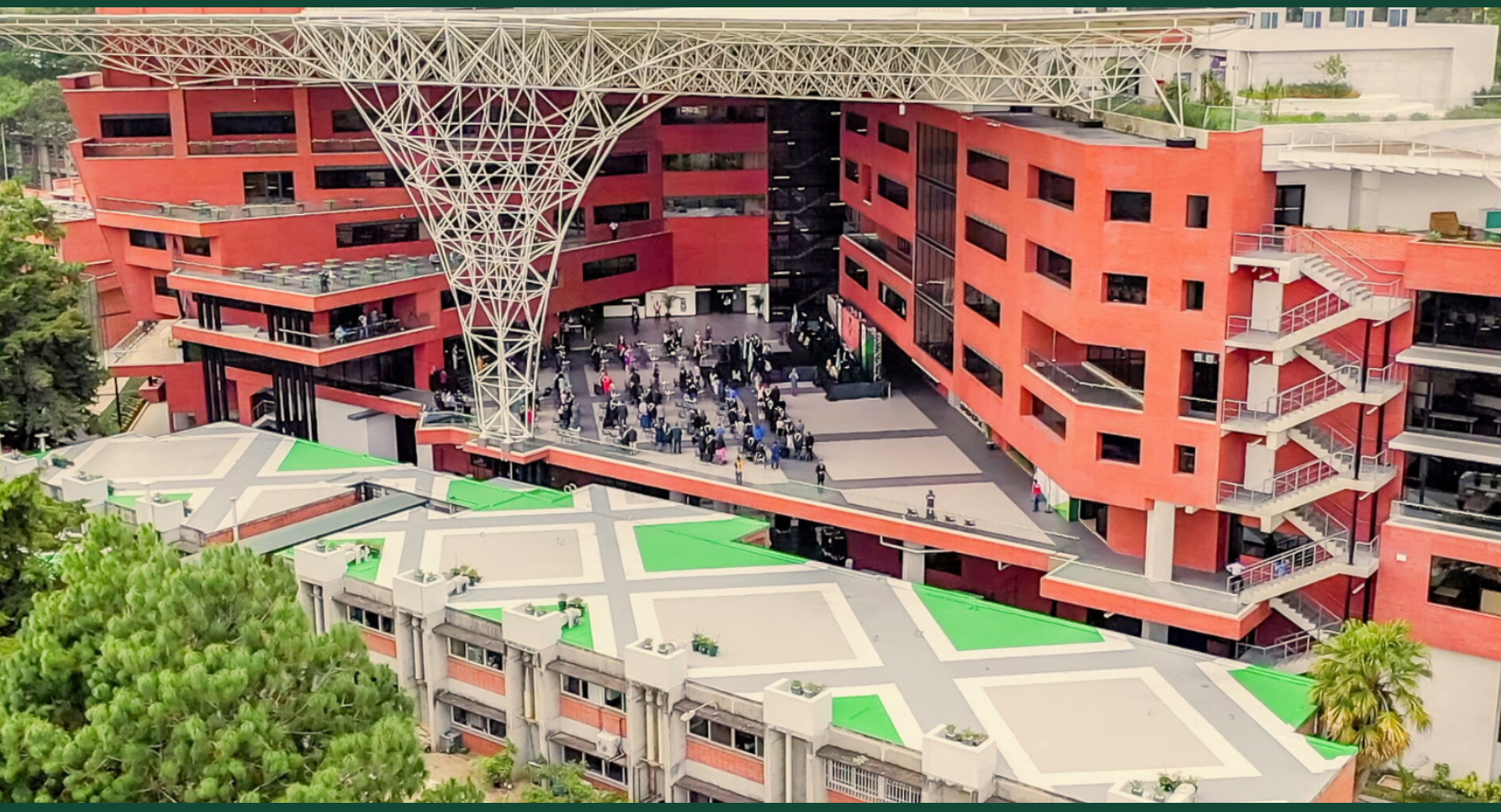
Desarrollo de herramientas de software para el control y monitoreo de agentes robóticos Pololu 3Pi+ mediante ROS2 dentro del ecosistema Robotat

Bryan Estuardo Ortiz Casimiro



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de herramientas de software para el control y monitoreo de agentes robóticos Pololu 3Pi+ mediante ROS2 dentro del ecosistema Robotat

Trabajo de graduación presentado por Bryan Estuardo Ortiz Casimiro
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

Vo.Bo.:

(f) 

M.Sc. Miguel Enrique Zea Arenales

(f) 

M.Sc. Carlos Alberto Esquit Hernández

Fecha de aprobación: Guatemala, 28 de noviembre de 2025.

Prefacio

Me gustaría agradecer profundamente a mi familia por todo el apoyo incondicional que me brindaron durante toda mi vida, así como por darme siempre la motivación para buscar la superación personal y la excelencia. Agradezco a mi padre, Estuardo Ortiz Barillas, por darme siempre todo lo necesario para superarme y motivarme con su apoyo y cariño. Muchas gracias por siempre creer en mí, incluso cuando yo no lo hice. Agradezco a mi madre, Irma Aracely Casimiro Regalado, por apoyarme y cuidar de mí en todo momento; gracias por creer en mí y por darme ánimos siempre que lo necesité. Agradezco a mi hermana, María Fernanda Ortiz Casimiro, por haber sido mi ejemplo a seguir y por presentarme a Dios desde joven. Gracias por su apoyo y cariño.

Asimismo, quiero agradecer a la Fundación Juan Bautista Gutiérrez por haber apostado por mi futuro y hacer posible el sueño de continuar con mis estudios universitarios. Agradezco infinitamente la oportunidad que me dieron, así como todo el acompañamiento que recibí por parte de su programa de coaching. Muchas gracias a cada una de mis *coaches* por haber sido una parte fundamental de mi desarrollo como persona.

A todos mis amigos de la universidad, quiero agradecerles por haber sido parte de mi vida en estos años. Gracias por el apoyo, las risas y los consejos. No hubiera sido lo mismo sin todas las personas que formaron mi grupo de amigos. Gracias porque siempre estuvieron ahí para mí.

Agradezco a mi asesor, Miguel Zea, no solo por guiarme durante la elaboración de mi trabajo de graduación, sino también por haberme mostrado el mundo del control y la robótica de la manera en que lo hizo. Agradezco todo el trabajo que ha realizado en la universidad.

Para finalizar, quiero agradecer a Dios por permitirme llegar hasta este momento, por haberme dado lo necesario para culminar esta etapa y por siempre cuidarme. Todo se lo debo a Dios y, sin Él, no sería posible nada de esto.

Índice general

Índice de figuras

Índice de cuadros

Resumen

La robótica móvil requiere herramientas de software para integrar el control y la comunicación multiagente, siendo ROS 2 (robot operating system) un estándar clave. No obstante, su uso en plataformas educativas de bajo costo se limita por los recursos reducidos de los microcontroladores. Para superar esto, se empleó micro-ROS, una extensión que permite llevar las funcionalidades de ROS 2 a dispositivos embebidos. El trabajo se centró en la integración de micro-ROS en robots móviles Pololu 3Pi+ dentro del laboratorio Robotat. La implementación se realizó en un microcontrolador ESP32-WROOM-32D, utilizando comunicación wifi (UDP) y un puente MQTT para interactuar con ROS 2, logrando soportar hasta diez agentes simultáneos. Se integró el sistema de captura de movimiento OptiTrack mediante Mocap4ros2 y un puente MQTT para la localización.

Se desarrolló un controlador punto a punto en ROS 2 para el Pololu 3Pi+, combinando un controlador PID para la orientación y un acercamiento exponencial para el error de posición. Este controlador se implementó tanto en versión individual como multiagente, probándose exitosamente con nueve robots simultáneamente. Los resultados validan la factibilidad de integrar ROS 2 y micro-ROS en el Pololu 3Pi+ dentro del ecosistema Robotat, estableciendo una arquitectura modular y escalable. Esto sienta una base sólida para futuros desarrollos de estrategias de control más sofisticadas y proyectos colaborativos de mayor alcance utilizando estos robots.

Palabras clave: robótica móvil, ROS 2, micro-ROS, Pololu 3Pi+,sistemas embebidos, sistemas de control.

Abstract

Mobile robotics requires software tools capable of integrating multi-agent control and communication, with ROS 2 (robot operating system) standing as a key standard. However, its use on low-cost educational platforms is limited by the constrained resources of microcontrollers. To address this, micro-ROS (an extension that brings ROS 2 capabilities to embedded devices) was employed. This work focused on integrating micro-ROS into Pololu 3Pi+ mobile robots within the Robotat laboratory. The implementation was carried out on an ESP32-WROOM-32D microcontroller, using WiFi (UDP) communication and an MQTT bridge to interface with ROS 2, supporting up to ten simultaneous agents. The OptiTrack motion-capture system was integrated through Mocap4ros2 and an MQTT bridge for localization.

A point-to-point controller for the Pololu 3Pi+ was developed in ROS 2, combining a PID controller for orientation with an exponential approach for position-error reduction. This controller was implemented in both single-agent and multi-agent versions, and successfully tested with nine robots operating simultaneously. The results validate the feasibility of integrating ROS 2 and micro-ROS on the Pololu 3Pi+ within the Robotat ecosystem, establishing a modular and scalable architecture. This provides a solid foundation for future development of more advanced control strategies and larger collaborative projects using these robots.

Keywords: mobile robotics, ROS 2, micro-ROS, Pololu 3Pi+, embedded systems, control systems.

CAPÍTULO 1

Introducción

La robótica móvil es un campo en constante evolución que demanda herramientas de software capaces de integrar distintos niveles de control y comunicación entre agentes. En la actualidad, uno de los estándares más relevantes en el ámbito académico e industrial es ROS 2 (robot operating system), que proporciona un marco modular y escalable para el desarrollo de aplicaciones robóticas distribuidas. Sin embargo, la adopción de este ecosistema en plataformas educativas y de bajo costo enfrenta limitaciones, particularmente cuando se requiere emplear microcontroladores con recursos reducidos. En este contexto, micro-ROS surge como una extensión que permite trasladar las funcionalidades de ROS 2 a dispositivos embebidos como el ESP32, ampliando de manera significativa el espectro de aplicaciones posibles.

El presente trabajo se centró en la integración de micro-ROS en robots móviles `Pololu 3Pi+`, dentro de la infraestructura del laboratorio Robotat. La finalidad fue establecer un puente robusto entre el entorno de desarrollo ROS 2 y los microcontroladores, de manera que los agentes puedan recibir variables de control y transmitir datos en tiempo real. De esta manera, se deja implementada una base de software e infraestructura que permitirá ampliar las fronteras del desarrollo robótico en la universidad, facilitando la incorporación de paquetes de control avanzados y fomentando la familiarización de estudiantes e investigadores con tecnologías que ya son un estándar en la industria.

La metodología empleada incluyó el diseño de firmware en PlatformIO, la configuración de nodos de comunicación mediante micro-ROS Agent, y la creación de un nodo de control dedicado capaz de operar en instancias paralelas para distintos agentes. El proceso de desarrollo implicó pruebas experimentales iterativas, particularmente en la parametrización de los controladores, aunque el énfasis principal del trabajo radicó en establecer la infraestructura de software y no en la optimización de los parámetros de control.

Los resultados más importantes de este trabajo consistieron en demostrar la factibilidad de integrar ROS 2 y micro-ROS en el `Pololu 3Pi+` dentro del ecosistema Robotat, así como en proveer una arquitectura modular y escalable que sienta las bases para futuros desarrollos. Con ello, se asegura que la universidad cuente con un punto de partida sólido

para la implementación de estrategias de control más sofisticadas, y se habilita la posibilidad de utilizar los `Pololu 3Pi+` en proyectos colaborativos de mayor alcance.

CAPÍTULO 2

Antecedentes

La Universidad del Valle de Guatemala (UVG) es pionera en el país y la región por su infraestructura en robótica. Un gran avance en la infraestructura de la universidad se debe al Robotat. Como detalla Perafán [1], el Robotat es un ecosistema robótico de captura de movimiento y comunicación inalámbrica que permite la interacción multiagente en una plataforma diseñada para rastrear la posición y orientación de agentes robóticos, desde manipuladores seriales hasta drones. Cuenta con una red inalámbrica que permite la comunicación interagente, el envío de comandos, lectura de sensores y lectura del sistema OptiTrack para captura de movimiento.

Estudiantes de la Universidad del Valle de Guatemala desarrollaron un explorador de orugas todo terreno, llamado Rover UVG. Este proyecto fue concebido antes de la creación del Robotat en la universidad. En sus inicios, el Rover UVG no contaba con un sistema de control, y carecía de cualquier tipo de autonomía. Fue hasta que Mencos [2], dotó de autonomía al Rover UVG, donde implementó Robot Operating System 2 (ROS 2) para el control del robot. Mencos creó una interfaz gráfica, con la capacidad de controlar y monitorear al robot, integrándolo a la red del Robotat, lo que facilita su operación remota y el acceso a las capacidades de captura de movimiento. Parte fundamental de este trabajo fue la selección de la computadora central del Rover UVG, capaz de ejecutar nativamente Ubuntu Linux 18.04. Esto mostro las demandas de hardware que trae consigo la implementación de ROS 2, ya que no cualquier equipo de procesamiento es adecuado para aplicaciones de robótica.

Con el tiempo, la universidad amplió su inventario robótico, adquiriendo diversos agentes, entre ellos los Pololu 3Pi+. Estos robots son controlados mediante el envío de comandos a la red del Robotat. En 2023, Pu [3] realizó su trabajo de graduación explorando diversas formas de simular y controlar a estos agentes. Dentro de su trabajo planteó el uso a futuro de ROS 2, el cual permite modularidad, escalabilidad y facilidad de desarrollo de aplicaciones robóticas avanzadas. Aunque el uso de ROS 2 es potenciador para estos agentes, no se puede ejecutar directamente desde los robots, ya que son controlados por microcontroladores, incapaces de correr Ubuntu Linux. La solución que Pu ofreció fue utilizar micro-ROS en el microcontrolador del Pololu 3Pi+ como nodo dentro de ROS 2. micro-ROS una versión

diseñada para sistemas embebidos de baja potencia, capas de comunicarse con una red ROS 2, que sí debe ser ejecutada desde una computadora o servidor.

Bin Li et al. [4] presentan el desarrollo de un robot móvil basado en un microcontrolador ESP32 (Figura 1), el cual es definido como un sistema de recursos extremamente limitados. El robot cuenta con motores, sensores para odometría, sensores ultrasónicos y comunicación wifi mediante el middleware DDS. Para el control del robot, así como para la implementación de algoritmos de cartografía y movimiento, se emplearon nodos desarrollados en ROS 2. Los autores demostraron que es posible implementar ROS 2 en un microcontrolador, habilitando no solo funciones de movimiento, sino también capacidades de mapeo a través del uso de micro-ROS.

Figura 1. Robot móvil controlado por ESP32 usando micro-ROS

Nota. La imagen muestra prototipo final del robot móvil controlado desde una aplicación ROS 2, mediante micro-Ros, para tener compatibilidad entre el microcontrolador ESP32 y la arquitectura de ROS 2. Esta imagen fue obtenida de [4].

Por su parte, Nguyen [5] desarrolló su tesis de maestría en colaboración con ABB, donde se implementaron múltiples microcontroladores ESP32 como nodos conectados a una red ROS 2. Cada microcontrolador se encargaba de gestionar tareas específicas dentro del robot manipulador de base móvil mYuMi (Figura 2). En este trabajo se evaluaron las capacidades de los microcontroladores ejecutando micro-ROS en condiciones de operación en tiempo real, obteniéndose resultados favorables. Nguyen concluyó que el uso de estos microcontroladores, en conjunto con micro-ROS, permite controlar robots complejos en entornos de operación con tiempos de respuesta cercanos al tiempo real.

Estos trabajos demuestran el potencial de ROS 2 y micro-ROS en sistemas de recursos limitados, como los basados en microcontroladores ESP32, aplicados en robots móviles. A partir de estos antecedentes, se propone implementar una solución similar en los robots Pololu 3Pi+, desarrollando una interfaz gráfica de control y monitoreo inspirada en la de Mencos, que permita extender las capacidades del robot hacia aplicaciones avanzadas, como las presentadas por Nguyen y Bin Li et al.

Figura 2. Robot manipulador de base móvil con integración de ESP32

Nota. La imagen muestra el robot de base móvil de ABB con colaboración de Nguyen, quien incluyó microcontroladores ESP32 para ejecutar diversas tareas, haciendo uso de micro-ROS. Esta imagen fue obtenida de [5].

CAPÍTULO 3

Justificación

El desarrollo de sistemas robóticos requiere una integración eficiente entre hardware y software, para facilitar la implementación de algoritmos y aprendizaje de tecnologías. El robot operating system 2 (ROS 2) se ha consolidado como un estándar global en la robótica, tanto a nivel académico como industrial, proporcionando una plataforma modular y escalable que permite la conexión entre nodos y dispositivos.

El control mediante ROS 2 para los robots Pololu 3Pi+ representa una oportunidad para ampliar sus capacidades y facilitar la enseñanza y evaluación de algoritmos robóticos. Esta implementación brindará a los estudiantes de ingeniería mecatrónica, en los cursos de robótica, una experiencia práctica en el uso de tecnologías de desarrollo robótico, permitiéndoles trabajar con un estándar utilizado en la industria e investigación. El uso de ROS 2, junto con la implementación del ecosistema del Robotat permitirá el desarrollo acelerado, modular y escalable de aplicaciones con robots diferenciales de recursos computacionales limitados.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar e implementar infraestructura de software para el control y monitoreo de agentes robóticos Pololu 3Pi+ mediante ROS 2 dentro del ecosistema Robotat

4.2. Objetivos específicos

- Implementar un nodo de ROS 2 para micro-ROS que gestione la comunicación con el ESP32 del agente robótico Pololu 3Pi+, permitiendo la ejecución de comandos de movimiento para el funcionamiento del robot.
- Desarrollar rutinas en software para el control y monitoreo del robot desde ROS 2.
- Integrar el uso del sistema de captura de movimiento con la infraestructura de ROS 2 para el desarrollo de aplicaciones de control en lazo cerrado.

CAPÍTULO 5

Alcance

El presente trabajo se enfoca en establecer una base de herramientas de *software* para el desarrollo de aplicaciones robóticas en los agentes robóticos Pololu 3Pi+ mediante el uso de ROS 2. Como punto de partida se retoma la propuesta de Jonathan Pu [3] de implementar micro-ROS en el ESP32 para el desarrollo de aplicaciones robóticas avanzadas.

