UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA MECATRÓNICA

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"LOCALIZACIÓN Y MAPEO SIMULTÁNEO (SLAM) PARA ENTORNOS DINÁMICOS"

Trabajo de titulación previo a la obtención de título de Ingeniero en Mecatrónica

Línea de investigación: Producción industrial y tecnología sostenible

Autor:

Franklin Xavier Arteaga Reina

Director:

PhD. Carlos Xavier Rosero Chandi

Ibarra - Ecuador 2025

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1003105762		
APELLIDOS Y NOMBRES:	Arteaga Reina Franklin Xavier		
DIRECCIÓN:	Ibarra – Ciudadela Municipal		
EMAIL:	fxarteagar@utn.edu.ec		
TELÉFONO FIJO:	065015895	**TELÉFONO MÓVIL:**	0990594975

DATOS DE LA OBRA	
TÍTULO:	"Localización y mapeo simultáneo (SLAM) para entornos dinámicos"
AUTOR:	Arteaga Reina Franklin Xavier
FECHA: DD/MM/AAAA	01/08/2025
PROGRAMA:	■ GRADO ☐ POSGRADO
TITULO POR EL QUE OPTA:	Ingeniero en Mecatrónica
DIRECTOR:	PhD. Carlos Xavier Rosero Chandi

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, 1 de Agosto de 2025.

EL AUTOR:

.....

Nombre: Franklin Xavier Arteaga Reina

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICAS
CARRERA DE INGENIERÍA EN MECATRÓNICA

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 1 de Agosto de 2025.

PhD. Carlos Xavier Rosero Chandi

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del Trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f) .....

PhD. Carlos Xavier Rosero Chandi
C.C.: 1002515821

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICAS
CARRERA DE INGENIERÍA EN MECATRÓNICA

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificado del trabajo de Integración Curricular "LOCALIZACIÓN Y MAPEO SIMULTÁNEO (SLAM) PARA ENTORNOS DINÁMICOS" elaborado por FRANKLIN XAVIER ARTEAGA REINA, previo a la obtención del título de INGENIERO EN MECATRÓNICA, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

(f) .....

PhD. Carlos Xavier Rosero Chandi
C.C.: 1002515821

(f) .....

MSc. Fernando Vinicio Valencia Aguirre
C.C.: 1003188669

DEDICATORIA

A mis padres, por ser el pilar fundamental en mi vida. Gracias por su amor incondicional, por los sacrificios silenciosos y por enseñarme con el ejemplo que el esfuerzo y la constancia siempre dan frutos. Su apoyo ha sido la base sobre la cual he construido cada uno de mis logros, y este trabajo no sería posible sin su guía y confianza. A mi hermana, por su alegría, su compañía y por estar siempre presente, incluso en los momentos en que no hacía falta decir nada para sentirse comprendido. Su cariño y motivación fueron clave para seguir adelante.

Esta tesis es también de ustedes, porque han estado conmigo en cada paso de este camino. Les agradezco profundamente por creer en mí incluso cuando yo dudaba, y por ser mi mayor motivación para no rendirme. Con todo mi cariño, les dedico este logro.

Franklin Xavier Arteaga Reina

AGRADECIMIENTO

Expreso mi más sincero agradecimiento al PhD. Xavier Rosero, director de esta tesis, por su guía constante, su disposición para compartir sus conocimientos y por orientarme con criterio y claridad durante todo el proceso.

Agradezco también al MSc. Fernando Valencia, asesor de este trabajo, por su valioso acompañamiento y su apoyo técnico en cada etapa del desarrollo.

Ambos han sido fundamentales en la realización de este proyecto, y su experiencia y compromiso académico han enriquecido profundamente mi formación. Gracias por confiar en mí y por fomentar en todo momento un ambiente de aprendizaje riguroso y motivador.

RESUMEN

El presente trabajo aborda el desafío de la localización y el mapeo simultáneo (Simultaneous Localization and Mapping - SLAM) en entornos dinámicos, mediante el diseño e implementación de un algoritmo funcional que integra múltiples técnicas de percepción y navegación. Se desarrolló una arquitectura SLAM dinámica que combina algoritmos preexistentes como RTAB-Map (para SLAM visual con cámara de profundidad y LiDAR), explore_lite (para exploración autónoma basada en detección de fronteras) y AMCL (para localización probabilística sobre un mapa previamente generado). Esta solución fue implementada en la plataforma TurtleBot2 equipada con una Jetson Nano, un sensor LiDAR y una cámara Kinect v1, utilizando el entorno operativo ROS. El sistema fue ajustado cuidadosamente para funcionar en tiempo real bajo limitaciones computacionales, asegurando eficiencia y precisión tanto en entornos simples como en escenarios dinámicos con obstáculos móviles. Se realizaron pruebas experimentales en tres entornos reales, evaluando la precisión del mapeo, la estabilidad de la localización y la capacidad del robot para evadir nuevos obstáculos. Los resultados obtenidos demuestran que la integración de estos módulos permitió el desarrollo de un algoritmo SLAM eficiente, capaz de adaptarse a variaciones en el entorno sin necesidad de reconstruir el mapa desde cero. Esta propuesta representa una contribución significativa al avance de la robótica móvil en Ecuador, al demostrar que es posible construir soluciones de navegación autónoma accesibles, robustas y replicables en contextos académicos e industriales.

Palabras clave: SLAM, robótica móvil, software, plataforma móvil, entorno dinámico.

ABSTRACT

This work addresses the challenge of Simultaneous Localization and Mapping (SLAM) in dynamic environments through the design and implementation of a functional algorithm that integrates multiple perception and navigation techniques. A dynamic SLAM architecture was developed that combines pre-existing algorithms such as RTAB-Map (for visual SLAM with depth camera and LiDAR), explore_lite (for autonomous exploration based on boundary detection), and AMCL (for probabilistic localization on a previously generated map). This solution was implemented on the TurtleBot2 platform equipped with a Jetson Nano, a LiDAR sensor, and a Kinect v1 camera, using the ROS operating environment. The system was carefully tuned to operate in real time under computational constraints, ensuring efficiency and accuracy in both simple environments and dynamic scenarios with moving obstacles. Experimental tests were carried out in three real environments, evaluating the accuracy of the mapping, the stability of the localization, and the robot's ability to avoid new obstacles. The results obtained demonstrate that the integration of these modules enabled the development of an efficient SLAM algorithm capable of adapting to variations in the environment without the need to rebuild the map from scratch. This proposal represents a significant contribution to the advancement of mobile robotics in Ecuador, demonstrating that it is possible to build accessible, robust, and replicable autonomous navigation solutions in academic and industrial contexts.

Keywords: SLAM, mobile robotics, software, mobile platform, dynamic environment.

INDICE DE CONTENIDOS

1. IDENTIFICACIÓN DE LA OBRA ii
2. CONSTANCIAS iii
CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR iv
APROBACIÓN DEL COMITÉ CALIFICADOR v
DEDICATORIA vi
AGRADECIMIENTO vii
RESUMEN viii
ABSTRACT ix
INDICE DE CONTENIDOS x
ÍNDICE DE FIGURAS xiii
ÍNDICE DE TABLAS xv
CAPITULO 1: INTRODUCCIÓN 1
1.1 Planteamiento del problema 1
1.2 Objetivos 2
1.2.1 General 2
1.2.2 Específicos 2
1.3 Alcance y delimitación 2
1.4 Justificación 3
CAPÍTULO 2: MARCO TEÓRICO 4
2.1 Antecedentes 4
2.2 Bases Teóricas 7
2.2.1 Robótica móvil 7
2.2.1.1 Evolución de la robótica móvil 7
2.2.2 Principios fundamentales de la robótica móvil 9
2.2.2.1 Planificación de la trayectoria 10
2.2.2.2 Localización 11
2.2.2.3 Percepción sensorial 12
2.2.2.4 Mapeo 14
2.2.2.5 Navegación autónoma 16
2.2.2.6 Inteligencia artificial aplicada a la robótica móvil 16

2.2.3 Plataformas robóticas móviles 17
2.2.3.1 Tipos de locomoción en plataformas móviles 17
2.2.4 Entornos de operación de robots móviles 19
2.2.4.1 Entorno estático 19
2.2.4.2 Entorno dinámico 19
2.2.4.3 Interacción del robot con el entorno 20
2.2.5 Técnicas de localización y mapeo 21
2.2.5.1 Filtros gaussianos 22
2.2.5.2 Filtros no paramétricos 24
2.2.6 Sensores para SLAM 25
2.2.7 Tipos de algoritmos para SLAM 31
2.2.7.1 Algoritmos basados en filtros 31
2.2.7.2 SLAM visual 35
2.2.8 Placas microcontroladas “Open Source” para SLAM 36
2.2.8.1 Raspberry Pi 36
2.2.8.2 Arduino 37
2.2.8.3 NVIDIA Jetson Nano 37
2.2.9 Software para robótica 38
2.2.9.1 Arquitectura general de ROS 38
2.2.9.2 Aplicaciones de ROS en robótica móvil 39
CAPÍTULO 3: MARCO METODOLÓGICO 40
3.1 Enfoque y tipos de investigación 40
3.2 Diseño de la Investigación 41
CAPÍTULO 4: RESULTADOS Y ANÁLISIS 44
4.1 Especificaciones del sistema a diseñar 44
4.2 Análisis comparativo de tecnologías 45
4.2.1 Sensor de distancia 45
4.2.1 CPU 46
4.3 Solución propuesta 47
4.4 Especificaciones del sistema diseñado 51
4.4.1 Plataforma robótica Turtlebot2 51
4.4.2 Sensor LiDAR A2M12 52

4.4.3 Cámara de profundidad Microsoft Kinect V1 52
4.4.4 Placa microcontrolada NVIDIA Jetson Nano 53
4.4.5 Power bank Sennbeyer 54
4.5 Puesta en marcha del sistema robótico 55
4.5.1 Prueba de funcionamiento del sensor LiDAR A2M12 55
4.5.2 Prueba de funcionamiento de la cámara de profundidad Kinect V1 56
4.5.3 Puesta en marcha del Turtlebot2 57
4.5.4 Diagrama de transformaciones TF 58
4.6 Mapeo con SLAM dinámico 60
4.6.1 Algoritmo de mapeo 61
4.6.2 Integración de explore lite 62
4.7 Escenarios experimentales 63
4.7.1 Escenario 1: Entorno simple sin obstáculos 63
4.7.2 Escenario 2: Entorno con pasillo y habitaciones 65
4.7.3 Escenario 3: Entorno con obstáculos añadidos 66
4.8 Navegación con AMCL sobre los mapas guardados 67
4.8.1 Proceso de guardado del mapa generado 67
4.8.2 Proceso de carga del mapa 68
4.8.3 Precisión de la localización 68
4.8.4 Evasión de obstáculos dinámicos 71
4.9 Análisis de resultados 74
CONCLUSIONES 75
RECOMENDACIONES 76
REFERENCIAS 77
ANEXOS 80

ÍNDICE DE FIGURAS

Fig. 2.1 Robot móvil Sojourner durante la misión Pathfinder para explorar Marte en 1997 [14]..... 8
Fig. 2.2 Robot de soldadura en planta de montaje de autos de KUKA [15] 9
Fig. 2.3 Trayectoria de un robot del punto S a T [16]..... 11
Fig. 2.4 Movimiento de un robot de accionamiento diferencial [14]..... 12
Fig. 2.5 Ejemplo de un proceso de precepción [18]..... 13
Fig. 2.6 Ejemplo de extracción de características a partir de un escáner láser [18]..... 14
Fig. 2.7 Información sobre el alcance [16]..... 15
Fig. 2.8 (a) Al detectar un objeto, el robot obtiene restricciones sobre su ubicación. (b) La restricción surge y la intersección de tres círculos determina un único punto para la ubicación del objeto [16]. 15
Fig. 2.9 Cronología de la robótica y la Inteligencia Artificial [20]..... 17
Fig. 2.10 Tipos de locomoción en robots móviles [13]..... 18
Fig. 2.11 Configuraciones de los robots móviles con ruedas [13] 18
Fig. 2.12 Interacción de un robot con su entorno [22] 20
Fig. 2.13 El proceso de SLAM implica estimar conjuntamente la ubicación del robot y los puntos de referencia, sin conocer sus posiciones exactas [26]. 21
Fig. 2.14 Sensores propioceptivos y exteroceptivos en un robot móvil [28] 27
Fig. 2.15 Ejemplo de un robot con sistema multisensorial [28]..... 27
Fig. 2.16 Sensor ultrasónico SRF04 y sus conexiones [29] 28
Fig. 2.17 Dibujo esquemático de un sensor láser con espejo giratorio [30]..... 29
Fig. 2.18 Principio de la triangulación laser ID [30]. 29
Fig. 2.19 Sensor óptico de triangulación de la serie GP de Sharp [31]..... 30
Fig. 2.20 Representación de la óptica de una cámara y su impacto en la imagen [30] 30
Fig. 2.21 Algoritmo del filtro de Kalman (EK) [23] 31
Fig. 2.22 Algoritmo del filtro extendido de Kalman (EKF) [23] 33
Fig. 2.23 Algoritmo del filtro de partículas [23] 34
Fig. 2.24 Diagrama de flujo de un enfoque SLAM visual estándar [33] 36
Fig. 2.25 Placa microcontrolada Raspberry Pi [34] 36

Fig. 2.26 Placa microcontrolada Arduino UNO [35] 37
Fig. 2.27 Placa microcontrolada NVIDIA Jetson Nano [36] 38
Fig. 4.1 Sistema de localización y mapeo simultáneo (SLAM) 49
Fig. 4.2 Base Kobuki de la plataforma Turtlebot2 50
Fig. 4.3 Parte inferior de la plataforma Turtlebot2 y baterías integradas 50
Fig. 4.4 Plataforma robótica Turtlebot2 52
Fig. 4.5 RPLIDAR A2M12 52
Fig. 4.6 Microsoft Kinect V1 53
Fig. 4.7 Placa NVIDIA Jetson Nano 54
Fig. 4.8 Power bank Sennbeyer 22.5W 55
Fig. 4.9 Datos del sensor lidar A2M12 publicados como /scan al nodo maestro ROS 56
Fig. 4.10 Prueba de funcionamiento de cámara de profundidad Kinect V1 57
Fig. 4.11 Diagrama de flujo de la secuencia para la puesta en marcha de la plataforma 58
Fig. 4.12 Diagrama de transformaciones del sistema 59
Fig. 4.13 Diagrama de flujo del sistema para mapeo 62
Fig. 4.14 Robot explorando sus fronteras con trayectoria automática (línea verde) 63
Fig. 4.15 Posición inicial del robot en el entorno 1 64
Fig. 4.16 Mapa final generado del entorno 1 64
Fig. 4.17 Robot en exploración del entorno 2 65
Fig. 4.18 Mapa final generado del entorno 2 66
Fig. 4.19 Incorporación de obstáculos durante la exploración del entorno 3 67
Fig. 4.20 Mapa final generado del entorno 3 67
Fig. 4.21 Posición inicial estimada por amcl 69
Fig. 4.22 Establecimiento del objetivo de navegación 70
Fig. 4.23 Trayectoria generada hacia el objetivo 70
Fig. 4.24 Trayectoria global inicial 72
Fig. 4.25 Desvío inicial para rodear el obstáculo 72
Fig. 4.26 Segundo desvío de trayectoria 73
Fig. 4.27 Ruta final ajustada para alcanzar el objetivo inicial 73

ÍNDICE DE TABLAS

Tabla 2.1: Clasificación de los sensores más utilizados en robótica según su objetivo de detección [propiocepción (PC)/exterocepción (EC)] y método (activo/pasivo) [18] 26
Tabla 2.2: Diferencias entre el algoritmo del filtro de Kalman y el filtro extendido de Kalman (EKF) [23] 33
Tabla 4.1: Comparativa de sensores de distancia para SLAM 46
Tabla 4.2: Comparativa de CPU para SLAM 47
Tabla 4.3: Componentes de la solución propuesta 51
Tabla 4.4: Descripción de los frames utilizados 60
Tabla 4.5: Comparativa de métricas en los distintos escenarios experimentales 74

CAPITULO 1: INTRODUCCIÓN

1.1 Planteamiento del problema

La localización y mapeo aplicado a entornos dinámicos, presenta un gran desafío en la optimización y maximización del rendimiento autónomo de la robótica móvil. El mapeo y generación de trayectorias de entornos no estructurados, ha experimentado un crecimiento significativo en las últimas décadas, revolucionando la navegación autónoma de los robots. Es así, que el uso de sistemas SLAM se propone como una potencial solución para tratar esta problemática [1].

