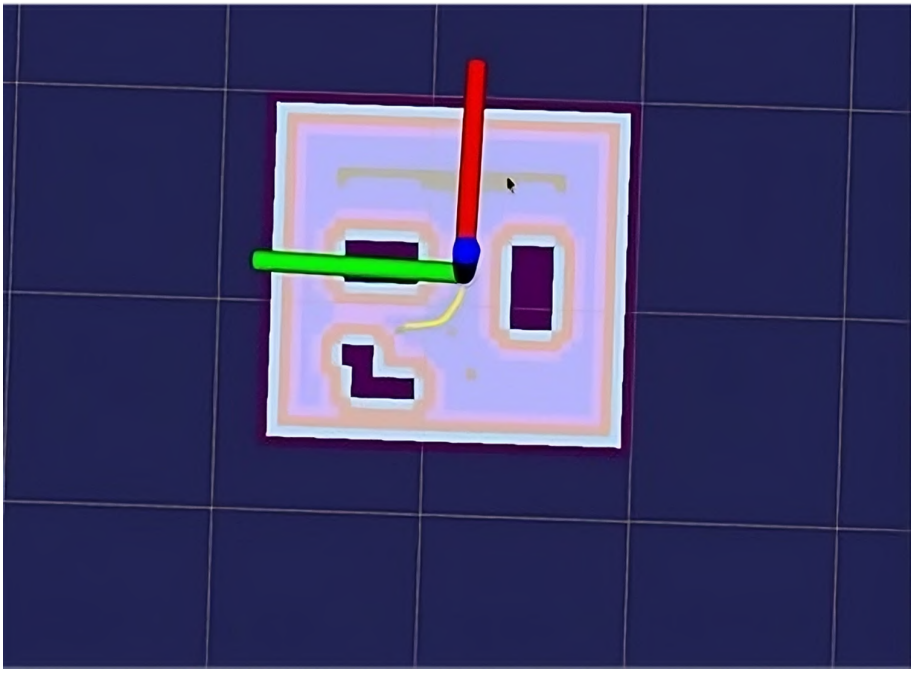


Figura 26: Mapa utilizado para las pruebas visto a través de Rviz.

La Figura 27 es una fotografía del entorno real visto en la Figura 26. La superficie es de madera y su textura lisa ayudó a evitar que las ruedas del AMR se atascaran en las grietas y hendiduras del suelo. Especialmente durante el ajuste de las velocidades lineales y angulares de trabajo de la AMR.

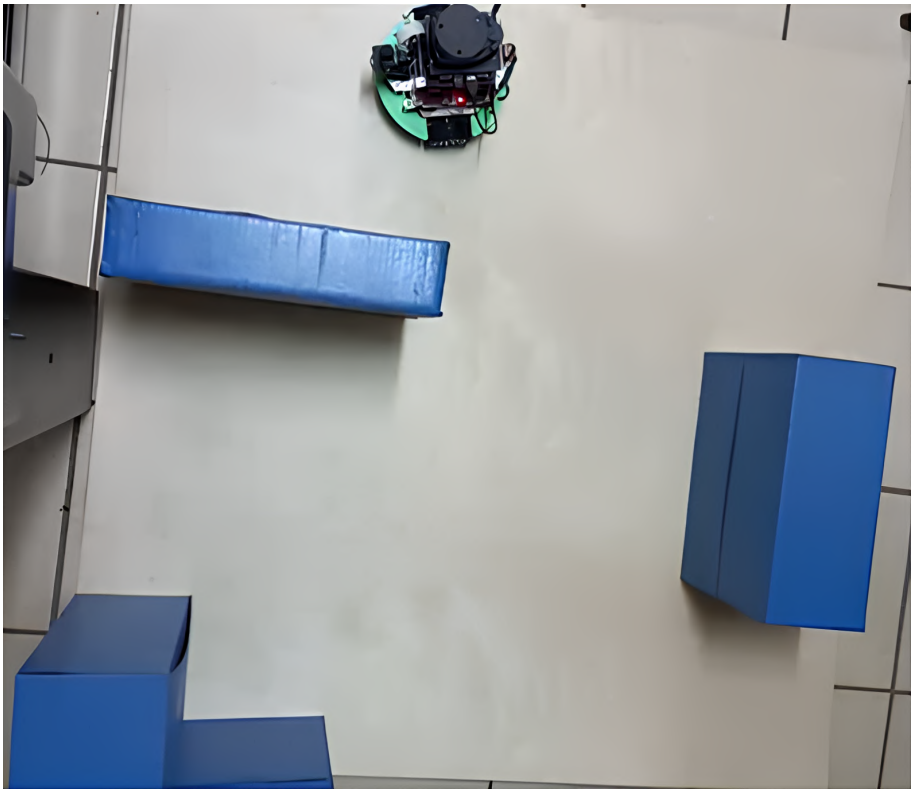


Figura 27: Entorno real por el que navega el robot durante las pruebas.

La Figura 28 es el mapa de planificación local. Eso ayuda a notar cuando un obstáculo dinámico se acerca al AMR. El mapa se lee a través del sensor LiDAR de forma similar al proceso cartográfico. La figura muestra un muro en la zona superior que no se tiene en cuenta en el mapa de prueba. El objeto situado delante del robot (eje rojo) es un obstáculo dinámico, y en la parte inferior del mapa se detecta una persona.