El sistema a utilizar está conformado por un robot Pololu 3Pi+ controlado mediante el microcontrolador ESP32-WROOM-32D. Su uso se limita al laboratorio Robotat de la Universidad del Valle de Guatemala. Dicho laboratorio cuenta con el sistema de captura de movimiento OptiTrack, así como la infraestructura de red necesaria para la comunicación. Para el desarrollo del *firmware* del ESP32 se emplea PlatformIO y para el entorno de desarrollo se utiliza ROS 2 Humble Hawksbill sobre Ubuntu 22.04 LTS.

El alcance del proyecto incluye la implementación de micro-ROS en el ESP32, de modo que el microcontrolador accione los actuadores del robot en base a las referencias de velocidad recibidas de la computadora central con ROS 2. En cuanto al sistema de captura de movimiento, su uso se limita a la suscripción de tópicos que contienen los datos de pose generados por el sistema Optitrack. Este trabajo no se enfoca en el desarrollo de la integración entre las cámaras de captura de movimiento y ROS 2, sino en el uso de la información para el control y monitoreo del robot. El desarrollo de la integración entre el sistema de captura de movimiento y ROS 2 corresponde al trabajo de graduación "Implementación de infraestructura para el control de enjambres robóticos con ROS 2 y captura de movimiento dentro del ecosistema Robotat" de Solis López [6], con quien se trabajo en conjunto para el desarrollo de infraestructura en el laboratorio Robotat de la Universidad del Valle de Guatemala.

Este trabajo no tiene como finalidad desarrollar o implementar algoritmos de control avanzado, sino implementar un controlador funcional que permita validar la arquitectura propuesta y servir como referencia para futuros desarrollos en ROS 2. Se busca habilitar a los robots de mayor capacidad computacional mediante el control generado desde ROS 2 y la actuación del robot por micro-ROS, dejando la infraestructura de *software* lista para que futuros proyectos puedan desarrollar aplicaciones de robótica más complejas.

El alcance de este proyecto se encuentra limitado por la disponibilidad del laboratorio Robotat para realizar pruebas. El laboratorio es un espacio de uso compartido por estudiantes de múltiples asignaturas, lo cual limita el tiempo disponible a escasos horarios. Asimismo, el Robotat se utiliza de manera compartida con otros alumnos que lo requieren, reduciendo la disponibilidad del espacio completo para realizar pruebas de funcionamiento. Aunado a lo anterior, su uso está sujeto a factores externos a la universidad, como feriados, cierres de calles, la situación política del país o incluso actividad sísmica. Por consiguiente, los procesos de validación quedan condicionados al acceso al laboratorio.

CAPÍTULO 6

Marco teórico

Este trabajo de graduación planteó el desarrollo de *software* diseñado para la integración con el *hardware* de los agentes robóticos Pololu 3Pi+ utilizados en la Universidad del Valle de Guatemala. Estos robots fueron modificados para que el control del robot fuera generado por el microcontrolador ESP32, el cual recibe datos de velocidad de ruedas y se encarga de enviarlos al robot de forma serial. Las herramientas de *software* implementadas en este proyecto abordaron temas de robótica y sistemas de control desarrollados en ROS 2, por lo que, a continuación, se definen y explican los conceptos y el uso de los componentes fundamentales de este proyecto.

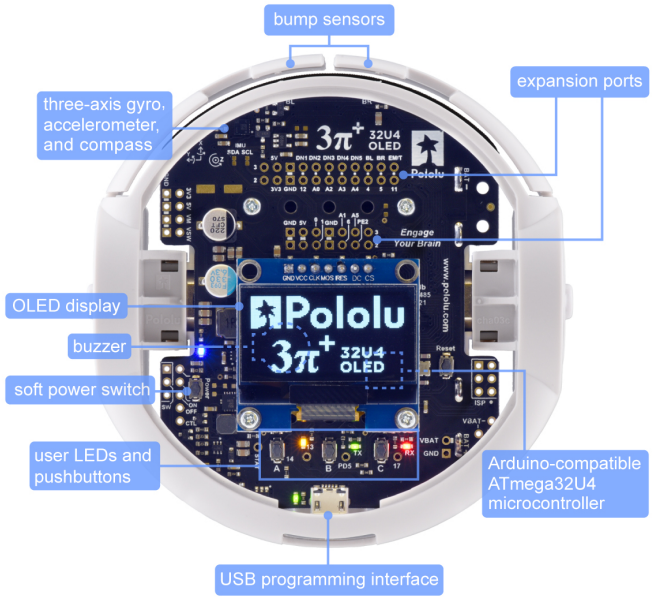
6.1. Pololu 3Pi+

El Pololu 3Pi+ es un robot móvil diferencial de 9.7 cm de diámetro, diseñado para aplicaciones educativas y desarrollo en robótica. Es programable a través de una interfaz USB y es compatible con el gestor de arranque (*bootloader*) de Arduino, brindando flexibilidad en su programación y configuración. Está controlado por el microcontrolador ATmega32U4, que permite un manejo eficiente de sus periféricos y recursos [7].

En las Figuras 3 y 4 se muestra la ubicación general de su *hardware*. El Pololu 3Pi+ cuenta con dos puentes H para el control de sus motores, así como *encoders* de cuadratura para implementar control en lazo cerrado. Incorpora una unidad de medición inercial (IMU) con acelerómetro, giroscopio y magnetómetro, además de sensores reflectivos inferiores, sensores de choque laterales, tres botones, LEDs, buzzer y pines de expansión que permiten conectar módulos adicionales [7].

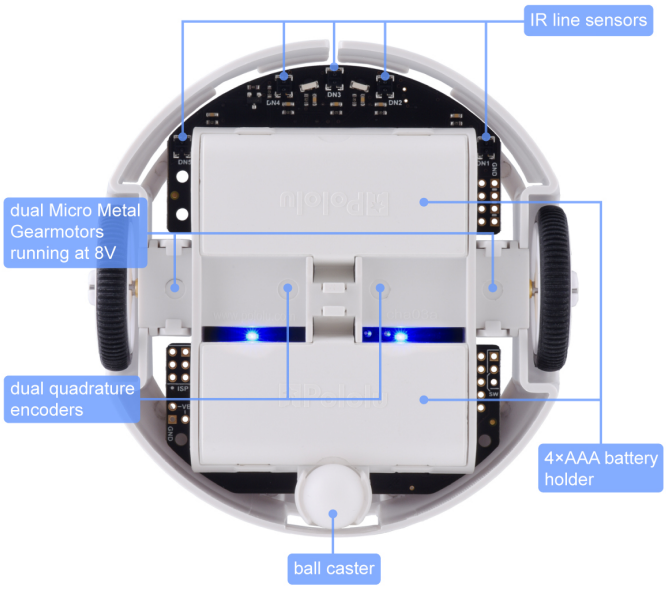
El fabricante ofrece tres versiones para el Pololu 3Pi+: *Standard*, *Turtle*, *Hyper*. Las diferencias entre versiones son la velocidad máxima de las ruedas. En la Universidad del Valle de Guatemala se cuenta con la versión *standard*, la cual permite una velocidad de hasta 1.5 m/s y es la versión de nivel de dificultad intermedia para su control [7].

Figura 3. *Hardware* del robot Pololu 3Pi+, vista superior



Nota. La imagen muestra la vista superior del robot, donde se señalan los principales sensores y dispositivos I/O. Imagen obtenida de [7].

Figura 4. *Hardware* del robot Pololu 3Pi+, vista inferior



Nota. La imagen muestra la vista inferior del robot, donde se señalan componentes importantes visibles desde abajo. Imagen obtenida de [7].

6.2. ESP32

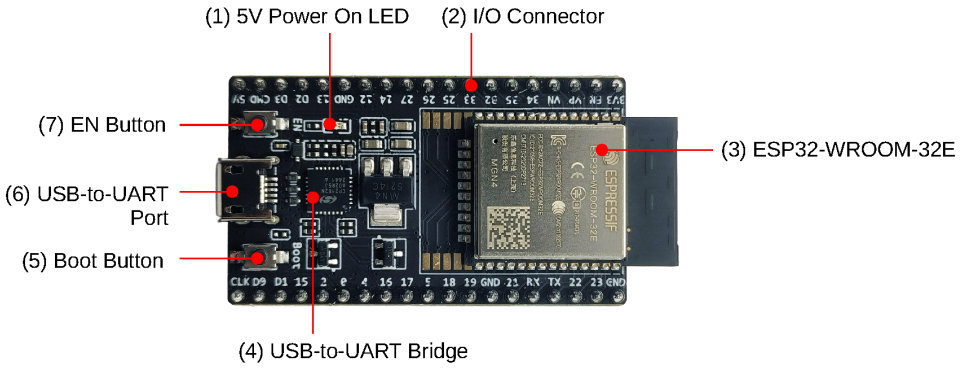
El ESP32 es una familia de microcontroladores de bajo costo y consumo, desarrollada por Espressif Systems y fabricada con tecnología de 40 nm. Integra conectividad wifi y Bluetooth [8]. El ESP32 utiliza procesadores Tensilica Xtensa LX6, disponibles en versiones de uno o dos núcleos, con frecuencias de operación entre 160 MHz y 240 MHz. Este también incluye un coprocesador de ultra baja potencia (ULP) que permite operaciones en modo de bajo consumo, y cuenta con aproximadamente 520 KiB de SRAM interna [9].

La familia de microcontroladores ESP32 es ideal para aplicaciones del internet de las cosas (IoT) por la comunicación inalámbrica integrada. Los periféricos que lo componen son ADC SAR de 12 bits (hasta 18 canales), DACs, sensores táctiles, interfaces I^2C, I^2S, SPI, UART, controladores SD/SDIO, PWM, Ethernet MAC, CAN, infrarrojos, además de un sensor Hall integrado [9]. Por estas características es un microcontrolador muy popular en la actualidad. La razón por la cual se eligió el ESP32 es por su soporte oficial con Micro-ROS, el cual corre sobre FreeRTOS y permite comunicación serial y wifi, ideal para aplicaciones con ROS 2 [10].

Para este trabajo de graduación se utilizó la placa de desarrollo ESP32_Devkitc_v4, con el modelo ESP32-WROOM-32D. Dicha placa de desarrollo fue diseñada con *pin headers* en ambos lados para una fácil interacción con los puertos de entrada y salida que ofrece el microcontrolador, permitiendo el uso de cables o la conexión directa en un *protoboard* [11].

En la Figura 5 se muestran los componentes principales para la interacción con la placa de desarrollo, la descripción detallada se presenta en el Cuadro 1. Asimismo, en la Figura 6 se muestra la configuración de los pines para la conexión. Es importante mencionar que esta configuración solo es válida para esta placa de desarrollo, por lo que se puede tener incompatibilidad si se usa otra.

Figura 5. Placa de desarrollo ESP32_Devkitc_v4



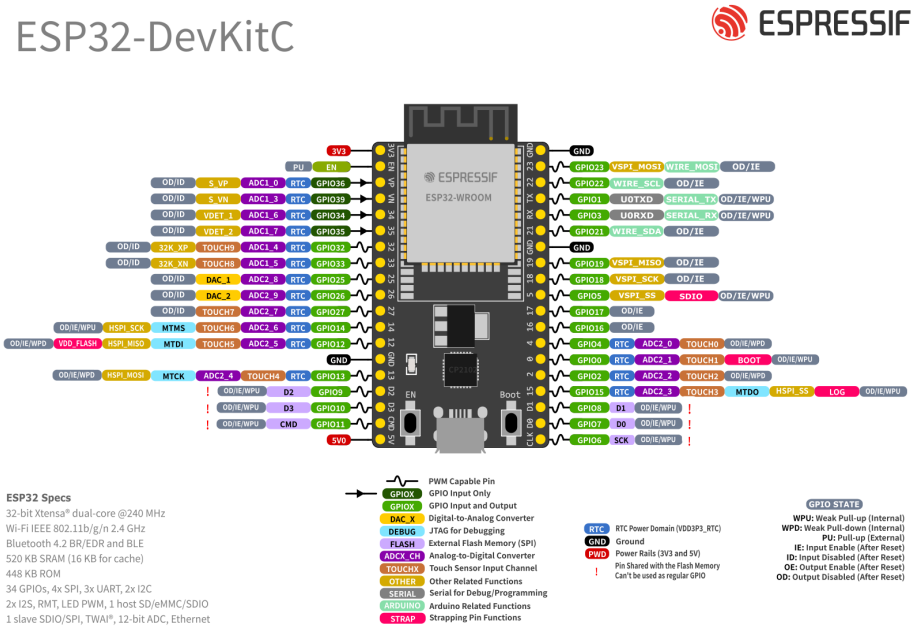
Nota. La imagen muestra la placa de desarrollo, donde se señalan los componentes importantes observados desde la vista superior. Imagen obtenida de [11].

Cuadro 1. Componentes principales del ESP32-DevKitC V4

N°	Componente clave	Descripción
1	LED de alimentación	Se enciende al recibir alimentación por el puerto USB o por una fuente externa de 5V.
2	Conector de entradas y salidas	La mayoría de pines están marcados en la placa, pero se puede programar para acceder a otras funciones: PWM, ADC, DAC, I2C, SPI, etc.
3	ESP32-WROOM-32E	El microcontrolador, esta placa puede tener diversas variantes, pero el que se usó en este trabajo fue el ESP32-WROOM-32D.
4	Puente USB a UART	Circuito integrado para la interconexión USB UART.
5	Botón de descarga	Se debe presionar para descargar *firmware* al ESP32, no siempre es necesario, dependerá del IDE que se utilice.
6	Puerto USB a UART	Puerto para alimentación y comunicación entre una computadora y el microcontrolador.
7	Botón de reseteo	Sirve para resetear a valores originales el *firmware* activo.

Nota. Descripción sobre los componentes clave de la placa de desarrollo de la Figura 5. Esta tabla se baso en la tabla proporcionada en [11].

Figura 6. Configuración de pines del ESP32_Devkitc_v4



Nota. La imagen muestra la placa de desarrollo, donde se señala la distribución de sus pines de entrada y salida. Imagen obtenida de [11].

6.3. Sistema de control y algoritmos robóticos

Los sistemas de control constituyen la base del funcionamiento de los robots móviles. Un sistema puede entenderse como un conjunto de componentes interconectados que reciben entradas, procesan información y generan salidas [12]. Cuando se busca que la salida siga un comportamiento deseado, se implementa un *sistema de control*, entendido como el conjunto de métodos que regulan el comportamiento dinámico de la planta o proceso físico [13]. En robótica, el control es indispensable para garantizar estabilidad, precisión en el movimiento y adaptación frente a perturbaciones externas [14].

En términos generales, los sistemas de control pueden clasificarse en lazo abierto y lazo cerrado, siendo estos últimos los más relevantes en robótica móvil por su capacidad de compensar errores e incertidumbres [13]. En el lazo cerrado, la salida se compara con la referencia deseada y el controlador ajusta su acción en función del error, lo que permite reducir perturbaciones externas y mejorar la precisión en el movimiento [14]. Sobre esta base se desarrollan los algoritmos que permiten a un robot percibir, planificar y actuar en su entorno, como los controladores PID, los métodos de localización y mapeo, y las estrategias de navegación punto a punto [12], [13], [15], [16].

6.3.1. Controlador PID

El controlador proporcional–integral–derivativo (PID) es uno de los algoritmos más utilizados en robótica móvil debido a su simplicidad y efectividad. Combina tres acciones de control: proporcional, integral y derivativa, que permiten corregir el error instantáneo, eliminar el error en estado estacionario y anticipar cambios bruscos, respectivamente. En robots diferenciales como el Pololu 3Pi+, se aplica comúnmente al control de velocidad de las ruedas [13].

La sintonización de sus ganancias es crítica para lograr un balance entre estabilidad, rapidez y precisión. Aunque existen controladores más avanzados, el PID sigue siendo una solución práctica en plataformas educativas y experimentales [14].

6.3.2. Mapeo y localización en entornos robóticos

La capacidad de un robot para desplazarse de manera autónoma depende de su habilidad para conocer su posición y, en muchos casos, construir un mapa del entorno, proceso conocido como *mapeo y localización* [12]. El enfoque más completo es el *simultaneous localization and mapping* (SLAM), que permite crear un mapa mientras se estima la posición [15]. Sin embargo, en entornos de laboratorio es común emplear sistemas externos de alta precisión como el *motion capture*.

En este trabajo, la localización se realizó con el sistema OptiTrack, que utiliza múltiples cámaras infrarrojas para rastrear marcadores reflejantes en tiempo real, logrando mediciones con errores de pocos milímetros, adecuados para validar algoritmos de control y navegación [16].

6.3.3. Estrategias de navegación y control punto a punto

La navegación robótica reúne las técnicas que permiten a un robot alcanzar un objetivo dentro de un entorno. Una estrategia básica es el control punto a punto, en el cual el robot recibe como referencia una posición objetivo y ajusta sus acciones de control para alcanzarla. En robots diferenciales, esta estrategia se implementa mediante diversos controladores de lazo cerrado. En este trabajo, el controlador empleado fue el PID para el comportamiento angular. [12], [15].

El control punto a punto destaca por su simplicidad y efectividad en escenarios estructurados, aunque en entornos dinámicos o desconocidos requiere complementarse con algoritmos de evasión de obstáculos o planificación global. Representa, por tanto, un paso fundamental hacia la navegación autónoma robusta [12], [15].

6.4. Robot operating system (ROS 2)

ROS 2 o robot operating system es un set de librerías de *software* y herramientas útiles para la construcción de aplicaciones robóticas. Es ampliamente usado en laboratorios de desarrollo e investigación. Cuenta con cobertura en diferentes rubros de robótica: agricultura, vehículos autónomos, logística de fábricas, robótica de servicio, industria pesada, drones y exploración planetaria [17].

Es una herramienta de código abierto que funciona con un marco de trabajo que permite gestionar la comunicación entre diferentes paquetes de la aplicación. ROS 2 cuenta con paquetes para múltiples aplicaciones, permitiendo al desarrollador crear comportamientos complejos con menos esfuerzo y tiempo. Además, cuenta con herramientas de línea de comandos para construir e inspeccionar la aplicación robótica, así como el flujo de comunicación. Algunas de sus herramientas más comunes son: generadores de gráficos y simuladores para gemelos digitales como Gazebo Sim [18].