Los entornos dinámicos se caracterizan por la presencia de obstáculos en constante cambio. La autonomía de un robot depende en gran medida de la capacidad para su auto localización y la generación de mapas actualizados y detallados del entorno en tiempo real para la toma de decisiones informadas. Los sistemas SLAM tradicionales generalmente basados en sensores enfrentan dificultades en estos entornos debido a la falta de robustez ante obstáculos móviles y a cambios en las condiciones ambientales [2]. La necesidad de operar en tiempo real agrega una presión adicional; ya que, los sistemas deben procesar grandes volúmenes de datos de manera eficiente. En base a lo anteriormente descrito, se plantea contribuir a la solución de la problemática planteada con la implementación de un sistema de localización y mapeo simultáneo (SLAM) para entornos dinámicos.

Con el sistema propuesto, se espera poder mejorar significativamente la autonomía de los robots móviles reduciendo el consumo de energía, contribuyendo así; al avance de la robótica móvil.

1.2 Objetivos

1.2.1 General

Desarrollar un algoritmo SLAM eficiente para entornos dinámicos.

1.2.2 Específicos

- Analizar los distintos tipos de técnicas de estimación y localización mediante el uso de una cámara de profundidad para navegación autónoma.
- Implementar el algoritmo de localización y mapeo seleccionado en una plataforma móvil para la generación de mapas detallados en tiempo real.
- Validar la efectividad y precisión del sistema SLAM desarrollado en un entorno no estructurado real.

1.3 Alcance y delimitación

La investigación tiene como objetivo el diseño de un algoritmo SLAM para la localización y mapeo de entornos dinámicos en tiempo real, el cual será implementado en una plataforma móvil. El sistema será adaptado a las condiciones específicas de un entorno no estructurado, buscando abordar y contrarrestar las falencias de los sistemas SLAM tradicionales en entornos con obstáculos móviles y cambios en las condiciones ambientales. Las restricciones físicas del sensor a utilizar pueden ser una limitante que afecte la capacidad del sistema para generar mapas detallados; en consecuencia, se pretende usar una cámara de profundidad que aporte robustez al sistema. El sistema será desarrollado en una placa microcontrolada “Open Source” con la finalidad de reducir el coste del mismo.

1.4 Justificación

En un contexto donde la automatización está en constante expansión y la interacción entre robots y humanos se vuelve cada vez más común, se torna fundamental desarrollar sistemas SLAM que puedan operar en tiempo real y superar las limitaciones inherentes de los sistemas tradicionales frente a obstáculos móviles y cambiantes en entornos reales [3]. La resolución de este problema permitiría mejorar de manera significativa la autonomía de los robots móviles, reduciendo el uso de recursos. El sistema generará un impacto positivo tanto en la sociedad como en el medio ambiente, contribuyendo así al avance continuo de la robótica móvil. En el ámbito económico, el desarrollo de un robot móvil con localización y mapeo SLAM puede tener un impacto sustancial para el sector académico e industrial del Ecuador; debido a que, la robótica colaborativa en los próximos años será un pilar fundamental en el desarrollo económico del país. El desarrollo de sistemas SLAM adaptados a los entornos dinámicos, puede abrir oportunidades para la implementación de este tipo de tecnología a nivel local y nacional.

La implementación de este tipo de sistemas podría conducir a la creación de empleos, al fortalecimiento del sector industrial y a la reducción de costos a los procesos automatizados. Finalmente, en el contexto investigativo, la adaptación y desarrollo de soluciones en robótica móvil contribuyen a la generación y transferencia de saberes; así como, a la promoción de la innovación. Además, al vincularse con la comunidad y enfocarse en criterios de sustentabilidad, la investigación no solo fortalecerá el desarrollo social y económico de la región y el país. En este documento, se profundiza en estudios preliminares cuyo objetivo es el diseño de un modelo SLAM altamente eficiente. Estos estudios se centran en la exploración exhaustiva de diversos tipos de algoritmos con el fin de identificar aquellos que ofrezcan el máximo rendimiento en términos de responsabilidad y contribución a un desarrollo tecnológico sostenible.

CAPÍTULO 2: MARCO TEÓRICO

2.1 Antecedentes

En este documento, se profundiza en estudios preliminares cuyo objetivo es el diseño de un modelo SLAM altamente eficiente. Estos estudios se centran en la exploración exhaustiva de diversos tipos de algoritmos con el fin de identificar aquellos que ofrezcan el máximo rendimiento en términos de precisión. Se ha prestado especial atención a los algoritmos que no solo manejan con destreza la localización y el mapeo en entornos dinámicos y complejos, sino que también abordan los desafíos inherentes a la incertidumbre y la variabilidad de los datos.

Se introduce el Filtro de Partículas (FP) para hacer frente a las limitaciones del filtro de Kalman, el cual adopta una distribución gaussiana para describir la variabilidad de un conjunto de datos. Los FP tradicionales tienen el problema del empobrecimiento de la muestra y una aproximación para resolver este problema es optimizar la distribución de la propuesta mostrada por las partículas. Este trabajo introduce un nuevo FP evolutivo basado en el Algoritmo de Optimización de Alta Dimensión por Oposición (OHDA) para reposicionar las partículas del FP en regiones de alta probabilidad para la estimación, lo cual puede contribuir en gran medida a desarrollar un modelo SLAM eficiente [3].

Por otra parte, se analizaron los métodos modernos de localización de robots terrestres móviles en entornos exteriores, prestando especial atención a los métodos basados en LiDAR, que permiten realizar una localización y mapeo simultáneos precisos, independientemente de las condiciones de iluminación, alcanzando un alto rendimiento en tiempo real para diferentes plataformas de hardware, en particular para la placa microcontrolada Nvidia Jetson AGX Xavier, lo cual muestra la robustez de este tipo de placas en aplicaciones SLAM [4].

Del mismo modo, se presenta sistema SLAM RGB-D en tiempo real para entornos altamente dinámicos con la ayuda de GPU Grid Map Projection, en un método de detección de puntos. El SLAM es un sistema sensible al tiempo para la robótica, y es difícil alcanzar el tiempo real, especialmente en entornos dinámicos porque es necesario rastrear objetos en movimiento y esto lleva mucho tiempo. Para resolver el problema del tiempo real, se propone un marco de detección de puntos dinámicos todo en paralelo para la localización y el mapeo visuales simultáneos (VSLAM) en entornos dinámicos basado en mapas de cuadrícula de ocupación 3D [5].

En otra investigación, se presenta un enfoque novedoso para aprovechar múltiples modalidades para una detección robusta y fiable en bucle de lazo cerrado. Este método se basa en el filtrado de partículas guiado por similitud (SGPF) para la búsqueda y validación de candidatos en lazo cerrado (LCC). Se validó la propuesta Multimodal Loop Closure (MMLC) utilizando dos modalidades de percepción basadas en las técnicas Bag-of-Words y Scan Context para el reconocimiento de lugares basado en cámaras y LiDAR, respectivamente [6].

Adicionalmente, se propone un método SLAM visual RGB-D unificado de fusión punto-línea-plano para la navegación de robots móviles en entornos estructurados y no estructurados, haciendo pleno uso de la información de tres tipos de características geométricas. En concreto, extrae características de punto, línea y plano utilizando imágenes capturadas por la cámara RGB-D y las expresa de manera uniforme. A continuación, se diseña un esquema de asociación mutua para la asociación de datos de características de punto, línea y plano, que no sólo considera la correspondencia de características homogéneas, es decir, pares punto-punto, línea-línea y plano-plano, sino que también incluye la asociación de características heterogéneas, es decir, pares punto-línea, punto-plano y línea-plano [7].

También, en otra investigación se explora la navegación autónoma de un robot móvil a través de un sensor láser (Hokuyo URG-04LX-UG01), acumulando datos para después tratarlos con el Filtro Extendido de Kalman (EKF). Sin embargo, con esta integración se obtuvo errores en los mapas en un rango de 20 a 30 cm cuando se compara los mapas del mismo entorno obtenidos consecutivamente. Es preciso tener presente que, para aplicaciones en entornos dinámicos, los errores se pueden incrementar; por lo cual, es necesario optar por una técnica de localización diferente para el mejor tratamiento de los datos obtenidos por el sensor [8]. De igual manera, en otra publicación se desarrolló un sistema SLAM, estableciendo una comunicación inalámbrica a través de la red Wireless propia de Robotino®, eliminando así la restricción de tener un medio físico como lo es un cable conectado a un computador. A pesar de ello, la velocidad de emisión y recepción de datos se ve limitada por la presencia de otras redes inalámbricas presentes. Este trabajo también incluye el EKF como técnica de localización y se propone el Filtro de Partículas como una alternativa más robusta para resolver problemas de localización y mapeo más detallados para entornos dinámicos [9].

Con el mismo enfoque, se logró la creación de un entorno virtual en Gazebo, mapeado mediante Rviz utilizando Localización simultánea y mapeo (SLAM). Una vez diseñado el entorno, se implementan dos algoritmos: el de árboles aleatorios de expansión rápida (RRT) y el de mapas de rutas probabilísticas (PRM). Estos algoritmos generan rutas de manera distinta: el primero crea caminos ramificados, mientras que el segundo toma muestras del entorno. Se llevaron a cabo pruebas en el SMA, variando los parámetros de los algoritmos y agregando obstáculos al entorno [10].

En una tesis de grado de la Universidad Técnica del Norte (UTN), se aborda el problema de localización y mapeo a través de la técnica de Exploración de fronteras y el sensor Kinect V1.

Obtuvieron como resultado una optimización del tiempo que toma el proceso, al usar la distancia media de la frontera y optimizando su trayectoria mediante un algoritmo de Dijkstra [11]. Del mismo modo, se llevó a cabo un estudio en el cual se planteó un estudio sobre las ventajas y desventajas del uso de sensores tradicionales frente a cámaras de profundidad en aplicaciones SLAM, las cuales se pusieron a prueba en el mapeo de un mismo entorno, dando como resultado que el uso de un sensor LiDAR, estimó trayectorias con una precisión del 97,98% con relación a las medidas reales del entorno [12].

2.2 Bases Teóricas

2.2.1 Robótica móvil

La robótica móvil se enfoca en el control y funcionamiento de vehículos que poseen un nivel total o parcial de autonomía. Actualmente, muchos de estos robots se emplean tanto en ambientes domésticos como en el sector industrial, aunque una parte considerable de los sistemas en funcionamiento sigue estando orientada a fines investigativos y se encuentra aún en etapa experimental. Las situaciones donde no es posible contar con intervención humana, ya sea por dificultades de acceso, elevados costos, operaciones prolongadas o condiciones poco adecuadas para las personas, han impulsado el uso exitoso de robots móviles en escenarios reales. El avance en esta área se sustenta en la integración de distintas disciplinas como la mecánica, la electrónica, la inteligencia artificial y la informática, permitiendo que estos sistemas ejecuten tareas con un notable nivel de autonomía [13].

2.2.1.1 Evolución de la robótica móvil

La robótica móvil ha atravesado un notable proceso de evolución, impulsado por avances tecnológicos y su aplicación en distintos sectores. Desde que surgió en la década de 1960, ha pasado

de ser un conjunto de dispositivos programables básicos a convertirse en sistemas autónomos avanzados, con la capacidad de desplazarse y desempeñar tareas en escenarios complejos. En las décadas de 1980 y 1990, esta rama de la robótica vivió un crecimiento importante, gracias al progreso de la inteligencia artificial y la computación. Fue en este periodo cuando los investigadores comenzaron a utilizar algoritmos de aprendizaje automático y técnicas de procesamiento de imágenes, lo que permitió que los robots pudieran asumir tareas más elaboradas y responder ante entornos variables. Además, en esos años aparecieron los primeros robots móviles con fines comerciales, como los sistemas automatizados de transporte utilizados en entornos industriales y logísticos [14].

Fig. 2.1 Robot móvil Sojourner durante la misión Pathfinder para explorar Marte en 1997 [14].

En el siglo XXI, la robótica móvil ha avanzado a un ritmo acelerado, gracias a factores como la reducción en el tamaño de los componentes tecnológicos, el incremento en la capacidad de procesamiento y la mejora continua de sensores y actuadores. Actualmente, los robots móviles pueden desenvolverse en una amplia gama de entornos, que van desde espacios interiores como

viviendas y oficinas, hasta zonas al aire libre e incluso misiones en el espacio. La incorporación de la inteligencia artificial ha sido clave para que estos robots puedan tomar decisiones por sí mismos, adaptarse a su entorno y colaborar de forma más eficiente tanto con personas como con otros robots.

Una de las áreas más representativas de la robótica móvil en esta etapa es la de los vehículos autónomos. Estos sistemas emplean una combinación de sensores avanzados, algoritmos de visión por computador y métodos de aprendizaje automático para desplazarse de manera segura por las vías urbanas, marcando una transformación significativa en los modelos actuales de transporte y movilidad en las ciudades [13].

Fig. 2.2 Robot de soldadura en planta de montaje de autos de KUKA [15]

2.2.2 Principios fundamentales de la robótica móvil

Una de las formas más básicas de representar teóricamente a un robot autónomo es mediante el concepto de robot puntual. Este modelo simplifica al robot considerándolo como un punto que se desplaza dentro de un entorno, generalmente descrito por un plano cartesiano continuo. Dentro de este, el robot puede representarse como un punto $(x, y) \in \mathbb{R}^2$. El dominio de operación del robot es

el plano. El punto (x, y) describe completamente el estado del robot y también se conoce como configuración del robot.

Desplazar al robot consiste en modificar dicho estado, pasando de una posición inicial (a, b) a una final (c, d). Aunque el plano es el entorno general de operación, no todo el dominio es accesible para el robot. Las posiciones a las que sí puede acceder conforman lo que se denomina espacio libre. A partir de estas definiciones es posible formular una serie de problemas clave en el estudio del comportamiento de los robots móviles.