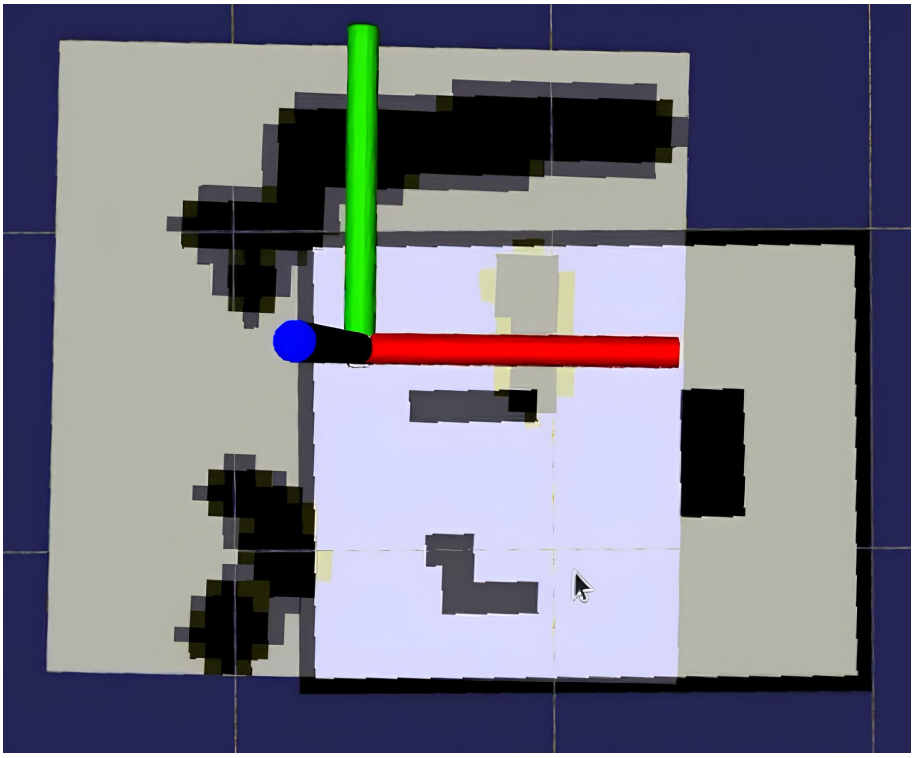


Figura 28: Mapa local que contiene los obstáculos dinámicos.

El mapa del laboratorio es el resultado de ejecutar el nodo ROS LiDAR proporcionado por Slamtec. Fue necesario ajustar parámetros como la posición inicial y la orientación del robot. A continuación, durante el proceso de mapeo, el algoritmo calcula la posición actual del robot basándose en las actualizaciones del LiDAR en tiempo real.

El resultado se guarda como un archivo YAML y una imagen tiff (Figura 29). El mapa muestra el plano bidimensional del entorno. El mapeado del laboratorio duró aproximadamente 7 minutos, y el área es de casi 33 m^2.

Aunque la velocidad del robot era significativamente baja (0,05 m/s), la velocidad angular causó errores menores en la estimación de la posición.

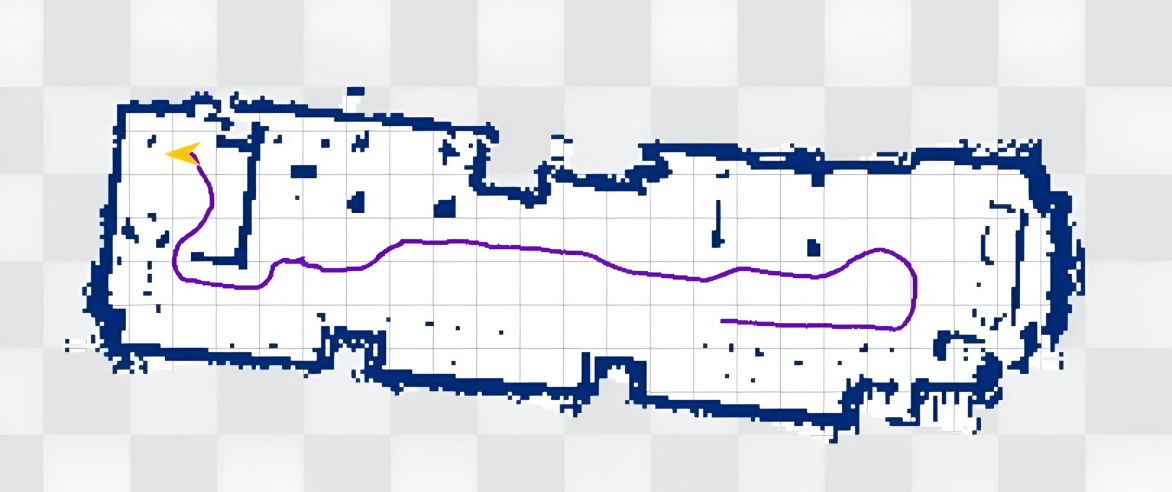


Figura 29: Mapa creado usando el algoritmo Hector SLAM.

El control del movimiento del AMR resulta de procesar la señal de salida del planificador de trayectorias a través de un controlador PID que se muestra en la Figura 12.

La necesidad de dicho control se debió a una falta de estabilidad en la salida del planificador de trayectorias que provocaba un retraso significativo al intentar seguir la ruta planificada. Dicho retraso también producía paradas innecesarias y, en ocasiones, replanificación de la ruta.

El mapa tiene un origen situado en la esquina inferior izquierda. Ese es el origen global del mapa y del sistema de odometría.

El gráfico de la Figura 30 es una simulación realizada para visualizar el comportamiento del controlador y afinar los coeficientes PID K_p, K_i y K_d. La señal resultante (en azul) es más suave y produce un mejor movimiento para el sistema de control de la base del AMR.

