Desarrollo de un Robot Móvil Autónomo para Entornos Logísticos en Interiores

Juan Francisco García Rodríguez
Integración de robótica y sistemas inteligentes
Tecnológico de Monterrey
Ciudad de México, México
a01660981@tec.mx

Jennifer Lizeth Avendaño Sánchez
Integración de robótica y sistemas inteligentes
Tecnológico de Monterrey
Ciudad de México, México
a01656951@tec.mx

Juan Antonio Mancera Velasco
Integración de robótica y sistemas inteligentes
Tecnológico de Monterrey
Ciudad de México, México
a01662634@tec.mx

Johan Donato Cabrera Martinez
Integración de robótica y sistemas inteligentes
Tecnológico de Monterrey
Ciudad de México, México
a01657922@tec.mx

***Abstract*—Este artículo presenta el desarrollo e integración de un Robot Móvil Autónomo (AMR) orientado a tareas logísticas en interiores, en el contexto de automatización de almacenes. El sistema fue construido mediante una arquitectura modular basada en herramientas de código abierto y el ecosistema ROS 2. Se implementó sobre una plataforma física originalmente no diseñada para navegación autónoma, la cual fue adaptada exitosamente para incorporar capacidades avanzadas de percepción, localización y control. La solución combina una unidad de procesamiento de alto desempeño con sensores inerciales y LiDAR, un controlador electrónico de velocidad programable y una unidad embebida delegada para el control de bajo nivel a través de micro-ROS. El robot ejecuta mapeo, navegación autónoma y control híbrido, alternando entre operación manual y autónoma según el contexto. Esta implementación constituye una referencia técnica sólida, replicable y compatible con los estándares modernos de desarrollo en robótica móvil. El sistema fue validado experimentalmente en un entorno logístico estructurado, demostrando navegación confiable, precisión operativa y potencial de escalabilidad.**

***Index Terms*—Robot Móvil Autónomo (AMR), ROS 2, Automatización Logística y de Almacenes, Fusión de Sensores, SLAM, Controladores VESC, Jetson Orin Nano.**

I. Introducción

La Industria 4.0, también conocida como la Cuarta Revolución Industrial, marca una transformación profunda en los procesos productivos mediante la integración de tecnologías digitales avanzadas. Este paradigma se caracteriza por la fusión de los mundos físico y virtual a través de sistemas ciberfísicos, sensores inteligentes, análisis de datos en tiempo real y automatización conectada, lo que permite una producción más flexible, eficiente y autónoma [1][2].

Dentro de este contexto, la Logística 4.0 surge como una respuesta estratégica a la creciente complejidad de las cadenas de suministro modernas. Su objetivo es mejorar la trazabilidad, la eficiencia operativa y la capacidad de adaptación de los sistemas logísticos mediante la incorporación de tecnologías como el Internet de las Cosas (IoT), la inteligencia artificial (IA), y la robótica móvil autónoma [3]. En particular, los Robots Móviles Autónomos (AMR) se han consolidado como herramientas clave para automatizar el transporte interno de mercancías, sobre todo en entornos industriales estructurados como almacenes y centros de distribución.

Este trabajo presenta el desarrollo e integración de un sistema AMR funcional, diseñado específicamente para operar en un entorno logístico controlado, compuesto por pasillos estrechos, estaciones de carga y zonas de entrega predefinidas. La solución fue implementada sobre una plataforma física preexistente, originalmente concebida para operaciones manuales, la cual fue adaptada con capacidades autónomas a través de herramientas de código abierto y una arquitectura modular.

La implementación se llevó a cabo bajo el ecosistema ROS 2, utilizando una Jetson Orin Nano como unidad principal de procesamiento, sensores LiDAR e IMU para la percepción del entorno, controladores VESC para la locomoción y un microcontrolador ESP32 ejecutando tareas de bajo nivel mediante micro-ROS. Esta configuración permitió establecer un sistema distribuido y escalable, alineado con los estándares modernos de la robótica móvil, donde las funciones de percepción, localización, planeación y control operan de manera desacoplada pero coordinada.

Este artículo documenta el proceso de diseño, implementación e integración del sistema, así como los resultados obtenidos durante su validación experimental. La solución propuesta busca demostrar la viabilidad técnica de una arquitectura AMR replicable, construida con tecnologías accesibles y conforme al ecosistema de desarrollo abierto que promueve ROS 2. Su diseño modular y alineado con estándares industriales facilita su adaptación a otros entornos logísticos estructurados, posicionándola como una posible referencia

técnica para desarrollos futuros tanto en contextos académicos como industriales.

II. Arquitectura del Sistema

A. Descripción física del Robocov

El sistema fue desarrollado sobre la plataforma robótica **Robocov**, un robot móvil diferencial con estructura de aluminio tipo perfil Bosch. Su forma general corresponde a un prisma rectangular de aproximadamente $0.84m \times 0.61m \times 0.25m$, con caras de acrílico negro que protegen los módulos internos. El acceso a componentes se realiza retirando únicamente la cara superior, lo que facilita el mantenimiento.

La locomoción se basa en dos motores *brushless* tipo hub montados directamente sobre ruedas traseras, mientras que el eje delantero está estabilizado con ruedas locas. Esta configuración diferencial resulta adecuada para entornos planos y estructurados, como pasillos logísticos. Sin embargo, la distribución de masa dominada por la batería de 48V ubicada al centro del chasis introdujo desafíos en frenado y estabilidad durante maniobras a alta velocidad.

Para cumplir con los requerimientos del caso de uso, se incorporó una caja superior para transportar paquetes con masa variable. Esta estructura se fijó a la cara superior del robot, manteniendo la estabilidad del centro de gravedad. Adicionalmente, se diseñó e imprimió un soporte en 3D para montar de forma alineada el sensor LiDAR RPLIDAR S3 y la cámara Logitech Brio 100 sobre el eje longitudinal del robot, a una altura de $0.62m$ desde el suelo, evitando interferencias con la carga.

A nivel superficial, se integró una caja de control que aloja la ESP32, circuitos auxiliares y una pantalla digital que permite visualizar el voltaje en tiempo real. Esta funcionalidad resultó útil para monitorear el nivel de carga de la batería de 48V y verificar la correcta entrega de energía al controlador VESC. Además, el sistema incluye un botón físico de paro de emergencia montado en la parte posterior, el cual desconecta exclusivamente la alimentación de los motores, preservando el estado de la Jetson y los sensores.

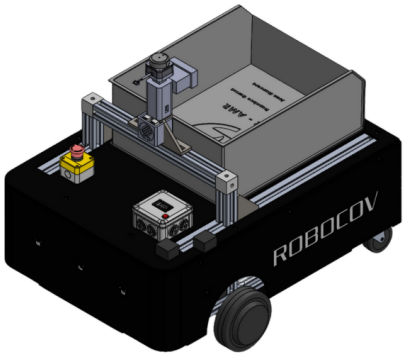


Fig. 1. Modelo físico del Robocov con caja transportadora y soporte elevado para sensores.

B. Topología general del sistema

A nivel funcional, la arquitectura del sistema se organiza en torno a una computadora a bordo: Jetson Orin Nano, que actúa como unidad de procesamiento principal dentro del ecosistema ROS 2. El flujo de datos y control sigue una estructura jerárquica distribuida, como se muestra en la Fig. 2.

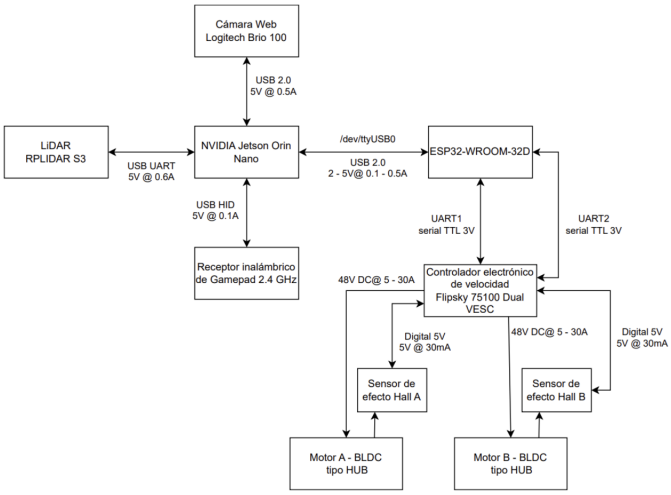


Fig. 2. Diagrama de conectividad del sistema con componentes principales.

La arquitectura resultante puede describirse como una topología en estrella con la Jetson Orin Nano al centro. Desde allí, se extienden conexiones USB, UART y digitales hacia todos los subsistemas. Esta organización facilita el diagnóstico, el mantenimiento y futuras expansiones del sistema, además de respetar principios de desacoplamiento y escalabilidad que caracterizan a los desarrollos ROS 2 modernos.

C. Unidad de procesamiento principal

La Jetson Orin Nano ejecuta la totalidad del stack ROS 2, incluyendo los nodos responsables de percepción visual, adquisición LiDAR, localización probabilística, mapeo, planificación de rutas y lógica de control. Los dispositivos conectados directamente a esta unidad incluyen:

- **Cámara Logitech Brio 100:** conectada por USB 2.0 (5V @ 0.5A), utilizada para seguimiento visual y percepción en carriles.
- **Sensor LiDAR RPLIDAR S3:** conectado mediante adaptador USB-UART (5V @ 0.6A), para adquisición de nubes de puntos en 2D.
- **Receptor inalámbrico de gamepad:** interfaz USB HID (5V @ 0.1A), para control manual en pruebas y validaciones.
- **ESP32-WROOM-32D**: mediante interfaz USB 2.0 (2V-5V @ 0.1A-0.5A), ejecuta micro-ROS y actúa como interfaz de bajo nivel para recibir comandos de velocidad y controlar los motores del Robocov.

D. Delegación de control de bajo nivel

A diferencia de plataformas como TurtleBot4, donde todos los módulos operan desde una sola unidad computacional, Robocov implementa una arquitectura distribuida. Las tareas de locomoción y retroalimentación fueron delegadas a un microcontrolador **ESP32-WROOM-32D** ejecutando micro-ROS, conectado a la Jetson por USB 2.0 en `/dev/ttyUSB0`.

El ESP32 recibe comandos tipo `Twist`, los convierte a velocidades de giro expresadas como eRPM, y transmite esta información por UART en nivel TTL (3.3V) al controlador VESC. Esta delegación reduce la carga computacional de la Jetson y permite mantener la modularidad del sistema.

E. Controladores de tracción y retroalimentación

El controlador Flipsky 75100 Dual VESC gestiona dos motores hub independientes, alimentados directamente desde una fuente de 48V DC capaz de entregar hasta 30A por canal. Cada motor cuenta con sensores de efecto Hall integrados, cuya información es leída por el VESC y reenviada al ESP32 para ser publicada como velocidades angulares en ROS 2. Esta retroalimentación es empleada para odometría y control de trayectoria.

F. Sistemas de paro de emergencia

Para garantizar la seguridad durante la operación del robot, se implementaron dos sistemas de paro de emergencia redundantes. El primero consiste en un botón físico de acción directa montado sobre el chasis del robot. Al ser presionado, interrumpe inmediatamente el suministro de voltaje hacia el controlador Flipsky 75100 Dual VESC, permitiendo una detención rápida en caso de comportamiento inesperado.

El segundo sistema opera de manera remota mediante un relevador de estado sólido controlado por radiofrecuencia. Este mecanismo permite al operador desactivar el sistema de tracción a distancia, resultando particularmente útil cuando el robot se encuentra en movimiento fuera del alcance inmediato del botón físico.

G. Transformaciones de coordenadas y marcos de referencia

En robótica móvil, es esencial manejar múltiples marcos de referencia para describir la posición y orientación de sensores, actuadores y del propio robot. Las transformaciones de coordenadas permiten convertir información entre estos marcos, utilizando matrices de rotación y traslación. El grupo especial SE(2) representa las transformaciones rígidas en el plano (2D), combinando rotaciones y traslaciones [4]. En sistemas como ROS, estas transformaciones se gestionan mediante la librería TF, que mantiene una estructura jerárquica de marcos de referencia en tiempo real [5].