- ¿Es posible pasar el robot de una configuración a otra permaneciendo dentro del espacio libre? Éste es el problema de la **planificación de la trayectoria** del robot.
- ¿Cómo puede el robot determinar su estado si dispone de mediciones locales del espacio libre? Este es el problema de la **localización**. El conocimiento completo del mundo del robot no es útil a menos que se sepa dónde se encuentra el robot en él.
- ¿Cómo puede determinar el robot qué partes de su entorno están ocupadas? Es decir, ¿cómo puede determinar si está libre? Este es el problema de la **percepción** del robot.
- ¿Cómo puede el robot determinar el espacio libre suponiendo que siempre sepa dónde está? Este es el problema de construir una representación del entorno del robot o **mapeo**. La existencia de un mapa suele suponerse implícitamente en la tarea de planificación de trayectorias [16].

2.2.2.1 Planificación de la trayectoria

La planificación de trayectorias representa un aspecto fundamental en aquellas aplicaciones donde los robots deben desenvolverse en entornos cambiantes o desconocidos. A través de los algoritmos de planificación, es posible que un robot calcule rutas que sean eficientes, evitando obstáculos y

optimizando parámetros como el tiempo requerido para el desplazamiento o el uso de energía. En el entorno de la Fig 2.3. El robot comienza en la configuración S = (a, b), y se desea moverlo a la configuración T = (c, d). La planificación consiste en encontrar una trayectoria que conecte ambos puntos dentro del espacio libre disponible. Desde un punto de vista formal, se busca una curva continua contenida en el espacio libre, cuyos extremos correspondan al punto de inicio (a, b) y al punto de destino (c, d). Para que una trayectoria válida exista, tanto la posición inicial como la meta deben ubicarse dentro del espacio libre. A partir de algún método de búsqueda, se pueden considerar diferentes criterios, como la ruta más corta o la más directa. Existen diversas estrategias para abordar el problema de planificación de trayectorias, muchas de las cuales se apoyan en aproximaciones que simplifican el problema original [16].

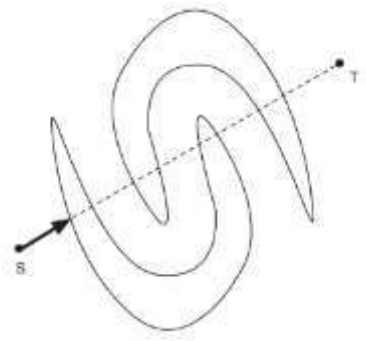


Fig. 2.3 Trayectoria de un robot del punto S a T [16]

2.2.2.2 Localización

La localización es un elemento esencial para el funcionamiento autónomo de un robot móvil. Esta capacidad le permite identificar su propia ubicación y orientación dentro del entorno en el que se encuentra [17]. Esto es fundamental para las tareas de navegación, donde el robot debe moverse de manera efectiva y segura, evitando obstáculos y llegando a sus destinos previstos.

Generalmente, la posición de un robot es representada por un vector,

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \tag{1}$$

para un robot de accionamiento diferencial, la posición puede estimarse a partir de una posición conocida integrando el movimiento (sumando las distancias de desplazamiento incrementales). Para un sistema discreto con un intervalo de muestreo fijo Δt, las distancias de desplazamiento incrementales $(\Delta x; \Delta y; \Delta\theta)$ son,

$$\Delta x = \Delta \mathrm{s} \cos\left(\theta + \frac{\Delta\theta}{2}\right), \tag{2}$$

$$\Delta \mathrm{y} = \Delta \mathrm{s} \sin\left(\theta + \frac{\Delta\theta}{2}\right), \tag{3}$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}, \tag{4}$$

$$\Delta \mathrm{s} = \frac{\Delta s_r + \Delta s_l}{2}, \tag{5}$$

donde $(\Delta x; \Delta y; \Delta\theta)$ representa el trayecto recorrido en el último intervalo de muestreo, $(\Delta s_r; \Delta s_l)$ representa las distancias recorridas para la rueda derecha e izquierda respectivamente y b es la distancia entre las dos ruedas del robot de tracción diferencial [14].

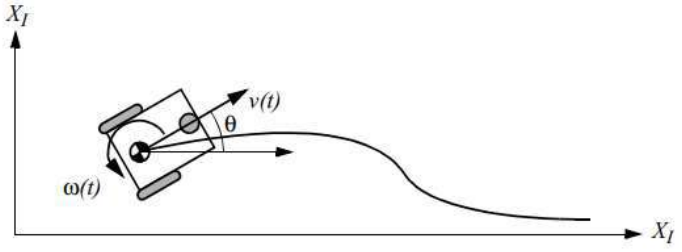


Fig. 2.4 Movimiento de un robot de accionamiento diferencial [14]

2.2.2.3 Percepción sensorial

El proceso de percepción en robótica suele basarse en dos tipos de información: por un lado, los datos digitales obtenidos por diversos sensores o transductores, y por otro, un modelo parcial del

entorno que incluye el estado del robot y elementos relevantes del entorno. Estos datos pueden presentarse como valores escalares, vectores en el tiempo, barridos, campos vectoriales o volúmenes 3D. En muchos casos, es necesario fusionar información proveniente de diferentes sensores. Por ejemplo, para estimar la posición de un robot móvil, se pueden combinar lecturas de codificadores, cámaras, GPS y sensores inerciales [17].

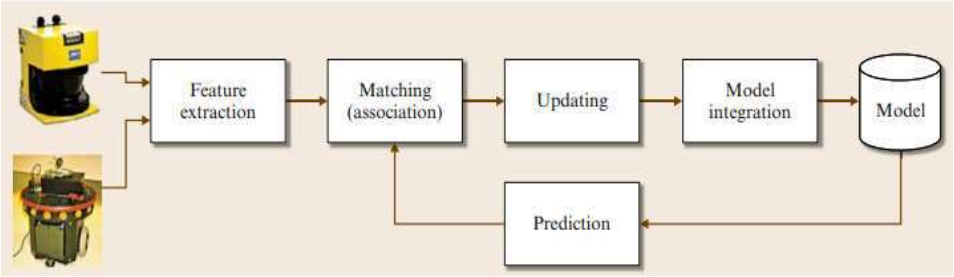


Fig. 2.5 Ejemplo de un proceso de precepción [17]

Este modelo considera las operaciones más comunes para combinar los datos sensoriales con un modelo general del entorno. El primer paso en el procesamiento de sensores suele ser el preprocesamiento y la extracción de características. El preprocesamiento tiene como objetivo reducir el ruido del sensor, corregir errores sistemáticos y resaltar la información relevante. En ocasiones, también es necesario alinear los datos en el tiempo o el espacio antes de integrarlos. Hay múltiples métodos para esta etapa, según el tipo de información. Un enfoque común es el ajuste de modelos, como se ilustra para un escáner láser en la Fig 2.6.

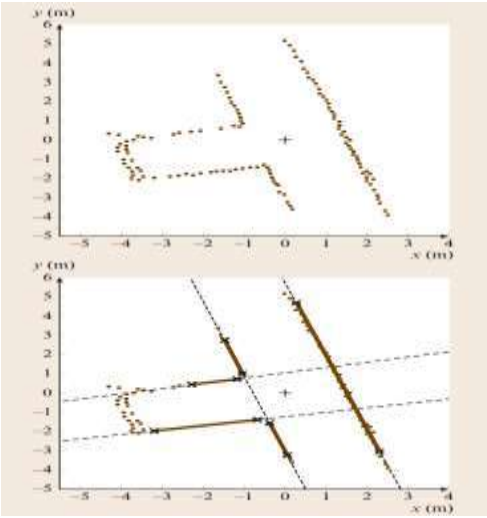


Fig. 2.6 Ejemplo de extracción de características a partir de un escáner láser [17]

2.2.2.4 Mapeo

Imaginando que un robot puntual se desplaza en un plano donde existen varios objetos fijos que emiten señales detectables por el robot, pero sin dirección, surge la pregunta de cómo puede este construir un mapa con sus ubicaciones. El sensor de alcance permite estimar la distancia a un objeto, lo que restringe su posible posición a un círculo en el plano (véase la Fig. 2.7), pero es insuficiente para permitir al robot localizar el objeto en un único punto. Sin embargo, al mover el robot y volver a detectar el objeto, se obtiene una nueva medición que impone otra restricción circular, como se observa en la Fig 2.8a. Esta intersección entre dos círculos define dos posibles ubicaciones del objeto. Con un tercer movimiento y otra medición, se genera una restricción adicional que permite reducir la incertidumbre a una única posición, como muestra la Fig 2.8b.

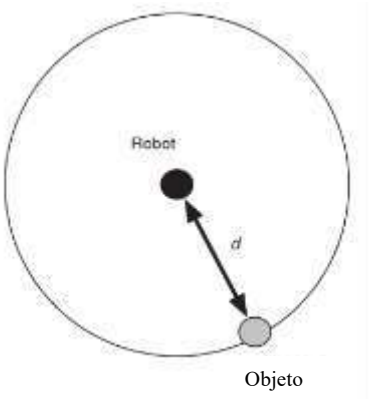
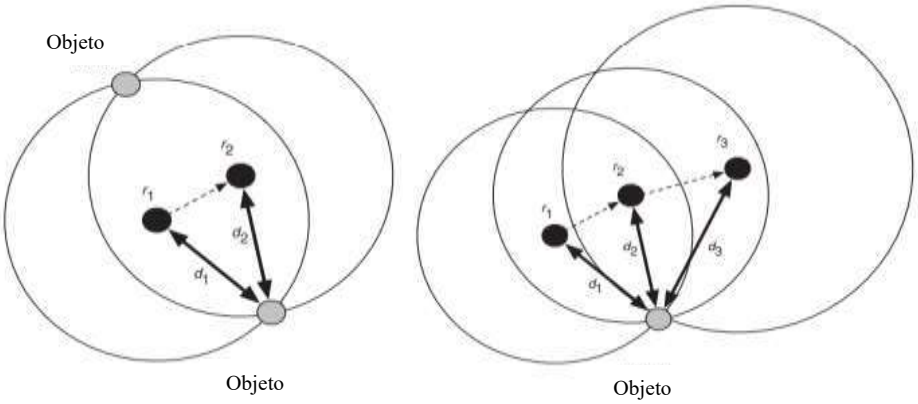


Fig. 2.7 Información sobre el alcance [16].

A medida que el robot se mueve, algunos objetos pueden entrar en el alcance del sensor y otras pueden salir de él. Sin embargo, tras dos movimientos del robot que den como resultado ubicaciones únicas en el espacio, se trazará la ubicación real de todos los objetivos que permanezcan en el radio de alcance del robot. Es importante observar que, para que la técnica funcione, el robot debe ser capaz de estimar continuamente su movimiento para integrar las restricciones a lo largo del tiempo.



(a) Después de un movimiento (b) Después de dos movimientos

Fig. 2.8 (a) Al detectar un objeto, el robot obtiene restricciones sobre su ubicación. (b) La restricción surge y la intersección de tres círculos determina un único punto para la ubicación del objeto [16].

2.2.2.5 Navegación autónoma

La navegación autónoma de robots móviles es un área multidisciplinaria que integra distintas tecnologías con el objetivo de permitir que los robots se desplacen e interactúen de forma independiente en su entorno. Esta capacidad requiere la combinación de sensores, sistemas de modelado y control, generación de mapas, planificación de trayectorias y toma de decisiones. Los sensores permiten al robot percibir su entorno, y mediante la fusión sensorial se mejora la calidad y utilidad de esa percepción. La elaboración de mapas y la planificación del movimiento utilizan tanto los datos sensoriales como las estrategias de control para aumentar la autonomía y eficiencia del robot.

La autonomía, en este contexto, se refiere a la capacidad del robot para tomar decisiones por sí mismo y ejecutar acciones sin intervención humana. Un robot autónomo toma y lleva a cabo sus decisiones de forma independiente, mientras que un robot teleoperado depende parcial o totalmente del control humano [18].

2.2.2.6 Inteligencia artificial aplicada a la robótica móvil

La disciplina enfocada en dotar a las máquinas de comportamientos inteligentes se conoce como inteligencia artificial (IA). Gracias a los avances en mecatrónica, electrónica e informática, la robótica ha logrado desarrollar funciones sensoriomotoras cada vez más complejas, permitiendo que los robots se adapten de mejor forma a entornos cambiantes.

Tradicionalmente, en la industria, el entorno se ajustaba a las necesidades de la máquina, que era calibrada para operar bajo condiciones muy controladas. Actualmente, los robots pueden adaptarse con mayor facilidad a entornos ya existentes. La autonomía robótica suele dividirse en tres componentes principales: percepción, planificación y ejecución, que incluye acciones como

manipular objetos, desplazarse y colaborar. La convergencia entre IA y robótica busca precisamente aumentar esta autonomía a través del aprendizaje, entendiendo la inteligencia como la capacidad del sistema para anticipar el futuro, ya sea planificando una tarea o interactuando activamente con su entorno mundo [19].

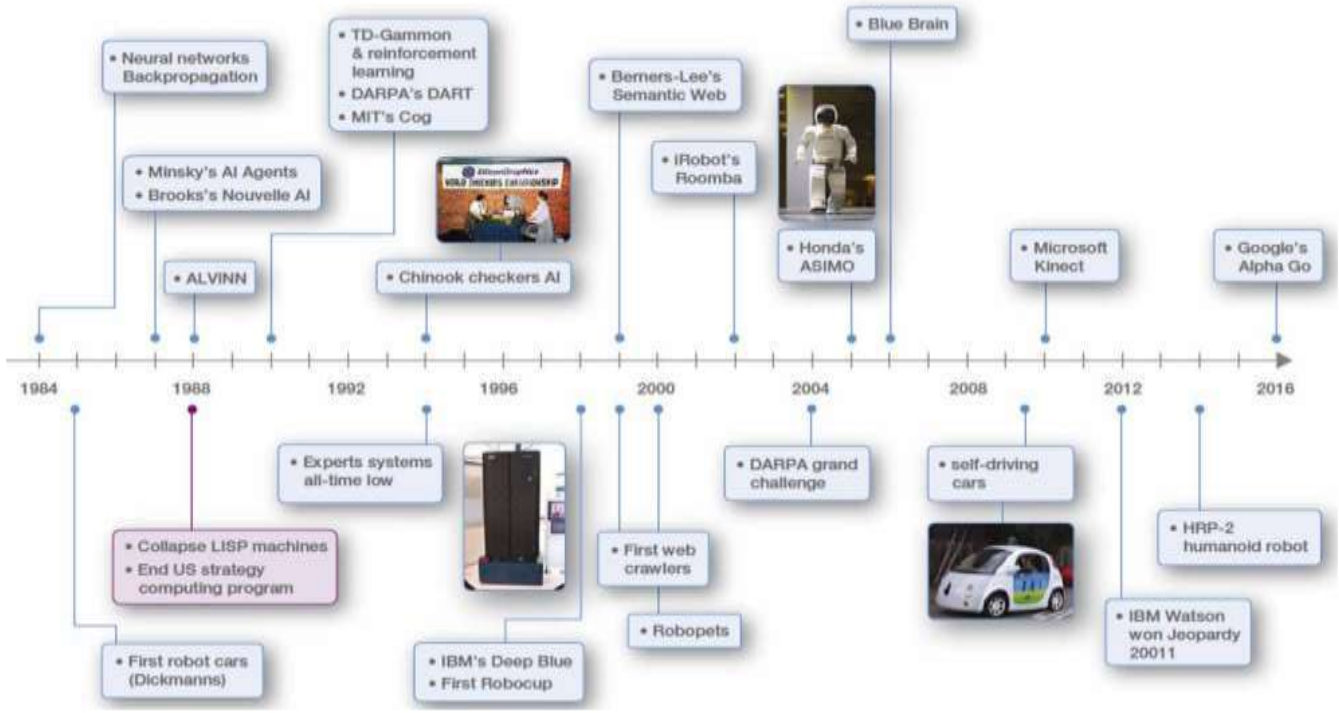


Fig. 2.9 Cronología de la robótica y la Inteligencia Artificial [19]

2.2.3 Plataformas robóticas móviles

2.2.3.1 Tipos de locomoción en plataformas móviles

Los robots móviles se pueden clasificar según su sistema de locomoción, que comúnmente se divide en tres categorías: robots con ruedas, con patas y con orugas. Aunque el movimiento mediante patas y orugas ha sido ampliamente estudiado, el mayor desarrollo se observa en los robots móviles con ruedas (RMR) [13]. Esta preferencia se debe a que las ruedas ofrecen varias

ventajas frente a las otras modalidades, como una mayor eficiencia energética en superficies planas y firmes, menor impacto en el terreno, y un diseño menos complejo al requerir menos componentes. En la Fig. 2.10 se ilustran los distintos métodos de locomoción.