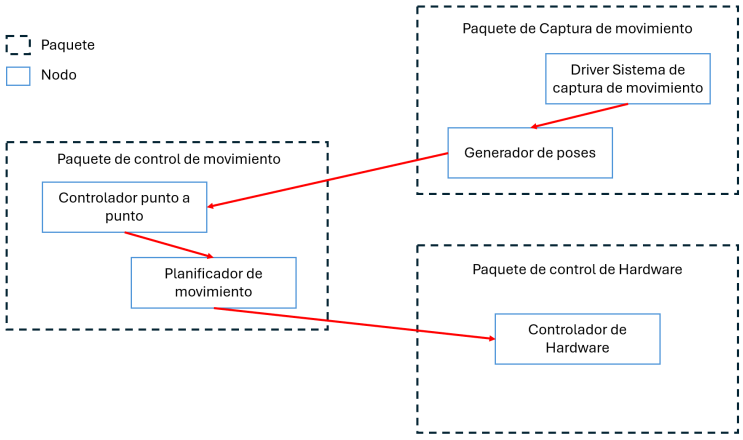
Las capacidades de este *software* son muy amplias, ya que facilita la integración de actuadores, sensores y sistemas de control, permitiendo que cada parte de la aplicación funcione de forma modular [17]. Para hacer posible esto, ROS 2 se maneja por medio de paquetes. Un paquete es una sub parte de la aplicación robótica independiente de las demás. Cada paquete se encarga de realizar una acción en concreto o un grupo de tareas relacionadas. Cada paquete está compuesto por nodos, los cuales son programas que ejecutan acciones más específicas de una tarea [18].

La comunicación entre sus diferentes nodos, resulta bastante sencilla al utilizar *topics* (tópicos) de ROS 2. Un tópico es un canal de comunicación que está dentro de la red ROS 2, a este canal de comunicación se le puede publicar o suscribir desde diferentes nodos de forma automática. Esto quiere decir, sin identificadores, claves de acceso o compatibilidad de protocolos de comunicación, etc. Cada tópico puede manejar un tipo de dato de ROS 2, por lo que en su creación y uso se debe considerar el propósito del nodo, para elegir el tipo de dato correcto. De esta manera, se facilita la comunicación directa entre todo tipo de nodos, desde controladores de *hardware* hasta algoritmos robóticos avanzados [18].

Además de tópicos, ROS 2 cuenta con *services* (servicios) y `actions` (acciones). Estos son mecanismos de comunicación más complejos que los tópicos, permitiendo mayor control sobre qué, cuándo y cómo se envían o reciben datos. Cuando se requieren acciones tipo cliente/servidor se usan *services*. El servidor espera la solicitud de un cliente para ejecutar una tarea y enviar la respuesta al cliente. Las solicitudes se pueden hacer desde nodos individuales o desde la línea de comandos. En caso de requerir completar tareas prolongadas que toman mayor tiempo de ejecución se utilizan `actions`. Cada acción recibe una tarea, y muestra el resultado obtenido y la retroalimentación durante la ejecución. Tanto las acciones como servicios, requieren un nombre, tipo y estructura de datos para funcionar de forma adecuada [18].

La organización de programas y código dentro de la aplicación queda a discreción del desarrollador, sin embargo, se debe seguir un orden y lógica para agrupar nodos dentro de los paquetes con el fin de desarrollar paquetes modulares e implementables en diversas aplicaciones. En la Figura 7 se muestra un ejemplo de la organización de paquetes en una aplicación robótica.

Figura 7. Estructura propuesta para organización de paquetes y nodos



Nota. La imagen muestra una estructura y organización de paquetes y nodos, como ejemplo del flujo de trabajo dentro de ROS 2. Elaboración propia.

Si bien ROS 2 se encuentra disponible para Ubuntu, Windows y macOS, sólo Ubuntu y Windows tienen soporte *Tier 1*, mientras que macOS tiene soporte *Tier 3*. Aún así, la comunidad de ROS 2 muestra tener una mejor experiencia en Ubuntu que Windows, ya que corre sin errores y no tiene problema con herramientas 2D y 3D, por eso este es el sistema operativo por defecto para desarrollo con ROS 2 [18].

Las aplicaciones robóticas, así como ROS 2 tiene como requisito hardware con memoria y capacidad de procesamiento elevadas. Por dicha razón, ROS 2 no puede ejecutarse en microcontroladores, solamente con computadoras o servidores con el sistema operativo Ubuntu. Esto es una limitante porque los microcontroladores son utilizados en casi todos los productos robóticos, ya que estos tienen acceso directo al *hardware*, trabajan en tiempo real estricto, con baja latencia, menor costo y presentan ahorro de energía frente a otros equipos de cómputo [19].

Micro-ROS es una adaptación de ROS 2 diseñada para funcionar en dispositivos con recursos de hardware limitados, como los microcontroladores. Bajo estas restricciones, mantiene compatibilidad con la arquitectura de ROS 2 gracias al uso de DDS-XRCE, un protocolo optimizado para comunicación en tiempo real en sistemas embebidos. La ventaja principal que ofrece es integrar directamente el control del hardware dentro del ecosistema de ROS 2, logrando un funcionamiento eficiente, de bajo consumo energético y adecuado para aplicaciones con restricciones de memoria y procesamiento [19].

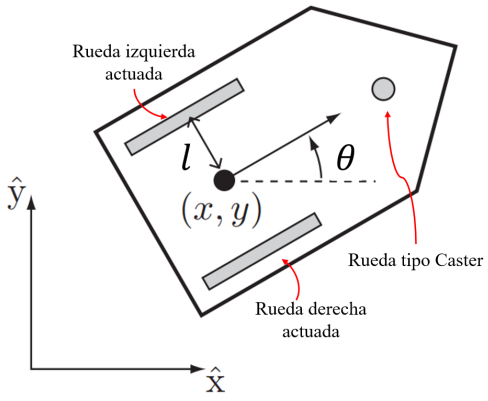
De esta manera, es posible desarrollar algoritmos robóticos complejos en una computadora o servidor utilizando ROS 2. Al mismo tiempo, el microcontrolador actúa únicamente como un nodo dentro de la aplicación. Su función es ejecutar solo las tareas de control directo y de más bajo nivel. Esto reduce significativamente la carga computacional del microcontrolador sin sacrificar la integración con el sistema general [19].

Micro-ROS tiene soporte para diversos microcontroladores y esto a su vez depende del entorno de programación. Se puede programar en Arduino IDE, PlatormIO, Espressif IDE, entre otros [20]. En la lista de microcontroladores que soportan micro-ROS aparece ESP32, el cual fue elegido para este trabajo por su uso previo en la universidad para el control de los agentes Pololu 3Pi+.

6.5. Modelado y control de robots diferenciales

Los robots móviles con ruedas constituyen uno de los mecanismos más utilizados en robótica debido a su simplicidad, eficiencia energética y facilidad de control. Entre los distintos tipos de configuraciones con ruedas se destacan los robots diferenciales, los cuales se caracterizan por poseer dos ruedas alineadas, cada una con accionamiento independiente, además de una tercera rueda de apoyo del tipo *caster* o esférica que únicamente cumple la función de mantener la estabilidad del robot en posición horizontal.

Figura 8. Modelo geométrico de un robot diferencial



Nota. La imagen muestra el modelo geométrico de un robot diferencial, donde se observan las partes que lo componen, así como las variables que describen su geometría y orientación. Esta imagen se basó en [15], con modificaciones de lenguaje y nomenclatura.

Una de las particularidades de los robots diferenciales es que se consideran sistemas no holonómicos. Esto significa que sus restricciones de movimiento no pueden expresarse únicamente en términos de coordenadas de posición, sino que dependen de las velocidades de las ruedas. En otras palabras, mientras las ruedas no se deslizan y se mantienen en rodadura, el movimiento del robot está determinado por las velocidades angulares de sus ruedas, pero su posición no puede deducirse de manera directa únicamente a partir de la configuración instantánea del sistema [15].

En la Figura 8 se muestra el esquema de un robot diferencial, en el cual se ilustran las variables principales del modelo: la distancia entre ruedas l, la orientación θ y la posición del centro del eje de las ruedas en el plano xy.

De acuerdo con el modelo cinemático propuesto en [15], la relación entre las velocidades de las ruedas y la configuración del robot está dada por la ecuación (1).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{r}{2} \begin{bmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_\ell \end{bmatrix}. \tag{1}$$

De este modelo se derivan las expresiones (2) y (3) para las velocidades del robot en el plano, así como la velocidad angular del robot (4).

$$\dot{x} = \frac{r}{2}\left(\dot{\phi}_r + \dot{\phi}_\ell\right)\cos\theta, \tag{2}$$

$$\dot{y} = \frac{r}{2}\left(\dot{\phi}_r + \dot{\phi}_\ell\right)\sin\theta, \tag{3}$$

$$\dot{\theta} = \frac{r}{2l}\left(\dot{\phi}_r - \dot{\phi}_\ell\right). \tag{4}$$

6.5.1. Control basado en el modelo del uniciclo

A partir del modelo del robot diferencial se procede a diseñar su control. Sin embargo, debido al carácter no holonómico del robot, el sistema no es completamente controlable en el espacio de estados [12]. Para abordar este problema, se emplea una aproximación clásica que consiste en representar al robot diferencial mediante el modelo del *uniciclo*, el cual constituye una simplificación que, aunque supone un único punto de apoyo, presenta un comportamiento cinemático equivalente al del robot diferencial [15].

El modelo del uniciclo se expresa como:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}, \tag{5}$$

donde v representa la velocidad lineal y w la velocidad angular del robot.

Controladores de orientación y velocidad lineal

Para garantizar que el robot se oriente correctamente hacia el objetivo, se implementó un controlador PID sobre la velocidad angular w. Dicho controlador está definido por la ecuación (6):

$$w = PID(e_o) = K_p e_o + K_i \int_0^t e_o(\tau)\, d\tau + K_d \dot{e}_o, \tag{6}$$

donde el error de orientación e_o se define por la ecuación (7):

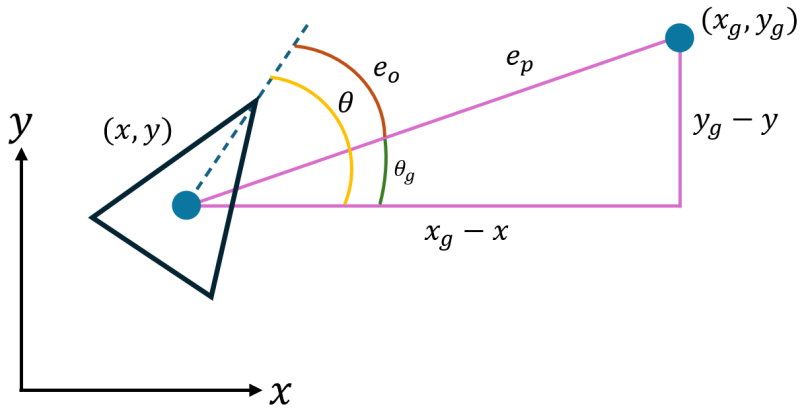
$$e_o = \arctan 2 \left(\frac{\sin(\theta_g - \theta)}{\cos(\theta_g - \theta)} \right). \tag{7}$$

El control de velocidad lineal v se realizó mediante un acercamiento exponencial, el cual busca reducir el error de posición e_p de forma rápida y progresiva a medida que el robot se aproxima al objetivo. Este control se presenta en la ecuación (8), donde v_0 representa la velocidad máxima permitida y α es un parámetro que regula la suavidad del frenado conforme el robot se acerca al objetivo.

$$v = k(e_p)e_p = v_0 \frac{1 - e^{-\alpha e_p^2}}{e_p}. \tag{8}$$

En la Figura 9 se muestra el esquema del modelo del uniciclo con respecto a un punto objetivo $g = (x_g, y_g)$, indicando la orientación deseada θ_g, el error de orientación e_o, error de posición e_p, dentro del marco de referencia (x, y) del robot.

Figura 9. Modelo del uniciclo orientado hacia un punto objetivo g



Nota. La imagen muestra el modelo del uniciclo, donde se observa el marco de referencia del robot, así como los errores de posición y orientación hacia un punto objetivo. Elaboración propia.

Regresando a las Ecuaciones (2), (3) y (4), se observa la velocidad lineal v del robot y a la velocidad angular w. De esta manera, se tienen las expresiones (9) y (10) que corresponden a las velocidades v y w del robot:

$$v = \frac{r}{2}\left(\dot{\phi}_r + \dot{\phi}_\ell\right), \tag{9}$$

$$w = \frac{r}{2l}\left(\dot{\phi}_r - \dot{\phi}_\ell\right), \tag{10}$$

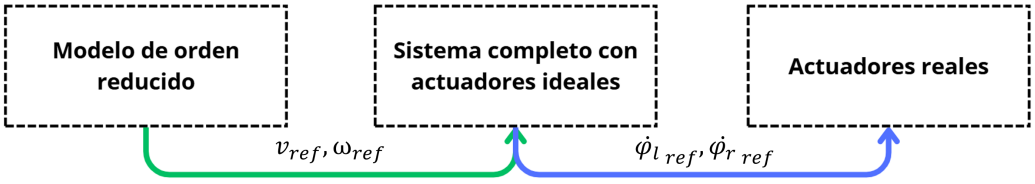
Estas expresiones forman un sistema de ecuaciones que permite obtener las velocidades angulares de cada rueda, como lo expresan las ecuaciones (11) y (12).

$$\dot{\phi}_r = \frac{v_{\text{ctrl}} + l\, w_{\text{ctrl}}}{r}, \tag{11}$$

$$\dot{\phi}_\ell = \frac{v_{\text{ctrl}} - l\, w_{\text{ctrl}}}{r}. \tag{12}$$

De esta manera, se utilizó el modelo del uniciclo para obtener las variables v y w controladas. Estas variables se implementaron en el sistema mediante el modelo del robot diferencial (1), el cual permite relacionar directamente las variables de control con las velocidades angulares de cada rueda. En consecuencia, el sistema de control global del robot diferencial puede entenderse como un control jerárquico: en un primer nivel se regulan v y w, que posteriormente se mapean a velocidades de rueda, y finalmente, cada rueda es controlada por un controlador de velocidad local (integrado en el Pololu 3pi+), así como se muestra en la Figura 10.

Figura 10. Arquitectura de control jerárquico del robot diferencial



Nota. La imagen muestra la relación entre el control del modelo de uniciclo (modelo de orden reducido) y el control del modelo real del robot diferencial. Elaboración propia.

CAPÍTULO 7

Desarrollo del nodo micro-ROS en el robot Pololu 3Pi+

A continuación se presenta la metodología empleada para el desarrollo del nodo micro-ROS, abarcando desde la configuración de la computadora con ROS 2 hasta la creación del *firmware* en el ESP32. Asimismo, se describen los criterios de evaluación utilizados para corroborar una implementación exitosa.

7.1. Metodología de trabajo para el desarrollo de *firmware* del ESP32

Dado que el desarrollo de *firmware* del ESP32 con micro-ROS está vinculado directamente con el uso de ROS 2, es necesario abordar primeramente la preparación del entorno de desarrollo. Esto incluye la etapa inicial de selección de sistema operativo y configuración de ROS 2.

7.1.1. Selección de distribución de ROS 2

Para la instalación de ROS 2 se debe elegir la versión de Ubuntu correcta para el desarrollo en ROS 2, lo que a su vez depende de la distribución de robot operating system 2 utilizada. Las distribuciones consisten en lanzamientos de paquetes que incorporan mejoras y correcciones, lo cual es desarrollado por la comunidad y enriquece las capacidades del software en cada iteración. Existen distribuciones con soporte de corto y largo plazo, con duración de uno o cinco años, respectivamente [18].

En el Cuadro 2 se presentan las distribuciones más recientes, junto con sus fechas de lanzamiento y fin de soporte, información obtenida del sitio oficial en [21]. No siempre resulta

recomendable utilizar la distribución más reciente, ya que deben considerarse factores como la vida remanente de soporte, la cantidad de documentación disponible y la compatibilidad con las herramientas asociadas a ROS2.

Se seleccionó la distribución Humble Hawksbill, comúnmente conocida como Humble. Esta distribución fue la primera en contar con soporte a largo plazo, lo cual constituyó un factor determinante en la elección, ya que garantiza mayor vigencia de las aplicaciones desarrolladas. Además, Humble dispone de una documentación sólida, tanto oficial como generada por la comunidad, incluyendo desarrolladores y creadores de contenido didáctico. Asimismo, en [3] se realizaron pruebas piloto con Humble para la implementación de micro-ROS, lo cual refuerza esta decisión. De esta forma, se asegura la continuidad en la línea de desarrollo, una vida útil prolongada de soporte y suficiente documentación para la integración de herramientas adicionales en ROS 2 [21].

Cuadro 2. Distribuciones recientes de ROS 2 y sus periodos de soporte

Distribución	**Fecha de lanzamiento**	**End of Life**
Foxy Fitzroy	5 de junio de 2020	20 de junio de 2023
Galactic Geochelone	23 de mayo de 2021	9 de diciembre de 2022
Humble Hawksbill	23 de mayo de 2022	mayo de 2027
Jazzy Jalisco	23 de mayo de 2024	mayo de 2029
Kilted Kaiju	23 de mayo de 2025	diciembre de 2026

Nota. Se listan las cinco distribuciones más recientes de ROS2, ordenadas de la más antigua a la más reciente. Elaboración propia con base en la documentación oficial [21].

Una vez definida la distribución a implementar, se procedió con la elección de la versión de Ubuntu correspondiente. En [22] se indica que la versión adecuada para ROS 2 Humble es Ubuntu 22.04 (Jammy). Por lo tanto, el siguiente paso para la preparación del entorno consistió en descargar este sistema operativo, ya fuera para instalarlo en una computadora dedicada, en arranque dual o en una máquina virtual. Con el fin de iniciar el desarrollo de forma ágil, desde cualquier equipo que cumpla los requisitos computacionales del Cuadro 3, se optó por utilizar una máquina virtual para alojar el entorno de desarrollo.

Cuadro 3. Requisitos mínimos de cómputo para ejecutar Ubuntu 22.04 con ROS 2 Humble

Recurso	**Requisito mínimo**	**Recomendado**
Procesador	2 núcleos	4 núcleos o superior
Memoria RAM	4 GB	8 GB
Almacenamiento	30 GB libres	50 GB SSD

Nota. Los requisitos se presentan para el uso de Ubuntu 22.04 con ROS 2 Humble, sin incluir simuladores tridimensionales. Elaboración propia con base en [18]

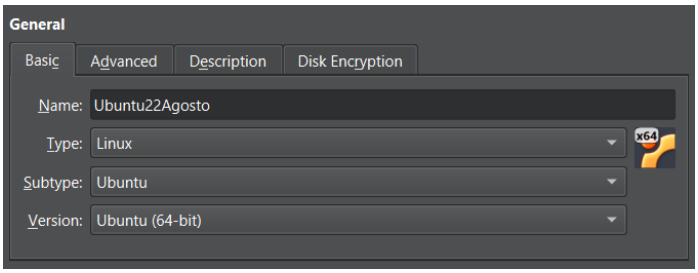
7.1.2. Configuración de máquina virtual Ubuntu

Como primer paso, se descargó el archivo .iso de Ubuntu 22.04 [23], seleccionando la imagen de escritorio (*desktop image*) y no la de servidor. Paralelamente, se descargó VirtualBox [24], un gestor de máquinas virtuales ampliamente utilizado. Al crear la máquina virtual, se recomienda asignar archivos bien identificados para su almacenamiento y otorgarle un nombre significativo.