En este proyecto, las transformaciones fueron definidas mediante un archivo URDF (*Unified Robot Description Format*), el cual constituye el estándar en ROS 2 para describir la estructura cinemática del robot. En el URDF, se especifican los *links* (elementos físicos como sensores, actuadores y el chasis) y los *joints*, que representan uniones entre dos *links*. En el caso de Robocov, las ruedas motrices están unidas mediante *joints* de tipo *continuous* respecto al /base_link, permitiendo rotación infinita en el eje Y, mientras que los sensores (LiDAR, cámara, IMU) y ruedas locas están conectados al marco /base_link mediante *joints* de tipo *fixed*, al no presentar movimiento relativo.

El árbol de transformaciones parte del marco global /map y continúa con la transformada estática `/map` $\rightarrow$ `/odom`. La transformada dinámica `/odom` $\rightarrow$ `/base_link` es publicada en tiempo real por el nodo de odometría. A partir de `/base_link` (ubicado en el centro entre las dos ruedas motrices) se definen transformadas fijas hacia cada uno de los sensores y actuadores, asegurando una representación espacial coherente y consistente para todo el sistema.

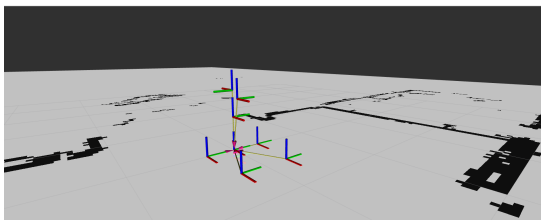


Fig. 3. Visualización de transformadas de Robocov.

Como se observa en la Fig. 3, están las transformadas del sistema visualizadas desde RViz. Cabe destacar que la transformada del sensor LiDAR presenta una rotación de π radianes sobre el eje z respecto a su orientación convencional. Esta inversión fue necesaria debido a la configuración interna del sensor, ya que el plano de escaneo se proyectaba en sentido inverso, resultando en visualizaciones incorrectas en RViz. Esta característica depende del diseño físico del LiDAR, el cual define el eje positivo de los datos de rango. Ajustar esta transformación garantiza la correcta alineación de los datos de escaneo con el entorno real del robot.

H. Diseño asistido por computadora (CAD)

Como parte fundamental del proceso de integración y adaptación a entornos logísticos, fue necesario diseñar e implementar estructuras físicas adicionales mediante modelado CAD. Una de las principales modificaciones fue la creación de una caja de transporte con dimensiones de $50 \cdot 40 \cdot 20cm$, colocada sobre la plataforma superior del robot. Esta caja permite simular escenarios de recolección y entrega de paquetes, y fue fabricada mediante impresión 3D para garantizar un peso ligero y un acoplamiento preciso.

Fig. 4. Caja de transporte utilizada por Robocov.

En la Fig. 4 se puede observar el diseño CAD de la caja, y en la Fig. 1 se muestra montada sobre el Robocov. Además, se diseñó una estructura adicional para el montaje del LiDAR y la cámara RGB-D, elementos clave para la navegación autónoma. Dicha estructura fue concebida para acoplarse fácilmente al perfil Bosch ya existente en la versión anterior del Robocov, asegurando compatibilidad mecánica sin comprometer la estabilidad del sistema.

También se diseñaron otras piezas y componentes estructurales con el fin de modelar completamente el robot en 3D, lo que facilitó tanto la planificación del ensamblaje como la visualización de la integración de sensores y electrónica.

III. Control y locomoción

A. Configuración inicial del control de velocidad

El control de velocidad del robot fue inicialmente implementado sobre una patineta tipo *hoverboard*, con el objetivo de revisar el uso de la controladora *VESC FlipSky Dual Mini FSESC6.7 Pro 140 A* sin comprometer el chasis heredado del robot *Robocov*. Esta primera etapa permitió validar la operación segura y funcional de la controladora y los motores sin riesgo mecánico.

La configuración inicial del sistema de tracción se realizó mediante el software *VESC Tool*, una plataforma ampliamente utilizada para la parametrización y monitoreo de controladores VESC mediante interfaz gráfica [6]. En la sección `Setup Motors FOC` (*Field Oriented Control*), se proporcionaron manualmente algunos parámetros básicos del sistema como entrada para guiar el proceso de identificación eléctrica automática del motor. Estos valores incluyeron:

- Tipo de motor: genérico
- Clasificación estimada: *Medium Outrunner (750g)*
- Configuración de batería: 13 celdas en serie (48V nominal)
- Capacidad estimada de batería: 20Ah
- Tipo de transmisión: accionamiento directo
- Diámetro aproximado de rueda: 160 mm

A partir de esta configuración, se ejecutó el procedimiento de detección de parámetros *FOC*, una técnica de control ampliamente utilizada en motores BLDC que mejora el rendimiento mediante el uso de referencias vectoriales para generar un par constante [7]. Este enfoque, basado en la teoría de control en el marco de referencia del rotor (DQ0), requiere estimar parámetros eléctricos clave del motor.

Los parámetros detectados incluyen:

- Corriente máxima de motor (`Motor Current Max`)
- Resistencia del motor (`Motor R`)
- Inductancias del motor (`Motor L, Lq, Ld`)
- Enlace de flujo magnético (`Flux linkage`)
- Compensación térmica de fuerza contraelectromotriz (`TEMP Comp`)
- Tipo y alineación de sensores; en nuestro caso, se seleccionó `Hall Sensor`.

Estos parámetros son esenciales para el diseño de los lazos internos de corriente, que suelen ser implementados mediante controladores PI ajustados automáticamente por el firmware del VESC una vez conocida la constante de enlace magnético `Lambda`. Este proceso se realiza dentro de la sección `FOC`, donde también puede repetirse la detección de `Motor R` y `Motor L` para mayor precisión.

Una vez identificados los parámetros, fue posible invertir la dirección de giro de los motores, configurar límites de corriente y voltaje seguros (50 A máx.), y establecer ajustes específicos en la pestaña `Motor Settings`. En paralelo, en la sección `Hall Sensors`, se llevó a cabo la detección automática para garantizar una conmutación precisa en el modo por sensor, fundamental para el arranque y control a bajas velocidades.

En la pestaña `App Settings`, se eligió el modo de control por `UART`, ajustando la comunicación a `115200 bps`. Para almacenar la configuración, se utilizaron los botones `M↓` y `A↓`, que graban los parámetros de las secciones de motor y aplicación, respectivamente, en la memoria no volátil del VESC.

Una funcionalidad destacada de *VESC Tool* es la ventana `Experiments`, que permite enviar directamente comandos de velocidad al motor para validación dinámica. Esta herramienta fue clave para detectar que por debajo de aproximadamente `900 ERPM`, el motor no respondía de forma efectiva, comportamiento típico en ciertos controladores BLDC por inestabilidades en el arranque sensorless [8].

Asimismo, la pestaña `Stream Realtime Data` permitió monitorear en tiempo real variables como corriente, tensión, temperatura y velocidad eléctrica del rotor, proporcionando retroalimentación útil para evaluar el comportamiento bajo distintas condiciones de carga.

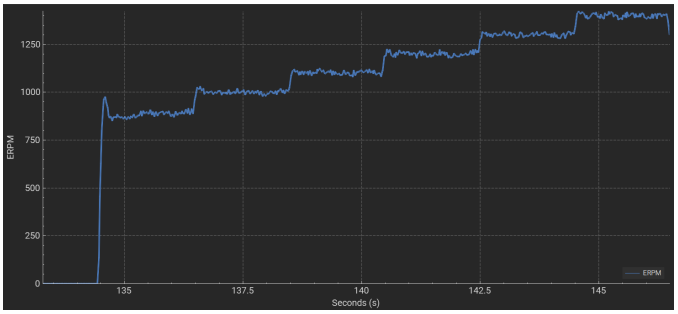


Fig. 5. eRPM del motor vs. tiempo (segundos).

La Fig. 5 muestra una gráfica de respuesta en lazo cerrado del motor, donde se observa cómo el valor de eRPM varía de manera escalonada en función del tiempo. Cada escalón representa un cambio en la referencia de velocidad enviada al motor a través de la interfaz UART, permitiendo evaluar la capacidad de seguimiento del sistema ante distintos niveles de exigencia. Esta prueba es útil para verificar la linealidad del control, así como la estabilidad del comportamiento ante variaciones abruptas de referencia.

B. Modelo cinemático diferencial

La cinemática diferencial permite describir el movimiento del robot a partir de las velocidades angulares de sus ruedas. Este tipo de configuración, ampliamente utilizada en robótica móvil, se basa en un modelo no holonómico que asume que las ruedas giran sin deslizamiento lateral, lo que impone restricciones a su movimiento [9].

Para el Robocov, se considera un radio de rueda $r = 0.08\,\text{m}$ y una distancia entre ruedas $L = 0.60\,\text{m}$. El modelo cinemático se expresa en coordenadas cartesianas y permite obtener la velocidad lineal v y la velocidad angular ω del chasis a partir de las velocidades angulares de las ruedas derecha (ω_R) e izquierda (ω_L):

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{L} & -\frac{r}{L} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \tag{1}$$

Este sistema lineal representa la cinemática directa del robot diferencial, donde la matriz de coeficientes relaciona el espacio de acción (velocidades de rueda) con el espacio de tarea (velocidades del chasis). Esta formulación es esencial para interpretar la odometría y planear trayectorias globales en el entorno.

Esto se puede expandir en forma escalar como:

$$v = \frac{r}{2}(\omega_R + \omega_L) \tag{2}$$

$$\omega = \frac{r}{L}(\omega_R - \omega_L) \tag{3}$$

Donde:

- v: velocidad lineal del centro del eje del robot [m/s]
- ω: velocidad angular respecto al eje vertical (yaw) [rad/s]
- $r = 0.08\,\text{m}$: radio de las ruedas
- $L = 0.60\,\text{m}$: distancia entre las ruedas
- ω_R, ω_L: velocidades angulares de las ruedas derecha e izquierda [rad/s]

Por otro lado, cuando se desea controlar al robot desde el espacio de tarea, es necesario calcular las velocidades angulares necesarias para que las ruedas generen un comando deseado de velocidad lineal v y angular ω. Esta operación corresponde a la cinemática inversa, cuyo modelo matricial es:

$$\begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{L}{2r} \\ \frac{1}{r} & -\frac{L}{2r} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4}$$

Este modelo inverso es especialmente útil en sistemas que generan comandos de velocidad desde un planificador global o una interfaz de usuario, permitiendo traducir dichos comandos a órdenes comprensibles por los motores del robot.

Este modelo es fundamental para el diseño de algoritmos de control, localización y navegación autónoma, ya que permite traducir el movimiento de las ruedas en desplazamientos en el plano y viceversa.

C. Control de ruedas en Micro-ROS

El nodo desplegado en la ESP32 con Micro-ROS se suscribe al tópico `/cmd_vel`, el cual contiene los comandos de velocidad lineal (v) y angular (ω) generados por el planificador global o un operador humano mediante interfaces como un *gamepad*. Para ejecutar estos comandos en los motores, es necesario transformarlos a velocidades angulares de rueda (ω_R, ω_L) utilizando el modelo de cinemática inversa diferencial previamente presentado en la Ecuación 4.

En la ESP32, esta transformación se resuelve en tiempo real, permitiendo el control autónomo o manual del robot con baja latencia. Aunque el modelo se define en forma matricial, su implementación escalar resulta más eficiente para microcontroladores, y se expresa como:

$$\omega_R = \frac{v}{r} + \frac{\omega \cdot L}{2r} \tag{5}$$

$$\omega_L = \frac{v}{r} - \frac{\omega \cdot L}{2r} \tag{6}$$

Donde $r = 0.08\,\text{m}$ es el radio de las ruedas y $L = 0.60\,\text{m}$ la distancia entre ellas.