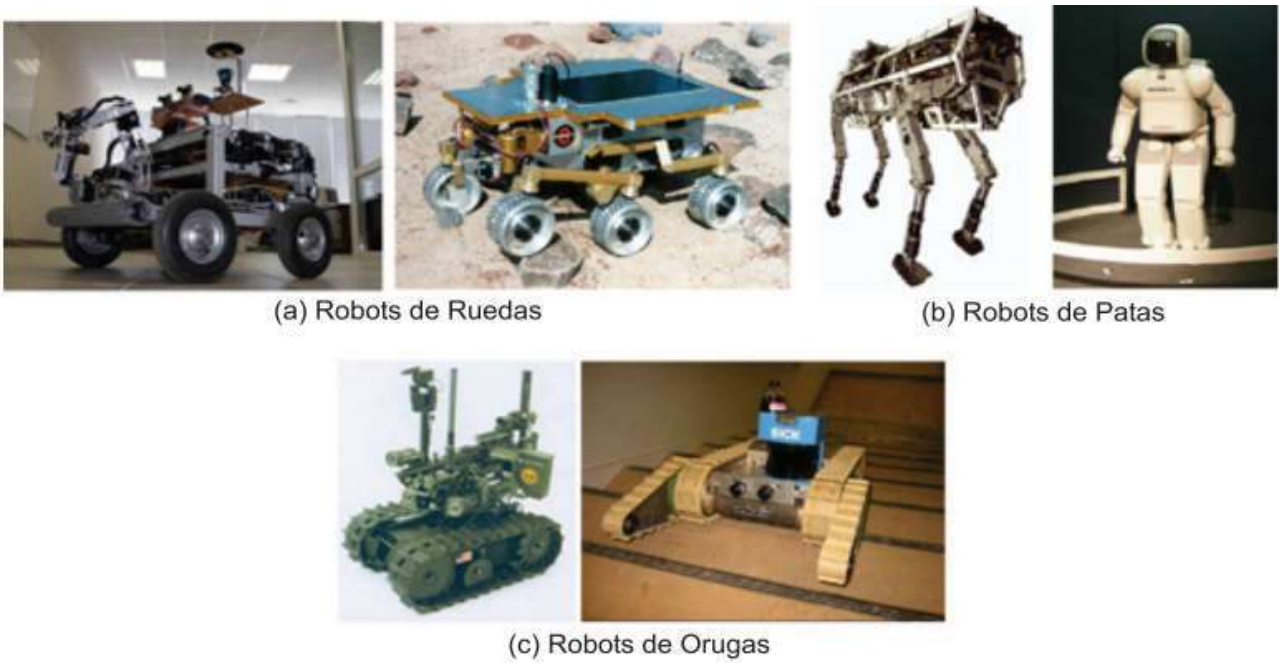


Fig. 2.10 Tipos de locomoción en robots móviles [13]

Hay varias configuraciones cinemáticas para robots móviles con ruedas, y su elección se basa en gran medida en el uso específico del robot. Sin embargo, comúnmente se utilizan algunas configuraciones clave, que incluyen: Ackerman, triciclo clásico, tracción diferencial, skid steer, tracción síncrona y omnidireccional. Estas configuraciones se pueden observar en la Fig 2.11.

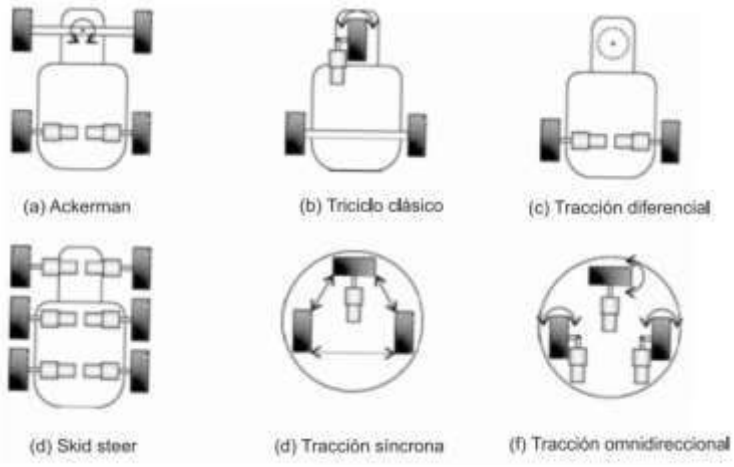


Fig. 2.11 Configuraciones de los robots móviles con ruedas [13]

2.2.4 Entornos de operación de robots móviles

2.2.4.1 Entorno estático

Un entorno estructurado o estático se refiere a un escenario o contexto donde las condiciones y reglas son fijas, predecibles y consistentes. En tales entornos, los cambios ocurren raramente o son mínimos, lo que permite una planificación y gestión más sencilla y directa. Un ejemplo clásico es una línea de producción en una fábrica, donde cada tarea, proceso y rol están claramente definidos y siguen un patrón constante.

2.2.4.2 Entorno dinámico

Los entornos no estructurados o dinámicos son aquellos que se caracterizan por un alto grado de incertidumbre y cambio constante. A diferencia de los entornos estructurados, en los entornos dinámicos, las reglas, condiciones y procesos no son fijos y pueden cambiar rápidamente, lo que requiere una mayor adaptabilidad y flexibilidad por parte de quienes operan dentro de ellos. En un entorno dinámico, los sistemas deben ser capaces de responder rápidamente a los cambios, aprovechar nuevas oportunidades y mitigar riesgos de manera efectiva.

2.2.4.3 Interacción del robot con el entorno

Existen dos tipos fundamentales de interacciones entre un robot y su entorno: El robot puede influir en el estado de su entorno a través de sus actuadores y puede recoger información sobre el estado a través de sus sensores, como se muestra en la Fig. 2.12.

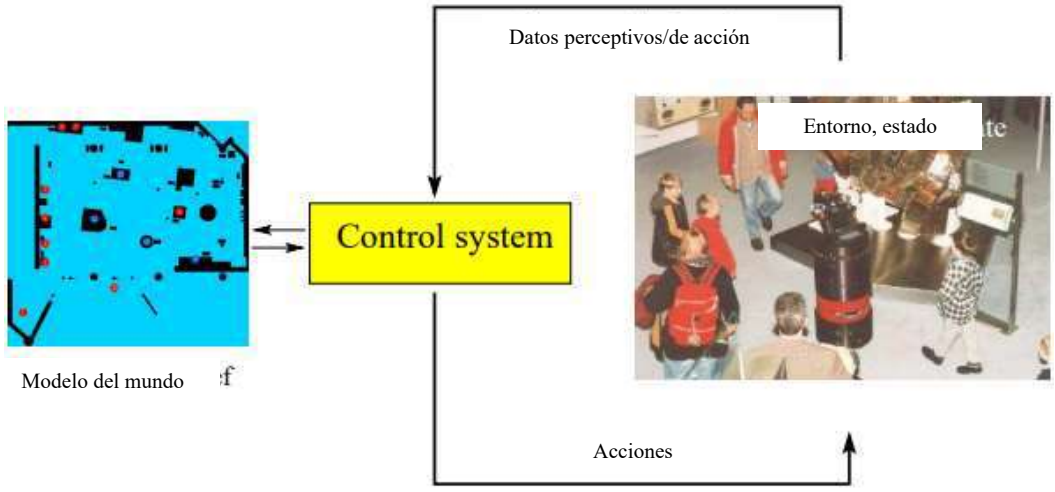


Fig. 2.12 Interacción de un robot con su entorno [20]

- **Medición de los sensores**. La percepción es el proceso mediante el cual el robot recoge información sobre su entorno a través de sus sensores. El resultado de esta interacción se denomina medición, aunque también puede llamarse observación o percepción.
- **Acciones de control**: Estas implican la aplicación activa de fuerzas sobre el entorno, como el desplazamiento del robot o la manipulación de objetos. Incluso cuando el robot no realiza ningún movimiento, su estado puede variar. Por consistencia, se considera que el robot siempre ejecuta una acción de control, aunque no active ninguno de sus motores [21] .

2.2.5 Técnicas de localización y mapeo

La localización y mapeo simultáneos (SLAM) consiste en construir de manera conjunta un modelo del entorno y estimar la posición del robot que se desplaza en él. El desafío principal de SLAM es que un robot móvil pueda ubicarse en un entorno desconocido y, al mismo tiempo, crear un mapa coherente de dicho entorno de forma incremental, mientras determina su propia posición dentro del mapa [22].

Una técnica fundamental en SLAM es el uso de sensores como cámaras y LIDAR para recopilar información del entorno. Estos datos se procesan mediante algoritmos que ayudan al robot a identificar referencias y calcular su ubicación relativa a ellas. El proceso de mapeo genera un modelo del entorno, que puede ser en dos o tres dimensiones, basado en la información obtenida [23].

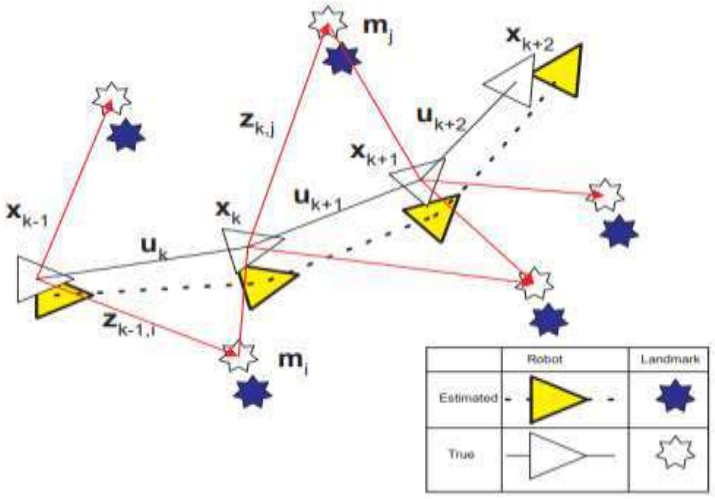


Fig. 2.13 El proceso de SLAM implica estimar conjuntamente la ubicación del robot y los puntos de referencia, sin conocer sus posiciones exactas [24].

Considerando un robot móvil que se desplaza por un entorno tomando observaciones relativas de

una serie de puntos de referencia desconocidos mediante un sensor situado en el robot, tal como se muestra en la Fig. 2.13. En un instante de tiempo k, se definen las siguientes cantidades:

- x_k: El vector de estado que describe la ubicación y orientación del vehículo.
- u_k: El vector de control, aplicado en el instante $k-1$ para conducir el vehículo al estado x_k en el instante k.
- m_i : Vector que describe la ubicación del punto de referencia i^{th}, cuya verdadera ubicación se supone invariante en el tiempo.
- z_{ik}: Una observación tomada desde el vehículo de la ubicación de i^{th} en el tiempo k. Cuando hay múltiples observaciones de hitos en un momento dado o cuando el hito específico no es relevante para la discusión, la observación se escribirá simplemente como z_k [24].

2.2.5.1 Filtros gaussianos

Los filtros gaussianos constituyen una importante familia de estimadores recursivos de estado. Fueron las primeras soluciones prácticas para espacios continuos, destacando por su eficiencia y facilidad de implementación. Estos filtros actualizan el estado en un tiempo que crece polinómicamente con la dimensionalidad del espacio de estados.

A. *Filtro de Kalman (KF)*

   Desarrollado en la década de 1950 por Rudolph Emil Kalman, este filtro es una técnica para el filtrado y la predicción en sistemas lineales. Está diseñado para estados continuos y no es adecuado para espacios discretos o híbridos [2].

El filtro de Kalman esta dado mediante la representación de momentos: En el momento t, la creencia está representada por la media μt y la covarianza. Los sucesores son gaussianos si se cumplen las dos propiedades siguientes.

1. La probabilidad del siguiente estado $p(x_t|u_t, x_{t-1})$ debe ser una función lineal en sus argumentos con ruido gaussiano añadido. Esto se expresa mediante la siguiente ecuación,

$$x(t) = A_t x_{t-1} + B_t u_t + \varepsilon_t. \quad (6)$$

   Aquí $x(t)$ y x_{t-1} son vectores de estado, u_t es el vector de control en el tiempo t. En la notación del autor, ambos son vectores verticales, es decir, son de la forma.

   A_t y B_t son matrices. A_t es una matriz cuadrada de tamaño n × n, donde n es la dimensión del vector de estado x_t. B_t es de tamaño n×m, siendo m la dimensión del vector de control u_t. Al multiplicar los vectores de estado y control por las matrices A_t y B_t, respectivamente, la función de transición de estado se convierte en lineal en sus argumentos. Así pues, el filtro de Kalman supone una dinámica lineal del sistema.

2. La probabilidad de medición $p(z_t|x_t)$ también debe ser lineal en sus argumentos, con ruido gaussiano añadido,

   Aquí C_t es una matriz de tamaño k × n, donde k es la dimensión del vector de medición

$$z_t = C_t x_t + \delta_t. \quad (7)$$

   vector z_t. El vector δ_t describe el ruido de medición. La distribución de δt es una gaussiana multivariante con media cero y covarianza Q_t.

B. Filtro extendido de Kalman (EKF)

Los supuestos de transiciones lineales de estado y mediciones con ruido gaussiano agregado son poco frecuentes en aplicaciones reales. Por ejemplo, un robot que se desplaza con velocidades constantes de traslación y rotación sigue una trayectoria circular, la cual no puede describirse mediante transiciones lineales de estado. Esta limitación, sumada a la hipótesis de creencias unimodales, hace que los filtros de Kalman simples sean inaplicables en la mayoría de los problemas robóticos, salvo en casos muy básicos.

El filtro extendido de Kalman (EKF) aborda esta restricción, al permitir funciones no lineales para describir la probabilidad del siguiente estado y las mediciones, mediante las funciones g y h, respectivamente,

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t, \tag{8}$$

$$z_t = h(x_t) + \delta_t. \tag{9}$$

Este modelo generaliza estrictamente el modelo lineal gaussiano subyacente en los filtros de Kalman, postulado en las ecuaciones (6) y (7). La función g sustituye a las matrices A_t y B_t en (8), y h sustituye a la matriz C_t en (9) [25].