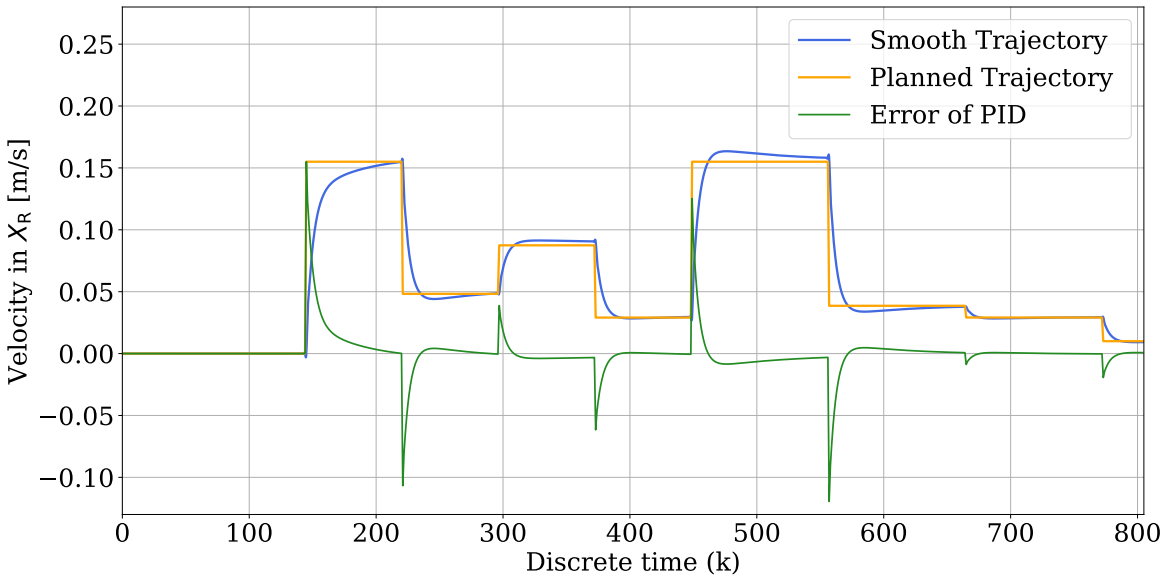


Figura 30: Velocidad de mando leída durante el movimiento del AMR (amarillo), la señal controlada por el PID (azul) y el error del controlador (rojo)

.

Los gráficos de la Figura 31- 32 se tomaron durante un comando de movimiento básico del robot. El mapa tiene un origen global compartido por el planificador de trayectorias, y sistema de odometría. Ese es el punto global al que hacen referencia para hacer los cálculos de movimiento y estimación de posición. El comportamiento de las señales indica que el AMR comenzó esa prueba en particular en un punto situado a 0,6 m del origen en el eje X, a 0,2 m del eje Y y a 0,18 rad del origen de orientación. Con respecto a la de Figura 18, se movió aproximadamente 0,2 m en el eje X. Y no se movió en el eje Y. La Figura 32 muestra los ajustes necesarios para seguir la ruta recta, aproximadamente una media de 0,05 rad (equivalente a $2,8°$).

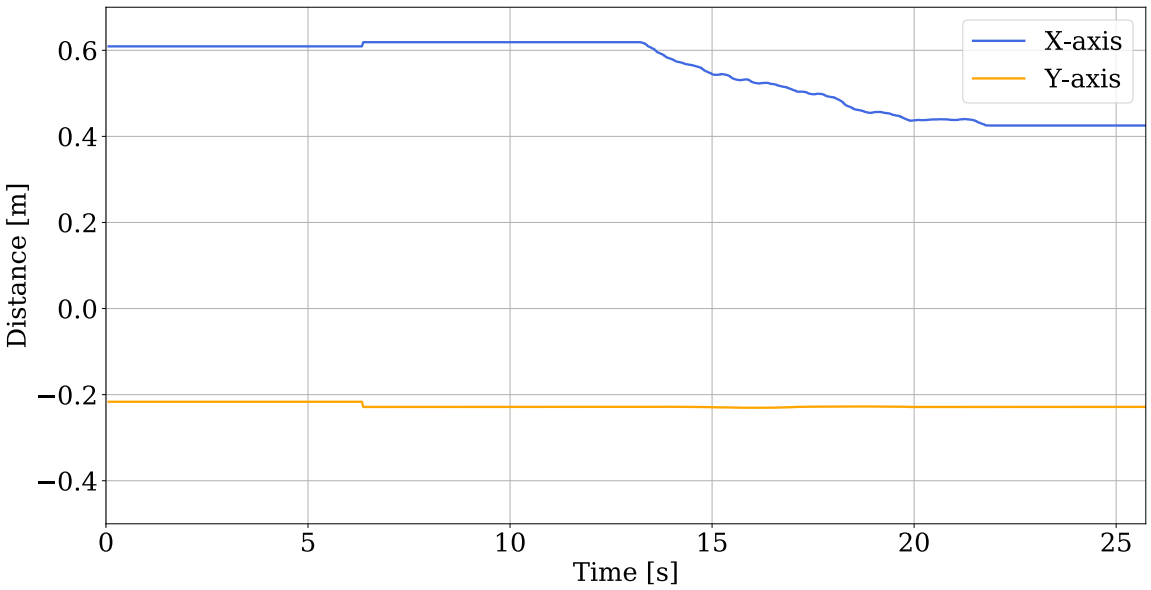


Figura 31: Lecturas de odometría del eje X (azul) y del eje Y (naranja) durante el movimiento del AMR.