Estas velocidades angulares se convierten a revoluciones por minuto (RPM) mediante:

$$\text{RPM} = \omega \cdot \frac{60}{2\pi} \tag{7}$$

Y posteriormente a eRPM (*electrical RPM*), unidad utilizada por los VESCs, al multiplicar por el número de polos eléctricos del motor (N_p):

$$\text{eRPM} = \text{RPM} \cdot N_p \tag{8}$$

Dado que no se contaba con la hoja de datos del fabricante, N_p fue estimado experimentalmente mediante pruebas de respuesta escalonada, observando la relación entre eRPMs enviados y el movimiento real del motor. Los valores calculados se envían al VESC a través del comando `setRPM()` de la librería `VESC_UART`, permitiendo un control directo de la velocidad de cada rueda.

Además, el mismo nodo ejecuta periódicamente un *callback* para leer las velocidades actuales reportadas por los VESCs en eRPM. Estas lecturas se convierten de nuevo a radianes por segundo utilizando las transformaciones inversas:

$$\omega = \left(\frac{\text{eRPM}}{N_p}\right) \cdot \frac{2\pi}{60} \tag{9}$$

Así, se obtienen las velocidades angulares reales de cada rueda (ω_R, ω_L), que son publicadas en los tópicos `/velocityEncR` y `/velocityEncL`. Para optimizar el uso de recursos en la ESP32, la publicación se realiza de forma alternada en cada ciclo de control.

D. Uso de gamepad para control manual

Para el control teleoperado de *Robocov* se emplea un gamepad convencional, integrado al sistema mediante el paquete `joy`, el cual forma parte del conjunto base de ROS 2. Este periférico cuenta con dos joysticks analógicos: uno dedicado al control de la velocidad lineal y otro a la velocidad angular. Cada joystick entrega valores normalizados en el rango $[-1, 1]$, donde 0 representa la posición neutral.

Un nodo adicional se encarga de mapear estos valores a velocidades físicas, aplicando una transformación lineal que contempla umbrales definidos por las capacidades mecánicas del robot. Las velocidades lineales están limitadas entre $\pm 0.44 m/s$ (mínimo operativo) y $\pm 0.88 m/s$ (máximo), mientras que las velocidades angulares se restringen entre ± 1.88 rad/s y ± 3.76 rad/s. Para evitar activaciones involuntarias y comportamientos inestables, se implementa una zona muerta (*dead zone*) alrededor del valor cero: cualquier entrada cuya magnitud sea inferior a la velocidad mínima definida se considera nula.

Fig. 6. *Joysticks* del gamepad para control manual.

En la Fig. 6, el *joystick* resaltado en color azul es el que se utiliza para el control lineal (adelante/atrás), mientras que el *joystick* señalado en color rosa funciona para movimientos angulares (izquierda/derecha). Además, se incorporó una combinación de seguridad para alternar entre el modo manual y el modo autónomo: es necesario presionar simultáneamente cuatro botones específicos del control. Esta configuración reduce significativamente el riesgo de alternaciones accidentales por contacto involuntario o ruido eléctrico. La disposición de estos botones se muestra en la Fig. 7.

Fig. 7. Botones para alternar entre modo manual o autónomo.

E. Controladores implementados

Durante el desarrollo se contempló inicialmente utilizar un único controlador para todo el entorno. Sin embargo, en pruebas reales se observó que el seguidor de trayectorias basado en *Pure Pursuit* con PID presentaba oscilaciones impredecibles al avanzar en línea recta, especialmente en pasillos estrechos. Estas desviaciones, aunque mínimas en espacios abiertos, representaban un riesgo de colisión en zonas delimitadas físicamente.

Para mitigar este comportamiento, se implementó un segundo controlador visual reactivo, orientado específicamente a mantener al robot centrado en carriles marcados con líneas amarillas. Este controlador permite un desplazamiento más contenido y robusto en entornos angostos, utilizando únicamente información visual en tiempo real sin depender de planeación previa.

Ambos controladores generan comandos de velocidad que se integran al sistema a través de un nodo de navegación híbrido, activando uno u otro según el contexto operativo.

F. Pure Pursuit + PID

Para la navegación local en espacios abiertos, se desarrolló un nodo en ROS 2 llamado `HybridPursuitPID`, el cual implementa un controlador híbrido que combina el algoritmo *Pure Pursuit* con una etapa de corrección proporcional-integral-derivativa (PID) sobre el error lateral. Esta combinación permite un seguimiento fluido y estable de trayectorias, incluso en presencia de errores de localización o trayectorias curvas generadas por un planificador global.

El algoritmo *Pure Pursuit* funciona identificando un punto objetivo (*lookahead point*) ubicado a una distancia fija delante del robot sobre la trayectoria. Esta distancia de anticipación se definió en 1.2 m. El punto se selecciona dinámicamente entre los puntos de la ruta, empezando desde el último índice conocido y avanzando hasta encontrar uno cuya distancia al robot exceda el *lookahead* definido:

$$\text{dist} = \sqrt{(x_{\text{lookahead}} - x)^2 + (y_{\text{lookahead}} - y)^2}$$

Una vez identificado el punto, se calcula su posición relativa en el marco local del robot mediante una rotación simple con el ángulo de orientación actual θ:

$$x_r = \cos(-\theta) \cdot dx - \sin(-\theta) \cdot dy \tag{10}$$

$$y_r = \sin(-\theta) \cdot dx + \cos(-\theta) \cdot dy \tag{11}$$

Aquí dx y dy son las diferencias en coordenadas absolutas entre el punto de seguimiento y la posición actual del robot.

La componente y_r representa el error lateral, que se utiliza como entrada a un controlador PID cuyos parámetros fueron calibrados experimentalmente como:

$$k_p = 1.2$$

$$k_i = 0.001$$

$$k_d = 0.2$$

La salida angular se calcula en forma discreta considerando:

$\texttt{error} = y_r$

$\texttt{integral} = \sum error \cdot dt$ (limitada entre ± 1.0)

$\texttt{derivada} = \frac{error - error_{\text{prev}}}{dt}$

Con estos términos, la salida angular del controlador es:

$$angular\_z = (k_p \cdot error + k_i \cdot integral + k_d \cdot derivada) \tag{12}$$

La velocidad lineal también se ajusta dinámicamente en función del error lateral:

$$linear\_x = 0.5 \cdot \left(1 - \min\left(\frac{|error|}{2.0}, 1.0\right)\right) + 0.35 \tag{13}$$

Esto garantiza que el robot avance lentamente cuando el error es grande, y con mayor velocidad cuando está bien alineado, pero sin exceder un máximo de 0.50 m/s.

El nodo se ejecuta a 20 Hz (cada 50 ms), y publica los comandos en el tópico `/cmd_vel`. Además, evalúa si el robot ha llegado a su meta final (por cercanía al último punto) para detener la locomoción de manera segura.

En pruebas realizadas, el controlador demostró una navegación precisa y robusta, especialmente en zonas abiertas del entorno logístico. Aunque se observaron ligeras oscilaciones en el seguimiento, estas no comprometieron ni la seguridad ni la trayectoria general. El sistema mantuvo la fluidez en cambios de dirección y curvas, ajustando continuamente la orientación del robot mediante el lazo de retroalimentación del PID.

Este enfoque híbrido es ampliamente utilizado en robótica móvil por su simplicidad, capacidad de generalización, y respuesta eficiente frente a errores pequeños de localización [10].

G. Seguidor de carril con detección HSV y momentos

Se implementó un nodo ROS 2 en Python llamado `LaneFollower`, encargado de realizar seguimiento de carril en zonas estrechas, como pasillos logísticos, sin necesidad de planeación global. Este nodo opera en tiempo real sobre imágenes capturadas por una cámara frontal, utilizando procesamiento de imágenes y control visual tipo PID para mantener al robot centrado dentro del carril.

La imagen recibida del tópico `/image_raw`, con resolución de 640×480 píxeles, se convierte del formato ROS a OpenCV (BGR8) mediante `CvBridge`. Luego, se transforma al espacio de color HSV, más robusto frente a variaciones de iluminación, y se aplica una máscara binaria con el rango de color característico del amarillo:

`lower_yellow` = [18, 120, 120]
`upper_yellow` = [35, 255, 255]

La segmentación se realiza con `cv2.inRange()`, generando una imagen binaria donde los píxeles correspondientes al carril son blancos. Para eliminar ruido, se aplica erosión morfológica con un kernel 5×5, lo cual mejora la definición de los bordes detectados.

Se define una región de interés (ROI) en el 30% inferior de la imagen (de $y = 336$ a $y = 480$), enfocando el procesamiento en el área más cercana al robot:

$$\texttt{ROI} = I_{\text{eroded}}[336:480,:]$$

En esta región, se detectan contornos mediante `cv2.findContours()`. Para cada contorno con área mayor a 100 píxeles, se calculan los momentos espaciales [11]:

$$m_{00} = \sum_x \sum_y I(x,y) \tag{14}$$

$$m_{10} = \sum_x \sum_y x \cdot I(x,y) \tag{15}$$

$$m_{01} = \sum_x \sum_y y \cdot I(x,y) \tag{16}$$

El centroide de cada contorno se obtiene como:

$$c_x = \frac{m_{10}}{m_{00}}, \quad c_y = \frac{m_{01}}{m_{00}} \tag{17}$$

Cuando se detectan al menos dos centroides, se consideran como bordes izquierdo y derecho del carril, y el centro se estima como:

$$\texttt{lane\_center} = \frac{c_{x,left} + c_{x,right}}{2} \tag{18}$$

El error lateral es la diferencia entre el centro de la imagen (320) y el centro del carril, corregido empíricamente por un desfase de montaje:

$$\texttt{error} = 320 - \texttt{lane\_center} - 20$$

Este error alimenta un controlador PID cuyos parámetros fueron calibrados experimentalmente como:

$$k_p = 0.01 \qquad k_i = 0.001 \qquad k_d = 0.002$$

El controlador opera en forma discreta, con:

`error` = valor actual del error lateral

`integral` = $\sum \texttt{error} \cdot dt$ (limitado a ± 1.0)

`derivada` = $\frac{\texttt{error} - \texttt{error}_{\text{prev}}}{dt}$

El intervalo de tiempo dt se calcula dinámicamente en cada iteración utilizando el reloj interno de ROS 2, lo cual permite mantener una temporización precisa para el cálculo del controlador PID.

La velocidad lineal del robot se mantiene constante en 0.50 m/s, mientras que la salida angular se calcula como:

$$angular\_z = (k_p \cdot error + k_i \cdot integral + k_d \cdot derivada) \tag{19}$$

Esta señal es publicada continuamente al tópico `/cmd_vel` para accionar el movimiento correctivo del robot.

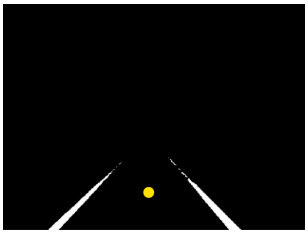


Fig. 8. Imagen binaria del carril detectado. El punto amarillo marca el centro de la imagen (no el centroide), añadido para referencia visual.

Este tipo de control visual reactivo es eficaz en entornos estructurados y limitados espacialmente, donde no es práctico depender de mapas ni localización absoluta. El uso de centroides para seguimiento visual es ampliamente utilizado en robótica por su simplicidad y eficiencia computacional [12].

H. Nodo de navegación híbrida

Dado que los controladores desarrollados tienen fortalezas en contextos distintos el *Pure Pursuit* combinado con PID para trayectorias abiertas, y el seguidor visual reactivo para carriles estrechos, se diseñó un nodo supervisor llamado `navigation_node.py` que permite alternar entre ambos modos de navegación según el estado operativo del robot.

Este nodo recibe la pose estimada del robot, el modo operativo deseado mediante el tópico `/flag` (por ejemplo, *true* para seguidor de carril o *false* para navegación por trayectoria), y una bandera global de activación desde `/self_driving`. Además, durante el modo de seguimiento de carril se implementa un comportamiento reactivo, el cual monitorea una señal de pausa por parte del LiDAR (`/pause`) cuando un obstáculo se encuentre a $1.5m$ de este, que permite detener temporalmente la marcha sin apagar el sistema.