2.2.5.2 Filtros no paramétricos

A diferencia de los filtros gaussianos, los filtros no paramétricos no asumen una forma fija para la distribución posterior, como la gaussiana. En su lugar, aproximan dicha distribución mediante un conjunto finito de muestras, cada una representando una región del espacio de estados.

Estos filtros son una alternativa popular cuando se necesita mayor flexibilidad en la representación de la distribución posterior, especialmente en casos donde las suposiciones gaussianas resultan demasiado restrictivas o poco precisas.

A. Filtro de partículas

El filtro de partículas es una implementación no paramétrica ampliamente usada en robótica y sistemas autónomos para la estimación del estado. Este método representa la distribución posterior mediante un conjunto finito de muestras, conocidas como partículas, que son generadas y distribuidas de forma que cubran el espacio de estados. Cada partícula corresponde a una posible configuración del sistema, y el conjunto completo ofrece una aproximación de la distribución posterior. En el filtro de partículas, las muestras de una distribución posterior se denominan partículas y se denotan como,

$$x_t = x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]}. \tag{10}$$

Cada partícula $x_t^{[M]}$ (con $1 \leq m \leq M$) es una instanciación concreta del estado en el momento t, es decir, una hipótesis sobre cuál puede ser el verdadero estado del mundo en el momento t. Aquí M denota el número de partículas del conjunto de partículas x_t. En la práctica, el número de partículas M suele ser un número grande, por ejemplo, M = 1.000. En algunas implementaciones M es una función de t.

Los filtros de partículas son especialmente útiles en situaciones donde el modelo del sistema o las mediciones son altamente no lineales o el espacio de estados es muy complejo [21].

2.2.6 Sensores para SLAM

Hay varias formas de clasificar los sensores en función de lo que miden y cómo lo miden. Por ejemplo, los sensores propioceptivos se utilizan para medir el estado interno de un robot, que puede incluir la posición diferentes grados de libertad, la temperatura, el voltaje de componentes clave, la corriente del motor, la fuerza aplicada a un efector, entre otros. Los sensores exteroceptivos por

su parte generan información sobre el entorno externo en términos de distancia a un objeto, fuerzas de interacción, densidad, entre otros.

Tabla 2.1: Clasificación de los sensores más utilizados en robótica según su objetivo de detección [propiocepción (PC)/exterocepción (EC)] y método (activo/pasivo) [17]

Clasificación	Tipo de sensor	Detección	Activo/Pasivo
Sensores táctiles	Interruptores	EC	Pasivo
	Barreras ópticas	EC	Activo
	Proximidad	EC	Pasivo/Activo
Sensores hápticos	Matrices de contacto	EC	Pasivo
	Fuerza	PC/EC	Pasivo
	Resistivos	EC	Pasivo
Sensores de motor/ejes	Encoders	PC	Pasivo
	Potenciómetros	PC	Pasivo
	Resolvers	PC	Activo
	Encoders ópticos	PC	Activo
	Encoders magnéticos	PC	Activo
	Encoders inductivos	PC	Activo
	Encoders capacitivos	EC	Activo
Sensores de rumbo	Brújulas	EC	Pasivo
	Giroscopios	PC	Pasivo
	Inclinómetros	EC	Activo/Pasivo
Alcance	Sensores capacitivos	EC	Pasivo
	Sensores magnéticos	EC	Activo/Pasivo
	Cámaras	EC	Activo/Pasivo
	Sonar	EC	Activo
	Sensores láser	EC	Activo
	Haz de luz	EC	Activo
Velocidad/movimiento	Sonido Doppler	EC	Activo
	Cámaras	EC	Pasivo
	Acelerómetros	EC	Pasivo
Identificación	Cámaras	EC	Pasivo
	Radiofrecuencia	EC	Activo
	Radar	EC	Activo
	Sonido	EC	Pasivo
	Ultrasonido	EC	Activo

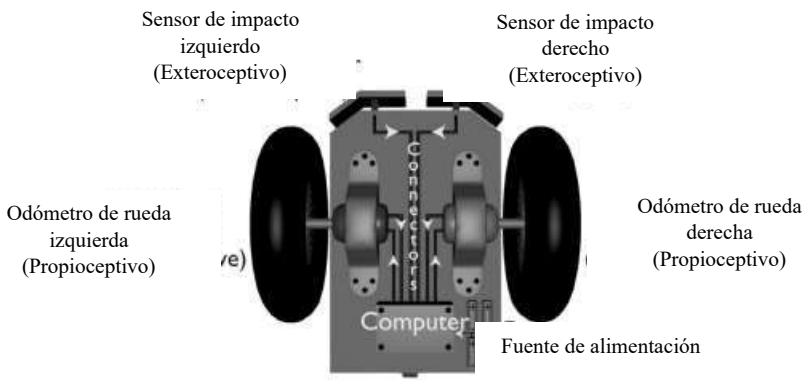


Fig. 2.14 Sensores propioceptivos y exteroceptivos en un robot móvil [26]

Los sensores pasivos captan la energía ambiental que incide sobre ellos. Ejemplos comunes incluyen sondas de temperatura, micrófonos y cámaras CCD o CMOS.

Por otro lado, los sensores activos emiten energía hacia el entorno y miden la respuesta recibida. Debido a que pueden controlar mejor la interacción con el ambiente, suelen ofrecer un desempeño superior. Sin embargo, esta detección activa implica ciertos riesgos, ya que la energía emitida puede influir en las características que el sensor intenta medir. Además, la señal del sensor activo puede verse afectada por interferencias externas fuera de su control.

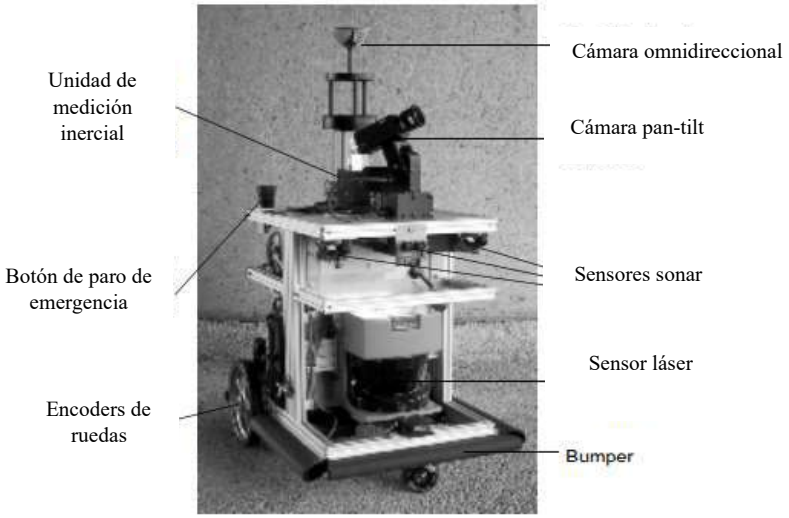


Fig. 2.15 Ejemplo de un robot con sistema multisensorial [26]

A. Sensores ultrasónicos

El principio básico de un sensor ultrasónico es transmitir un paquete de ondas de presión (ultrasónicas) y medir el tiempo que tarda este paquete de ondas en reflejarse y volver al receptor. La distancia del objeto que provoca la reflexión puede calcularse en función de la velocidad de propagación del sonido y del tiempo de vuelo.

Fig. 2.16 Sensor ultrasónico SRF04 y sus conexiones [27]

B. Sensor laser (time-of-flight)

El sensor láser es un dispositivo de tiempo de vuelo que mejora considerablemente el alcance comparado con los sensores ultrasónicos al utilizar luz láser en lugar de ondas sonoras. Está compuesto por un transmisor que proyecta un haz colimado hacia el objetivo y un receptor que detecta la luz reflejada, generalmente alineada coaxialmente con el haz emitido. Conocidos también como radar óptico o lidar (light detection and ranging), estos sensores calculan la distancia midiendo el tiempo que tarda la luz en ir y volver del objeto. Un mecanismo mecánico con espejos, a menudo giratorios y oscilantes, permite barrer el haz para escanear la escena en uno o varios planos, incluso en 3D.

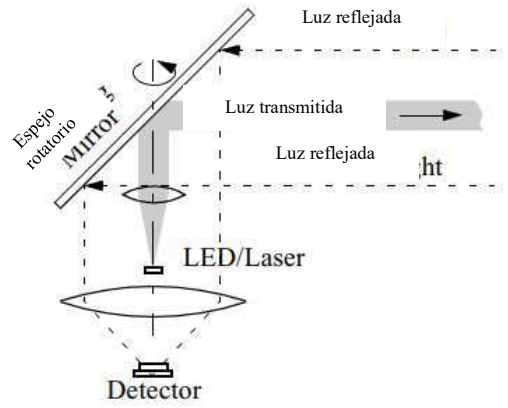


Fig. 2.17 Dibujo esquemático de un sensor láser con espejo giratorio [28]

C. Sensor de Triangulación óptica (sensor ID).

El principio de la triangulación óptica en ID es sencillo, como se muestra en la Fig. 2.18. Un haz colimado (por ejemplo, un LED infrarrojo enfocado, rayo láser) hacia el objetivo. La luz reflejada es recogida por una lente y se proyecta sobre un dispositivo sensible a la posición (PSD) o una cámara lineal.

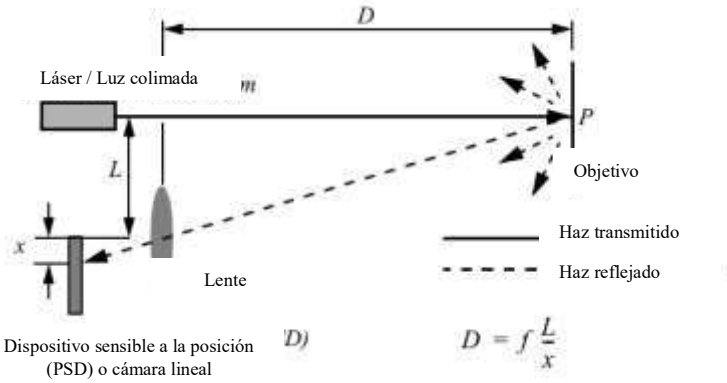


Fig. 2.18 Principio de la triangulación laser ID [28].

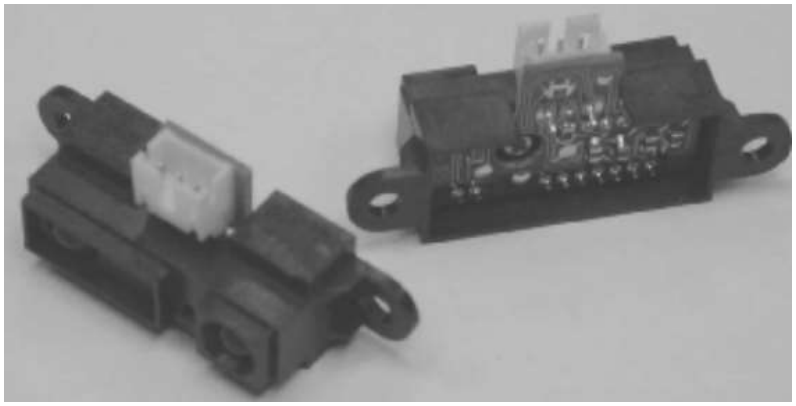


Fig. 2.19 Sensor óptico de triangulación de la serie GP de Sharp [29]

D. Sensores visuales de alcance

La detección de distancias es fundamental en robótica móvil, pues constituye un dato clave para la correcta evitación de obstáculos. No obstante, la obtención de información de profundidad a partir de imágenes visuales presenta desafíos importantes.

La solución más común es recuperar la profundidad mediante la captura de múltiples imágenes desde diferentes perspectivas, con el fin de extraer suficiente información para estimar la distancia. Para que una imagen sea nítida, el plano de enfoque debe coincidir con el plano de la imagen; de lo contrario, el punto (x, y, z) se verá desenfocado, como se ilustra en la Fig 2.20.

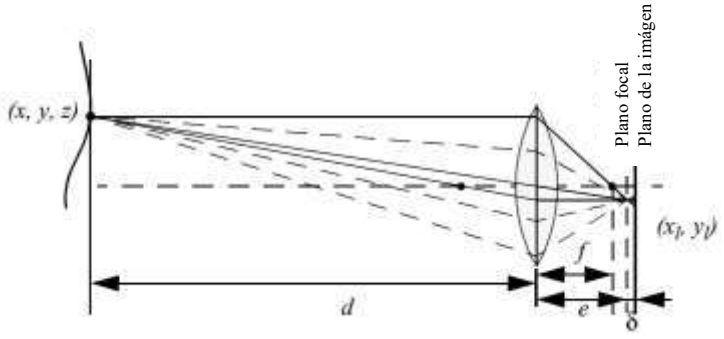


Fig. 2.20 Representación de la óptica de una cámara y su impacto en la imagen [28]

2.2.7 Tipos de algoritmos para SLAM

2.2.7.1 Algoritmos basados en filtros

A. Algoritmo del Filtro de Kalman

El algoritmo del Filtro de Kalman es una técnica matemática que proporciona una solución eficiente y recursiva para el problema de la estimación lineal. Es utilizado ampliamente en aplicaciones de control y navegación para estimar el estado interno de un proceso en tiempo real a partir de una serie de mediciones que contienen ruidos. En la Fig. 2.21 se representa el algoritmo del filtro de Kalman.

1: **Algorithm Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)**:**
2: $\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$
3: $\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$
4: $K_t = \bar{\Sigma}_t\, C_t^T (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\, \bar{\mu}_t)$
6: $\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$
7: return μ_t, Σ_t

Fig. 2.21 Algoritmo del filtro de Kalman (EK) [21]

A continuación, se describen cada una de las líneas de código referentes al filtro de Kalman.

1. Inicialización
    - El algoritmo inicia con una estimación previa del estado del sistema (u_{t-1}) y la incertidumbre asociada a esa estimación (Σ_{t-1}). Además, recibe como entradas la señal de control actual (u_t) y la medición actual (z_t).
2. Predicción
    - $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$: Esta es la función de predicción del estado. Calcula la estimación previa del estado actual $\bar{\mu}_t$ utilizando la estimación del estado anterior

μ_{t-1}, la matriz de transición de estado A_t, y la matriz de control B_t junto con la señal de control u_t. Es una función lineal.

- $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$: Esta fórmula actualiza la covarianza de la predicción del error. Estima la incertidumbre previa del estado actual Σ_t usando la incertidumbre del estado anterior Σ_{t-1}, la matriz de transición de estado A_t, y la covarianza del ruido del proceso R_t.

3. Actualización

- $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$: Calcula la ganancia de Kalman K_t, que es un factor que pondera la importancia de la nueva medición versus la predicción. C_t es la matriz de observación que relaciona el estado con la medición, y Q_t es la covarianza del ruido de la medición.
- $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$: Esta es la actualización del estado. Corrige la predicción del estado $\bar{\mu}_t$ con la ganancia de Kalman K_t y el residuo entre la medición real z_t y la medición esperada $C_t \bar{\mu}_t$. Esto resulta en la estimación corregida del estado μ_t.
- $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$: Actualiza la covarianza del error de la estimación Σ_t . Disminuye la incertidumbre de la predicción $\bar{\Sigma}_t$ basándose en la ganancia de Kalman K_t y la matriz de observación C_t . I representa la matriz identidad del tamaño apropiado.

4. Resultado

- $return\ \mu_t, \Sigma_t$: Finalmente, el algoritmo devuelve la estimación corregida del estado μ_t y la covarianza actualizada del error Σ_t [21].