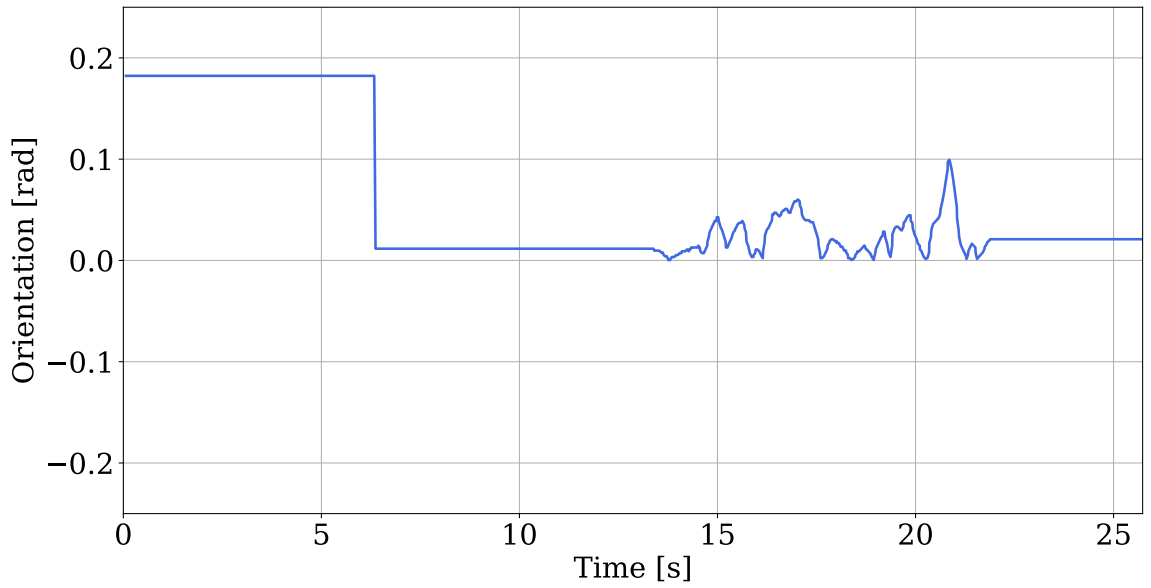


Figura 32: Lectura de orientación de odometría de ROS durante el movimiento de AMR.

5.3.4 Algoritmos de aprendizaje evaluados

El primer método utilizado para la detección de objetos es la CNN ResNet18 por medio de las librerías Pytorch y Torchvision y aprendizaje por transferencia. En la Tabla 2 se

muestran los hiperparámetros empleados para el entrenamiento.

Tabla 2: Parámetros de entrenamiento para modelo ResNet18.

Parámetro	Valor
Optimizador	SDG
Tasa de aprendizaje	0.001
Momento	0.9
Épocas	10
Tamaño de lote	8

El conjunto de datos creado para el entrenamiento se divide en dos clases: *bloqueado* y *libre*. La primera clase incluye 306 imágenes, y la segunda 328 con un tamaño de 224×224 píxeles. Los resultados del proceso de entrenamiento se resumen en la Figura 33, que proporciona información detallada sobre la capacidad del modelo para clasificar la imagen de la cámara CSI del robot en una de las dos clases. La Figura 33a muestra los resultados de la matriz de confusión del modelo ResNet18, en este caso, obtuvo tres falsos positivos para la clase *bloqueado* y 0 falsos negativos para la clase *libre*. Un segundo método para la clasificación de objetos es YOLOv3. Como se menciona en la introducción, se reportan resultados positivos en aplicaciones que se ejecutan en tiempo real. Además, ayuda a identificar objetos específicos dentro del campo de visión del robot. Este método obtuvo un resultado similar al del modelo ResNet18, con la diferencia de que indica la ubicación de cada obstáculo en la escena. Como se muestra en la Figura 33b, el modelo no detectó correctamente el entorno bloqueada en 8 de 100 intentos.

Training Set			
TARGET / OUTPUT	Blocked	Free	SUM
Blocked	94 47.00%	3 1.50%	97 96.91% 3.09%
Free	0 0.00%	103 51.50%	103 100.00% 0.00%
SUM	94 100.00% 0.00%	106 97.17% 2.83%	197 / 200 98.50% 1.50%

(a) Matriz de confusión resultante con el modelo ResNet18.

Training Set			
TARGET / OUTPUT	Blocked	Free	SUM
Blocked	92 46%	0 0%	92 100% 0%
Free	8 4%	100 50%	108 92.59% 7.4%
SUM	100 92% 7.99%	100 100% 0%	192 / 200 96% 4%

(b) Matriz de confusión resultante con el modelo YOLOv3.

Figura 33: Matriz de confusión resultante del entrenamiento de los modelos de DL.

5.3.5 Métricas de evaluación

La Tabla 3 contiene las métricas de entrenamiento del modelo ResNet18. La Exactitud media es del 98,5 %, con una Precisión del 96,91 % para la clase *bloqueado* y del 100 % para la clase *libre*. La Recuperación es del 100 % para la clase *bloqueado* y del 97,17 %

para *libre*.

Tabla 3: Métricas de rendimiento del modelo ResNet18.

Nombre de clase	Precisión	1-Precisión	Recuperación	1-Recuperación	F1-Score
Bloqueado	0.9691	0.0309	1.0000	0.0000	0.9843
Libre	1.0000	0.0000	0.9717	0.0283	0.9856
Exactitud	0.9850				
Tasa de errores de clasificación	0.0150				
F1 macro	0.9850				
F1 ponderado	0.9850				

La Tabla 4 contiene las métricas de entrenamiento del modelo YOLOv3. La Exactitud media es del 96 %, con una Precisión del 100 % para la clase *bloqueado* y del 92.59 % para la clase *libre*. La Recuperación es del 92 % para la clase *bloqueado* y del 100 % para *libre*.

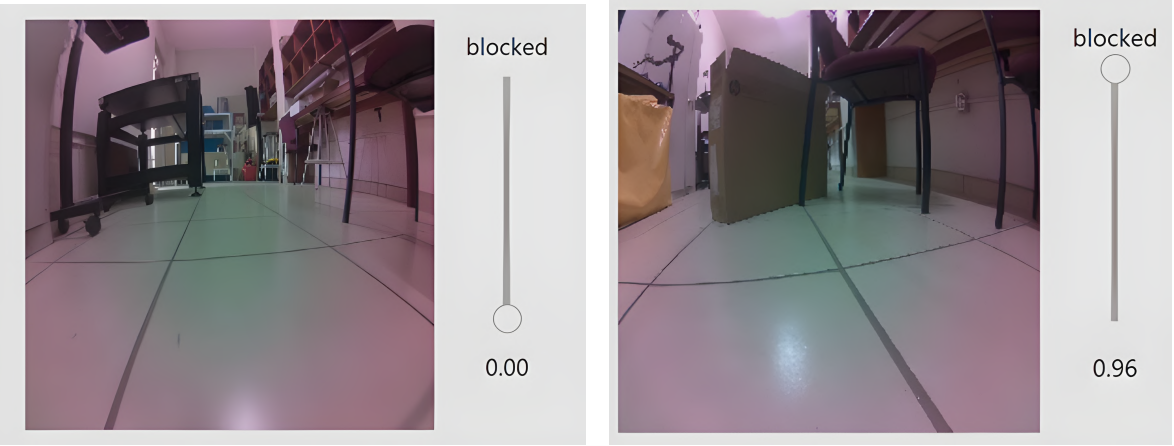
Tabla 4: Métricas de rendimiento de l modelo YOLOv3.