A nivel de lógica, el nodo evalúa constantemente el modo de operación solicitado y selecciona qué controlador debe ejecutar comandos de velocidad. Si se encuentra en modo trayectoria, escucha el tópico `/waypoints` y utiliza el controlador Pure Pursuit para seguir los puntos definidos. Si el modo corresponde al seguidor de carril, entonces toma como referencia la desviación visual detectada en tiempo real por la cámara para mantenerse centrado entre las franjas amarillas.

Tanto el controlador de Pure Pursuit como el seguidor visual están programados para detectar cuando el robot ha alcanzado el último punto de la trayectoria o punto objetivo. En ese caso, se detiene la marcha por completo mediante la publicación de un mensaje `Twist` con velocidades nulas, y se espera a recibir un nuevo objetivo o un cambio a modo manual. Esta lógica considera el radio físico del robot $(0.8, m)$ como margen para definir la llegada al punto deseado, lo que asegura que se detenga apropiadamente sin necesidad de una

precisión absoluta milimétrica. De este modo, se garantiza que el robot no continúe ejecutando comandos innecesarios una vez finalizada su tarea.

La salida final del nodo consiste en mensajes tipo `Twist` publicados en el tópico `/cmd_vel`, los cuales son consumidos por el microcontrolador ESP32 para su conversión en eRPMs y transmisión al VESC. Esta arquitectura modular permite desacoplar los controladores individuales de la lógica de decisión, facilitando futuras extensiones como la integración con un planificador global, un gestor de flotas o la adaptación dinámica basada en percepción.

IV. Percepción y localización

A. Dead reckoning (Odometría de rueda)

Este modelo fue implementado dentro de un nodo en ROS 2, el cual suscribe las velocidades angulares de cada rueda y las transforma en velocidades lineales y angulares del chasis, de acuerdo con la cinemática diferencial. Después de actualizar la pose (x, y, θ) del robot, se publica un mensaje tipo `Odometry` en el tópico `/odom`. El nodo considera un radio de rueda de $0.08m$ y una separación entre ruedas de $0.60m$, valores que corresponden al diseño físico del robot. Además, se publica la posición angular acumulada de las ruedas como `JointState`, lo que permite una visualización coherente en RViz.

La pose (x, y, θ) del robot se actualiza integrando las velocidades como se observa en las siguientes ecuaciones:

$$x = x_{-1} + v \cdot cos(\theta) \cdot dt \tag{20}$$

$$y = y_{-1} + v \cdot sin(\theta) \cdot dt \tag{21}$$

$$\theta = (\theta_{-1} + \omega \cdot dt)\%(2 \cdot \pi) \tag{22}$$

Estas ecuaciones corresponden al modelo de odometría diferencial, y suponen desplazamientos cortos por paso de integración. El uso del operador módulo en la Ecuación (16) asegura que la orientación se mantenga dentro del rango $[0, 2\pi]$, lo cual es útil para evitar discontinuidades angulares en representaciones gráficas o algoritmos de control. Este modelo también es esencial para algoritmos de control de trayectoria y navegación autónoma.

B. Limitaciones de la odometría y estimación probabilística del estado

Aunque el modelo de odometría diferencial permite estimar la pose del robot a partir de las velocidades angulares de las ruedas, esta técnica presenta limitaciones significativas en escenarios reales. Factores como el deslizamiento de las ruedas, imperfecciones mecánicas, errores en los encoders o irregularidades del terreno generan acumulación de error conforme el robot se desplaza. Esto se traduce en desviaciones crecientes entre la posición estimada y la real, especialmente en trayectorias largas o cuando el robot ejecuta múltiples giros [14].

Para mitigar este problema, se recurre a la formulación probabilística de la localización, donde el estado del robot x_t (posición y orientación) se representa como una distribución de probabilidad en lugar de un valor puntual. En este marco, cada nueva acción de control u_t (como avanzar o girar) y cada observación sensorial z_t (como una lectura de LiDAR) se integran mediante el filtro Bayesiano:

- **Predicción:**

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) \cdot bel(x_{t-1})\, dx_{t-1}$$

  Se estima el nuevo estado en función del movimiento ejecutado y del estado anterior.
- **Corrección:**

$$bel(x_t) = \eta \cdot p(z_t \mid x_t) \cdot \overline{bel}(x_t)$$

  La creencia se ajusta según la coherencia entre la observación actual y el estado estimado.

Esta formulación permite combinar múltiples fuentes de información, incluso si son ruidosas o parcialmente confiables, y generar una estimación robusta del estado del robot. En robótica móvil, esta aproximación es la base de algoritmos ampliamente utilizados como el Filtro de Kalman Extendido (EKF) y el Filtro de Partículas implementado en AMCL [15] [16].

En las siguientes subsecciones se describe cómo se aplicaron estos enfoques dentro del sistema Robocov, utilizando principalmente odometría de rueda y sensores de entorno (LiDAR), con capacidad para integrar sensores inerciales o visuales en el futuro. Gracias a esta arquitectura probabilística, el robot logra mantener una localización precisa incluso ante errores sistemáticos o incertidumbre sensorial.

C. Extended Kalman Filter (EKF)

La estimación precisa del estado de un robot móvil es un aspecto fundamental en tareas de navegación autónoma, ya que permite integrar datos de distintos sensores para obtener una representación coherente y robusta del movimiento real del sistema [15]. Para este propósito se utilizó un filtro de Kalman extendido (EKF), una técnica probabilística basada en modelos dinámicos no lineales, ampliamente empleada en robótica móvil para la fusión sensorial [14].

En esta implementación se utilizó el paquete `robot_localization` de ROS 2 [13], que permite ejecutar un EKF en tiempo real para estimar la pose y la velocidad del robot. El filtro fue configurado para operar en modo tridimensional con una frecuencia de actualización de 40 Hz, utilizando como fuente principal de información la odometría generada por el modelo cinemático diferencial.

Los datos considerados incluyeron las componentes x y y de la posición, la orientación en *yaw* (θ), la velocidad lineal (v) y la velocidad angular (ω). Aunque en la versión final no se utilizó una unidad de medición inercial (IMU), esta sí fue implementada en fases previas del desarrollo. Su inclusión permitió en su momento mejorar significativamente la estabilidad de la estimación, especialmente al corregir la acumulación de errores angulares que provocaban desplazamientos ficticios en el eje y.

No obstante, debido a inestabilidades en el hardware y pérdida de datos confiables en la IMU, se decidió continuar el desarrollo únicamente con la odometría de rueda. Aun así, el sistema fue diseñado de forma modular, y está listo para reincorporar datos inerciales u otras fuentes como cámaras o GNSS, en caso de requerirse mayor precisión o robustez frente a condiciones adversas.

D. Transformaciones para la estimación de pose

En contraste con las transformaciones internas del robot, que definen la estructura física y cinemática de Robocov, esta sección aborda las transformaciones requeridas para la localización dentro del entorno. En este caso, se hace uso de una jerarquía de marcos de referencia que incluye `map`, `odom` y `base_link`, los cuales deben estar correctamente definidos y actualizados para lograr una estimación precisa y coherente de la pose del robot.

La transformación `odom` $\rightarrow$ `base_link` es calculada por el filtro EKF implementado mediante el paquete `robot_localization`. Esta transformación se basa principalmente en la odometría del robot y proporciona una estimación local de su desplazamiento relativo. Posteriormente, esta estimación es utilizada por el algoritmo AMCL (*Adaptive Monte Carlo Localization*), el cual incorpora información del mapa y de los sensores (como el LiDAR) para corregir errores acumulados. AMCL es responsable de publicar la transformación `map` $\rightarrow$ `odom`, cerrando así la cadena de transformaciones necesarias para referenciar al robot globalmente.

Esto permite que Robocov tenga una localización robusta y adaptable, separando claramente la estimación local (por odometría y EKF) de la corrección global (por AMCL).

E. Adaptive Monte Carlo Localization (AMCL)

Para permitir que el robot se ubicara con precisión dentro de un entorno previamente mapeado, se empleó el algoritmo *Adaptive Monte Carlo Localization* (AMCL), una técnica basada en filtros de partículas que estima la pose del robot dentro de un mapa estático. AMCL mantiene una nube de partículas, cada una representando una posible ubicación y orientación del robot. Estas partículas se actualizan constantemente utilizando la información proveniente del sistema de odometría y del sensor LiDAR, ajustando su peso según la correspondencia entre las observaciones reales y el mapa conocido.

El algoritmo de AMCL se basa en dos fases principales:

- **Predicción (motion update):** Cada partícula $x_t^{[i]}$ se propaga según el modelo de movimiento del robot, típicamente derivado de la odometría:

$$x_t^{[i]} \sim p(x_t \mid u_t, x_{t-1}^{[i]})$$

- **Corrección (measurement update):** A cada partícula se le asigna un peso $w_t^{[i]}$ que representa la probabilidad de que esa hipótesis de pose sea correcta dado el escaneo actual z_t y el mapa:

$$w_t^{[i]} = p(z_t \mid x_t^{[i]})$$

Estas operaciones permiten ajustar dinámicamente la nube de partículas para representar con mayor probabilidad las poses más consistentes con los sensores. Este enfoque probabilístico ofrece robustez frente a incertidumbres, errores acumulativos y condiciones del entorno ambiguas.

En el caso de Robocov, AMCL se configuró para operar con un rango de entre 2000 y 4000 partículas, lo que permitió alcanzar un equilibrio adecuado entre precisión de localización y carga computacional, considerando las capacidades de procesamiento de la Jetson Orin Nano. Se seleccionó el modelo de movimiento `diff`, acorde a la cinemática diferencial del robot, y se ajustaron los parámetros de ruido (`alpha1` a `alpha5`) para representar adecuadamente la incertidumbre asociada tanto a movimientos lineales como rotacionales. Asimismo, se establecieron umbrales bajos para las actualizaciones de posición y orientación, lo que permitió que el algoritmo reaccionara rápidamente a pequeños desplazamientos del robot.

El nodo de AMCL proviene del paquete de navegación `nav2` de ROS 2, el cual implementa una versión optimizada del filtro de partículas para sistemas móviles. Este nodo toma como entrada la odometría fusionada generada por el filtro de Kalman extendido (EKF), así como los escaneos del LiDAR 2D publicados en el tópico `/scan`. A partir de esta información, mantiene una estimación continua de la pose del robot respecto al marco global `map`, y publica la transformación correspondiente entre `map` y `odom`. Además, la estimación de pose global generada por el filtro se publica directamente como un mensaje de tipo `PoseWithCovarianceStamped` en el tópico `/amcl_pose`, lo que permite a otros nodos del sistema acceder fácilmente a la ubicación del robot con su respectiva incertidumbre.

Gracias al mapa previamente generado con `slam_toolbox`, y a una calibración adecuada de los sensores, AMCL permitió que Robocov mantuviera una localización estable incluso en entornos con geometrías repetitivas, como pasillos logísticos. Durante las pruebas, se observó que el sistema lograba recuperar la ubicación del robot tras reubicaciones manuales o tras pequeñas pérdidas de comunicación entre la Jetson y el módem, demostrando la robustez del enfoque.

F. ArUco

Los marcadores ArUco son patrones binarios cuadrangulares diseñados para ser detectados de forma robusta mediante visión por computadora. Cada marcador codifica un identificador único y permite, tras la detección de sus esquinas, calcular su pose relativa con respecto a la cámara mediante técnicas de calibración intrínseca y algoritmos de estimación PnP (Perspective-n-Point) [**?**]. Estas características los hacen ideales para tareas de localización, mapeo y navegación asistida en robótica móvil [**?**].