B. Filtro extendido de Kalman (EKF)

La Fig. 2.22 presenta el algoritmo EKF. En muchos aspectos, este algoritmo es similar al algoritmo del filtro de Kalman indicado en la Fig. 2.21. Las diferencias más importantes se resumen en la Tabla 2.2.

1: **Algorithm Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$
4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$
7: return μ_t, Σ_t

Fig. 2.22 Algoritmo del filtro extendido de Kalman (EKF) [21]

Tabla 2.2: Diferencias entre el algoritmo del filtro de Kalman y el filtro extendido de Kalman (EKF) [21].

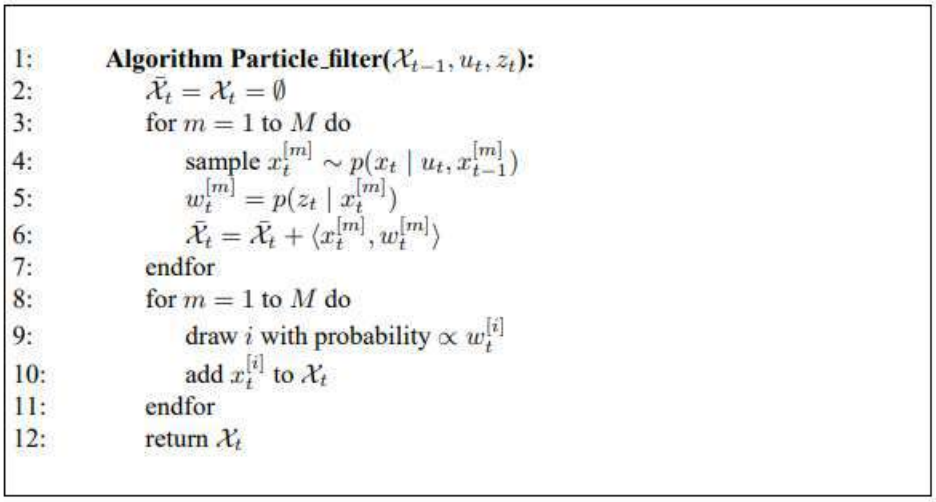
	Filtro de Kalman	**Filtro extendido de Kalman**
Predicción de estado	$A_t\mu_{t-1} + B_t\mu_t$	$g(\mu_t, \mu_{t-1})$
Predicción en la medición	$C_t\bar{\mu}_t$	$h(\bar{\mu}_t)$

Es decir, las predicciones lineales de los filtros de Kalman se sustituyen por sus generalizaciones no lineales en los EKF. Además, los EKF utilizan los jacobianos G_t y H_t en lugar de las correspondientes matrices del sistema lineal A_t, B_t y C_t de los filtros de Kalman. El jacobiano G_t corresponde a las matrices A_t y B_t, y el jacobiano H_t corresponde a C_t.

B. Algoritmo del filtro de partículas

El algoritmo del filtro de partículas es una aplicación no paramétrica el cual aproxima la posterior mediante un número finito de parámetros. Es un método de estimación secuencial y es útil para

estimar el estado de sistemas no lineales y/o no gaussianos. El filtro mantiene un conjunto de hipótesis (partículas) sobre el estado del sistema y las actualiza en base a la evidencia de las mediciones.



1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Fig. 2.23 Algoritmo del filtro de partículas [21]

A continuación, se describen cada una de las líneas de código referentes al filtro de partículas en la Fig 2.23.

1. Inicialización

   - Comienza con un conjunto vacío de partículas $\bar{X}_t$ para el tiempo actual t y el conjunto de partículas del tiempo anterior $\bar{X}_{t-1}$

2. Muestreo

   Para cada partícula m desde 1 hasta M:

   - $sample\ x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$:Genera una muestra (partícula) x[m]t del espacio de estados basada en la partícula anterior $x_{t-1}^{[m]}$ y la señal de control actual u_t, utilizando la densidad de probabilidad de transición de estados p.

- $w_t^{[m]} = p(z_t|x_t^{[m]})$: Calcula el peso $w_t^{[m]}$ de la partícula $x_t^{[m]}$ utilizando la función de probabilidad de la medición actual z_t, dada la partícula $x_t^{[m]}$.
- $\bar{X}_t = \bar{X}_t + (x_t^{[m]}, w_t^{[m]})$: Agrega la partícula y su peso al conjunto de partículas para el tiempo actual t.

3. Muestreo de importancia

   Después de que todas las partículas han sido muestreadas y pesadas según su importancia:

   Para cada partícula m desde 1 hasta M:

   - $draw\ i\ with\ probability \propto w_t^{[i]}$: Selecciona una partícula i con una probabilidad proporcional a su importancia $w_t^{[i]}$.
   - $add\ x_t^{[i]}\ to\ X_t$:Añade la partícula seleccionada al conjunto final de partículas X_t.

4. Resultado

   - $return\ X_t$: Devuelve el conjunto actualizado de partículas X_t [21].

2.2.7.2 SLAM visual

La localización y mapeo simultáneos visuales (vSLAM) es una tecnología fundamental para la automatización en robótica, que permite a los robots funcionar de manera autónoma en entornos desconocidos. Este método consiste en construir un mapa del entorno mientras se rastrea la posición del robot, utilizando principalmente sensores visuales. Estos dispositivos se destacan por su tamaño compacto, bajo consumo energético y alta riqueza en la información capturada.

A pesar de los avances logrados, vSLAM aún enfrenta retos como el ruido en los sensores, variaciones en la iluminación y movimientos rápidos del robot [30]. Los progresos en visión por computadora han mejorado significativamente su desempeño, sobre todo en aplicaciones de

navegación interior, realidad virtual y entornos complejos. Sin embargo, siguen siendo desafíos importantes mantener un seguimiento estable y garantizar una eficiencia adecuada en el procesamiento en tiempo real.

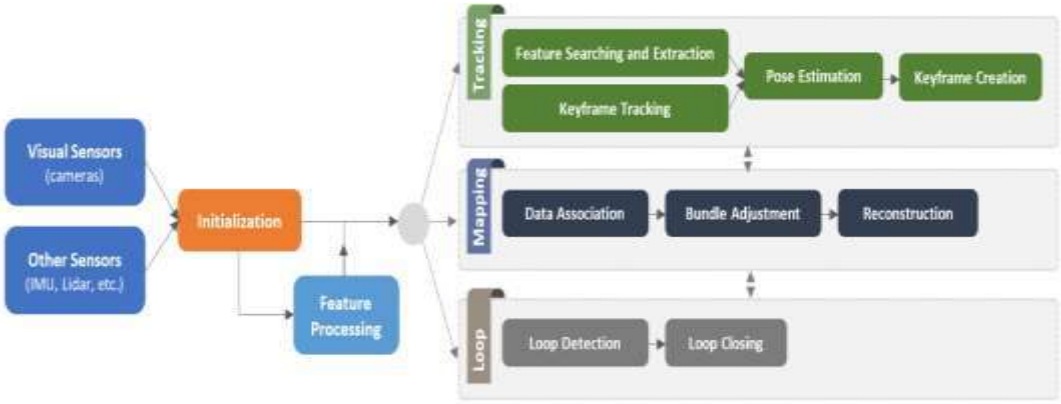


Fig. 2.24 Diagrama de flujo de un enfoque SLAM visual estándar [30]

2.2.8 Placas microcontroladas "Open Source" para SLAM

2.2.8.1 Raspberry Pi

El Raspberry Pi es un microordenador económico y compacto que ha transformado la informática y la enseñanza en electrónica. Gracias a su procesador con buen rendimiento y sus múltiples opciones de conectividad, es ampliamente utilizado en proyectos de robótica, particularmente en aplicaciones de SLAM. Su compatibilidad con diversos sistemas operativos y lenguajes de programación facilita la implementación flexible de algoritmos complejos para el mapeo y la localización.

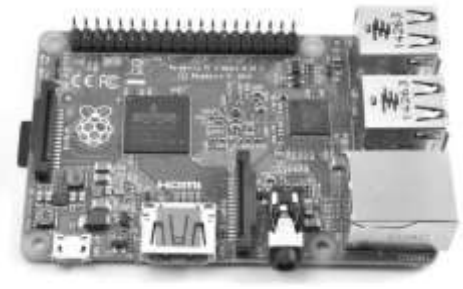


Fig. 2.25 Placa microcontrolada Raspberry Pi [31]

2.2.8.2 Arduino

Arduino es una plataforma de microcontroladores de código abierto, conocida por su facilidad de uso y su amplia comunidad. Aunque no es tan potente como otros microcontroladores en términos de procesamiento, es extremadamente útil para controlar la electrónica de un sistema de SLAM, como la gestión de sensores y motores. Su simplicidad y costo-efectividad lo hacen ideal para prototipos de SLAM en aplicaciones donde la complejidad computacional no es crítica.

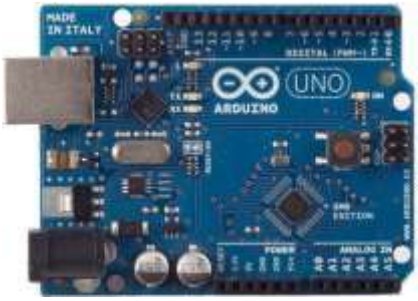


Fig. 2.26 Placa microcontrolada Arduino UNO [32]

2.2.8.3 NVIDIA Jetson Nano

El NVIDIA Jetson Nano es un pequeño pero potente ordenador de una sola placa diseñado para aplicaciones de inteligencia artificial y aprendizaje automático. Gracias a su potente CPU y GPU con arquitectura NVIDIA Maxwell, es ideal para proyectos de SLAM que requieren procesamiento intensivo, como el reconocimiento de objetos y la visión por computadora en tiempo real. En el contexto de SLAM, el Jetson Nano puede ejecutar algoritmos complejos, incluyendo aquellos basados en aprendizaje profundo, procesamiento de imágenes y reconocimiento de objetos en tiempo real.

Fig. 2.27 Placa microcontrolada NVIDIA Jetson Nano [33]

2.2.9 Software para robótica

El desarrollo de sistemas robóticos modernos requiere de plataformas de software que permitan la integración modular de sensores, actuadores, algoritmos de control y herramientas de visualización. Uno de los frameworks más extendidos en la comunidad robótica es ROS (Robot Operating System), el cual proporciona una infraestructura de comunicación entre procesos distribuida, facilitando el desarrollo, prueba e implementación de algoritmos para robótica móvil.

2.2.9.1 Arquitectura general de ROS

ROS está basado en una arquitectura distribuida compuesta por nodos, los cuales son procesos que realizan tareas específicas, como adquirir datos de sensores o generar comandos de movimiento. Estos nodos se comunican entre sí a través de tópicos, servicios, y el sistema de transformaciones TF, que permite mantener una relación espacial coherente entre los distintos marcos de referencia del robot.

Además, ROS ofrece herramientas esenciales para depuración y visualización, como Rviz para representación en tiempo real del entorno y rqt_graph para examinar la red de nodos y tópicos [34].

2.2.9.2 Aplicaciones de ROS en robótica móvil

Gracias a su enfoque modular y su comunidad activa, ROS cuenta con múltiples bibliotecas y algoritmos ya implementados para tareas de SLAM, navegación autónoma, localización, seguimiento de trayectorias, entre otros. Esto permite a los desarrolladores enfocarse en la integración de estos componentes según los requerimientos del sistema, sin necesidad de desarrollar todo desde cero. Además, ROS es compatible con una amplia variedad de sensores, plataformas móviles y sistemas operativos, lo que lo convierte en una opción flexible y robusta para la investigación y desarrollo en robótica móvil [34].

CAPÍTULO 3: MARCO METODOLÓGICO

3.1 Enfoque y tipos de investigación

El presente trabajo de integración curricular tiene como objetivo el desarrollo de una investigación aplicada, ya que, a través de la aplicación de los conocimientos adquiridos a lo largo de la carrera de ingeniería en mecatrónica, se pretende hacer uso de estas para dar una solución [35], el cual es la optimización de los sistemas SLAM tradicionales, mejorando la eficiencia en tiempo real del sistema ante entornos dinámicos.

Del mismo modo, el enfoque de este trabajo también se basa en una investigación documental, debido a que se realiza una búsqueda minuciosa de bibliografía de fuentes escritas, tales como libros, tesis, artículos científicos, reportes, entre otras, para la búsqueda de datos relevantes que contribuyan al cumplimiento de los objetivos general y específicos de este trabajo de integración curricular [36]. A su vez, también se pretende realizar una investigación descriptiva, como base en la investigación documental, ya que, se pretende realizar una comparación de los distintos tipos de técnicas de localización y mapeo, sensores y placas micro controladas, para la selección de las mejores alternativas en cada campo que contribuyan a un sistema robusto capaz de realizar su función eficientemente [37].

Todo esto converge en una investigación experimental, puesto que, una vez implementado el sistema en un entorno real, se podrá evaluar y validar la eficiencia tanto del algoritmo y los materiales seleccionados mediante el mapa del entorno generado y su aproximación a la realidad, así como, la posición de la plataforma robótica dentro de los límites del mapa [35].

3.2 Diseño de la Investigación

Para la ejecución del presente trabajo de integración curricular se plantearon distintas actividades individuales que contribuyan al cumplimiento de los objetivos planteados y de esta manera desarrollar un algoritmo eficiente para SLAM el cual se implementara en una placa micro controlada “Open Source”, funcionando en una plataforma móvil totalmente autónoma.

3.2.1. Fase 1: Análisis de los tipos de técnicas de estimación y localización, mediante el uso de una cámara de profundidad para navegación autónoma.

Actividad 1.1: Investigación sobre sensores de detección de objetos y cámaras de profundidad; en esta actividad, se busca la realización de una investigación documental sobre los distintos tipos de sensores y cámaras de profundidad disponibles en el mercado ecuatoriano.

Actividad 1.2: Investigación y comparación de los distintos tipos de técnicas de estimación y localización; en esta actividad se realiza una investigación descriptiva sobre los distintos tipos de técnicas de localización y mapeo, haciendo una comparación.

Actividad 1.3: Selección de técnica de localización y mapeo; el propósito de esta actividad consiste en, una vez realizada la investigación documental y descriptiva, seleccionar la mejor alternativa a técnica para SLAM.

Actividad 1.4: Selección de cámara de profundidad; esta actividad tiene como propósito la selección de una cámara de profundidad adecuada que permita el mapeo de un entorno 3d y que esté disponible en el mercado ecuatoriano.

3.2.2. Fase 2: Implementación del algoritmo de localización y mapeo seleccionado en una plataforma móvil para la generación de mapas detallados en tiempo real.

Actividad 2.1: Investigación y comparación de distintos tipos de placas microcontroladas; esta actividad es de tipo documental y descriptiva, ya que, permite tener un enfoque más amplio de todos los tipos de placas microcontroladas existentes y su aplicación en sistemas SLAM.

Actividad 2.2: Selección de modelo de placa microcontrolada; el propósito de esta actividad consiste en la selección de la mejor alternativa de una placa microcontrolada que permita una aplicación SLAM y sea robusta para un correcto funcionamiento del sistema en entornos dinámicos reales.

Actividad 2.3: Desarrollo del algoritmo SLAM; esta actividad es de tipo experimental, ya que, una vez seleccionada la técnica de localización y mapeo, se desarrollará un algoritmo de localización y mapeo en una plataforma "Open Source".