Aunque Ubuntu es un sistema operativo optimizado para recursos computacionales, se siguieron configuraciones específicas en la máquina virtual con el fin de evitar inconvenientes.

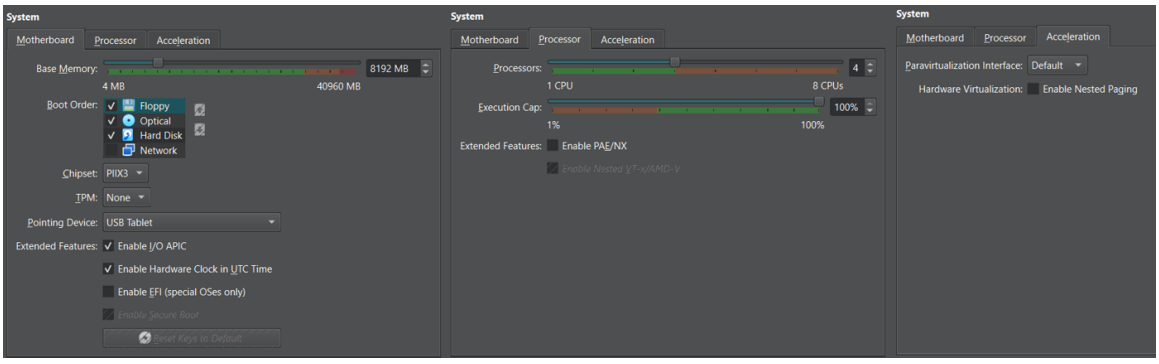
En la Figura 11 se presenta la información general de la máquina virtual creada para este proyecto. Se configuró el portapapeles de forma bidireccional en la pestaña *advanced*, permitiendo copiar y pegar texto entre la máquina principal y la virtual. La Figura 12 muestra todas las pestañas de configuración del sistema de la máquina virtual. Para equipos con 8 GB de memoria RAM, se recomienda asignar 4 GB a la máquina virtual; si la memoria RAM supera los 8 GB, se considera adecuado asignar 6 GB como mínimo. En cuanto a los procesadores, se asignó la mitad del total de núcleos de la máquina anfitriona. La Figura 13 muestra la configuración recomendada para la imagen y el video.

Figura 11. Configuración general de la máquina virtual



Nota. La imagen muestra una captura de pantalla con la información general básica de la máquina virtual creada para este proyecto. Elaboración propia.

Figura 12. Configuración del sistema de la máquina virtual

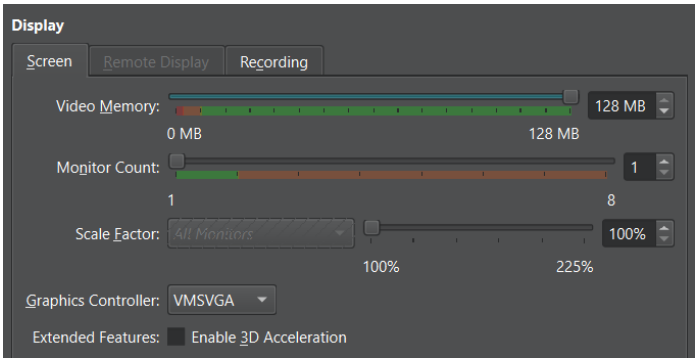


Nota. La imagen muestra una captura de pantalla con la información de sistema de la máquina virtual creada para este proyecto. Elaboración propia.

Durante la configuración se asignó un espacio de disco virtual de 50 GB, aunque el valor mínimo recomendado puede ser de 35 GB. Este tamaño permite instalar aplicaciones adicionales y puede modificarse en cualquier momento según las necesidades del proyecto.

Se configuró una carpeta compartida, lo que facilita el intercambio de archivos entre la máquina principal y la virtual, así como la realización de copias de seguridad. Esta opción, aunque no es indispensable, resultó ser útil y sencilla de implementar.

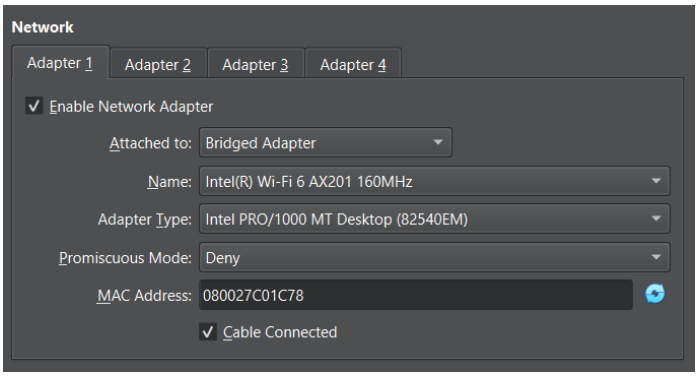
Figura 13. Configuración del *display* de la máquina virtual



Nota. La imagen muestra una captura de pantalla con la información del *display* de la máquina virtual creada para este proyecto. Elaboración propia.

La Figura 14 muestra la configuración de red utilizada, seleccionando la opción *bridged adapter*. Los demás campos de configuración no mencionados se dejaron con valores predeterminados, ya que no afectan de manera significativa el rendimiento de la máquina virtual.

Figura 14. Configuración de red de la máquina virtual



Nota. La imagen muestra una captura de pantalla con la configuración de red de la máquina virtual creada para este proyecto. Elaboración propia.

Tras completar la configuración, se procedió con la instalación de Ubuntu siguiendo el asistente de instalación. Se configuró el idioma inglés para evitar inconsistencias con la documentación disponible, que se encuentra mayoritariamente en inglés. Para la conexión a

internet se seleccionó la opción *use wired connection*, de modo que la máquina virtual accede a la red a través de la máquina principal. Esta configuración demostró ser estable y permitió recibir datos en ROS 2 sin inconvenientes.

Posteriormente, se actualizó el sistema operativo mediante la línea de comandos, siguiendo las instrucciones del Código 7.1:

```
1         sudo apt update
2         sudo apt upgrade
```

Código 7.1. Actualización de paquetes Ubuntu

Una vez actualizado el sistema operativo, se configuró e instaló *guest additions CD image* para mejorar la interacción con la máquina virtual. Esta herramienta permite ajustar automáticamente la ventana de la máquina virtual al tamaño de la pantalla y facilita copiar y pegar contenido entre la máquina principal y la virtual. Aunque no es estrictamente necesario, su instalación resulta recomendable para optimizar la experiencia de uso.

El procedimiento descrito se basó en lo explicado por Renard [18]. Si bien se siguió su metodología de manera general, se ajustaron la asignación de recursos y otras configuraciones con el fin de mejorar el rendimiento y adaptarlo a un equipo con mayores capacidades. Por ello, se recomienda utilizar la configuración presentada en esta sección, ya que fue validada para cumplir con los requerimientos del proyecto.

7.1.3. Instalación y configuración de ROS2

Con el sistema operativo y la máquina virtual configurados, se procedió a la instalación del sistema operativo robótico 2 (ROS 2), siguiendo la guía oficial de la documentación [25]. La instalación se completó mediante la ejecución de los comandos en la terminal, seleccionando la versión ROS Desktop, que incluye todos los paquetes necesarios para el desarrollo de aplicaciones en ROS 2.

Para utilizar los comandos, funciones y librerías de ROS 2, fue necesario realizar un *source* del entorno mediante la instrucción `source /opt/ros/humble/setup.bash`. Este comando carga la configuración de ROS 2 en la sesión activa de la terminal, de modo que cada nueva sesión requeriría volver a ejecutarlo para indicar al sistema el intérprete y las rutas de los paquetes instalados. En caso de omitir esta instrucción, ROS 2 no es capaz de identificar la distribución, localizar los ejecutables ni reconocer los paquetes disponibles.

Con el fin de evitar repetir este paso en cada apertura de terminal, se añadió la instrucción de manera permanente al archivo `~/.bashrc` mediante el siguiente comando:

```
1     echo 'source /opt/ros/humble/setup.bash' >> ~/.bashrc
```

De esta manera, ROS 2 quedó disponible automáticamente en todas las sesiones posteriores, permitiendo que cualquier terminal ejecute ROS 2 sin errores. Para evaluar el correcto funcionamiento de la instalación, se pueden seguir los tutoriales de la documentación oficial [25]. A modo de ejemplo, se ejecutó el siguiente comando:

```
1     ros2 run turtlesim turtlesim_node
```

Este comando genera una ventana azul con una tortuga. Dado que al ejecutar el comando se observa la ventana mencionada, la instalación se consideró correcta.

Una vez finalizado el proceso de instalación de ROS 2, se procedió a preparar el entorno de desarrollo de software. Para ello, se recomienda instalar el editor Visual Studio Code, un editor ligero, multiplataforma y con soporte para múltiples lenguajes de programación, que permite depurar código e integra una terminal en la misma interfaz, lo cual resulta especialmente útil al trabajar con ROS 2. La instalación se realizó mediante la línea de comandos con la instrucción:

```
1    sudo snap install code --classic
```

De forma complementaria, se instaló el gestor de terminales Terminator, el cual resulta muy útil en proyectos con ROS2, ya que este entorno suele requerir la ejecución simultánea de múltiples terminales. Terminator permite dividir la ventana en paneles y organizar mejor las sesiones de trabajo, lo que facilita la supervisión de nodos y procesos de manera ordenada.

Posteriormente, con el fin de optimizar el trabajo en ROS2, se añadieron extensiones a Visual Studio Code que ampliaron sus capacidades:

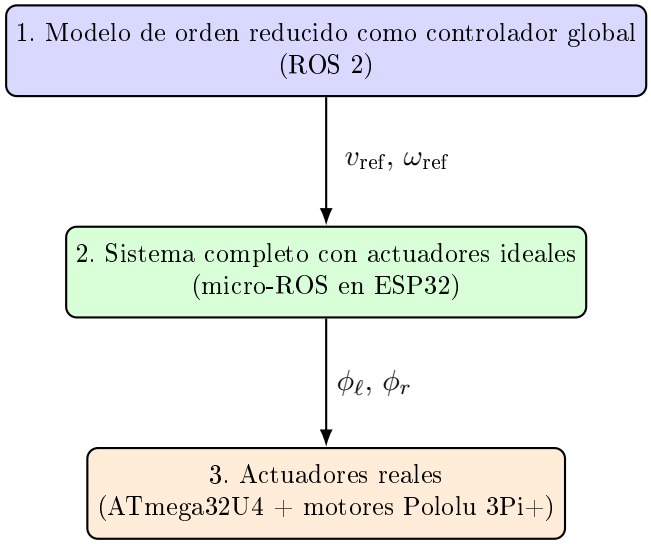
- Robot Developer Extensions for ROS 2, de Ranch Hand Robotics LLC, que facilita la interacción con paquetes, mensajes y services de ROS 2.
- Python, de Microsoft, necesaria para la ejecución y depuración de *scripts* en Python.
- CMake, de twxs, utilizada para compilar paquetes de ROS 2 escritos en C++ y generar los archivos de construcción correspondientes.
- PlatformIO IDE, de PlatformIO, destinada al desarrollo de *firmware* en microcontroladores, incluyendo los proyectos que serán utilizados más adelante para implementar micro-ROS en el ESP32.

En conjunto, estas herramientas convierten al entorno conformado por Visual Studio Code y Terminator en una plataforma de trabajo integrada, estable y eficiente, adecuada tanto para la creación de aplicaciones en ROS 2 como para la posterior programación de microcontroladores.

7.1.4. Jerarquía de control implementada en el algoritmo del *firmware* del ESP32

Según lo discutido en la subsección 6.5.1, el control del robot diferencial, y por ende del Pololu 3Pi+, se realiza mediante una arquitectura de control por capas de jerarquía (Figura 10), esta jerarquía se encuentra contextualizada en la Figura 15, con una comunicación organizada como se muestra en la Figura 16.

Figura 15. Jerarquía de control del robot Pololu 3Pi+ con integración de micro-ROS



Nota. En la figura se presenta la jerarquía de control del Pololu 3Pi+, desde la aplicación global en ROS 2 hasta la ejecución física en los actuadores reales. Elaboración propia.

La capa superior de control (capa 1) ejecuta un controlador global responsable del cumplimiento de tareas. Esta capa se ejecuta en ROS 2 debido a sus robustas capacidades computacionales, permitiendo la implementación de paquetes complejos de navegación, SLAM, planificación de movimiento, o controladores avanzados basados en el modelo del uniciclo (6.5.1) e inteligencia artificial.

La segunda capa de control (capa 2), incorporada en el microcontrolador ESP32, toma el sistema completo asumiendo actuadores ideales y es la encargada de convertir las señales de control global a velocidades de rueda. Finalmente, la tercera capa de control (capa 3), ejecutada en el microcontrolador ATmega32U4 del Pololu 3Pi+, recibe estas variables de velocidad e implementa un controlador de velocidad por rueda, formando parte de la infraestructura previa del laboratorio.

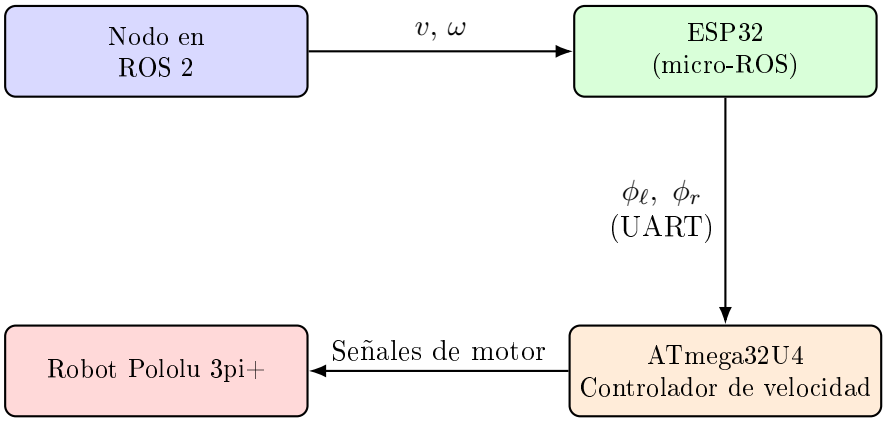
La comunicación entre la primera y segunda capa se realiza mediante la red wifi del Robotat. Esta conecta ROS 2 con el ESP32 utilizando Micro-ROS DDS XRCE, un protocolo diseñado para enlazar dispositivos con recursos limitados a la red global DDS de ROS 2. Por este canal, se envía el tipo de dato `geometry_msgs/msg/Twist`, que contiene la velocidad tridimensional del cuerpo rígido de interés. A pesar de que este mensaje puede transmitir seis valores de velocidad (lineal en los ejes x, y, z; y angular en ángulos de Euler), solo se utiliza la velocidad lineal del eje x, v (m/s), y la velocidad angular en torno al eje z, ω (rad/s), las cuales son comunicadas desde un nodo de ROS 2. La comunicación entre la segunda y la tercera capa se implementa mediante comunicación serial con empaquetamiento binario. Este protocolo, al ser una solución preexistente en el laboratorio, se mantuvo sin modificaciones. A través de este enlace, el microcontrolador ESP32 recibe las variables de velocidad v y ω calculadas en la capa superior.

El ESP32 es el responsable de transformar las variables globales v y ω en las velocidades angulares individuales de cada rueda (ϕ_ℓ y ϕ_r), expresadas en revoluciones por minuto (rpm). Para esto, utiliza el modelo de robot diferencial y uniciclo (como se detalla en la Sección 6.5.1), realizando la conversión de rad/s a rpm. Estas variables fueron declaradas en el *firmware* como `phi_ell` y `phi_r`.

De esta manera, el ESP32 dirige el comportamiento del robot en un lazo de control abierto, ya que no cuenta con información de retroalimentación de la velocidad real de los motores. No obstante, esta limitación no es crítica, pues la capa 1 compensa esta discrepancia al emplear el sistema de captura de movimiento del Robotat para establecer un control de lazo cerrado global, tema que se abordará en el siguiente capítulo.

Finalmente, la tercera capa de control utiliza los encóders de cuadratura integrados en el robot Pololu 3Pi+ para retroalimentar la velocidad real de los actuadores. Este lazo de control local eleva el manejo de los motores más allá de un modelo ideal, brindando una retroalimentación interna rápida. Esta acción interna tiene un tiempo de respuesta significativamente menor que el control de la primera capa, lo que asegura una mejor estabilidad y respuesta local del sistema.

Figura 16. Arquitectura de comunicación entre ROS 2, micro-ROS, ESP32, ATmega y el robot Pololu 3Pi+



Nota. En la figura se presenta un diagrama con la arquitectura de comunicación para el robot Pololu 3Pi+. Elaboración propia.

De este modo, queda formalmente establecida la jerarquía de control del robot Pololu 3Pi+ y la distribución de sus funcionalidades en las tres capas (global, intermedia y local). Esta arquitectura multicapa, no solo guió el desarrollo y la implementación del firmware en el microcontrolador ESP32, sino que también sirve como marco conceptual esencial para la implementación de los controladores que serán abordados en los capítulos siguientes.

7.1.5. Configuración del proyecto e instalación del agente micro-ROS

El primer paso para el desarrollo del *firmware* con micro-ROS consistió en definir el entorno de programación. Para ello, se optó por utilizar PlatformIO, un ecosistema de desarrollo integrado en visual studio code. PlatformIO proporciona un sistema de gestión de

librerías, compilación y despliegue para múltiples plataformas de hardware, lo cual lo hace especialmente adecuado para proyectos de robótica.

La elección de PlatformIO, en lugar del entorno clásico de Arduino IDE o de Espressif IDF, se debió principalmente a que en la universidad ya existía experiencia previa en esta herramienta, lo cual facilita la estandarización de proyectos. De hecho, en trabajos anteriores como [3], se documenta una prueba básica de comunicación entre micro-ROS y ROS 2 utilizando PlatformIO como entorno de desarrollo.

Además de esta experiencia previa, PlatformIO presenta ventajas adicionales. Una de las más relevantes es que los pines de la placa ESP32 están correctamente identificados en este entorno, lo que reduce confusiones frecuentes al trabajar con Arduino IDE, donde la nomenclatura de los pines puede variar según el modelo de placa utilizado.