Durante el desarrollo del sistema Robocov, se implementó un nodo funcional para la detección de ArUco en ROS 2, utilizando OpenCV y una cámara previamente calibrada. La calibración incluyó la obtención de la matriz intrínseca y los coeficientes de distorsión óptica, lo que permitió obtener estimaciones confiables de distancia y orientación en condiciones visuales controladas.

Inicialmente se evaluó integrar los ArUco como fuente adicional de localización. Sin embargo, se determinó que el diseño de navegación del robot y el campo de visión de la cámara no garantizaban una detección consistente de los marcadores durante el desplazamiento continuo. Por tanto, se descartó su uso en la fusión de pose con el filtro EKF o AMCL, priorizando la estabilidad de localización ya alcanzada con odometría filtrada y escaneos LiDAR.

Posteriormente, se consideró utilizar los ArUco como señal visual para conmutar entre controladores de navegación (seguidor de carril y pure pursuit + PID). No obstante, se optó por una solución más eficiente basada en el análisis directo de la posición global estimada por el sistema de localización, activando automáticamente el modo apropiado según la zona del mapa.

A pesar de no ser incluidos en la versión final de navegación, el sistema de detección y calibración de ArUco permanece disponible como recurso confiable para futuras extensiones, ya sea en escenarios de localización asistida o como interfaz visual para interacciones en entornos estructurados.

V. Mapeo y navegación

A. Simultaneous Localization and Mapping (SLAM)

El problema de `SLAM` (*Simultaneous Localization and Mapping*) es uno de los desafíos centrales en robótica móvil. Consiste en que un robot construya un mapa del entorno mientras estima su propia ubicación dentro de él, todo ello a partir de sensores ruidosos y en tiempo real. Esta capacidad es indispensable cuando el entorno es desconocido o cambia dinámicamente, y no se cuenta con mapas previos ni con una posición inicial confiable [19].

Las soluciones SLAM modernas emplean técnicas probabilísticas, entre las que destacan el filtro de Kalman extendido (EKF-SLAM), los filtros de partículas (como FastSLAM) y los enfoques basados en gráficas de poses, que optimizan la trayectoria global del robot mediante cierres de bucle [20].

En el caso de Robocov, el procesamiento se lleva a cabo en una Jetson Orin Nano, que ejecuta los nodos de percepción, odometría y mapeo. El robot integra datos de un sensor LiDAR 2D (para obtener medidas del entorno) y de los encoders de ruedas (conectados a una ESP32 que transmite por UART). Estos datos se fusionan para calcular la odometría del robot, publicada en el tópico `/odom`.

Para resolver SLAM, se utilizó el paquete `slam_toolbox` de ROS 2 [21], en su modo asíncrono. Esta herramienta emplea un sistema basado en gráficas de poses (*pose-graph SLAM*), capaz de construir y optimizar mapas 2D en tiempo real. Recibe como entradas los escaneos del LiDAR (`/scan`) y la odometría, generando un mapa ocupacional binario donde se representan los obstáculos y espacios navegables. Además, mantiene continuamente actualizada la transformación entre los marcos `map` y `odom`, lo cual es fundamental para lograr una localización coherente y confiable.

Al concluir la fase de exploración, se utilizó la herramienta `map_saver_cli` para guardar el mapa resultante, que consta de un archivo `.pgm` (imagen en escala de grises) y un archivo `.yaml` (metadatos como resolución y origen). Un ejemplo puede observarse en la Fig. 9.

Fig. 9. Mapa generado mediante `slam_toolbox`.

Este mapa fue posteriormente utilizado durante la fase de navegación autónoma mediante AMCL, operando en modo de localización pura. Esta estrategia de dividir las fases de mapeo y localización permite mejorar la eficiencia del sistema, especialmente en tareas repetitivas. Durante las pruebas, se observó que el sistema generaba mapas precisos y mantenía una localización estable del robot. Todo el proceso fue visualizado y validado en tiempo real mediante RViz.

B. Planeador global: A*

Para que un robot autónomo pueda desplazarse de manera eficiente entre dos puntos dentro de un entorno conocido, es necesario contar con un planificador global que genere rutas viables evitando obstáculos. Una de las técnicas más empleadas para este propósito es el algoritmo `A*`, que encuentra

caminos óptimos sobre una grilla, considerando tanto la distancia recorrida como una estimación heurística de la distancia restante hasta el destino. Este enfoque balancea eficiencia computacional con calidad de solución, siendo ampliamente utilizado en robótica móvil y videojuegos [22].

El mapa utilizado por el planeador fue previamente generado mediante SLAM, pero requiere un preprocesamiento para ser funcional en entornos logísticos. El sensor LiDAR del robot solo percibe elementos en su plano de escaneo, por lo que estructuras elevadas como anaqueles pueden pasar desapercibidas. Para compensar esto, se aplicó una operación morfológica de dilatación sobre el mapa binario, reduciendo el área navegable y creando un margen de seguridad alrededor de los obstáculos. El resultado se muestra en la Fig. 10.

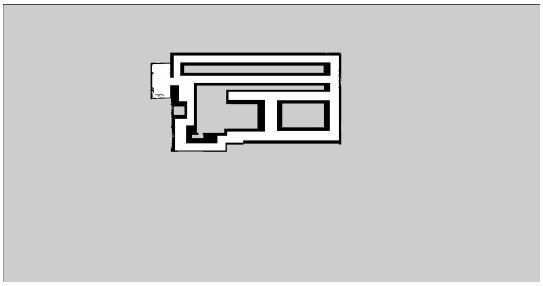


Fig. 10. Mapa delimitado y dilatado.

El nodo encargado de la planificación fue desarrollado en ROS 2 y trabaja directamente con el mapa binarizado. Comienza cargando un archivo `.yaml` con los parámetros del mapa (resolución, origen y bandera `negate`) y su imagen asociada en formato `.pgm`. Esta imagen se convierte a una matriz binaria, donde el valor 1 indica celdas libres y el 0 celdas ocupadas. Para alinear correctamente el sistema de coordenadas del mundo con la imagen, se invierte verticalmente esta matriz.

El nodo se suscribe a la pose del robot estimada por AMCL (`/amcl_pose`) y a la meta enviada desde RViz (`/clicked_point`). Además, escucha dos señales de control: `/waiting_path` (espera de nueva ruta) y `/self_driving` (modo autónomo activo). La trayectoria generada se publica en `/waypoints` como una secuencia de poses.

Las conversiones entre coordenadas del mundo y de imagen se realizan mediante las siguientes expresiones:

$$p_x = \left\lfloor \frac{x - x_0}{r} \right\rfloor, \quad p_y = \left\lfloor \frac{y - y_0}{r} \right\rfloor \tag{23}$$

$$x = p_x \cdot r + x_0, \quad y = p_y \cdot r + y_0 \tag{24}$$

donde (x_0, y_0) es el origen del mapa y r su resolución.

Si el punto de inicio o la meta se encuentran sobre celdas ocupadas, se ejecuta una búsqueda en amplitud (BFS) para encontrar el punto libre más cercano en una vecindad de 8 direcciones. Esto previene errores de planificación derivados de pequeñas imprecisiones en la localización.

El algoritmo A* se ejecuta sobre la grilla binaria, utilizando un costo uniforme por celda. Si no se encuentra una ruta válida, se cancela la planificación. En caso contrario, se transforma la trayectoria a coordenadas del mundo y se suaviza con interpolación tipo *spline*. El número de puntos interpolados se ajusta según la longitud total del recorrido.

Finalmente, el camino suavizado se publica en `/waypoints`, desde donde lo recupera el controlador para ejecutar el movimiento del robot. Esta integración permite una navegación autónoma segura, adaptable a la geometría del entorno.

C. Detección controlada

Una vez definida una ruta hacia el objetivo final mediante el algoritmo de planificación A*, el sistema navega a través de distintos escenarios, alternando entre controladores en función de la posición actual del robot en coordenadas (x, y). En áreas abiertas, se utiliza un controlador combinado *Pure Pursuit + PID*, mientras que al ingresar a pasillos delimitados por líneas amarillas, identificados mediante la ubicación del robot dentro del mapa, se activa un controlador visual reactivo basado en visión por computadora.

En estos pasillos angostos, la estructura física del entorno impide realizar maniobras laterales de evasión, por lo que la estrategia óptima consiste en detener el robot y esperar a que el obstáculo, como una persona u objeto temporal, libere el camino. Esta lógica de pausa se activa exclusivamente cuando el controlador visual reactivo (*Lane Follower*) está en uso.

Para habilitar esta funcionalidad, se desarrolló un nodo encargado de analizar continuamente los datos del sensor LiDAR, enfocándose en un rango frontal limitado de $170°$ a $190°$. Este rango corresponde al frente real del robot, ya que, como se mencionó, el sensor LiDAR fue instalado con una rotación de π radianes respecto a su orientación original. Esta corrección permite al sistema interpretar correctamente las lecturas del entorno.

Cuando se detecta un obstáculo a una distancia menor o igual a $1.5m$ dentro de dicho rango, el nodo publica el valor booleano *True* en el tópico `/pause`. Este mensaje es recibido por el nodo supervisor del controlador visual, que responde deteniendo temporalmente el avance del robot hasta que la trayectoria esté libre nuevamente.

La integración de este comportamiento reactivo básico permite al robot adaptarse dinámicamente a situaciones inesperadas en zonas críticas, mejorando la seguridad y confiabilidad durante la navegación. Aunque actualmente esta lógica está limitada a los pasillos estrechos y al seguimiento de carril, representa una base sólida para futuros desarrollos. En etapas posteriores, este esquema podría extenderse para aplicarse a lo largo de todo el recorrido, combinando la planeación global

con una respuesta local más sofisticada frente a obstáculos dinámicos.

VI. Validación experimental

A. Trayectorias ejecutadas dentro del almacén

Las pruebas del sistema de navegación autónoma se llevaron a cabo directamente en el almacén industrial, en el cual se realizaron múltiples recorridos bajo condiciones representativas de operación logística. El espacio incluía pasillos angostos delimitados por racks metálicos, zonas abiertas destinadas a carga y descarga, curvas pronunciadas y obstáculos tanto estáticos como móviles.

Se ejecutaron diversas trayectorias, incluyendo recorridos en forma de "U", curvas en "L", rutas con múltiples cambios de dirección y trayectorias cerradas que requerían ida y retorno al punto de inicio. Varias de estas rutas pasaban por los dos pasillos estrechos indicados en el mapa de la Fig. 10, e involucraban giros cerrados y presencia ocasional de personas en movimiento dentro del área de pruebas.

El robot operó utilizando el sistema de navegación autónoma previamente descrito, compuesto por un planificador global basado en A* y una arquitectura de control. Esta arquitectura alterna automáticamente entre el *lane follower* visual en pasillos estrechos y el controlador de trayectoria basado en Pure Pursuit más un PID de error lateral en zonas abiertas. La transición entre ambos controladores se realizó de manera autónoma según la ubicación del robot, sin intervención externa.

Durante las pruebas, el robot mostró un comportamiento estable y robusto, manteniendo trayectorias suaves y sin colisiones incluso en situaciones de navegación compleja. La operación en el entorno real validó la eficacia del sistema de percepción, planeación y control ante las variaciones propias del ambiente físico.

En todas las trayectorias ejecutadas, el robot fue capaz de detenerse dentro del radio de tolerancia de $0.8m$ definido alrededor del objetivo, cumpliendo con un 100% de éxito en la llegada a destino. Esto evidencia que el sistema de localización, basado en AMCL, actualiza y corrige correctamente la posición estimada del robot a lo largo del trayecto, permitiendo una navegación confiable hasta los puntos meta.

En ciertos casos, se observó una ligera oscilación en la orientación del robot o un paso cercano a algunos obstáculos estáticos. Sin embargo, gracias al preprocesamiento del mapa con dilatación morfológica, que amplía artificialmente el contorno de los obstáculos, el robot siempre mantuvo una distancia segura y evitó colisiones. Este resultado confirma la efectividad de las medidas preventivas incorporadas en la etapa de planeación global y local.