Nombre de clase	Precisión	1-Precisión	Recuperación	1-Recuperación	F1-Score
Bloqueado	1	0	0.92	0.0799	0.9583
Libre	0.9259	0.074	1	0	0.9615
Exactitud	0.96				
Tasa de errores de clasificación	0.04				
F1 macro	0.9599				
F1 ponderado	0.9599				

5.3.6 Evaluación de rendimiento del sistema propuesto

La Figura 34 muestra dos posibles situaciones para el campo de visión del robot. En la Figura 34a, no hay obstáculos delante del robot, mientras que en la Figura 34b, hay una caja, siendo detectada exitosamente como obstáculo con una exactitud del 96 %

por el modelo ResNet18. La interfaz visual utilizada para este experimento se obtuvo de [68].



(a) Sin obstáculos. (b) Un obstáculo.

Figura 34: Campo visual del robot móvil para ambos escenarios posibles.

Figura 35: Prediction of the YOLOv3 model for both classes.

Un segundo método para la clasificación de objetos es YOLO V3. Como se ha mencionado en el Capítulo 2, ofrece resultados favorables en aplicaciones que implican trabajo en tiempo real. Además, las GPU de Jetson Nano permiten un enfoque de aprendizaje profundo, que ayuda a identificar objetos específicos dentro del campo de visión del robot. La Figura 36 demuestra la validez de YOLO V3 ya que

detecta diferentes objetos como sillas y una persona. La resolución utilizada para este experimento fue de 1280x720 píxeles.

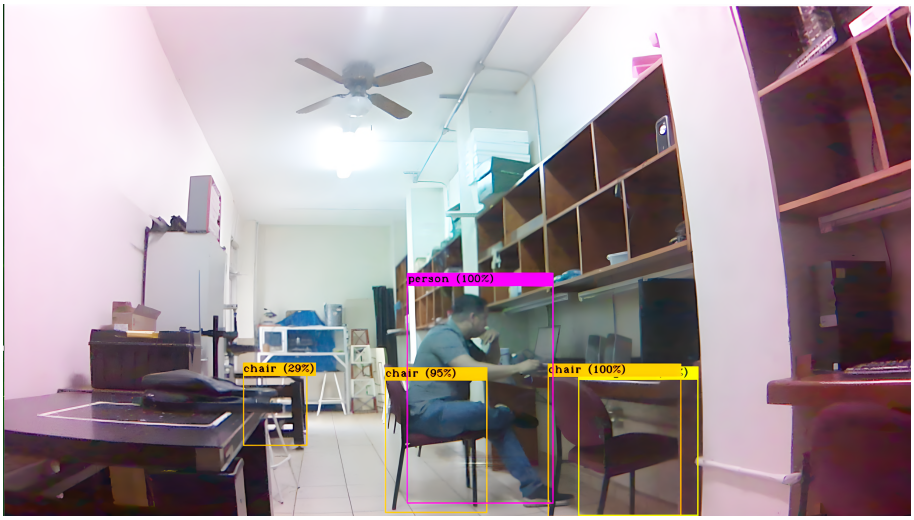


Figura 36: YoloV3 trabajando en tiempo real a través de la cámara CSI.

Capítulo 6

Conclusiones

6.1 Conclusiones generales

La revisión bibliográfica mostró que la mayoría de las investigaciones relacionadas con la RL y DL en robótica móvil tienen como objetivo resolver la planificación de trayectorias en entornos desconocidos para robots que navegan por áreas de difícil acceso sin tener en cuenta la posible interferencia de obstáculos dinámicos. Esto demuestra el creciente interés por diseñar herramientas que simplifiquen el uso de esta tecnología y algoritmos en aplicaciones reales. En este estudio, se confirma que el DL y RL son metodologías fiables para resolver problemas de navegación mediante un enfoque heurístico. La construcción y mejora del prototipo JetBot AMR significó el diseño del hardware y la integración del software. La implementación de capacidades cognitivas más profundas requirió la adición de dispositivos específicos que ayudaran a interpretar el entorno del robot, esto fue posible al agregar sensores como IMU en conjunto con los encoders mediante el filtro Kalman. La plataforma ROS proporcionó un entorno que permitió la posibilidad de añadir progresivamente dispositivos externos y sus funcionalidades, dividiendo el proyecto en fases, partiendo de las características mecánicas más fundamentales como las dimensiones del prototipo, y la ubicación de cada sensor, para ir añadiendo cada vez más características como la fusión de sensores y los algoritmos DL. En términos de consumo de energía, después de la experimentación,

se concluye que el robot puede trabajar por más de 2 horas, extendiendo su autonomía si la carga del procesador es menor. Demostrando que el sistema embebido tiene un alto consumo de energía, mientras que la etapa de potencia pudo trabajar adecuadamente durante el tiempo ya mencionado. El uso de dos baterías diferentes facilitó la fase de pruebas porque el aislamiento del sistema embebido y la etapa de potencia garantizó un aislamiento que protegió la integridad de cada componente ante posibles fallos o cortocircuitos.

El sistema de visión artificial propuesto utilizando ResNet18 alcanzó una exactitud media del 98,5 %, con una precisión del 96,91 %, recuperación del 98.58 % y F1-Score del 98,5 %. El sistema de visión con YOLOv3 obtuvo una Exactitud del 96 %, con una Precisión del 98.45 %, una Recuperación del 96 % y un F1-score del 95.99 %.