En el anexo se encuentra el repositorio oficial, dentro de la carpeta “MicroRos firmware"se encuentra el proyecto de micro-ROS desarrollado en PlatformIO. Dentro del proyecto, las secciones más importantes son el programa principal (main.c), así como el archivo de configuración del PlatformIO (platformio.ini). Como primer paso, se debe realizar la correcta configuración de este archivo, ya que resulta fundamental para el éxito del proyecto. Dentro del archivo se definen parámetros clave como la placa de desarrollo utilizada, las dependencias, velocidad de comunicación serial y el puerto de comunicación. A continuación se muestra la configuración utilizada en este trabajo, así como su descripción:

```
[env:az-delivery-devkit-v4]
platform = espressif32
board = az-delivery-devkit-v4
framework = arduino
monitor_speed = 115200
board_microros_distro = humble
board_microros_transport = wifi
lib_deps =
soburi/TinyCBOR@^0.5.3-arduino2
;micro-ROS/micro_ros_arduino
https://github.com/micro-ROS/micro_ros_platformio
```

- **platform = espressif32**: define la plataforma de hardware utilizada, en este caso microcontroladores ESP32.
- **board = az-delivery-devkit-v4**: especifica el modelo de la placa de desarrollo (ESP32 DevKit V4 de AZ-Delivery).
- **framework = arduino**: selecciona el *framework* de programación compatible con Arduino, lo cual simplifica la implementación del *firmware*.
- **monitor_speed = 115200**: velocidad de comunicación serie entre la PC y el ESP32, utilizada para depuración y monitoreo.
- **board_microros_distro = humble**: indica la versión de ROS 2 con la que se desea compatibilidad (en este caso, la distribución Humble).
- **board_microros_transport = wifi**: define el medio de transporte para la comunicación entre micro-ROS y ROS 2, en este caso mediante conexión WiFi.
- **lib_deps**: lista las dependencias externas que deben descargarse automáticamente. Entre ellas se incluyen:

- `TinyCBOR`: biblioteca para codificación y decodificación de datos en paqueres binario, con el formato CBOR.
- `micro_ros_platformio`: integración oficial de micro-ROS para PlatformIO.
- (comentada) `micro_ros_arduino`: alternativa de librería para entornos Arduino.

Esta configuración asegura que el ESP32 pueda ejecutarse como un agente micro-ROS conectado a la distribución ROS 2 Humble, utilizando wifi como canal de comunicación y CBOR como formato de datos para las referencias de velocidad. Esto último corresponde a configuraciones previas de la integración entre el ESP32 y ATmega32U4, por lo que no se modificó.

Para establecer el puente de comunicación entre ROS 2 y micro-ROS es necesario ejecutar el **agente micro-ROS**. Este componente actúa como un nodo dentro de ROS 2 que se encarga de gestionar el enlace con el microcontrolador. De esta manera, todo lo que se publique o suscriba a través del nodo del agente se transfiere directamente al firmware de micro-ROS que corre en el ESP32.

La instalación del agente requiere en primer lugar la descarga de las dependencias de ROS 2 mediante el comando bash:

```
rosdep install --from-paths src --ignore-src -y
```

Posteriormente, se debe clonar el repositorio oficial con:

```
cd ros2_ws/src
git clone https://github.com/micro-ROS/micro-ROS-Agent.git
```

Este repositorio se encuentra disponible en el sitio oficial de micro-ROS [26]. Una vez clonado, el agente debe compilarse con `colcon build` y cargarse en el entorno de la siguiente manera:

```
colcon build
source install/local_setup.bash
```

7.1.6. Desarrollo del programa principal (main.c) para el *firmware* del ESP32

La estructura general del programa esta divida tanto en funciones como en tareas de ejecución paralela con *multithreading*, a continuación se describen las funciones y tareas utilizadas en el programa:

- **Cargar librerías adecuadas:** las librerías y dependencias que se usaron en el trabajo permiten acceder a todas las funcionalidades de micro-ROS, así como el control del *harware* mediante la misma sintaxis de Arduino IDE.
- **Declaración de suscriptores y publicadores:** en esta parte se debe indicar cuantas suscripciones se generan, así como el tipo de datos que contendrá dicha suscripción.

- **Función de limite de velocidad:** esta función, como su nombre lo indica, limita la velocidad máxima que se envía serialmente, se ejecuta con fines de proteger el a robot de posibles colisiones e inestabilidad.
- **Tarea de envió de datos:** la función `encode_send_wheel_speeds_task` es una de las tareas que se ejecuta al hacer *multithreading* o multihilo. Es responsable por la codificación y envió de datos mediante CBOR, lo único que se hace para cargar los datos necesarios es escribir las velocidades deseadas en las variables `phi_ell` y `phi_r`. No se debe modificar esta función.
- **Función de retorno del suscriptor micro-ROS:** la función `cmd_vel_callback` emplea micro-ROS y se ejecuta automáticamente, al recibir un mensaje desde ROS 2, de la suscripción del tópico `cmd_vel_n`. La informaición contenida en la suscripción se utiliza para generar el calculo de las velocidades de las ruedas usando el modelo discutido en (1)
- **Tarea de carga de valores de velocidad actuales:** La tarea `ros2_cmdvel_task` es el segundo hilo que se corre en el microcontrolador. Es responsable por cargar continuamente nuevas velocidades a las variables de velocidad de rueda, haciendo uso de la función para limitar su velocidad.
- **Configuración (`setup`):** en este apartado se configuran los procesos paralelos o hilos, se configuró la comunicación serial y se configuró el entorno micro-ROS. Esta última configuración se usa para asignar memorias, definir y crear el nodo `MicroROS_node`, generar la suscripción del tópico `cmd_vel_11` y definir a los ejecutores.
- **Bucle principal:** contiene unicamente la instrucción

```
1    RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(20)));
```

que ejecuta un bucle no bloqueante. Espera 20 ms para recibir datos desde ROS 2 y revisa tareas pendientes de los ejecutores. Es útil para el manejo de errores de comunicación por DDS [27].

7.1.7. Criterios de evaluación para el cumplimiento de la integración de micro-ROS para el control del Pololu 3Pi+

La metodología para el desarrollo y la implementación del nodo micro-ROS se estructuró mediante una serie de validaciones consecutivas e incrementales que permitieron verificar la funcionalidad de la arquitectura por etapas. Esta estrategia buscó minimizar la incertidumbre y aislar el error en cada prueba, comenzando con validaciones básicas hasta escalar a pruebas completas. No fue hasta la comprobación exitosa de todos los criterios se concluyó la fase de desarrollo del nodo. Esto garantizó la capacidad para cumplir con la arquitectura jerárquica de control propuesta. Los pasos de validación seguidos fueron los siguientes:

- Validar la conexión del agente micro-ROS en ROS 2 de forma serial.
- Validar la conexión del agente micro-ROS en ROS 2 de forma inalámbrica.
- Validar la recepción de datos entre ROS 2 y el microcontrolador ESP32.
- Validar la jerarquía de control mediante la teleoperación del robot utilizando ROS 2.

7.2. Resultados de la implementación de micro-ROS en el Pololu 3Pi+

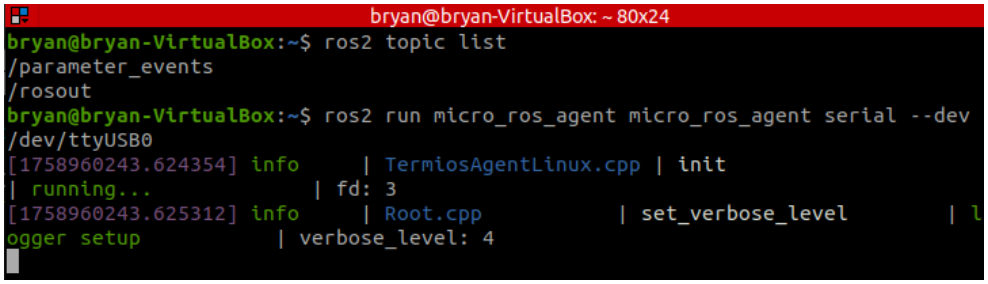
Posterior a la implementación del *firmware* en el microcontrolador ESP32, se continuó con la realización de pruebas para corroborar el adecuado funcionamiento de la conexión entre ROS 2 y el ESP32 mediante el *firmware* micro-ROS, así como el control de movimiento del robot utilizando la terminal de ROS 2.

Como primera prueba de comunicación, se utilizo el nodo publicador de ejemplo, brindado por PlatformIO en [20]. Dicho ejemplo se implementó en el firmware del ESP32, se cargó y probó la comunicación de forma cableada mediante el comando:

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0
```

La Figura 17 muestra, en primera instancia, los tópicos por defecto de ROS 2 mediante el comando `ros2 topic list`. Luego, se observan los mensajes generados al ejecutar el nodo micro-ROS de forma cableada, hasta este punto no hay conexión con el microcontrolador. Elaboración propia.

Figura 17. Ejecución del agente micro-ROS mediante comunicación serial

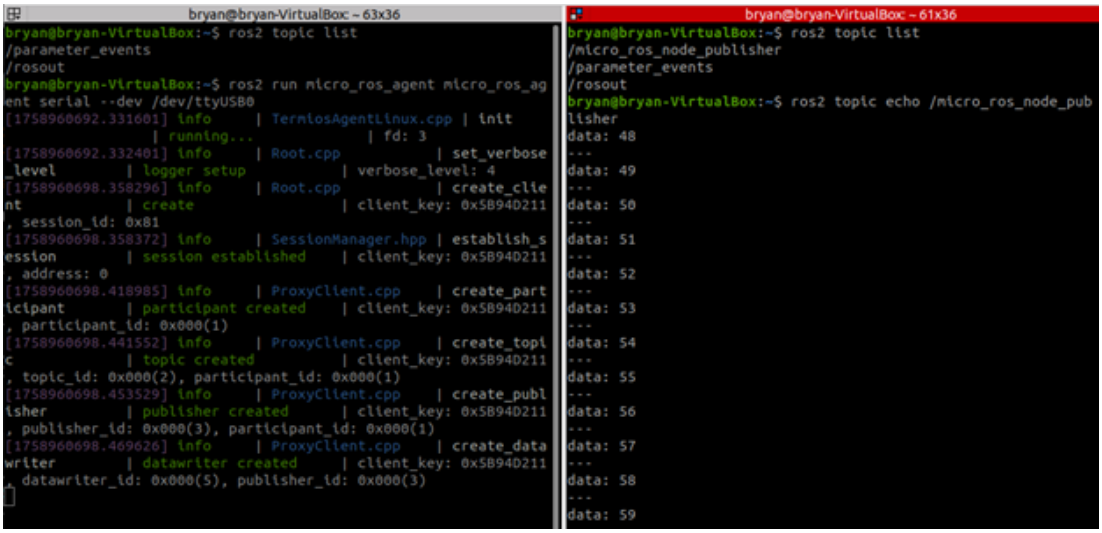


Nota. En la terminal únicamente se observan los tópicos por defecto de ROS 2: `/rosout` y `/parameter_events`, sin conexión del nodo de micro-ROS.
Elaboración propia.

No es hasta que se reinicia el microcontrolador ESP32 que aparecen más mensajes en la ventana donde se ejecutó el agente micro-ROS. Esto indica que se estableció la conexión entre ROS 2 y el ESP32. Seguidamente, se ejecutó nuevamente el comando `ros2 topic list` para verificar que el microcontrolador publica datos en un tópico creado por el mismo. En la Figura 18 se observan dos terminales de Terminator. En la ventana izquierda se observa la ejecución del agente micro-ROS, esta vez ya conectado. En la ventana derecha, se observa la aparición del tópico `/micro_ros_node_publisher` generado por el ESP32. De igual forma, se observa un eco de los mensajes que este publica, evidenciando que el microcontrolador ejecutó su tarea a la perfección: publicar un contador de segundos en la red ROS 2 mediante DDS.

Esta prueba valida la correcta configuración de PlatformIO para el ESP32-WROOM-32D, así como la correcta instalación del agente micro-ROS en el espacio de trabajo de ROS 2.

Figura 18. Validación de conexión del agente micro-ROS mediante comunicación serial



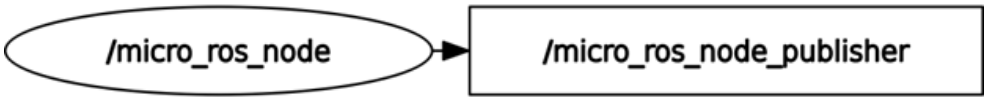
Nota. En la terminal izquierda se aprecia el estado del agente al detectar la conexión del ESP32. En la derecha, al repetir `ros2 topic list`, aparece el tópico `/micro_ros_node_publisher`, así como los datos publicados por el nodo. Elaboración propia.

Tras validar el sistema por medio de conexión serial, se procedió a realizar pruebas de comunicación inalámbrica mediante wifi . Para este caso, el agente debe ejecutarse utilizando el protocolo `udp4`, especificando el puerto que se configuró en el ESP32. Además, el *firmware* del ESP32 debe contener la dirección IP de la computadora en la cual se ejecuta el agente micro-ROS. Mediante el siguiente comando se ejecuta el agente micro-ROS con comunicación inalámbrica:

```
ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
```

En la Figura 19 se muestra el diagrama de nodos y tópicos generado por `rqt_graph` de ROS 2. Este diagrama permite observar de forma global la comunicación en ROS 2. Encerrados en elipses se encuentran los nodos activos, mientras que encerrados en rectángulos se observan los tópicos, así como la dirección de comunicación con flechas. De esta forma, también se valida la correcta implementación de micro-ROS, esta vez, de forma inalámbrica, observando la publicación del nodo accesible desde la terminal.

Figura 19. Diagrama de nodos y tópicos generado con `rqt_graph`



Nota. El diagrama refleja la relación entre el nodo `micro_ros_node_publisher` y el resto del sistema, confirmando la comunicación establecida de forma inalámbrica. Elaboración propia.

De esta forma, se validó la correcta transmisión de datos por medio de micro-ROS, esta vez mediante wifi. Esta modalidad permitió la comunicación a distancia, por lo que se estableció como la configuración por defecto de aquí en adelante.

Consideraciones de conexión al correr ROS 2 desde una máquina virtual

Se bebe tener en cuenta que el uso de una máquina virtual divide los recursos de la computadora, según se realizó la configuración. Por ello, tanto los dispositivos conectados por los puertos USB o por wifi, pueden no ser reconocidos directamente por la máquina virtual. El procedimiento en Oracle VirtualBox resulta bastante sencillo, pero se incluye a continuación:

- **Conexión serial:** configurar los dispositivos reconocidos por la máquina virtual en el botón *devices* (dispositivos), ubicado en la cinta de herramientas superior. En este botón dirigirse a la opción USB, donde se debe seleccionar el dispositivo conectado a algún puesto USB de la computadora. El nombre puede depender, pero para el microcontrolador ESP32-WROOM-32D, el nombre es: `Silicon Labs CP2102 USB to UART Bridge Controller [0100]` .
- **Conexión mediante WiFi:** dentro de la configuración del firmware cargado al ESP32 se debe incluir la dirección IP de la máquina virtual, que no es la misma dirección de la máquina principal. Para identificar la dirección se debe escribir el comando `ip a` en cualquier terminal de Ubuntu, este despliega información sobre la red, así como la IP. Este proceso se recomienda hacer antes de cada conexión, ya que puede cambiar a lo largo del tiempo, a menos que se configure una IP fija desde el *router* de la red.

7.2.1. Control de los actuadores del robot Pololu 3Pi+ desde ROS 2

Según la jerarquía de control discutida previamente, la transmisión de datos entre el ESP32 y ROS 2 requiere de un nodo controlador para que este publique las referencias de velocidad (v, ω) en el tópico `cmd_vel`. Dicho nodo corresponde al control global del sistema, responsable de transformar las tareas de alto nivel en referencias de movimiento. Sin embargo, para las pruebas de funcionamiento iniciales no fue necesario implementar este nodo, ya que se empleó el nodo de operación a distancia provisto por ROS 2:

```
1    ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Este comando permitió publicar directamente mensajes tipo `twist` en el tópico `cmd_vel`, generando los valores de velocidad lineal y angular a partir de la interacción con el teclado de la computadora. Para esta prueba, fue necesario que el tópico al cual se suscribió el agente micro-ROS tuviera el nombre `cmd_vel`.

El cuadro 4 muestra el esquema de teclas utilizadas en `teleop_twist_keyboard`, donde cada comando se traduce en un movimiento específico o en un ajuste de velocidad:

Cuadro 4. Comandos de teclado para el nodo `teleop_twist_keyboard`

Tecla	Acción
u	Avance + giro izquierdo
i	Avance
o	Avance + giro derecho
j	Giro izquierdo
k	Detener
l	Giro derecho
m	Retroceso + giro izquierdo
<	Retroceso
>	Retroceso + giro derecho
q/z	±10 % vel. total
w/x	±10 % vel. lineal
e/c	±10 % vel. angular

Nota. El cuadro muestra las teclas, identificadas con su función, utilizadas para controlar al robot y su velocidad. Elaboración propia.

La ejecución, junto con la experimentación de los diferentes tipos de movimiento y velocidades, permitió validar la arquitectura de comunicación desde la capa de control global (ROS 2) hasta la capa de control local (controladores de velocidad angular del Pololu 3Pi+). Se comprobó que el mapeo de velocidades (v, ω) a velocidades de rueda (ϕ_ℓ, ϕ_r) se ejecutó correctamente. Por lo tanto, esta resulta ser la prueba final para corroborar que el *firmware* desarrollado para el ESP32 es capaz de comunicarse con ROS 2 y con el robot para controlar sus motores y *hardware* de forma inalámbrica y en tiempo real.

7.2.2. Procedimiento para la ejecución del Pololu 3Pi+ con micro-ROS

Como parte de los resultados, se generó una lista de pasos para lanzar el nodo y operar al robot desde ROS 2 sin ningún inconveniente. La puesta en marcha del robot Pololu 3pi+ con *firmware* de micro-ROS se realizó siguiendo un procedimiento secuencial. A continuación se presenta la descripción de los pasos realizados, así como un diagrama guía que resume visualmente el flujo de ejecución de la Figura 20. Esta integración proporciona al lector una referencia completa, tanto textual como gráfica, del proceso necesario para operar el robot de manera efectiva.

1. **Cargar el *firmware* en el Pololu 3Pi+.** El *firmware* se cargó en la placa principal del robot, configurando el microcontrolador ATmega32U4 para suscribirse al tópico `cmd_vel`. Esta acción fue esencial para que el Pololu 3Pi+ pudiera recibir correctamente los comandos de velocidad en formato binario enviados desde el ESP32. Sin este paso, el robot no podría ejecutar los comandos de movimiento.