Para una referencia visual más clara del desempeño del sistema, se invita a dar clic aquí para observar un video demostrativo del funcionamiento del robot. Dicho video presenta una sincronización precisa entre tres perspectivas distintas: una cámara a bordo del robot, una vista externa desde atrás que permite observar el comportamiento global de navegación, y una visualización en RViz donde se aprecia la trayectoria ejecutada, las transformaciones activas y el estado del sistema de localización. En esta grabación se ejecuta una vuelta completa a lo largo de una trayectoria cerrada, lo que permite apreciar la transición fluida, la precisión en el seguimiento de ruta y la robustez general del sistema autónomo en condiciones reales.

B. Comparación entre poses por `/odom` y `amcl_pose`

La odometría basada únicamente en los encoders de rueda presentó un error de deriva considerable, especialmente en el eje lateral y, donde se observaron desviaciones acumulativas incluso en trayectorias lineales. En cambio, el eje longitudinal x y la orientación θ se mantuvieron dentro de márgenes aceptables durante trayectos cortos, aunque en desplazamientos más extensos el error aumentaba significativamente. Este comportamiento se atribuye a factores como la alta inercia del robot, derivada de su peso, y el comportamiento de las ruedas locas, que generan desviaciones no deseadas incluso al comandar únicamente velocidad lineal hacia adelante. Por ello, la odometría por sí sola resulta insuficiente para una localización confiable en el entorno.

Respecto al algoritmo AMCL, inicialmente se empleó una configuración con un número de partículas entre 1000 y 3000, lo cual provocaba una respuesta algo lenta del sistema y una corrección tardía de la pose. En esta etapa era evidente un salto en la representación del robot en el mapa cada vez que AMCL corregía la posición, indicando que las actualizaciones no eran suficientemente frecuentes ni precisas. Posteriormente, se ajustaron los parámetros del algoritmo, aumentando el número de partículas a un rango entre 2000 y 4000 y elevando la frecuencia de actualización. Con esta configuración, el sistema logró una localización continua y estable, manteniendo la pose del robot correctamente alineada al mapa durante todo el trayecto. Además, se observó que la elipse de confianza en RViz, tanto en posición como en orientación, permaneció relativamente pequeña y constante, lo que evidencia la robustez del sistema de localización probabilística en condiciones reales.

C. Evaluación de precisión en seguimiento de carril

Para validar el rendimiento del controlador visual reactivo, se analizaron los desplazamientos del robot dentro de pasillos delimitados por líneas amarillas en el piso. El sistema utilizó una cámara frontal orientada al suelo que capturaba imágenes a 30 FPS, procesadas en tiempo real mediante técnicas de binarización adaptativa y cálculo del centroide de las líneas detectadas.

La evaluación se realizó exclusivamente en tramos rectos, ya que los pasillos no incluían curvas. En condiciones ade-

cuadas de iluminación, el robot se mantuvo prácticamente alineado con el eje central del carril, sin mostrar oscilaciones apreciables. El controlador proporcional angular, basado en el error entre el centroide y el centro de la imagen, permitió una trayectoria estable y precisa.

Sin embargo, el desempeño del sistema se veía afectado por la iluminación ambiental. En horarios donde la luz solar incidía directamente en el piso, el sistema podía perder una o ambas líneas amarillas, afectando la estimación del centroide. Aunque los pasillos contaban con iluminación artificial, esta no siempre era suficiente para compensar las variaciones provocadas por la luz natural. La máscara HSV utilizada para segmentar el color amarillo fue afinada para mejorar la detección bajo condiciones normales, pero aún presentaba limitaciones frente a estos cambios.

Afortunadamente, el robot contaba con un mecanismo de seguridad ya descrito anteriormente, el cual se activaba mediante el tópico `/pause`. Cuando el sistema no detectaba correctamente las dos líneas amarillas y se desviaba de la trayectoria, cambiaba este tópico a *True* al identificar un obstáculo a menos de 1.5 metros. Esto detenía al robot de forma segura, evitando colisiones con los anaqueles y asegurando la integridad tanto del entorno como del propio vehículo.

D. Curvas de velocidad, latencia en control y cambio entre modos

Durante los experimentos, se registraron tanto las referencias de velocidad lineal y angular como las velocidades reales alcanzadas por el robot, enviadas a través del bus UART desde el nodo de control al microcontrolador ESP32 encargado de la ejecución motriz. Las curvas observadas reflejaron un seguimiento consistente y preciso, con buena correspondencia entre los comandos y velocidades reales, lo cual indica un desempeño robusto de la arquitectura de control.

En cuanto a la latencia del sistema, la respuesta fue perceptiblemente rápida y constante, incluso durante los cambios automáticos entre controladores. Esta conmutación fue gestionada por el nodo supervisor, que evaluaba continuamente las coordenadas (x, y) del robot mediante una suscricpión a `amcl_pose`. Al ingresar a zonas predefinidas del mapa, como pasillos, se desactivaba el controlador global y se activaba el seguidor visual de carril. Esta transición se realizaba de manera fluida, sin interrupciones ni comportamientos erráticos, y sin necesidad de utilizar señales visuales externas como marcadores ArUco, que en pruebas preliminares demostraron poca fiabilidad.

Cabe destacar que, en algunos casos, se observaron limitaciones en el rendimiento del sistema cuando el robot se alejaba del punto de acceso WiFi, el cual consistía en un router configurado como extensor de señal para compartir la red de datos móviles del operador. Esta distancia afectaba la calidad del enlace inalámbrico entre la Jetson Orin Nano y otros dispositivos de supervisión remota, como la computadora del operador, ocasionando desconexiones intermitentes por SSH y dificultando la visualización en tiempo real mediante RViz. Aunque los sensores y actuadores principales del robot se comunican por interfaces físicas (como UART), estas caídas de red impactaban negativamente la interacción y el monitoreo del sistema durante la operación.

Adicionalmente, para el cambio entre los modos manual y autónomo mediante los cuatro botones del joystick, fue necesario implementar una rutina de debouncing. Inicialmente, al presionar los botones simultáneamente, el cambio de modo se ejecutaba múltiples veces debido a la alta frecuencia de muestreo del nodo joy. Se solucionó añadiendo un retardo de $300ms$ tras cada detección válida, lo que estabilizó completamente la transición entre modos y permitió un control confiable del sistema operativo.

E. Lecciones aprendidas durante las pruebas

Durante el proceso de validación experimental, se identificaron varias limitaciones del sistema que permitieron mejorar tanto el diseño como la operación del robot.

En primer lugar, se observó que al ejecutar el archivo de lanzamiento principal que incluye todos los nodos del sistema, la carga computacional alcanzaba los límites de capacidad de la Jetson Orin Nano, especialmente al procesar visión por computadora y simultáneamente mantener múltiples nodos activos. Esto no solo afectaba el rendimiento en tiempo real, sino también la estabilidad de algunos procesos.

Una de las principales lecciones fue la discrepancia entre el modelo cinemático ideal y el comportamiento real del robot. Esto se debía a factores como el peso total del sistema (estructura, batería, sensores) y a la fricción dinámica del suelo, que afectaban la respuesta de los actuadores. Se abordó este problema ajustando los parámetros del modelo y recalibrando las ganancias del controlador PID, logrando así una representación más fiel del comportamiento del robot.

En cuanto a la comunicación, se detectó que el microcontrolador con Micro-ROS, que es responsable de recibir los comandos de velocidad y publicar la odometría, en ocasiones se desactivaba espontáneamente. Este evento, aunque poco frecuente, podía ocurrir tanto al iniciar el sistema como tras un tiempo prolongado de operación, provocando una pérdida de control sobre el robot. Aunque no se logró identificar la causa exacta, se especula que podría deberse a desbordamientos de buffer, fallos de sincronización o interrupciones en la comunicación serie. Para mitigar o prevenir cualquier accidente, se tienen los sistemas de paro de emergencia previamente explicados, los cuales interrumpen el flujo energético al controlador de motores.

También se comprobó que el botón de reset en la ESP32 es indispensable, ya que al iniciar el sistema, Micro-ROS no comienza correctamente hasta que se realiza este reinicio manual. Esta acción se automatizó parcialmente mediante

la instalación de un botón físico de reinicio en la caja de control, y adicionalmente se integró la opción de reinicio por radiofrecuencia, permitiendo reestablecer Micro-ROS sin necesidad de acceso físico.

Por otro lado, se identificaron limitaciones en la conectividad Wi-Fi durante las pruebas en campo. En particular, se detectó una pérdida progresiva de estabilidad conforme el robot se alejaba del punto de acceso. Si bien esto no comprometía la operación interna del robot, sí dificultaba las tareas de supervisión remota, como el acceso por SSH y la visualización en RViz desde una estación externa. Para evitar estos inconvenientes, se recomienda trabajar bajo una red inalámbrica dedicada y de buena cobertura, especialmente en entornos amplios o con obstáculos físicos.

Respecto a la detección visual, se desarrolló un sistema funcional basado en marcadores ArUco con el objetivo de identificar zonas específicas o alternar entre modos de control. No obstante, durante las pruebas se observó que pequeñas oscilaciones en la trayectoria del robot podían hacer que la cámara no captara el marcador en el momento esperado, comprometiendo así la conmutación confiable entre controladores. Por esta razón, se optó por una solución más robusta, basada en la evaluación directa de la pose estimada del robot dentro del mapa. Aun así, el sistema de detección ArUco permanece completamente operativo, con cámara calibrada y estimación de pose funcional, lo cual lo convierte en un recurso valioso para futuras aplicaciones en condiciones más controladas de iluminación y movimiento.

Otro hallazgo fue la fragilidad estructural de la caja superior del robot. Mediante simulaciones en SolidWorks, se estimó que la caja puede soportar hasta $100N$ (aproximadamente $10kg$) antes de comenzar a deformarse significativamente. Se recomienda evitar colocar componentes adicionales sobre ella sin un refuerzo estructural.

Finalmente, se comprobó que tener las transformaciones correctamente configuradas entre los distintos marcos de referencia es esencial para el correcto funcionamiento de los algoritmos de navegación. Desde el mapeo con `slam_toolbox`, pasando por localización con AMCL y filtros como el EKF, todos dependen de la correcta propagación de transformadas para estimar y actualizar con precisión la pose del robot dentro del entorno.

VII. Discusión

A. Fortalezas del sistema

El sistema desarrollado presenta diversas fortalezas que lo hacen atractivo para aplicaciones logísticas, especialmente en tareas de entrega y recolección de paquetes dentro de un almacén, así como en entornos que lo requieran:

- **Modularidad:** Cada componente del sistema, desde el control de bajo nivel con Micro-ROS hasta la navegación autónoma con ROS 2, está diseñado de forma independiente pero perfectamente integrable. Esta arquitectura modular permite modificar, sustituir o mejorar subsistemas específicos sin comprometer la funcionalidad general del robot.
- **Bajo costo:** Se priorizó el uso de componentes accesibles, como la ESP32, motores comerciales, sensores de bajo costo y piezas impresas en 3D, lo cual reduce significativamente el costo total del sistema sin sacrificar sus capacidades esenciales.
- **Escalabilidad:** Gracias al uso de ROS 2 y su capacidad para manejar múltiples nodos distribuidos, el sistema puede escalar fácilmente hacia configuraciones más complejas, como multirrobot, integración con brazos manipuladores o incorporación de sensores avanzados como cámaras estéreo o LiDAR 3D. Asimismo, su diseño abierto permite adaptarse a plataformas móviles diferenciales de diferentes tamaños.
- **Fiabilidad:** A pesar de las limitaciones de hardware, el sistema ha demostrado un funcionamiento robusto durante pruebas prolongadas. El diseño eléctrico protegido, las rutinas de reconexión y las herramientas de diagnóstico contribuyen a una operación confiable en escenarios reales, donde condiciones como variaciones de red o pequeños fallos de hardware pueden presentarse.

Estas fortalezas constituyen una excelente primera aproximación para la futura implementación de Robots Móviles Autónomos (AMRs) en ambientes industriales como el almacén de Glaxo, permitiendo evaluar retos reales de navegación, comunicación y operación en un entorno controlado antes de escalar a soluciones de mayor nivel.