Actividad 2.4: Selección de plataforma móvil; esta actividad se refiere al tipo de plataforma necesaria para garantizar una navegación autónoma del sistema en entornos dinámicos.

Actividad 2.5: Ensamble de la placa microcontrolada y de la cámara de profundidad en la plataforma móvil; esta actividad tiene como propósito el montaje de todos los materiales previamente seleccionados y diseñados en la plataforma móvil, garantizando el correcto movimiento de la plataforma móvil en un entorno real.

3.2.3. Fase 3: Validación de la efectividad y precisión del sistema SLAM desarrollado en un entorno no estructurado real.

Actividad 3.1: Pruebas de funcionamiento de la cámara de profundidad integrada a la placa microcontrolada; esta actividad tiene como objetivo el realizar las primeras pruebas de funcionamiento de la cámara de profundidad montada en la plataforma móvil para el escaneo de un entorno 3d real.

Actividad 3.2: Pruebas de funcionamiento del sistema en ambientes controlados y no controlados; en esta actividad se realizarán pruebas de funcionamiento del sistema SLAM desarrollado, en entornos sin obstáculos móviles y con obstáculos móviles.

Actividad 3.3: Análisis de la eficiencia del algoritmo SLAM; en esta actividad se analizará tanto la efectividad del sistema SLAM desarrollado, como la de los componentes seleccionados para este fin, mediante la generación de mapas en tiempo real del entorno.

Actividad 3.4: Ajustes y correcciones; en esta actividad se desarrollarán los ajustes y correcciones necesarias al algoritmo SLAM para mejorar la precisión del mapa generado respecto al entorno real.

Actividad 3.5: Validación del sistema; mediante esta actividad, se validará la efectividad del sistema propuesto, a través del mapa generado y su aproximación con la realidad, así como, la correcta localización de la plataforma móvil en dicho mapa.

Actividad 3.6: Desarrollo del documento de Trabajo de Integración Curricular; el propósito de esta actividad es el progreso en el documento final escrito con los formatos y normas establecidas por la docente de trabajo de integración curricular.

CAPÍTULO 4: RESULTADOS Y ANÁLISIS

En el presente capítulo, se presentan los resultados obtenidos tras la implementación del sistema de localización y mapeo simultáneo (SLAM) en la plataforma Turtlebot2, utilizando la placa microcontrolada Jetson Nano como controlador principal. Se analiza el desempeño del sistema en entornos dinámicos, evaluando la precisión y eficiencia de los algoritmos de mapeo y localización en tiempo real. A partir de las pruebas realizadas, se comparan los resultados obtenidos con los criterios establecidos, valorando el nivel de precisión en la detección de obstáculos y la capacidad de adaptación ante cambios en el entorno.

4.1 Especificaciones del sistema a diseñar

Criterios:

- Integración de Hardware: Uso de la plataforma Turtlebot2 como base móvil autónoma para la prueba del sistema SLAM.
- Jetson Nano: Control principal del Turtlebot2, con capacidad para procesar datos en tiempo real provenientes del LiDAR y ejecutar los algoritmos de SLAM.
- Desempeño del sistema: El sistema debe generar mapas detallados del entorno en tiempo real, manteniendo la precisión y actualización constante frente a cambios.
- Eficiencia energética: Optimización del consumo energético de la Jetson Nano, el Turtlebot2 y el sensor LiDAR, buscando mantener un equilibrio entre el rendimiento y la duración de la batería.
- Requerimientos de software: El sistema debe estar basado en ROS, aprovechando las bibliotecas existentes para la navegación autónoma y la integración de hardware. Los

algoritmos de SLAM deberán estar optimizados para la Jetson Nano, aprovechando su capacidad de procesamiento paralelo en caso necesario.

- Validación del sistema: Validar el sistema en un entorno controlado y real, con la introducción de obstáculos móviles para probar la robustez del mapeo y la localización.

Restricciones:

- Batería del Turtlebot2: Duración limitada que condiciona la autonomía operativa del robot y los experimentos en campo.
- Entornos interiores: El Turtlebot2 está diseñado para operar en superficies planas y estables, lo que limita su efectividad en terrenos irregulares o exteriores.
- Interferencia en el LiDAR: Factores como superficies reflectantes o interferencias ópticas pueden afectar la precisión del sensor.
- Software y compatibilidad: La integración de ROS con la Jetson Nano podría estar limitada por problemas de compatibilidad de versiones, requiriendo configuraciones específicas y personalizaciones en los drivers y controladores.

4.2 Análisis comparativo de tecnologías

4.2.1 Sensor de distancia

La capacidad de detección de objetos es crucial para el cumplimiento de los requerimientos de este trabajo de integración curricular, por lo cual, se han escogido tres opciones para solventar esto. En la Tabla 4.1 se detallan dichas opciones y sus características cruciales para aplicaciones SLAM.

Tabla 4.1: Comparativa de sensores de distancia para SLAM

Característica	RPLIDAR A2M12	LiDAR TFmini	Sensor de distancia Sharp
Tipo de tecnología	LiDAR 360 grados	LiDAR de un solo haz (medición en un solo punto)	Sensor infrarrojo unidireccional
Rango de detección	Hasta 12 metros	Hasta 12 metros	Entre 10 y 150 cm
Resolución y precisión	0.5 grados, precisión de 1-3cm	1 grado, precisión de 1-2cm (en distancias cortas)	Baja resolución, precisión moderada en distancias cortas
Campo de visión	360 grados (barrido completo)	Unidireccional	Unidireccional (ángulo muy limitado)
Velocidad de adquisición	Hasta 8000 muestras por segundo	100Hz	Variable, significativamente menor que LiDAR
Consumo energético	2.5W	1W	0.5W
Costo	$180 USD	$40-80 USD	$10-20 USD
Aplicaciones comunes	Mapeo completo, SLAM, navegación autónoma en entornos complejos	Detección de obstáculos, SLAM limitado, robots pequeños	Detección de obstáculos simples, aplicaciones básicas de robótica
Compatibilidad con ROS	Excelente soporte y documentación	Buen soporte, requiere integración personalizada	Soporte limitado, requiere adaptadores

De estas tres opciones, el RPLIDAR A2M12 es el más completo y adecuado para aplicaciones avanzadas de SLAM y mapeo en 360 grados. El LiDAR TFmini es una alternativa económica con un alcance similar, pero está limitado a una sola dirección y carece de barrido, lo que restringe su uso para tareas de mapeo. El sensor infrarrojo Sharp es el más básico y barato, útil solo para detección de obstáculos en un ángulo limitado, pero insuficiente para aplicaciones SLAM.

4.2.1 CPU

La elección de la plataforma de hardware es fundamental para la implementación de SLAM. Existen diversas opciones en el mercado, cada una con características específicas que pueden influir en el rendimiento del sistema. A continuación, se presenta un análisis comparativo entre tres opciones populares: La placa NVIDIA Jetson Nano, Raspberry Pi 4 y el Arduino Mega. En la Tabla 4.2 se especifica los componentes esenciales para el desarrollo de un sistema SLAM robusto.

Tabla 4.2: Comparativa de CPU para SLAM

Característica	NVIDIA Jetson Nano	Raspberry Pi 4	Arduino Mega
Arquitectura	ARM Cortex-A57 + GPU NVIDIA Maxwell	ARM Cortex-A72 (4 núcleos)	Microcontrolador AVR ATmega2560
Capacidad de procesamiento	CPU de 4 núcleos a 1.43 GHz + 128 núcleos CUDA para procesamiento paralelo	CPU de 4 núcleos a 1.5 GHz	16 MHz, sin capacidad de procesamiento paralelo
Capacidad para SLAM	Excelente (procesamiento paralelo para visión artificial y redes neuronales)	Aceptable para tareas básicas de SLAM y navegación autónoma	Limitada, no adecuada para procesamiento de SLAM debido a baja potencia
Memoria RAM	4 GB LPDDR4	Hasta 8 GB LPDDR4	8 KB SRAM
Soporte de GPU	NVIDIA GPU Maxwell (128 núcleos CUDA)	No tiene GPU dedicada	No tiene GPU dedicada
Consumo energético	5-10W	3-5W	Menos de 1W
Compatibilidad con ROS	Excelente soporte para ROS y visión artificial	Compatible con ROS, aunque con menos rendimiento en visión artificial	Muy limitada, difícil integración con ROS
Costo aproximado	$200 USD	$35-75 USD	$30 USD
Aplicaciones comunes	Robótica avanzada, visión artificial, SLAM, procesamiento paralelo	Proyectos de robótica básica, control de sensores, proyectos de SLAM básico	Control de sensores y motores básicos, aplicaciones de baja complejidad

La placa Jetson Nano ha sido seleccionada para este proyecto debido a su capacidad de procesamiento avanzado que incluye una GPU dedicada, ideal para procesamiento de imágenes provistas por la cámara de profundidad, y tareas intensivas de visión artificial en tiempo real, esenciales para SLAM. Tanto la Raspberry Pi 4 y el Arduino Mega, carecen del poder de procesamiento necesario para SLAM dinámico y en tiempo real. Por lo tanto, la Jetson Nano es la opción más robusta para cumplir con los requisitos de este trabajo de integración curricular.

4.3 Solución propuesta

La solución propuesta se basa en el desarrollo de un sistema de localización y mapeo simultáneo (SLAM) para entornos dinámicos, utilizando componentes accesibles, como la placa NVIDIA Jetson Nano, un sensor LiDAR y la plataforma móvil Turtlebot2. Este sistema se implementa mediante el uso de software de código abierto, específicamente el Robot Operating System (ROS)

sobre Ubuntu 18.04, aprovechando sus librerías especializadas y herramientas de comunicación entre dispositivos. El sistema se controla mediante una Jetson Nano, lo que proporciona el procesamiento necesario para ejecutar algoritmos complejos de percepción y control. En cuanto al software, se ha empleado la versión Melodic de ROS, que facilita la integración de sensores, la comunicación entre nodos y el desarrollo de algoritmos de SLAM (Simultaneous Localization and Mapping). El Turtlebot2 se desplaza por el entorno, capturando datos de profundidad a través del LiDAR y la cámara de profundidad Microsoft Kinect (integrada en el Turtlebot2), que luego son procesados en tiempo real para construir y actualizar un mapa del entorno y permitir la localización precisa del robot en este. Esta configuración de hardware y software permite un SLAM eficiente y adaptable en entornos complejos.

Esta solución busca ofrecer un enfoque eficiente y reproducible para aplicaciones de robótica autónoma, brindando flexibilidad y adaptabilidad ante entornos cambiantes mediante herramientas accesibles y ampliamente respaldadas por la comunidad de desarrollo.

En las Figs 4.1, 4.2 y 4.3 se muestran cada uno de los componentes esenciales que componen el sistema, mientras que en la Tabla 4.3 se describen, de manera general cada uno de los elementos.

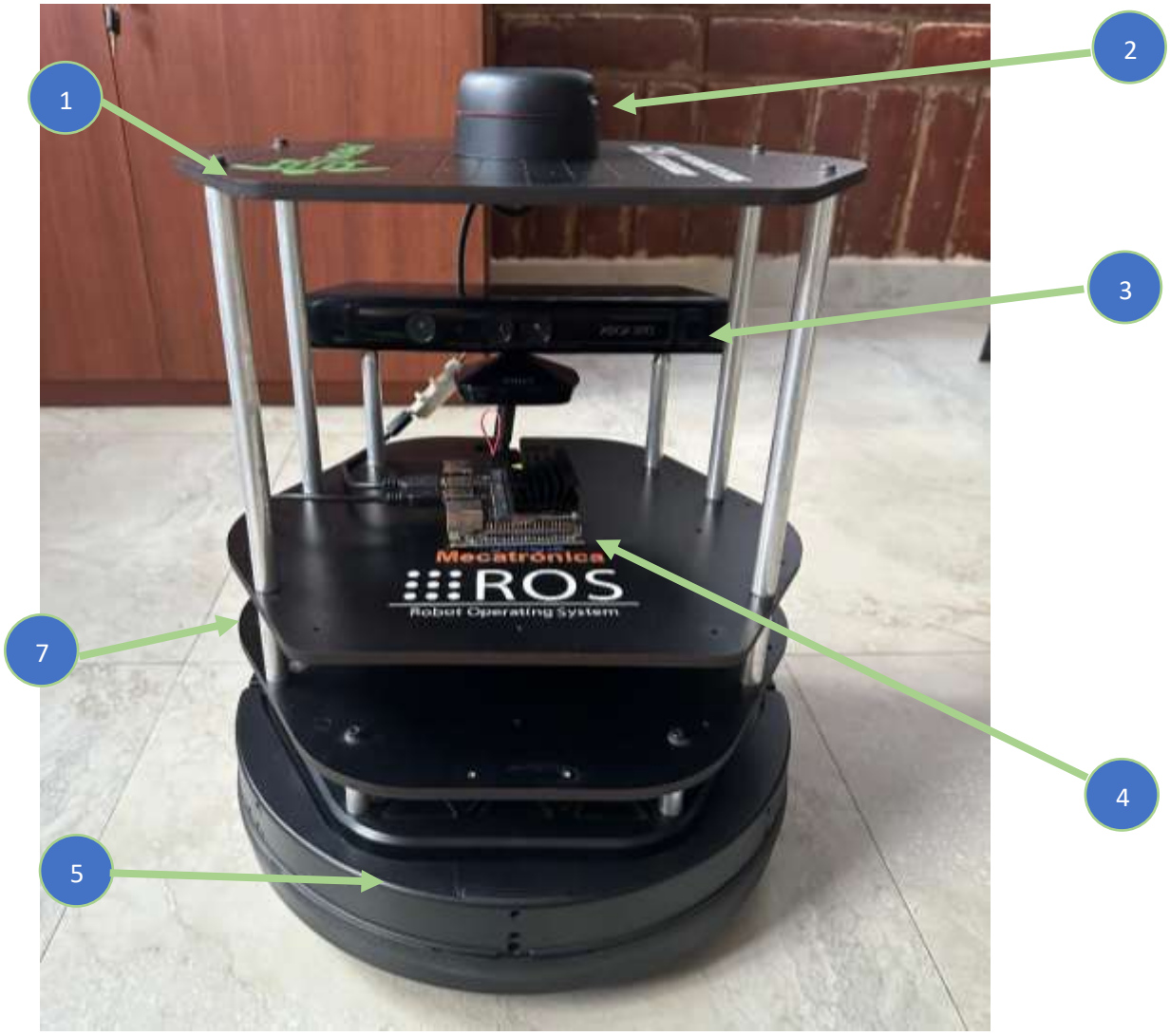


Fig. 4.1 Sistema de localización y mapeo simultáneo (SLAM)

En la Fig. 4.2 se muestra la base Kobuki sobre la que está montada la plataforma robótica Turtlebot2, mediante la cual, a través de USB se conecta a la placa microcontrolada NVIDIA Jetson Nano para la acción de control. De igual manera, la base cuenta con distintas salidas de voltaje y corriente que serán vitales para la energización de los distintos componentes del sistema, como son, la cámara de profundidad Microsoft Kinect v1, el sensor RPLIDAR A2M12 y la placa Jetson Nano.

Fig. 4.2 Base Kobuki de la plataforma Turtlebot2

En la Fig. 4.3 se muestra la parte inferior de la base Kobuki, en la que se encuentra su sistema de locomoción dado por una configuración de tracción diferencial, con dos ruedas acopladas a dos motores de corriente continua cada una, así como dos ruedas pivotantes. También se puede acceder a las baterías de iones de litio integradas a la plataforma robótica Turtlebot2.