La cámara digital empleada para capturar el conjunto de datos para el entrenamiento del modelo de DL, proporcionó una resolución bastante exacta de 3280 x 2464 pixeles, sin embargo, con la finalidad de aligerar el procesamiento del sistema embebido, se optó por reducirla a 640x480 pixeles porque se demostró que dicha resolución es suficiente para el uso que se le dió. Con estas mejoras, se ha demostrado que es posible ejecutar simultáneamente el algoritmo de planificación de trayectorias AMCL junto con el detector de obstáculos sin impactar signitifcativamente en la latencia del sistema.

De acuerdo con la descripción realizada en este estudio sobre el hardware que se está utilizando para entrenar los modelos de DL, es posible afirmar que futuras mejoras de dicho hardware proporcionarán una plataforma confiable para realizar procesamiento intensivo utilizando procesadores y GPUs de última generación, especialmente en el campo de los sistemas embebidos.

6.2 Trabajo futuro

El interés principal que surge al concluir la investigación, es la mejora del sistema de control de la base móvil, debido a que es limitado, y se observó que esa etapa determina en gran medida la exactitud de los movimientos del robot, esto puede ser por medio de un mejor sistema de control de la etapa de potencia, y reduciendo las comunicaciones con dispositivos externos al sistema embebido, como puede ser el microcontrolador. Esto tiene el potencial de optimizar el software y lograr un mejor desempeño del sistema.

Por otro lado, el uso de otro tipo de modelos de aprendizaje puede ayudar a optimizar otras etapas del sistema, ya que de momento solamente se experimentó con la evasión de obstáculos en la planificación de trayectorias de tipo local. Sin embargo, es posible usar modelos de DL enfocados a una aplicación específica como puede ser en la toma de decisiones en labores de inspección, análisis de señales en tiempo real, e interacción con el usuario. Todas estas tareas aportan la capacidad de mejorar la adaptabilidad de un AMR en aplicaciones a nivel industrial, agrícolas y en investigaciones multidisciplinarias.

Finalmente, en el futuro se proyecta aplicar la metodología expuesta en este trabajo para hacer sistemas automatizados donde colaboren múltiples robots, y con base en la información presentada en el Capítulo 3, esto es posible trabajando con algoritmos heurísticos basados en aprendizaje.

Apéndice A

Publicaciones derivadas del trabajo de tesis

- **Producción científica**

  J. Galarza-Falfan, E. E. García-Guerrero, O. A. Aguirre-Castro, O. R. López-Bonilla, U. J. Tamayo Pérez, J. R. Cárdenas-Valdez, C. Hernández-Mejia, S. Borrego-Domínguez, E. Inzunza-González. 2024, *Path planning for autonomous mobile robot using intelligen algorithms*. Sometido en Technologies.

- **Acceso universal de conocimiento**

  J. Galarza-Falfan, E. E. García-Guerrero, O. R. López-Bonilla, O. A. Aguirre-Castro, U. J. Tamayo Pérez, C. Hernández-Mejia, J. R. Cárdenas-Valdez, E. Inzunza-González. 2023. *Path planning for industrial autonomous mobile robot using reinforcement learning algorithms*. Congreso internacional EDIESCA 2023, Monterrey, Nuevo León.

  J. Galarza-Falfan, O. A. Aguirre-Castro, E. Inzunza-González. *Inteligencia artificial y la seguridad de la información en internet*. 2023. Presentación en la Semana Nacional de Ciencia y Tecnología 2023. Ensenada, Baja California.

J. Galarza-Falfan, et. al. *Planificación de trayectorias usando algoritmos de aprendizaje automático*. 2023. Presentación en Expo Ciencia y Tecnología - FIAD. Ensenada, Baja California.
Enlace: https://www.youtube.com/watch?v=yb2cB14dZrc.

J. Galarza-Falfan, et. al. *Planificación de trayectorias usando algoritmos de aprendizaje automático*. 2023. Presentación en Noche de las Ciencias 2023. Ensenada, Baja California.

J. Galarza-Falfan. *Filtro no lineal para imágenes digitales*. 2022. Presentación en Expo Ciencia y Tecnología - FIAD. Ensenada, Baja California.
Enlace: https://www.youtube.com/watch?v=9Fe-akmqv-8.

Bibliografía

[1] Nils J. Nilsson. A mobile automation: an application of artificial intelligence techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 509–520, Washington, DC, 1969. x, 1, 2, 4, 12

[2] Davide Scaramuzza, Roland Siegwart, and Illah Reza Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2nd edition, 2011. x, 1, 8, 9, 10, 42

[3] Cui Zhi. Research on cartographer algorithm based on low cost lidar. *International Journal of Engineering Research and*, V8, 10 2019. x, 11

[4] Sophie Hall, Fridolin Wild, and Tjeerd Scheper. *Real-Time Auditory Biofeedback System for Learning a Novel Arm Trajectory: A Usability Study*, pages 385–409. Springer, Cham, 11 2019. x, 13

[5] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain 1, 1958. x, 19

[6] Oscar Adrian Aguirre-Castro, Everardo Inzunza-González, Enrique Efrén García-Guerrero, Esteban Tlelo-Cuautle, Oscar Roberto López-Bonilla, Jesús Everardo Olguín-Tiznado, and José Ricardo Cárdenas-Valdez. Design and construction of an rov for underwater exploration. *Sensors (Switzerland)*, 19(24), 2019. x, 33

[7] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. x, 36

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. x, 37

[9] Ravi Raj and Andrzej Kos. A comprehensive study of mobile robot: History, developments, applications, and future research perspectives. *Applied Sciences*, 12(14), 2022. 1, 2, 8

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010. 2

[11] Subhranil Das and Sudhansu Kumar Mishra. A machine learning approach for collision avoidance and path planning of mobile robot under dense and cluttered environments. *Computers and Electrical Engineering*, 103:108376, 2022. 4

[12] Jichao Cui and Guanghua Nie. Motion route planning and obstacle avoidance method for mobile robot based on deep learning. *Journal of Electrical and Computer Engineering*, 2022:5739765, 2022. 4, 23

[13] M. Kheirandish, E. Azadi Yazdi, H. Mohammadi, and M. Mohammadi. A fault-tolerant sensor fusion in mobile robots using multiple model kalman filters. *Robotics and Autonomous Systems*, 161:104343, 3 2023. 4