B. Desempeño alcanzado vs. expectativas de precisión

En términos generales, el sistema alcanzó el desempeño previsto durante las pruebas experimentales. La navegación autónoma, la detección de carriles y el switching entre modos funcionaron de manera consistente y confiable, permitiendo la operación segura del robot dentro del entorno del almacén.

Una de las áreas más satisfactorias fue el control manual mediante el gamepad, donde el usuario percibe un control robusto e intuitivo, similar al manejo de un carro a control remoto. La respuesta del sistema es rápida y estable, incluso a velocidades relativamente altas, lo cual facilitó tareas de prueba, calibración y operación en zonas estrechas o ante imprevistos.

En cuanto a la navegación autónoma, se logró cumplir el objetivo clave de que el robot pudiera desplazarse hacia cualquier punto del mapa generado, sin colisiones ni bloqueos. A pesar de que en algunos tramos se observan ligeras oscilaciones, estas no comprometen la seguridad ni la trayectoria general del robot, que sigue correctamente el camino generado por el planeador de rutas.

Por otro lado, una de las áreas donde las expectativas no se cumplieron completamente fue en el comportamiento del robot

bajo el controlador combinado PurePursuit + PID. Aunque se realizaron múltiples ajustes y sintonización de parámetros (k_p, k_d, k_i), no se logró una trayectoria suficientemente suave, observándose oscilaciones notables, especialmente en curvas o al acercarse a los objetivos. Este fenómeno se atribuye tanto a la dinámica real del sistema como a limitaciones del propio esquema de control en interacción con el entorno.

Finalmente, un aspecto que superó las expectativas fue el consumo energético. A pesar de contar con múltiples sensores activos, varios nodos de ROS funcionando en paralelo y los motores en movimiento constante, la batería del robot ofreció una autonomía mayor a la esperada. Esto permitió sesiones de prueba prolongadas sin interrupciones, lo cual contribuyó significativamente a la eficiencia del proceso de validación.

Por lo tanto, se puede reafirmar que el desempeño global fue adecuado para validar el enfoque, cumpliendo con los objetivos funcionales del proyecto y sirviendo como base sólida para futuras mejoras.

C. Comparación con Turtlebot y AMRs comerciales

El sistema desarrollado se diseñó como una alternativa accesible y personalizable frente a plataformas educativas como Turtlebot y sistemas comerciales de alto rendimiento empleados por empresas como Amazon Robotics o FedEx.

A diferencia del Turtlebot, que está orientado principalmente a aplicaciones educativas y de investigación en entornos controlados, el presente sistema incorpora mejoras orientadas al ambiente logístico, como mayor robustez estructural, y un diseño modular que permite la fácil integración de sensores, actuadores o algoritmos adicionales. Además, el bajo costo relativo del sistema desarrollado lo hace más viable para proyectos de escalamiento y pruebas piloto en almacenes reales.

En comparación con AMRs comerciales de uso industrial, que cuentan con sensores de alta gama, hardware de procesamiento dedicado y software propietario altamente optimizado, este sistema ofrece una primera aproximación práctica con tecnologías *open source* como ROS 2 y Micro-ROS. Aunque el rendimiento en términos de velocidad, precisión y confiabilidad puede ser menor, el enfoque propuesto permite validar funciones críticas como la navegación autónoma, planeación de rutas, control manual y adaptación al entorno, usando recursos más modestos.

Si bien no compite directamente con soluciones logísticas totalmente desplegadas en entornos como los centros de distribución de Amazon o FedEx, donde se requiere precisión milimétrica, alta capacidad de carga y tiempos de respuesta mínimos, el sistema representa una base sólida y adaptable que puede escalarse progresivamente hacia aplicaciones reales, incluyendo pruebas en almacenes medianos como los de Glaxo.

VIII. Conclusiones y trabajo futuro

Este proyecto logró el desarrollo e integración exitosa de un Robot Móvil Autónomo (AMR) funcional, modular y accesible, diseñado específicamente para operar en entornos logísticos reales, como el almacén de Glaxo. En un periodo intensivo de doce semanas, se construyó un sistema completo y operativo, capaz de realizar navegación autónoma tanto en espacios amplios como en pasillos estrechos, utilizando únicamente herramientas de software libre, hardware comercialmente accesible y técnicas avanzadas de percepción y control.

Uno de los principales aciertos fue la arquitectura modular del sistema, que permitió trabajar de forma desacoplada en subsistemas como el control de motores, la adquisición sensorial, la fusión de datos, la planeación de rutas y la navegación autónoma. Esta separación funcional no solo facilitó el desarrollo por etapas, sino que además sentó las bases para futuras ampliaciones y adaptaciones en otros entornos o plataformas. Todo el sistema fue diseñado bajo los principios de compatibilidad, escalabilidad y replicabilidad, demostrando que es posible construir soluciones industriales ligeras basadas en tecnologías abiertas como ROS 2 y micro-ROS.

A nivel técnico, se logró una localización robusta mediante la combinación de odometría diferencial, fusión con EKF y corrección global con AMCL, lo cual garantizó desplazamientos confiables dentro del mapa. La precisión alcanzada se mantuvo dentro de márgenes adecuados incluso tras recorridos largos, confirmando la estabilidad del sistema de localización. Además, se implementaron con éxito dos esquemas de navegación adaptativa: un controlador visual reactivo tipo PID para pasillos, y un algoritmo híbrido de PurePursuit + PID para trayectorias globales planificadas mediante A*. La transición entre ambos modos se gestionó de manera eficiente según la ubicación estimada del robot, sin requerir intervención humana.

La planeación de rutas se resolvió mediante un nodo propio basado en A*, con suavizado por interpolación y verificación de zonas seguras para evitar errores por imprecisión de sensores o mapas. El mapeo del entorno se realizó con `slam_toolbox` en modo asíncrono, generando mapas ocupacionales confiables que luego fueron utilizados en navegación pura mediante localización probabilística.

A nivel de control, el uso de una ESP32 corriendo micro-ROS permitió separar la capa de bajo nivel del procesamiento principal, asegurando tiempos de respuesta bajos para la ejecución de comandos de movimiento y adquisición de velocidades reales. Esta separación también favorece el mantenimiento y escalabilidad del sistema, ya que permite integrar nuevas funciones sin recargar el nodo central.

En las pruebas, el robot fue capaz de completar rutas de ida y vuelta sin colisiones, con desviaciones inferiores a su propio radio de operación, y con una respuesta estable incluso

a velocidades altas en modo manual. La interfaz mediante gamepad resultó intuitiva y eficaz para pruebas y validaciones rápidas. Además, se observó un consumo energético menor al esperado, lo que se tradujo en una autonomía mayor durante sesiones de prueba extensas.

De cara al futuro, el sistema abre la puerta a múltiples líneas de evolución:

- Integración de un gestor de flota (*fleet manager*) para coordinar múltiples robots en un mismo entorno, optimizando recursos y reduciendo tiempos de espera.
- Aplicación del sistema a tareas automatizadas como inventariado dinámico, recolección y entrega de paquetes, o asistencia en líneas de producción ligeras.
- Incorporación de sensores de percepción avanzada, como cámaras RGB-D o LiDAR 3D, combinados con redes neuronales para enriquecer la interpretación del entorno.
- Optimización de la planeación y la gestión energética mediante algoritmos adaptativos, incluyendo puntos de recarga autónoma o rutas de bajo consumo.

Este trabajo representa una validación práctica del potencial de los AMRs construidos con tecnologías abiertas, demostrando que es posible lograr plataformas robustas en tareas clave como localización, navegación, percepción y operación remota, incluso bajo restricciones de tiempo, presupuesto y recursos computacionales. Su implementación en un entorno real, con condiciones logísticas representativas, posiciona al sistema como una base sólida para futuras soluciones escalables en la automatización industrial.

REFERENCES

[1] IBM, "What is Industry 4.0?", 2024. [En línea]. Disponible en: https://www.ibm.com/mx-es/topics/industry-4-0. [Consultado: 7 de junio de 2025].

[2] SAP, "What is Industry 4.0?", 2024. [En línea]. Disponible en: https://www.sap.com/products/scm/industry-4-0/what-is-industry-4-0.html. [Consultado: 7 de junio de 2025].

[3] El País, "La logística 4.0 transforma la cadena de suministro", 6 de junio de 2025. [En línea]. Disponible en: https://elpais.com/economia/horitzo-4-0/2025-06-06/i-si-la-gran-apagada-fos-logistica.html. [Consultado: 7 de junio de 2025].

[4] W. Meeussen, "REP 105: Coordinate Frames for Mobile Platforms," ROS.org, 2010. [Online]. Available: https://www.ros.org/reps/rep-0105.html

[5] "tf2/Tutorials," ROS Wiki. [Online]. Available: https://wiki.ros.org/tf2/Tutorials

[6] Benjamin Vedder. "VESC Tool: Graphical Interface for VESC Motor Controllers." [Online]. Available: https://vesc-project.com/vesc_tool

[7] Z. Chen and D. Xu, "FOC of Permanent Magnet Synchronous Motor – A Tutorial," IEEE Industrial Electronics Society, 2016.

[8] K. Rajashekara, "Sensorless Control of BLDC Motors – Challenges and Solutions," IEEE Transactions on Industry Applications, vol. 41, no. 2, 2005.

[9] Clearpath Robotics, "Kinematics of a Differential Drive", 2023. [Online]. Available: https://docs.clearpathrobotics.com/differential_drive_kinematics.html

[10] Purdue SIGBots Wiki, "Basic Pure Pursuit," Purdue University, 2022. [Online]. Available: https://wiki.purduesigbots.com/software/control-algorithms/basic-pure-pursuit.

[11] OpenCV, "Contour Features," OpenCV Documentation, 2024. [Online]. Available: https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html.

[12] G. Bradski and A. Kaehler, *Learning OpenCV 4: Computer Vision with Python*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2020.

[13] T. Moore, "robot_localization package for ROS," http://wiki.ros.org/robot_localization, consultado en junio de 2025.

[14] S. Thrun, W. Burgard y D. Fox, *Probabilistic Robotics*, MIT Press, 2005. Disponible en: https://www.probabilistic-robotics.org/

[15] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*, Wiley, 2006. Disponible en: https://onlinelibrary.wiley.com/doi/book/10.1002/0470045345

[16] F. Dellaert, D. Fox, W. Burgard y S. Thrun, "Monte Carlo Localization for Mobile Robots," Proc. IEEE ICRA, 1999. Disponible en: https://www.cc.gatech.edu/~dellaert/pubs/Dellaert99icra.pdf

[17] J. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014. [Online]. Available: https://doi.org/10.1016/j.patcog.2014.01.005

[18] OpenCV.org, "Detection of ArUco Markers," OpenCV Documentation, 2024. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[19] Robotics and Perception Group, "What is SLAM? Simultaneous Localization and Mapping," *Robotics and Perception Lab, University of Zurich*, 2023. [Online]. Available: https://rpg.ifi.uzh.ch/research_slam.html

[20] ROS Industrial Consortium, "Understanding SLAM in ROS 2," *ROS-Industrial*, 2022. [Online]. Available: https://rosindustrial.org/news/2022/11/14/understanding-slam-in-ros-2

[21] S. Macenski, "Slam Toolbox — SLAM for the dynamic world," *navigation.ros.org*, 2023. [Online]. Available: https://navigation.ros.org/configuration/packages/slam_toolbox.html

[22] GeeksforGeeks, "A* Search Algorithm," 2023. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/

ANEXOS

A. Enlaces

- Video de funcionamiento en modo autónomo
- Repositorio completo de código y documentación