2. **Ejecutar el nodo *micro-ROS agent*.** Se inició el agente micro-ROS en el ordenador, especificando la dirección IP correspondiente al ESP32 mediante el comando:

```
ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
```

3. **Conectar el ESP32.** Se presionó el botón `EN` en el ESP32 para establecer la conexión con el agente. Esta acción es crítica, ya que inicializa el microcontrolador y permite que el agente detecte el nodo `micro_ros_node_publisher`. En caso de desincronización, apagar y encender el robot o reiniciar el ESP32 restableció la comunicación, garantizando la transmisión correcta de los datos.

4. **Ejecutar el nodo de teleoperación.** Se utilizó el comando:

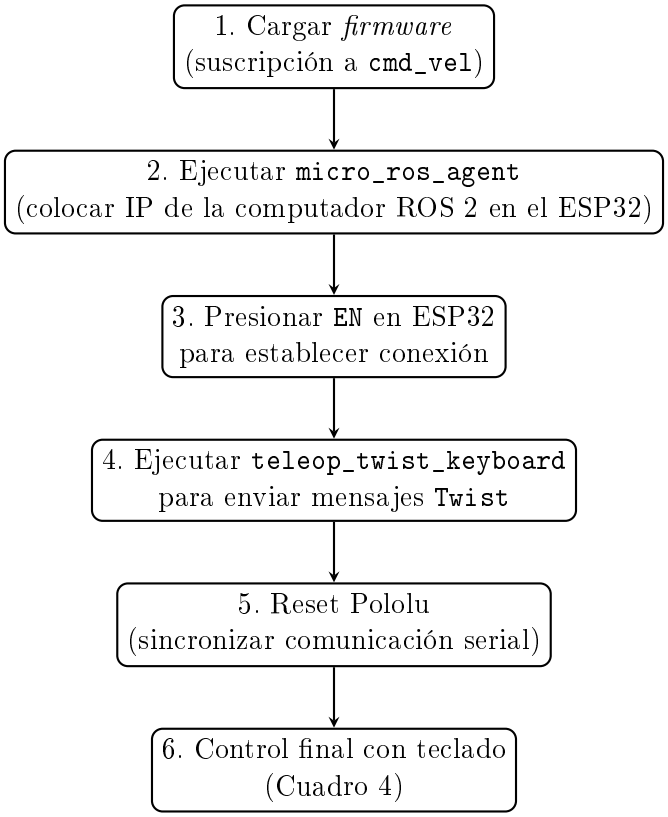
```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

   para enviar mensajes de tipo `twist` al tópico `cmd_vel`. Esta herramienta permitió controlar el robot de manera remota, facilitando la evaluación inicial de su comportamiento ante diferentes magnitudes de velocidad lineal y angular.

5. **Reiniciar el Pololu 3Pi+.** Se presionó el botón de `reset` en la placa principal del Pololu. Este paso aseguró la correcta sincronización de los datos enviados desde el ESP32, evitando pérdidas o desajustes en la ejecución de los comandos de velocidad.

6. **Controlar el robot con el teclado.** Finalmente, se realizó el control del robot mediante el teclado, siguiendo los comandos especificados en el Cuadro 4. Esta etapa permitió observar de manera práctica la respuesta dinámica del robot y validar que las consignas de velocidad se ejecutaran correctamente en la plataforma física.

Figura 20. Flujo integrado de pasos para la ejecución del Pololu 3Pi+ con micro-ROS



Nota. La figura mostra de forma secuencial el procedimiento para poner en marcha el robot Pololu 3pi+ con *firmware* basado en micro-ROS. Se destaca la importancia de inicializar el ESP32 y reiniciar el Pololu para garantizar la correcta transmisión de datos. Elaboración propia.

7.3. Conclusiones sobre la implementación de micro-ROS en ESP32

Se realizaron pruebas de comunicación entre el microcontrolador ESP32 y ROS 2, validando la implementación completa del agente micro-ROS mediante comunicación serial o wifi.

Se logró el control del comportamiento físico del robot desde una computadora con ROS 2, utilizando la arquitectura micro-ROS. Esto no solo permitió la operación del robot a distancia, sino que también facilitó la experimentación con el comportamiento del sistema bajo distintas referencias de velocidad.

Se desarrolló un procedimiento secuencial y probado para la puesta en marcha del agente micro-ROS en ROS 2 y el ESP32. Esta metodología garantiza el correcto funcionamiento del sistema y la operatividad del robot.

CAPÍTULO 8

Desarrollo e implementación de nodos de control en ROS 2 para el agente robótico Pololu 3Pi+

En este capítulo se presenta el desarrollo e implementación del sistema de control en lazo cerrado para el robot Pololu 3Pi+, utilizando la infraestructura del laboratorio Robotat y una unidad de cómputo configurada para ejecutar ROS 2. Este sistema corresponde a la `capa de control global` descrita en la Sección 6.5.1. La finalidad de esta capa fue controlar tanto la velocidad lineal v como la angular ω del robot de manera autónoma, permitiendo que el agente robótico se desplace desde un punto inicial hacia un punto objetivo sin intervención teleoperada. El sistema utilizó el sistema de captura de movimiento OptiTrack para obtener la retroalimentación del estado del robot, así como para recibir la ubicación del objetivo al cual debe dirigirse.

El controlador implementado cumplió con las siguientes características: dar continuidad a líneas de investigación previas; servir como prueba de concepto para el desarrollo de aplicaciones ROS 2; adaptarse a la infraestructura existente en la universidad; y ser de fácil implementación , de modo que pueda utilizarse en cursos de robótica y sistemas de control. Por estas razones,se empleó un esquema de *control punto a punto*, el cual ya había sido utilizado en [2]. Dicho esquema sirvió como punto de partida para validar el sistema propuesto, compuesto por una arquitectura de control jerárquico y la integración con el sistema de captura de movimiento.

El controlador desacopla la velocidad lineal v de la velocidad angular ω, lo que permite definir un número reducido de parámetros de control. En particular, el comportamiento angular del robot se reguló mediante un controlador PID, mientras que el error de posición lineal se compensó utilizando un acercamiento exponencial. De esta manera, el controlador punto a punto genera un comportamiento en el cual el robot corrige su orientación a la vez que avanza hacia el objetivo, manteniendo su velocidad hasta encontrarse suficientemente cerca del mismo (comportamiento configurable). Como consecuencia, el robot realiza

una trayectoria en forma de espiral, cuyo grado de curvatura y precisión dependen de las constantes del controlador.

8.1. Nodo de control para un solo agente

Se desarrollo el nodo `controller_3pi` dentro del paquete `pololu_controller` para el control de agentes individuales. La finalidad de este nodo fue controlar tanto la velocidad lineal v como la angular ω del robot de manera autónoma, permitiendo que el agente robótico se desplace desde un punto inicial hacia un punto objetivo sin intervención teleoperada. Según la ecuación (6) el controlador PID para el comportamiento angular tiene como parámetros ajustables K_p, K_i y K_d. De manera similar, en la ecuación (8) se observa que los parámetros ajustables para regular la velocidad de acercamiento son v_0 y α.

Se utilizó la información del sistema de captura de movimiento para obtener la retroalimentación de pose (posición x,y, así como la orientación jaw) del robot. Esto se realizó mediante la suscripción del tópico `/pololu_n/pose`, donde `n` se sustituyó por el identificador del robot. De forma similar, se obtuvo la posición x,y del objetivo o meta, mediante la suscripción del tópico `/Meta_n/pose`, donde `n` se sustituyó por el identificador de metas. Los identificadores de los agentes, así como los identificadores de metas se muestran en el Cuadro 5.

Cuadro 5. Identificadores de agentes e identificadores de metas

Identificador de agentes	Identificador de metas disponibles
3	1, 2, 3, 4, 5
4	1, 2, 3, 4, 5
6	1, 2, 3, 4, 5
7	1, 2, 3, 4, 5
8	1, 2, 3, 4, 5
9	1, 2, 3, 4, 5
11	1, 2, 3, 4, 5
12	1, 2, 3, 4, 5
13	1, 2, 3, 4, 5

Nota. El cuadro muestra los identificadores de los agentes y las metas que cada uno puede utilizar Elaboración propia.

Con esta información, el nodo es capaz de calcular la discrepancia entre la ubicación de la meta y el robot, generando velocidades controladas v en m/s. Asimismo, se calcula el angulo entre el robot y la meta según el marco de referencia del Robotat, luego el controlador genera valores de velocidad angular ω en rad/s controlados para alinear el eje x del robot con la dirección en la que debe moverse para encarar al objetivo. Este proceso es ejecutado en el bucle de control, configurado por defecto a una frecuencia de 30 Hz. Esto permite al robot ajustar su comportamiento ante variaciones en la posición del robot, permitiendo seguir metas dinámicas.

Para ejecutar las variables controladas v y ω el nodo publica un mensaje del tipo

`geometry_msgs/Twist` en el tópico `cmd_vel_n`. Es importante notar que los datos que entran al nodo representan poses en tres dimensiones, que no tienen el mismo formato que los datos que salen, ya que estos representan velocidades. Por esta razón, el nodo extrae solo los datos numéricos útiles, haciendo la conversión de cuaterniones unitarios a angulos de Euler, para luego escribir datos numéricos en el formato `cmd_vel_n` que usa angulos de Euler.

8.1.1. Parámetros del nodo ROS 2

Los nodos desarrollados en ROS 2 cuentan con herramientas que permiten parametrizar diferentes valores usados en la aplicación. Estos parámetros se usan para dar un comportamiento inicial al robot, pero también pueden ser modificados mientras el robot esta ejecutando una tarea. Con el fin de desarrollar un controlador flexible y de uso didáctico en la universidad, todos los nodos desarrollados incluyen parámetros para la personalización, tanto del comportamiento de robot, como del sistema de control.

Parámetros del agente robótico

Los parámetros del robot deben ser configurados al momento de lanzamiento del agente, ya que estos parámetros dictaminan hacia cual robot se aplicará el control. Es muy importante revisar que estos parámetros sean correctos, de lo contrario el nodo no ejecutará el control de la forma esperada.

- `pose_topic`: parámetro para elegir sobre qué agente se trabajará; sirve para seleccionar el agente al cual se le dará seguimiento. Este parámetro espera `/pololu_n/pose`, donde `n` es el identificador del robot.
- `goal_topic`: parámetro usado para elegir la meta hacia la cual se dirigirá el agente. El parámetro espera `/Meta_n/pose`.
- `cmd_vel_topic`: parámetro utilizado para publicar desde ROS 2 hacia el ESP32 las variables controladas calculadas por el nodo. El parámetro espera `/cmd_vel_n`, donde `n` debe coincidir con el identificador utilizado en `pose_topic`.
- `offset_deg`: parámetro que representa un ángulo de desfase entre el eje x del robot y el eje x del marco de referencia del laboratorio. Se utiliza para corregir dicho desfase y alinear ambos ejes. Cada agente tiene un desfase diferente, por lo que se debe revisar este parámetro en el Cuadro 6.Se espera un valor de punto flotante en grados.

Cuadro 6. Ángulos de desfase (`offset_deg`) para cada robot según su identificador

Identificador del robot	Ángulo de desfase (°)
3	270.1823
4	278.0994
5	271.1894
6	270.0347
7	265.9134
8	270.0128
9	265.7413
11	263.0
12	263.0
13	261.0

Nota. El cuadro muestra el desfase angular entre el eje del robot y el eje del sistema de referencia del laboratorio para cada agente. Elaboración propia.

Parámetros del controlador

Para la implementación inicial del controlador, los valores de las variables de control eran desconocidos. Por tal motivo, el ajuste de estos parámetros se realizó de forma individual y secuencial. Se optó por una estrategia que permitiera modificar las ganancias del controlador mientras el robot se encontraba en ejecución, sin necesidad de detener y ejecutar el nodo. Esta aproximación resultó útil, ya que facilitó la prueba de desempeño de los parámetros en tiempo real, permitiendo un proceso iterativo rápido para su sintonización.

Además, esta característica de ajuste en tiempo real es de gran utilidad en el curso de Sistemas de Control I de la universidad, pues proporciona a los estudiantes una herramienta práctica para el desarrollo de destreza en la sintonización de controladores sin la necesidad de implementar el algoritmo de control desde cero.

- `kp0`: representa el término proporcional del controlador PID. Se espera un valor de punto flotante.
- `ki0`: representa el término integrativo del controlador PID. También se espera un valor de punto flotante.
- `kd0`: representa el término derivativo del controlador PID. Su valor debe ser de punto flotante.
- `alpha`: representa el parámetro α del acercamiento exponencial. Se espera un valor de punto flotante.
- `v0`: representa el parámetro v_0 del acercamiento exponencial. Debe ser un valor de punto flotante, preferentemente no mayor a 2, ya que corresponde a la velocidad máxima del robot en metros por segundo. Si este parámetro excede dicho valor, el firmware aplicará su propio límite de velocidad seguro.

La metodología seguida para sintonizar el controlador fue la siguiente:

1. Lanzar el nodo con una meta.
2. Inicializar todas las constantes en `0.0` y sintonizar primero el comportamiento lineal.
3. Incrementar el término proporcional hasta que el robot gire siguiendo el movimiento de la meta. Ajustarlo para obtener una respuesta rápida.
4. Si la respuesta presenta oscilaciones, incrementar el término derivativo en un orden de magnitud menor que el término proporcional.
5. Si la respuesta presenta error estacionario, donde el robot no se alinea con la meta, incrementar el término integrativo.
6. Cuando el comportamiento angular sea satisfactorio, ajustar la velocidad lineal mediante v_0 hasta alcanzar la velocidad máxima deseada, haciéndolo de manera gradual.
7. Se observará que el robot no disminuye su velocidad al acercarse a la meta y colisionará con ella.
8. Para corregir esto, incrementar α, de modo que el robot desacelere al estar cerca de la meta. Un valor de α demasiado grande impedirá que el robot alcance su velocidad máxima incluso estando lejos, mientras que un valor muy pequeño hará que colisione con la meta.

A continuación se presenta, a modo de ejemplo, el comando para ejecutar el agente con identificador 3, utilizando la meta 5 para el sistema de control:

```
ros2 run pololu_controller cp2p_pololu --ros-args -p pose_topic:=/
    pololu_3/pose  -p goal_topic:=/Meta_5/pose  -p cmd_vel_topic:=/
    cmd_vel_3  -p offset_deg:=270.1823
```

De igual forma, se muestran los comandos para modificar los parámetros del controlador mientras el nodo se ejecuta:

```
ros2 param set /controller_3pi kp0 0.0
ros2 param set /controller_3pi ki0 0.0
ros2 param set /controller_3pi kd0 0.0
ros2 param set /controller_3pi v0 0.0
ros2 param set /controller_3pi alpha 0.0
```

8.1.2. Resultados experimentales

Mediante la realización de pruebas experimentales, se determinaron las constantes del controlador que se detallan en el Cuadro 7. Se obtuvieron dos conjuntos de parámetros con un rendimiento satisfactorio, cada uno válido para un rango específico de velocidad lineal. Aunque el segundo conjunto de parámetros no presenta oscilaciones, no se recomienda su uso en las condiciones actuales. Esto se debe a que el robot experimenta deslizamiento (derrape) debido a la baja tracción con la superficie de la plataforma del laboratorio Robotat.

Si bien este efecto no es un error inherente al controlador, compromete la fiabilidad del comportamiento del sistema. Sin embargo, se incluye este conjunto ya que, al mejorar las condiciones de tracción de las ruedas, podría utilizarse para lograr un desempeño más rápido y preciso.

Cuadro 7. Conjuntos de parámetros y su comportamiento observado

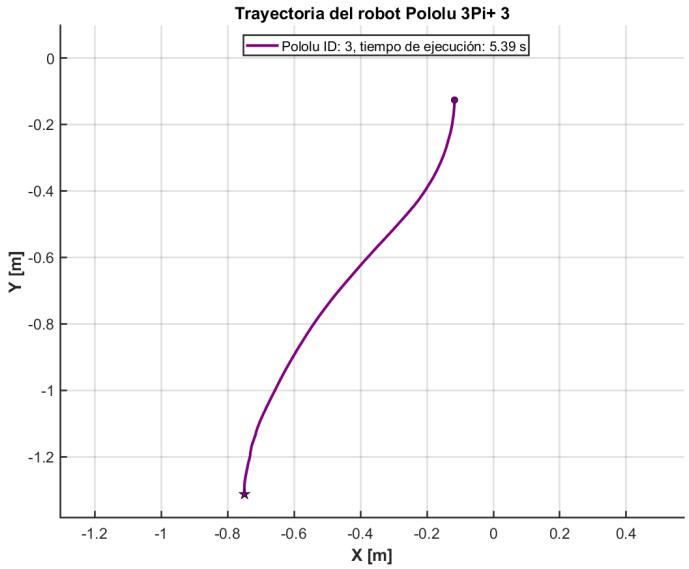
Parámetro	Valor	Descripción del comportamiento
Conjunto de constantes 1		
v_0	0.6 – 0.8	Comportamiento rápido, estable y sin oscilaciones.
kpO	2.0	
kiO	0.0	
kdO	0.0	
α	6.0	
Conjunto de constantes 2		
v_0	0.8 – 1.5	Comportamiento más rápido y agresivo; sin oscilaciones, pero las llantas derrapan por falta de tracción.
kpO	2.0	
kiO	0.0	
kdO	0.01	
α	0.0	

Nota. El cuadro resume dos configuraciones distintas del controlador y su efecto en la dinámica del robot. Elaboración propia.

A continuación, en la Figura 21 se presenta el comportamiento de un agente Pololu 3Pi+ (ID: 3) siendo controlado por el nodo `controller_3pi` implementado en ROS 2, utilizando las constantes de los controladores del conjunto 1 del Cuadro 7, con una velocidad v_0 de 0.6 m/s. Asimismo, en la Figura 22 se observa el comportamiento del mismo agente, con las mismas constantes de control, pero con una velocidad de 0.75 m/s. En estas pruebas, la meta se mantuvo estática y se marca con una estrella, mientras que el punto de inicio se marca con un punto.

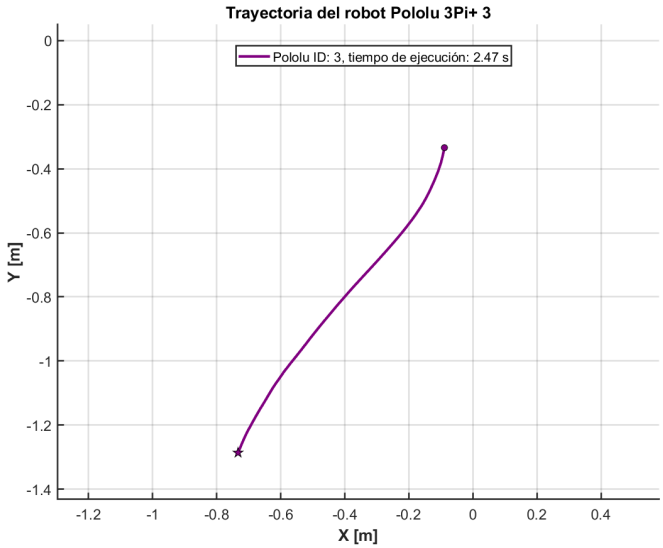
De igual manera, se realizaron múltiples pruebas del controlador con meta dinámica. Estas pruebas se efectuaron moviendo la meta lejos del robot, causando que el robot ajustara su comportamiento para alcanzarla. En la Figura 23 se presenta el comportamiento del agente utilizando los parámetros del Conjunto 1 del Cuadro 7. En la figura se muestran dos trayectorias: una representa el desplazamiento de la meta y otra presenta el movimiento del robot. De manera visual, se puede apreciar cómo el controlador ajusta el comportamiento del robot para acercarse a la meta.

Figura 21. Trayectoría del robot Pololu 3Pi+ ejecutando el nodo `controller_3pi` para una meta estática



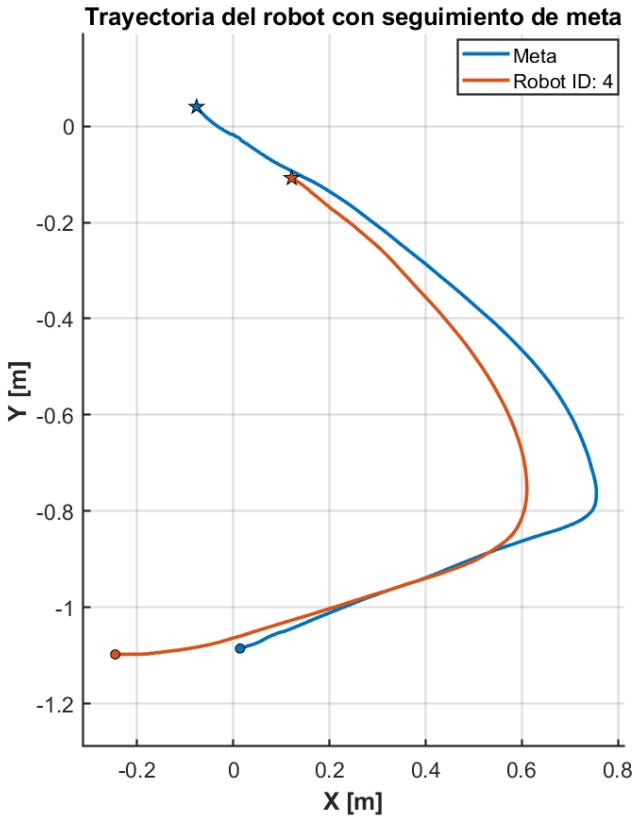
Nota. La figura muestra la trayectoria seguida por el robot con ID 3 implementando el controlador punto a punto para un agente con una meta estática. Primera prueba Elaboración propia.

Figura 22. Trayectoría del robot Pololu 3Pi+ ejecutando el nodo `controller_3pi` para una meta estática



Nota. La figura muestra la trayectoria seguida por el robot con ID 3 implementando el controlador punto a punto para un agente con una meta estática. Segunda prueba. Elaboración propia.

Figura 23. Trayectoría del robot Pololu 3Pi+ ejecutando el nodo `controller_3pi` para una meta dinámica



Nota. La figura muestra la trayectoria seguida por el robot con ID 3 implementando el controlador punto a punto para un agente con una meta dinámica. Elaboración propia.

Con estas pruebas, se valida el correcto funcionamiento del nodo de control para agentes individuales. Se corroboró que la arquitectura de control por capas genera los datos controlados al observar el desempeño del robot. Todo esto se logró haciendo uso del sistema de captura de movimiento de forma compatible con ROS 2. Además de emplear el sistema de captura de movimiento para la retroalimentación del lazo de control, también se utilizó para obtener la referencia posición de la meta del controlador punto a punto.

8.2. Escalabilidad del sistema: control de múltiples agentes

ROS 2, tiene capacidad de hacer aplicaciones de robótica y control más complejas que las realizadas en el nodo de control del agente individual. La arquitectura de ROS 2 permite hacer aplicaciones modulares y escalables que implementan múltiples nodos dentro de una misma aplicación. Si bien el alcance de este proyecto no consideró el control de múltiples agentes robóticas de forma simultanea, se consideró apropiado escalar el nodo de control individual, con el fin de probar los beneficios del desarrollo robótico con ROS 2.

A continuación, se presentan dos diferentes formas en las que se pueden controlar múltiples agentes con el mismo controlador individual, así como un nodo adicional, diseñado específicamente para el manejo de nueve agentes funcionando en paralelo.

8.2.1. Múltiples instancias del nodo

De forma directa, se puede utilizar el mismo nodo con múltiples instancias corriendo en diferente terminal. Este metodo de trabajo no requiere más que correr la cantidad de controladores deseados para los robots que se utilizaran. La única condición para realizar este procedimiento es indicarle a ROS 2 un nombre bajo el cual se ejecuta el nodo. Si bien el nodo se llama igual, ROS 2 generá conflicto al tener el mismo nodo suscrito y publicando a diferentes tópicos. Por eso se le asigna un nombre temporal para la ejecución de cada nodo, esto no cambia el nombre del nodo, solo el identificador con el que ROS 2 lo reconoce, permitiendo separar recursos.

A continuación se presenta como ejemplo tres comandos para ejecutar el controlador en tres diferentes robots.

```
ros2 run pololu_controller cp2p_pololu --ros-args -p pose_topic:=/
    pololu_11/pose -p goal_topic:=/Meta_1/pose -p cmd_vel_topic:=/
    cmd_vel_11 -p offset_deg:=263.0     -r __node:=P11controller
```

```
ros2 run pololu_controller cp2p_pololu --ros-args -p pose_topic:=/
    pololu_12/pose -p goal_topic:=/Meta_1/pose -p cmd_vel_topic:=/
    cmd_vel_12 -p offset_deg:=263.0     -r __node:=P12controller
```

```
ros2 run pololu_controller cp2p_pololu --ros-args -p pose_topic:=/
    pololu_13/pose -p goal_topic:=/Meta_4/pose -p cmd_vel_topic:=/
    cmd_vel_13 -p offset_deg:=261.0     -r __node:=P13controller
```

Es importante notar que cada comando se debe ejecutar en diferentes terminales. La instrucción `-r __node:=` es la encargada de otorgarle un nombre de ejecución al nodo, por lo que es muy importante utilizarla.

En este ejemplo se ejecutó el nodo de control en los agentes con identificación 11, 12 y 13. Como se observa en el ejemplo, esta forma de ejecutar varios nodos de control permite asignar diferente meta a cada robot, pero tambien se puede asignar la misma.

Es importante notar que cada comando se debe ejecutar en terminales separadas. La instrucción `-r __node:=` es la encargada de otorgarle un nombre de ejecución al nodo, por lo que es muy importante utilizarla.

En este ejemplo, se ejecutó el nodo de control en los agentes con identificación 11, 12 y 13. Como se observa en el ejemplo, esta forma de ejecutar varios nodos de control permite asignar una meta diferente a cada robot, pero **también** es posible asignar la misma.

8.2.2. Archivos de lanzamiento

Si bien la ejecución de múltiples instancias del nodo es sencilla, puede volverse una tarea prolongada cuando se utilizan bastantes nodos, especialmente al requerir su inicialización repetida para realizar pruebas de funcionamiento. La ejecución secuencial de múltiples comandos consume una cantidad considerable de tiempo. ROS 2 ofrece una solución eficiente para esto mediante los archivos de lanzamiento (`.launch`). Estos archivos permiten inicializar múltiples nodos utilizando un único comando, lo cual es fundamental para el manejo de aplicaciones robóticas complejas. El archivo de lanzamiento también facilita la configuración de los nodos con parámetros predefinidos en archivos de tipo `.yaml`, aunque esto no limita la configuración dinámica de los nodos.

A partir de esto, se desarrolló un archivo de lanzamiento específico capaz de ejecutar hasta nueve instancias del controlador individual. Este archivo permite indicar los identificadores de los robots que se desean utilizar, aceptando la lista de identificadores en dos formatos: especificando la cantidad de agentes o proporcionando una lista explícita.

El siguiente comando ejecuta los nodos correspondientes a los primeros cinco identificadores de la lista predefinida en el Cuadro 5, por ejemplo: 3, 4, 6, 7, 8:

```
ros2 launch pololu_controller multi_controller_launch.py agentes:=5
```

Alternativamente, el siguiente comando ejecuta los nodos correspondientes a la lista explícita de identificadores proporcionada:

```
ros2 launch pololu_controller multi_controller_launch.py agentes:="
    3,7,12"
```

Esta funcionalidad simplifica la ejecución de múltiples nodos de control, eliminando la necesidad de utilizar varias terminales, concentrando la operación en un solo comando, lo que facilita enormemente las pruebas con varios agentes.

Esta forma de ejecutar múltiples controladores resultó útil; sin embargo, según las pruebas realizadas, se observó que las constantes del controlador no se comportaron de la misma manera al aumentar el número de nodos en ejecución. Conforme se lanzaron más nodos, el sistema operó de manera más lenta, lo que provocó que la respuesta de todos los controladores se volviera más lenta y menos precisa. Para ejecutar más de dos nodos en paralelo fue necesario ajustar nuevamente los parámetros del controlador.

Esto ocurrió porque, al ejecutar varios nodos individuales en una misma máquina, cada uno compitió por los recursos de CPU, tiempo de procesamiento, manejo de hilos y comunicación interna de ROS 2. En consecuencia la frecuencia efectiva de los ciclos de control disminuyó y se generaron retardos en la recepción de datos. Estos retrasos hicieron que el controlador reaccionara con menor rapidez, alterando el comportamiento esperado las ganancias del controlador. Además, el aumento en la publicación y suscripción simultánea incrementó la carga dentro del sistema de comunicación, añadiendo latencia adicional.

Debido a estas limitaciones, este método no se consideró adecuado para aplicaciones que involucraran más de tres robots utilizando el controlador punto a punto para agentes individuales. En las pruebas experimentales se logró operar hasta seis robots, pero con una

respuesta notablemente más lenta.

En conclusión, aunque este método no resultó ideal para aplicaciones multiagente, se mantuvo la implementación con el fin de mostrar el uso de los archivos `launch` y su utilidad para automatizar la ejecución de múltiples nodos.

8.2.3. Nodo de control de múltiples agentes

Con el fin de mitigar la latencia introducida por la ejecución de múltiples nodos, se implementó un único nodo para el control de múltiples agentes. Este nodo realizó el mismo control que el nodo `controller_3pi`, pero permitió ejecutar el control para nueve robots simultáneamente con el mismo desempeño que se tuvo al utilizar el nodo individual. Este enfoque presentó varias ventajas sobre el lanzamiento de nueve nodos diferentes.

La diferencia entre la ejecución de nueve nodos separados comparada con ejecutar un nodo que calculó nueve ciclos de control fue significativa. Cuando se usaron nodos separados, cada nodo requirió un temporizador de control independiente, utilizando más recursos de memoria y procesamiento. En contraste, cuando se ejecutó el nodo multiagente, solo se tuvo un temporizador de control que iteró sobre los nueve robots secuencialmente, lo que minimizó la memoria utilizada y la sobrecarga del sistema operativo.

Dado que el controlador utilizó una frecuencia de control de 30 Hz, el nodo contó con 33.33 ms disponibles para ejecutar las instrucciones del controlador en cada ciclo. El controlador se compuso de operaciones básicas como cálculo de distancias, ángulos y las leyes de control PID, las cuales se calcularon en un tiempo considerablemente menor al tiempo disponible por ciclo, incluso cuando se ejecutaron para los nueve robots. La baja complejidad computacional de estas operaciones permitió mantener la frecuencia de control deseada con un amplio margen antes del siguiente ciclo de control.

La programación se realizó de forma que las funciones de recepción de datos y publicación de comandos no bloquearan el ciclo de control, permitiendo que este se ejecutara de manera asíncrona. A su vez, cada controlador tuvo su espacio de memoria designado para el estado y los errores acumulados, por lo que no se mezclaron variables entre robots en ningún momento. Al utilizar un solo nodo, la cantidad de información impresa en la terminal disminuyó drásticamente, facilitando el monitoreo del sistema. Este nodo tuvo la capacidad de ejecutar más de nueve controladores debido a la baja complejidad numérica del algoritmo de control, pero para las pruebas realizadas, solo se tuvo acceso a nueve robots funcionales.

Si bien este nodo cumplió correctamente con el sistema de control para múltiples agentes, se pueden mencionar ciertas desventajas al compararlo con múltiples nodos individuales. La desventaja principal fue que el fallo del nodo repercutió en todos los robots, deteniendo el control de todos simultáneamente. Adicionalmente, identificar y corregir errores en el comportamiento de un robot específico resultó más difícil debido a que los mensajes de todos los robots se generaron en un mismo flujo.

Sin embargo, este nodo demostró ser adecuado para el manejo de múltiples agentes, mostrando una respuesta idéntica a la obtenida con el nodo individual para cada robot. Por esta razón, este nodo se recomendó para implementar algoritmos de enjambre o aplicaciones

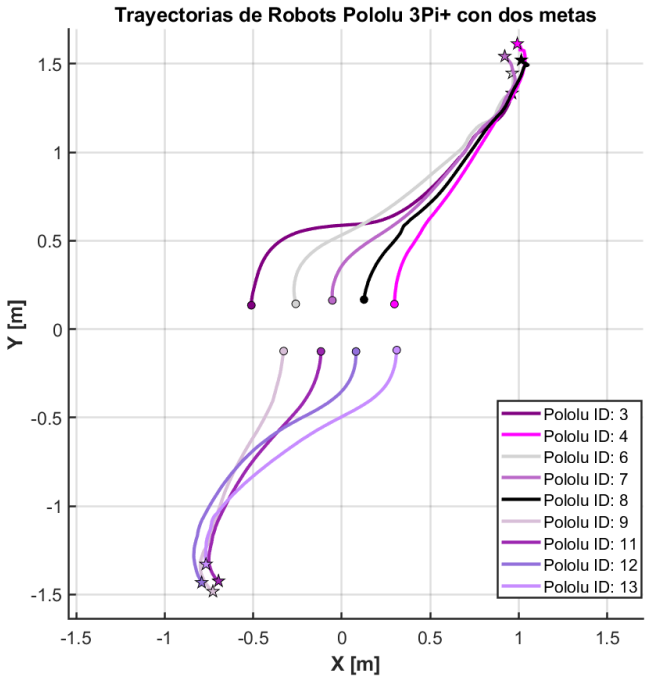
con múltiples agentes que requieran coordinación centralizada.

8.2.4. Resultados experimentales

Los resultados del controlador para múltiples agentes fueron generados tras correr el comando `ros2 run pololu_controller multi_agent_controller`. Para las diversas pruebas de funcionamiento, se ejecutó el control de nueve robots con diferentes configuraciones de metas. Primeramente, se realizaron pruebas con dos metas, como se observó en la Figura 24. Luego, se procedió al uso de tres metas, como se mostró en la Figura 25. Por último, se utilizaron cuatro metas, como se presentó en la Figura 26.

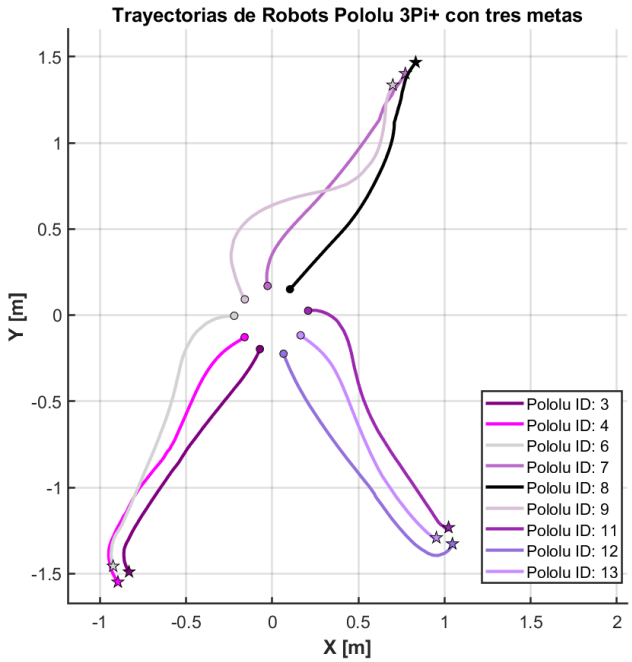
Estos resultados mostraron el comportamiento de los robots Pololu 3Pi+ siendo controlados por el mismo nodo, ejecutando tareas de forma simultánea. Se validó la correcta implementación del nodo, permitiendo su uso para futuras aplicaciones más complejas.

Figura 24. Trayectorias de nueve agentes robóticos con dos metas



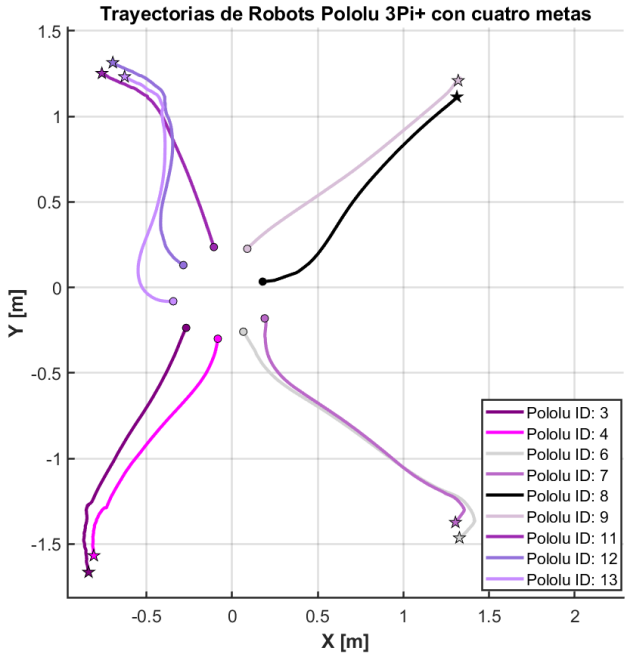
Nota. La figura presenta el comportamiento de nueve agentes robóticos al utilizar el nodo de control multiagente con dos metas. Elaboración propia.

Figura 25. Trayectorias de nueve agentes robóticos con tres metas



Nota. La figura presenta el comportamiento de nueve agentes robóticos al utilizar el nodo de control multiagente con tres metas. Elaboración propia.

Figura 26. Trayectorias de nueve agentes robóticos con cuatro metas

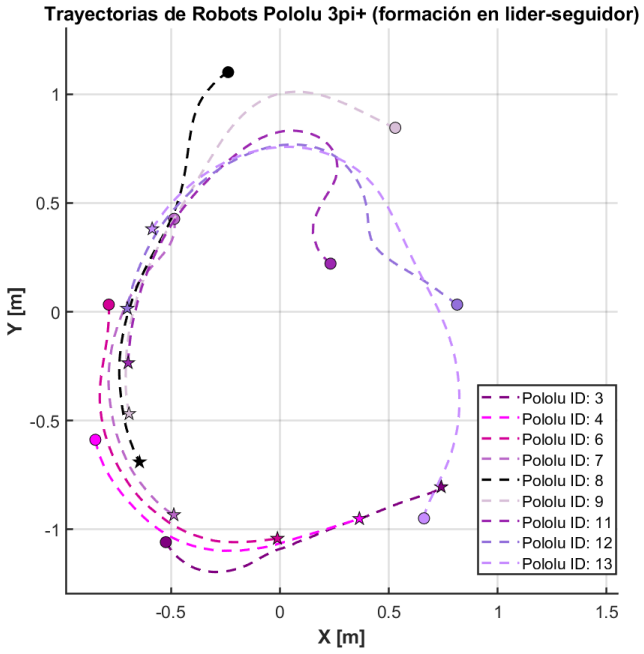


Nota. La figura presenta el comportamiento de nueve agentes robóticos al utilizar el nodo de control multiagente con cuatro metas. Elaboración propia.

8.3. Aplicación demostrativa: control líder-seguidor

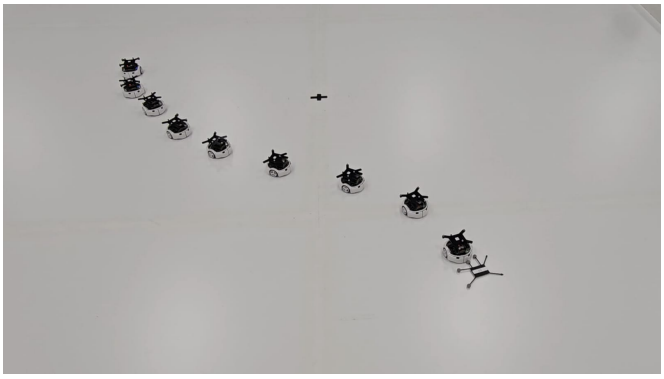
Haciendo uso del control de múltiples agentes, se desarrolló un nodo de aplicación demostrativo. Este nodo implementó el mismo sistema de control y la misma lógica; la modificación que se realizó fue en la asignación de metas a cada robot. Según el listado de identificadores de los robots presentado en el Cuadro 5, el primer robot tuvo como meta un marcador del sistema de captura de movimiento, el segundo robot persiguió al primero, el tercero al segundo y así consecutivamente. Básicamente, el nodo no cambió, solo que las metas se tomaron como las posiciones de los mismos robots, provocando que estos se persiguieran de forma ordenada. Este nodo demostrativo también incluyó el parámetro `safety_distance`, utilizado para evitar que los robots se acercaran demasiado y colisionaran. Por defecto, este parámetro se configuró en 14 cm, lo que mostró ser una distancia adecuada para el seguimiento. En la Figura 27 se apreció el comportamiento líder-seguidor, donde se observó cómo las posiciones finales mantuvieron un distanciamiento entre robots. El comportamiento de los nueve agentes fue del tipo líder-seguidor y generó una formación de fila cuando se siguieron entre sí, como se observó en la Figura 28.

Figura 27. Trayectorias de nueve agentes robóticos con el nodo demostrativo



Nota. La figura presenta el comportamiento de nueve agentes robóticos al utilizar el nodo demostrativo. Elaboración propia.

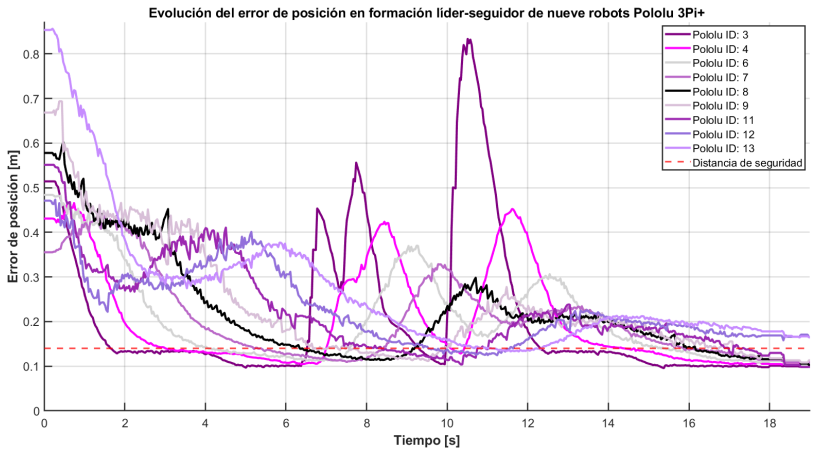
Figura 28. Fotografía del comportamiento de los robots con el nodo demostrativo



Nota. La figura presenta el comportamiento de nueve agentes robóticos al utilizar el nodo demostrativo, presentando una formación de fila. Elaboración propia.

Como validación del correcto funcionamiento, tanto del controlador como de la distancia de seguridad para evitar colisiones, se generó la Figura 29, donde se graficó el error de posición de los nueve robots. El error de posición convergió hacia la distancia de seguridad mostrada por la línea punteada roja. Se observaron varios picos en el error, ya que durante la prueba, el marcador de referencia para el primer robot se alejó constantemente de los robots para mantener el movimiento. Si bien la convergencia no fue exacta en la distancia de seguridad, fue cercana y no representó una mala implementación. La razón fue que, al estar en constante movimiento, los robots no alcanzaron la meta debido a que esta se movió más rápido de lo que ellos se desplazaron. Si la meta se hubiera detenido, los robots se habrían acercado entre sí hasta alcanzar la distancia predefinida.

Figura 29. Error de posición de los robots con respecto al líder de cada robot



Nota. La figura presenta el error de posición o la distancia euclidiana entre la posición del robot y su referencia de posición. En este caso, la referencia de cada robot se denominó líder, mientras que seguidor se denominó al robot que persiguió al líder. Elaboración propia.

Este nodo demostrativo permitió mostrar la versatilidad del nodo de control de múltiples agentes, así como el buen desempeño de los robots realizando tareas coordinadas. Dado que los robots se organizaron en fila, los errores de posición se propagaron a lo largo de la formación. Por lo tanto, el funcionamiento adecuado observado en los nueve robots indicó que los errores fueron mínimos, ya que, de lo contrario, se habrían acumulado y amplificado a través de la cadena de seguidores. Este comportamiento líder-seguidor demostrado abre la puerta a diversas aplicaciones futuras. Entre estas se encuentran la simulación de tráfico vehicular con diferentes tiempos de respuesta por agente, permitiendo estudiar fenómenos como la formación de congestionamientos. Asimismo, este sistema podría utilizarse para simular evacuaciones en emergencias, donde grupos de personas siguen a líderes designados hacia zonas seguras. Otras aplicaciones potenciales incluyen el transporte autónomo de materiales en cadena, la formación de convoyes robóticos para exploración, y el estudio de dinámicas colectivas en sistemas multiagente. La capacidad demostrada del nodo para coordinar múltiples robots de manera estable y eficiente lo convierte en una herramienta valiosa para investigaciones futuras en robótica colaborativa.

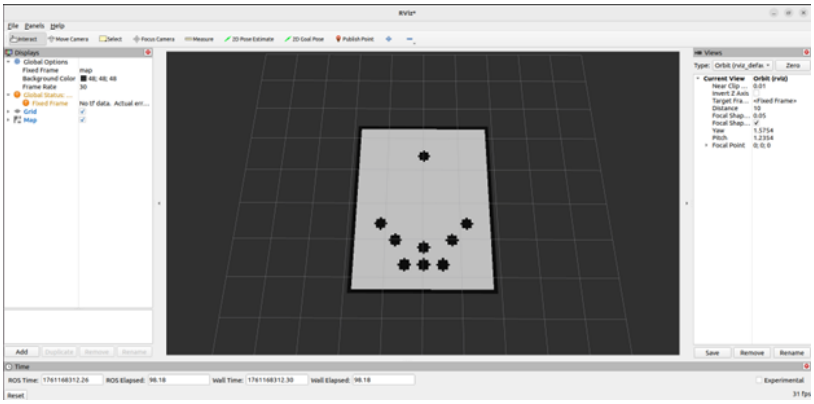
8.4. Nodo generador de mapa del laboratorio Robotat

Se desarrollo un nodo generador de mapa de la plataforma Robotat, donde se visualiza la ubicación de los agentes roboticos ubicados dentro de la plataforma, así como todos los marcadores del sistema de captura de movimiento. Este nodo permite visualizar en tiempo la pose de los robots, así como generar archivos `.pgm` y `.yaml`. Dichos archivos cumplen el formato adecuado para representar mapas de obstáculos y ocupación del paquete de navegación Nav2 de ROS 2.

La motivación de la creación de este mapa es implementar algoritmos de navegación y planificación de movimiento, ejecutando tareas de evasión de obstáculos dinámicos y estáticos. Dado que estas aplicaciones se encuentran fuera del alcance del proyecto, se diseño el nodo generador de mapa, para servir como antecedente al uso de estos paquetes de navegación.

En la Figura 30 se observa una captura, donde se muestra la aplicación RVIZ2 de ROS 2, en esta aplicación se visualiza el mapa generado por el nodo y cuenta con actualización en tiempo real. El comando para ejecutar el nodo es: `ros2 run map_generator map_generator_node`

Figura 30. Captura del mapa generado para visualización del Robotat



Nota. La figura el entorno RVIZ2 con la visualización del mapa del laboratorio Robotat generado por el nodo de mapeo Elaboración propia.

Esta implementación permite observar el comportamiento de los robots y marcadores en forma de obstaculos compatibles con el paquete Nav2. Por las caracteristicas del mapa, los obstaculos mostrados tienen un tamaño radial definido por el usuario de forma de parametro `obstacle_radius`.

De esta forma este nodo sirve como base para la futura implementación de aplicaciones con planificación de movimiento y mapeo.

CAPÍTULO 9

Conclusiones

El trabajo desarrollado permitió cumplir de manera satisfactoria los objetivos planteados al inicio de la investigación. La integración de micro-ROS en el microcontrolador ESP32 del robot Pololu 3pi+ demostró ser factible y funcional dentro del ecosistema Robotat, confirmando la hipótesis de que es posible extender las capacidades de ROS 2 a plataformas educativas de bajo costo mediante el uso de microcontroladores.

En términos de los objetivos específicos, se alcanzaron los siguientes resultados:

- Se implementó micro-ROS en el microcontrolador ESP32-WROOM-32D mediante comunicación wifi, utilizando el protocolo UDP y un puente MQTT para la integración con ROS2. El firmware se desarrolló en PlatformIO y permitió recibir referencias de velocidad lineal y angular desde ROS2, con soporte para hasta diez agentes simultáneos. Esta implementación facilitó la comunicación bidireccional entre los robots Pololu 3Pi+ y el sistema de control principal.
- Se implementó el sistema de captura de movimiento OptiTrack en los nodos de control de ROS2 para obtener la posición y orientación de los robots en tiempo real. La implementación se realizó mediante Mocap4ros2 con una tasa de actualización de 90 Hz, y se utilizó un puente MQTT para ROS2 que proporcionó datos a 30 Hz. La configuración constó de quince tópicos de pose que permitieron el seguimiento simultáneo de múltiples agentes y marcadores de referencia para los nodos de control.
- Se desarrolló un controlador punto a punto en ROS2 para el robot Pololu 3Pi+, combinando un controlador PID para la orientación y un acercamiento exponencial para el error de posición. El controlador se implementó en dos versiones: un nodo para control individual y un nodo para control multiagente. Este último se probó exitosamente con nueve robots de forma simultánea, demostrando la escalabilidad y eficiencia del sistema desarrollado.

CAPÍTULO 10

Recomendaciones

Se recomienda utilizar el microcontrolador ESP32 para enviar datos de odometría por medio de micro-ROS. El uso de estos datos permitiría alimentar paquetes de navegación y planificación de movimiento. A su vez, se puede utilizar la odometría del robot para reforzar los conceptos de fusión de sensores del curso de robótica 2 de la Universidad del Valle de Guatemala

Se recomienda sensores adicionales como LiDAR y cámaras, lo cual permitiría dotar a los robots de percepción del entorno más allá del sistema de captura de movimiento. Esto abre la posibilidad de implementar algoritmos de mapeo y localización simultánea (SLAM), que constituyen un componente esencial en sistemas móviles autónomos al brindar mayor independencia y robustez frente a entornos cambiantes.

De igual manera, se recomienda explorar el uso de paquetes avanzados de navegación y planificación de movimiento disponibles en ROS 2, como `Nav2`, los cuales integran de manera modular funcionalidades de planificación de trayectorias, evasión de obstáculos y control adaptativo. La utilización de estos paquetes permitiría ampliar el alcance de la arquitectura presentada en este trabajo hacia aplicaciones más sofisticadas y cercanas a los estándares de la robótica moderna. Con ello, el ecosistema universitario podría disponer de una plataforma completa de investigación que facilite la transición hacia proyectos de mayor complejidad y escalabilidad.

CAPÍTULO 11

Referencias

[1] C. Perafán, «Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica,» Tesis de licenciatura, Universidad del Valle de Guatemala, Guatemala, 2021.

[2] D. Mencos, «Integración de una computadora central en el Rover UVG para la ejecución de ROS,» Tesis de licenciatura, Universidad del Valle de Guatemala, Guatemala, 2023.

[3] J. Pu, «Desarrollo de herramientas de programación y simulación para los agentes robóticos Pololu 3Pi+ dentro del ecosistema Robotat,» Tesis de licenciatura, Universidad del Valle de Guatemala, Guatemala, 2024.

[4] B. Li, Z. Ma e Y. Zhao, «2D Mapping of Mobile Robot Based on micro-ROS,» en *2022 International Symposium on Robotics, Intelligent Manufacturing Technology (ISRIMT)*, vol. 2402, IOP Publishing, 2022, pág. 012 030. DOI: 10.1088/1742-6596/2402/1/012030.

[5] P. Nguyen, «MICRO-ROS FOR MOBILE ROBOTICS SYSTEMS,» 30.0 credits, Master of Science in Engineering - Robotics, School of Innovation Design y Engineering, Västerås, Sweden, jun. de 2022.

[6] T. Solis, «Implementación de infraestructura para el control de enjambres robóticos con ROS 2 y captura de movimiento dentro del ecosistema Robotat,» Tesis inédita. Universidad del Valle de Guatemala, 2025, 2025.

[7] Pololu Corporation, *Pololu 3pi+ 32U4 User's Guide*, Accessed: 2025-08-18, Pololu Corporation, 2023. dirección: https://www.pololu.com/docs/pdf/0J83/3pi_plus_32u4.pdf.

[8] Espressif Systems, *ESP32 Technical Reference Manual Version 5.5*, https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf, Manual técnico con detalles de hardware, arquitectura y periféricos del ESP32, 2025.

[9] Espressif Systems, *ESP32 Series Datasheet Version 5.0*, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, Hoja de datos oficial con especificaciones generales del ESP32, 2025.

[10] micro-ROS Project / Open Robotics, *micro-ROS porting to ESP32*, https://micro.ros.org/blog/2020/08/27/esp32/, Port oficial de micro-ROS al ESP32-DevKitC-32E con soporte para FreeRTOS y transporte serial/Wi-Fi, 2020.

[11] Espressif Systems (Shanghai) Co., Ltd, *ESP32-DevKitC V4 User Guide*, Accessed: 2025-09-16, Espressif Systems, 2025. dirección: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html.

[12] B. Siciliano y O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[13] G. F. Franklin, J. D. Powell y A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 6th. Prentice Hall, 2010.

[14] N. S. Nise, *Control Systems Engineering*, 6th. John Wiley & Sons, 2011.

[15] K. M. Lynch y F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[16] N. Correll, K. Bekris, D. Berenson y D. Rus, *Introduction to Autonomous Robots*. MIT Press, 2022.

[17] Open Robotics, *ROS (Robot Operating System)*, https://www.ros.org/, Accessed: 2025-09-17, 2025.

[18] E. Renard, *ROS2 from Scratch*, 1.ª ed. Packt Publishing, 2024, ISBN: 978-1-83588-140-8.

[19] eProsima, *micro-ROS: ROS 2 for microcontrollers*, https://micro.ros.org/, Accessed: 2025-09-19, 2025.

[20] micro-ROS, *micro-ROS library for Platform.IO*, https://github.com/micro-ROS/micro_ros_platformio, Repositorio GitHub; licencia Apache-2.0. Accessed: 2025-09-19, 2025.

[21] *Distributions*, https://docs.ros.org/en/rolling/Releases.html, Consultado: 23 de septiembre de 2025, Open Robotics, 2025.

[22] *ROS2 Humble Hawksbill — Releases*, https://docs.ros.org/en/rolling/Releases/Release-Humble-Hawksbill.html, Consultado: 23 de septiembre de 2025, Open Robotics, 2022.

[23] C. Ltd., *Ubuntu 22.04.5 LTS (Jammy Jellyfish) Release*, https://releases.ubuntu.com/jammy/, Accedido: 23 de septiembre de 2025, 2022.

[24] O. Corporation, *VirtualBox Downloads*, https://www.virtualbox.org/wiki/Downloads, Accedido: 23 de septiembre de 2025, 2025.

[25] O. Robotics, *Ubuntu (deb packages)*, https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html, Accedido: 24 de septiembre de 2025, 2025.

[26] micro-ROS, *micro-ROS Agent: ROS 2 node wrapping the Micro XRCE-DDS Agent*, https://github.com/micro-ROS/micro-ROS-Agent, Repositorio GitHub. Accessed: 2025-09-19, 2019.

[27] Google Large Language Model. «Conversación sobre la función rcl executor spin some,» Google, visitado 29 de sep. de 2025. dirección: https://g.co/Bard.

CAPÍTULO 12

Anexos

En este anexo se presenta el enlace al repositorio de GitHub que contiene el código fuente completo desarrollado durante este trabajo. El repositorio incluye el firmware de micro-ROS, los nodos de control de ROS 2, archivos de configuración y documentación técnica necesaria para la reproducción de los resultados presentados.

12.1. Repositorio de Github

A continuación se presenta el enlace al repositorio de Github en donde se presenta todo el código desarrollado en este proyecto, junto con instrucciones de uso:

https://github.com/ByranOrtiz/Herramientas-de-software-para-control-de-Pololu-3Pi-con-ROS2-y-micro-ROS-en-el-ESP32.git