[14] Anantha Sai Hari Haran Injarapu and Suresh Kumar Gawre. A survey of autonomous mobile robot path planning approaches. In *International Conference on Recent Innovations in Signal Processing and Embedded Systems (RISE)*, pages 624–628. IEEE, 2017. 4

[15] Mohd Nayab Zafar and J. C. Mohanta. Methodology for path planning and optimization of mobile robots: A review. In *Procedia Computer Science*, volume 133, pages 141–152. Elsevier B.V., 2018. 4

[16] D Keirsey, E Koch, J McKisson, A Meystel, and J Mitchell. Algorithm of navigation for a mobile robot. In *Proceedings. 1984 IEEE International Conference on Robotics and Automation*, volume 1, pages 574–583, 1984. 4

[17] C Lamini, Y Fathi, and S Benhlima. H-mas architecture and reinforcement learning method for autonomous robot path planning. In *2017 Intelligent Systems and Computer Vision (ISCV)*, pages 1–7, 2017. 4

[18] Antonio Savio Silva Oliveira, Marcello Carvalho dos Reis, Francisco Alan Xavier da Mota, Maria Elisa Marciano Martinez, and Auzuir Ripardo Alexandria. New trends on computer vision applied to mobile robot localization. *Internet of Things and Cyber-Physical Systems*, 2:63–69, 1 2022. 5

[19] S A S Mohamed, M H. Haghbayan, T Westerlund, J Heikkonen, H Tenhunen, and J Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7:97466–97486, 2019. 9

[20] Chaoqun Wang, Lili Meng, Sizhen She, Ian M Mitchell, Teng Li, Frederick Tung, Weiwei Wan, Max Q H Meng, and Clarence W de Silva. Autonomous mobile robot navigation in uneven and unstructured indoor environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. 10

[21] Weihua Chen and Tie Zhang. An indoor mobile robot navigation technique using odometry and electronic compass. *International Journal of Advanced Robotic Systems*, 14:1729881417711643, 5 2017. doi: 10.1177/1729881417711643. 10

[22] Gábor Péter, Bálint Kiss, and Viktor Tihanyi. Vision and odometry based autonomous vehicle lane changing. *ICT Express*, 5:219–226, 12 2019. 11

[23] T. Ran, L. Yuan, and J. B. Zhang. Scene perception based visual navigation of mobile robot in indoor environment. *ISA Transactions*, 109:389–400, 3 2021. 11

[24] Thomas Bräunl. *Lidar Sensors*, pages 53–57. Springer International Publishing, Cham, 2023. 11

[25] Xi Hu and Rayan H. Assaad. A bim-enabled digital twin framework for real-time indoor environment monitoring and visualization by integrating autonomous robotics, lidar-based 3d mobile mapping, iot sensing, and indoor positioning technologies. *Journal of Building Engineering*, 86, 2024. 11

[26] Alireza Mohseni, Vincent Duchaine, and Tony Wong. Improvement in monte carlo localization using information theory and statistical approaches. *Engineering Applications of Artificial Intelligence*, 131, 5 2024. 12, 15

[27] Rémy Guyonneau, Franck Mercier, and Vincent Boucher. Robotic system for indoor illuminance map generation. *Journal of Building Engineering*, 86:108800, 2024. 12

[28] Sattra Piyapunsutti, Eisen Lance De Guzman, and Ronnapee Chaichaowarat. Navigating mobile manipulator robot for restaurant application using open-source software. In *2023 IEEE International Conference on Robotics and Biomimetics, ROBIO 2023*. Institute of Electrical and Electronics Engineers Inc., 2023. 12

[29] El Houssein Chouaib Harik and Audun Korsaeth. Combining hector slam and artificial potential field for autonomous navigation inside a greenhouse. *Robotics*, 7, 5 2018. 12

[30] Tong Zhang, Jianyu Xu, Hao Shen, Rui Yang, and Tao Yang. Rmsc-vio: Robust multi-stereoscopic visual-inertial odometry for local visually challenging scenarios. *IEEE Robotics and Automation Letters*, 9(5):4130 – 4137, 2024. 12

[31] Ayan Paul, Rajendra Machavaram, Ambuj, Dheeraj Kumar, and Harsh Nagar. Smart solutions for capsicum harvesting: Unleashing the power of yolo for detection, segmentation, growth stage classification, counting, and real-time

mobile identification. *Computers and Electronics in Agriculture*, 219, 4 2024. 12, 13

[32] Xue Iuan Wong and Manoranjan Majji. Extended Kalman Filter for Stereo Vision-Based Localization and Mapping Applications. *Journal of Dynamic Systems, Measurement, and Control*, 140(3):030908, 11 2017. 12, 13

[33] Qiyang Zhang, Xiangying Che, Yijie Chen, Xiao Ma, Mengwei Xu, Schahram Dustdar, Xuanzhe Liu, and Shangguang Wang. A comprehensive deep learning library benchmark and optimal library selection. *IEEE Transactions on Mobile Computing*, 23(5):5069 – 5082, 2024. Cited by: 0. 13

[34] Gerasimos G. Samatas and Theodore P. Pachidis. Inertial measurement units (imus) in mobile robots over the last five years: A review. *Designs*, 6(1), 2022. 13

[35] Lentin Joseph. *Robot Operating System for Absolute Beginners.* Springer International Publishing, 01 2018. 14, 15

[36] Andrei Vukolov. Wiki. Export Date: 16 April 2024. 14

[37] Andrei Vukolov. Wiki. Export Date: 16 April 2024. 14

[38] Addison Sears-Collins. Coordinate frames and transforms for ros-based mobile robots. 14

[39] Wiki. Export Date: 16 April 2024. 15

[40] Lentin Joseph and Jonathan Cacace. *Mastering ROS for Robotics Programming - Second Edition: Design, build, and simulate complex robots using the Robot Operating System.* Packt Publishing, 2nd edition, 2018. 16