B. Imágenes

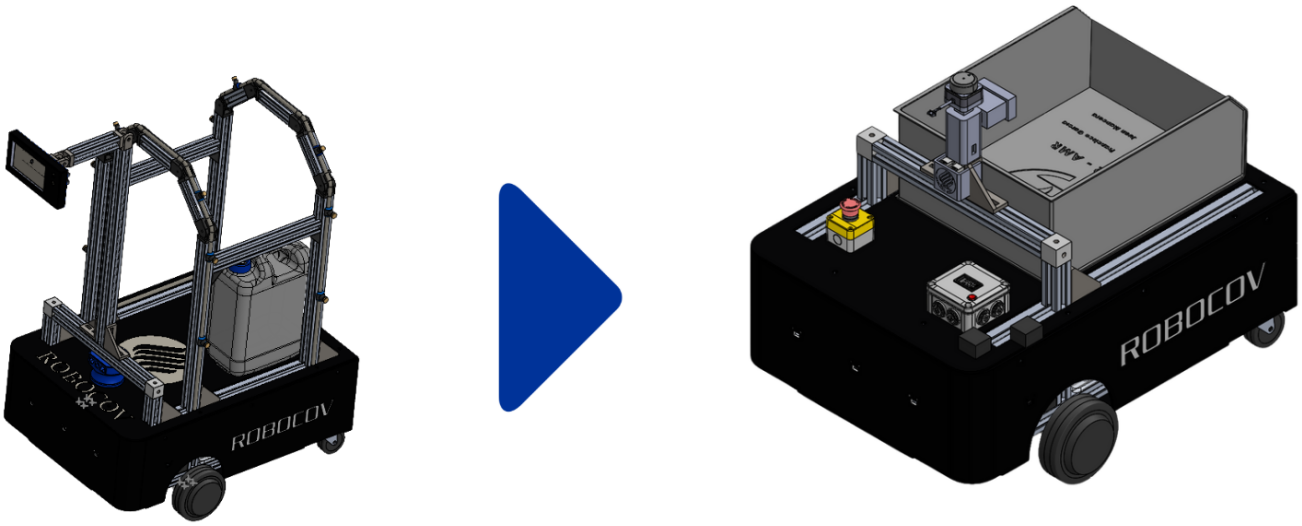


Fig. 11. Antes y después de Robocov visualizado mediante CAD.

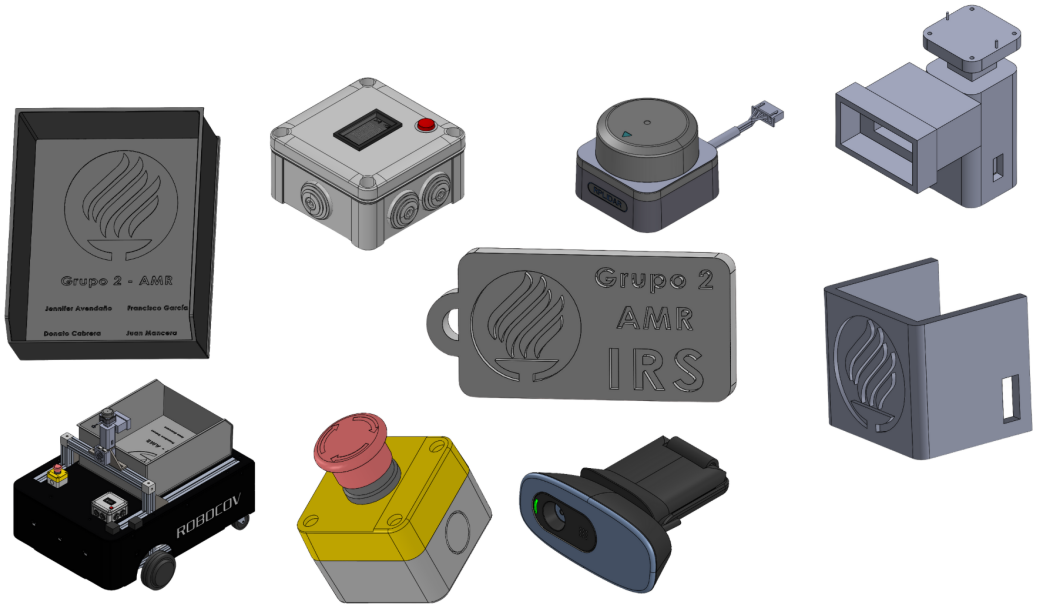
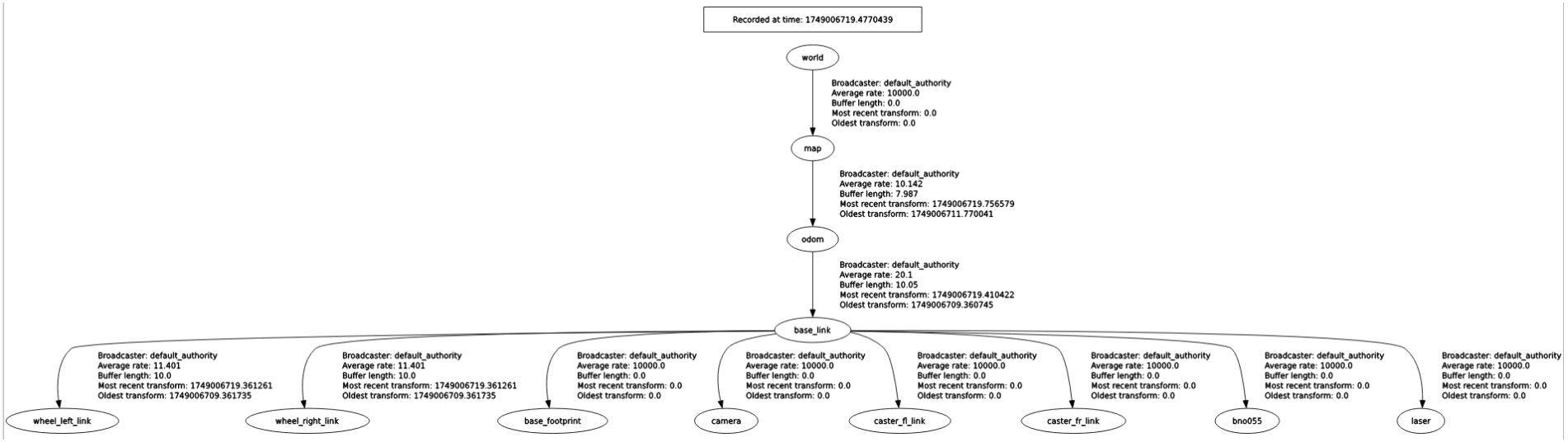
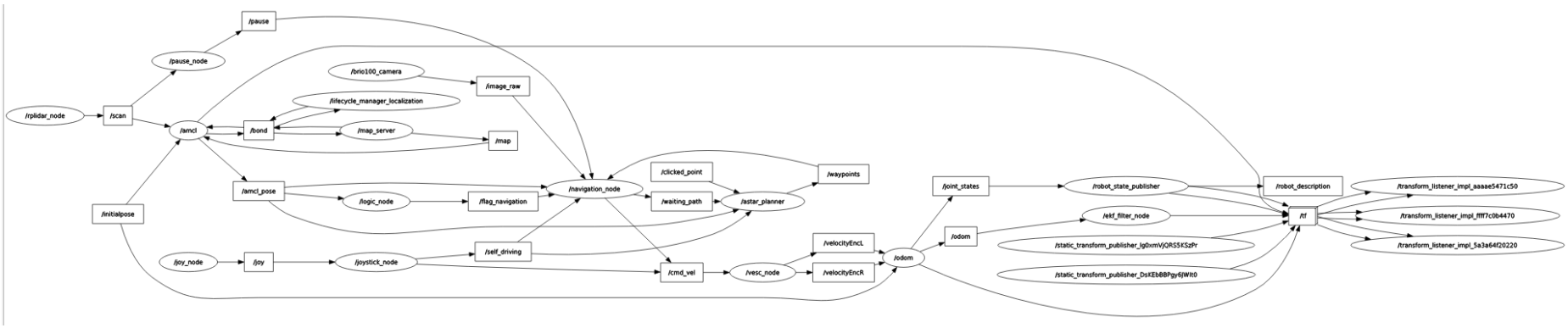


Fig. 12. Piezas y sensores utilizados en Robocov diseñadas mediante CAD.

Árbol de transformadas del sistema



Grafo de nodos y tópicos de ROS 2



Esquemático simplificado del sistema

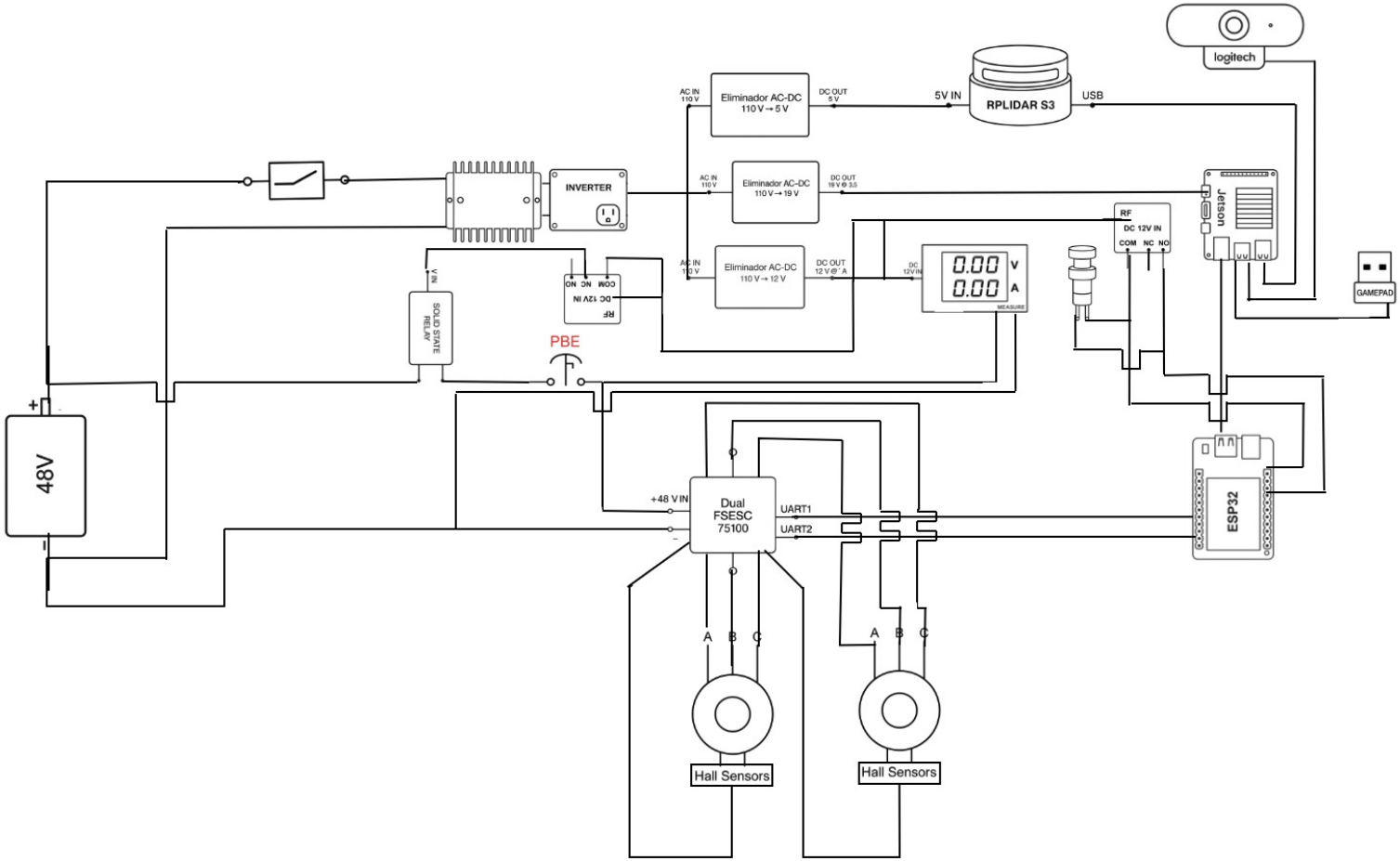


Diagrama de bloques del sistema Robocov