[41] R. Tommy, A.N. Manaparampil, and R. Michael. *Building Smart Robots Using ROS: Design, Build, Simulate, Prototype and Control Smart Robots Using ROS, Machine Learning and React Native Platform.* BPB Publications, 2022. 16

[42] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997. 17

[43] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, USA, 1995. 17

[44] Bohdan Macukow. Neural networks – state of art, brief history, basic models and architecture. In Khalid Saeed and Władysław Homenda, editors, *Computer Information Systems and Industrial Management*, pages 3–14. Springer International Publishing, 2016. 18

[45] Chaymaa Lamini, Youssef Fathi, and Said Benhlima. H-mas architecture and reinforcement learning method for autonomous robot path planning. In *2017 Intelligent Systems and Computer Vision (ISCV)*, pages 1–7, 2017. 20

[46] Xiaogang Ruan, Chenliang Lin, Jing Huang, and Yufan Li. Obstacle avoidance navigation method for robot based on deep reinforcement learning. In *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, volume 6, pages 1633–1637, 2022. 21, 23, 26

[47] Kevin Braathen de Carvalho, Hiago B. Batista, Iure L. De Oliveira, and Alexandre S. Brandao. A 3d q-learning algorithm for offline uav path planning with priority shifting rewards. In *2022 19th Latin American Robotics Symposium, 2022 14th Brazilian Symposium on Robotics and 2022 13th Workshop on Robotics in Education, LARS-SBR-WRE 2022*, pages 169–174. Institute of Electrical and Electronics Engineers Inc., 2022. 21, 26

[48] K B de Carvalho, I R L de Oliveira, D K D Villa, A G Caldeira, M Sarcinelli-Filho, and A S Brandão. Q-learning based path planning method for uavs using priority shifting. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 421–426, 2022. 21

[49] Xiangdong Zheng, Yuxin Wu, Lianhong Zhang, Maobin Tang, and Fusheng Zhu. Priority-aware path planning and user scheduling for uav-mounted mec networks: A deep reinforcement learning approach. *Physical Communication*, 62:102234, 2024. 21

[50] José del R. Millán. Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot. *Robotics and Autonomous Systems*, 15(4):275–299, 1995. Reinforcement Learning and Robotics. 21

[51] Vosviewer. Available at https://www.vosviewer.com/, Accessed: June 2023. 22

[52] Linh Kastner, Teham Bhuiyan, Tuan Anh Le, Elias Treis, Johannes Cox, Boris Meinardus, Jacek Kmiecik, Reyk Carstens, Duc Pichel, Bassel Fatloun, Niloufar Khorsandi, and Jens Lambrecht. Arena-bench: A benchmarking suite for obstacle avoidance approaches in highly dynamic environments. *IEEE Robotics and Automation Letters*, 7:9477–9484, 10 2022. 23, 24

[53] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5:6932–6939, 10 2020. 24

[54] M. Park, P. Ladosz, and H. Oh. Source term estimation using deep reinforcement learning with gaussian mixture model feature extraction for mobile sensors. *IEEE Robotics and Automation Letters*, 7(3):8323–8330, 2022. 24

[55] Z. Zheng, C. Cao, and J. Pan. A hierarchical approach for mobile robot exploration in pedestrian crowd. *IEEE Robotics and Automation Letters*, 7(1):175–182, 2022. 24

[56] Y. Chen, U. Rosolia, W. Ubellacker, N. Csomay-Shanklin, and A.D. Ames. Interactive multi-modal motion planning with branch model predictive control. *IEEE Robotics and Automation Letters*, 7(2):5365–5372, 2022. 24

[57] Y. Yin, Z. Chen, G. Liu, and J. Guo. A mapless local path planning approach using deep reinforcement learning framework. *Sensors*, 23(4), 2023. 24

[58] M. Park, S.Y. Lee, J.S. Hong, and N.K. Kwon. Deep deterministic policy gradient-based autonomous driving for mobile robots in sparse reward environments. *Sensors*, 22(24), 2022. 24

[59] D. Kozjek, A. Malus, and R. Vrabič. Reinforcement-learning-based route generation for heavy-traffic autonomous mobile robot systems. *Sensors*, 21(14), 2021. 24

[60] M. Pei, H. An, B. Liu, and C. Wang. An improved dyna-q algorithm for mobile robot path planning in unknown dynamic environment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(7):4415–4425, 2022. cited By 6. 24, 25

[61] A. Sivaranjani and B. Vinod. Artificial potential field incorporated deep-q-network algorithm for mobile robot path prediction. *Intelligent Automation and Soft Computing*, 35(1):1135–1150, 2023. 25

[62] Xia Wang, Jun Liu, Chris Nugent, Ian Cleland, and Yang Xu. Mobile agent path planning under uncertain environment using reinforcement learning and probabilistic model checking. *Knowledge-Based Systems*, 264:110355, 2023. 26

[63] Laiyi Yang, Jing Bi, and Haitao Yuan. Dynamic path planning for mobile robots with deep reinforcement learning. *IFAC-PapersOnLine*, 55:19–24, 1 2022. 26

[64] Kiwon Yeom. Collision avoidance for a car-like mobile robots using deep reinforcement learning. *International Journal of Emerging Technology and Advanced Engineering*, 11:22–30, 11 2021. 26

[65] J Hu, H Niu, J Carrasco, B Lennox, and F Arvin. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69:14413–14423, 2020. 26

[66] J Xiang, Q Li, X Dong, and Z Ren. Continuous control with deep reinforcement learning for mobile robot navigation. In *Proceedings - 2019 Chinese Automation Congress, CAC 2019*, pages 1501–1506. Institute of Electrical and Electronics Engineers Inc., 2019. 26

[67] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), may 2022. 32

[68] Automatic obstacle avoiding - waveshare wiki. Available at https://www.waveshare.com/wiki/Automatic_Obstacle_Avoiding, Accessed: January 2023. 37, 65

[69] Addison Sears-Collins. How to set up the ros navigation stack on a robot. 48