Fig. 4.3 Parte inferior de la plataforma Turtlebot2 y baterías integradas

Tabla 4.3: Componentes de la solución propuesta

Nro. de elemento	Parte	Descripción	Función principal
1	**Chasis**	Estructura con plataformas de montaje de material ABS y soportes para organizar y fijar componentes.	Soporta y organiza todos los componentes del robot
2	**LiDAR A2M12**	Sensor laser 360° con un rango de medición de 0.2m a 12m y una resolución angular de 0.225°.	Realiza mapeo y detección de obstáculos en el entorno.
3	**Microsoft Kinect v1**	Sensor de cámara RGB-D que captura el color y profundidad del ambiente.	Permite visión 3D y reconocimiento de los objetos en el ambiente.
4	**NVIDIA Jetson Nano**	Computadora compacta con CPU y GPU integrada y múltiples interfaces de conexión.	Procesa los datos provenientes de los sensores, ejecuta algoritmos y controla el movimiento de la plataforma robótica
5	**Base Kobuki**	Base motorizada con ruedas en configuración de tracción diferencial, sensores de movimiento, encoders y giroscopio integrados.	Proporciona locomoción, odometría y datos de posición de la plataforma robótica
6	**Baterías**	Baterías de iones de litio de 14.8V y 4400mAh.	Proporciona energía a la base Kobuki, así como a los componentes conectados a ella.
7	**Power Bank**	Fuente de alimentación para la placa Jetson Nano y el sensor LiDAR A2M12	Proporciona una alimentación energética adecuada para la placa y el sensor lidar (5V y 4.5A)

4.4 Especificaciones del sistema diseñado

4.4.1 Plataforma robótica Turtlebot2

La plataforma robótica TurtleBot2 es un robot móvil de código abierto diseñado para la investigación, la educación y el desarrollo de aplicaciones de robótica. Es compatible con el Robot Operating System (ROS) y cuenta con una base móvil Kobuki, que proporciona locomoción, sensores de proximidad y de contacto. Además, integra hardware adicional como sensores LiDAR, cámaras y computadoras a bordo, permitiendo realizar tareas avanzadas de navegación, mapeo y percepción. Gracias a su versatilidad y facilidad de uso, el TurtleBot2 es ampliamente utilizado en proyectos académicos y experimentales en robótica.

Fig. 4.4 Plataforma robótica Turtlebot2

4.4.2 Sensor LiDAR A2M12

El sensor LiDAR RPLIDAR A2M12 es un dispositivo de detección láser diseñado para aplicaciones de mapeo y navegación en robótica. Utiliza tecnología de tiempo de vuelo (ToF) para medir distancias a su alrededor, proporcionando un escaneo 2D de alta resolución con un rango de detección de hasta 12 metros. Su diseño compacto y bajo consumo energético lo hacen ideal para plataformas móviles como drones y robots.

Fig. 4.5 RPLIDAR A2M12

4.4.3 Cámara de profundidad Microsoft Kinect V1

La cámara de profundidad Kinect v1 es un dispositivo desarrollado originalmente por Microsoft para el sistema Xbox 360, que ha encontrado amplio uso en la investigación y la robótica. Combina

un sensor RGB con un sensor infrarrojo y un proyector de patrones, lo que le permite capturar imágenes en 3D y detectar profundidades con precisión. Esta capacidad facilita aplicaciones como la detección de personas, la reconstrucción de entornos y el control por gestos

Fig. 4.6 Microsoft Kinect V1

4.4.4 Placa microcontrolada NVIDIA Jetson Nano

La Jetson Nano es una computadora de desarrollo compacta creada por NVIDIA, diseñada para ejecutar aplicaciones de inteligencia artificial y computación de alto rendimiento en dispositivos integrados. Está equipada con una GPU NVIDIA Maxwell de 128 núcleos CUDA, un procesador ARM Cortex-A57 de cuatro núcleos y 4 GB de memoria RAM, lo que la hace ideal para tareas intensivas como visión por computadora, aprendizaje profundo y robótica. Su soporte para bibliotecas y frameworks populares como TensorFlow, PyTorch y ROS, combinado con su bajo consumo energético, la convierte en una solución potente y económica para proyectos de inteligencia artificial en tiempo real.

Fig. 4.7 Placa NVIDIA Jetson Nano

4.4.5 Power bank Sennbeyer

La fuente de alimentación usada para la placa microcontrolada Jetson nano es la Sennbeyer 22.5W la cual ofrece una salida de voltaje óptima para su correcto funcionamiento, proporcionando 5V y 4.5A de corriente, sobrepasando por 0.5A la corriente necesaria para que la placa esté en su máximo rendimiento, lo cual se compensa con la instalación de un ventilador encima del disipador en la placa microcontrolada. La conexión se realiza con un cable tipo barril 5.5mm x 2.1mm a USB-A.

Fig. 4.8 Power bank Sennbeyer 22.5W

4.5 Puesta en marcha del sistema robótico

4.5.1 Prueba de funcionamiento del sensor LiDAR A2M12

Para probar el funcionamiento del sensor LiDAR A2M12, se instaló la biblioteca correspondiente y se compiló el espacio de trabajo (catkin_ws). Posteriormente, se ejecutó el comando:

- sudo chmod 777 /dev/ttyUSB0

Este comando nos permite dar permisos de administrador al puerto USB denominado ttyUSB0, ya que, sin este permiso, el sensor lidar podría no arrancar o dar datos erróneos, posteriormente ejecutamos el comando para arrancar el sensor lidar A2M12

- roslaunch rplidar_ros rplidar_a2m12.launch

Una vez que el sensor esté funcionando y girando, podemos comprobar que los datos del sensor se publican correctamente al nodo maestro de ROS a través del comando:

- rostopic list

Si todo esta correctamente funcionando podremos visualizar que los datos se publican como /scan como se visualiza en la siguiente figura.

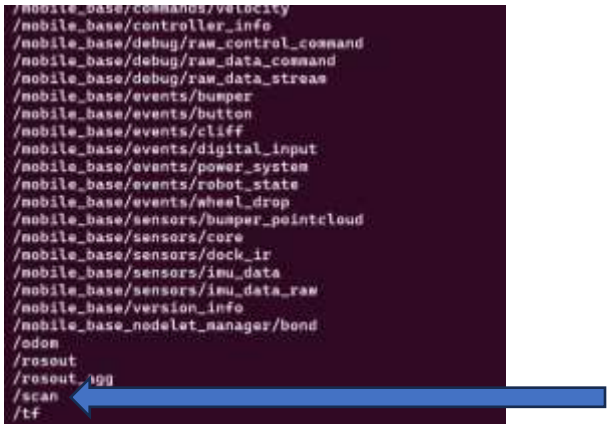


Fig. 4.9 Datos del sensor lidar A2M12 publicados como /scan al nodo maestro ROS

4.5.2 Prueba de funcionamiento de la cámara de profundidad Kinect V1

Para verificar el correcto funcionamiento de la cámara de profundidad Kinect v1 después de la instalación del paquete ROS correspondiente ejecutamos el siguiente comando:

- rosrun image_view image_view image:=/camera/raw

Al hacer esto podremos visualizar en nuestra terminal, la imagen en formato raw proporcionada por el sensor Kinect, lo que indica que está funcionando correctamente como se visualiza en la siguiente figura.

Fig. 4.10 Prueba de funcionamiento de cámara de profundidad Kinect V1

4.5.3 Puesta en marcha del Turtlebot2

Para la puesta en marcha de la plataforma robótica Turtlebot2 haciendo uso de un teclado para el control e integrando los dispositivos tecnológicos antes mencionados se debe seguir los pasos detallados en el diagrama de flujo de la Fig. 4.11. Es importante mencionar que, el primer paso es inicializar "roscore" en una terminal, el cual es el núcleo del sistema operativo de robots ROS y se inician los componentes esenciales necesarios para su correcto funcionamiento. Se debe abrir una nueva terminal, tanto para la confirmación entre la comunicación de la plataforma robótica con la placa microcontrolada, como para la tele operación a través de teclado, ya que todos estos procesos son continuos.

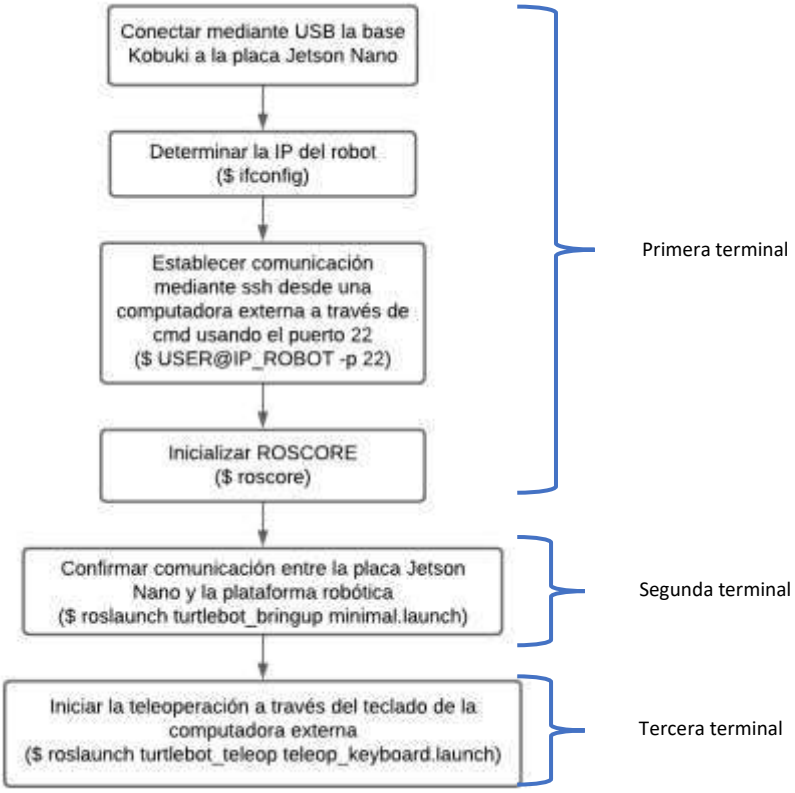


Fig. 4.11 Diagrama de flujo de la secuencia para la puesta en marcha de la plataforma

4.5.4 Diagrama de transformaciones TF

El sistema de transformaciones TF (Transform Frames) en ROS permite mantener una conexión espacial coherente entre la base robótica Kobuki y los sensores utilizados. En el presente proyecto, la correcta configuración del árbol de transformaciones es esencial para que el sistema responda correctamente al mostrar la generación del mapa en tiempo real mientras se ejecuta la navegación autónoma.

En la Fig. 4.12 se muestra la correcta configuración del árbol TF implementado en la plataforma robótica Turtlebot2, mientras que en la Tabla 4.4 describe las transformaciones estáticas y dinámicas de cada frame utilizado.

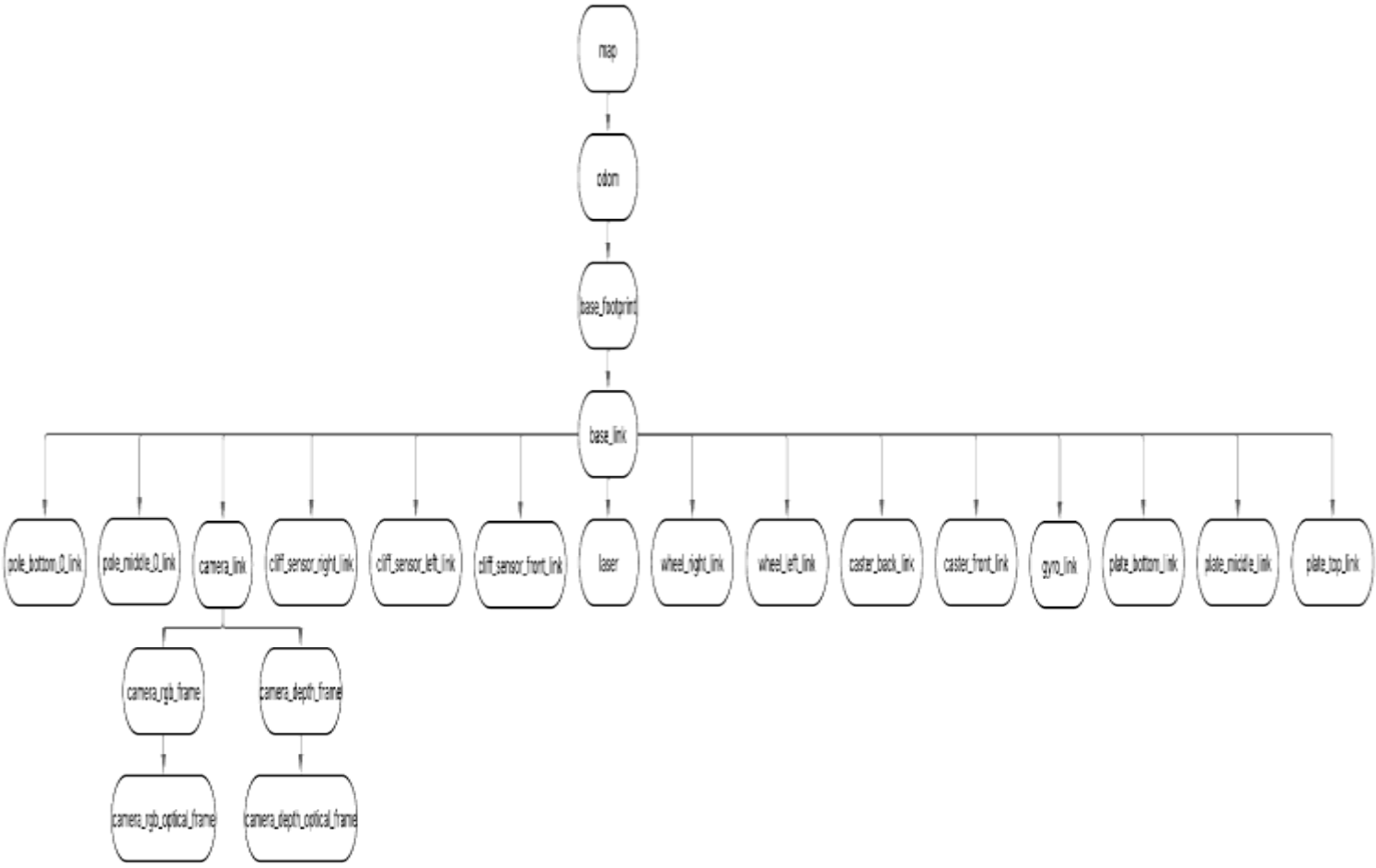


Fig. 4.12 Diagrama de transformaciones del sistema

Tabla 4.4: Descripción de los frames utilizados

Frame origen	Frame destino	Tipo de transformación	Descripción
Map	**Odom**	Dinámica	Corrección de la posición global del robot sobre el mapa
Odom	**Base_footprint**	Dinámica	Publicada al poner en marcha la base Kobuki
Base_footprint	**Base_link**	Estática	Publicada al poner en marcha la base Kobuki
Base_link	**laser**	Estática	Sensor lidar montado sobre la base Kobuki
Base_link	**Camera_link**	Estática	Cámara de profundidad montada sobre la base Kobuki
Camera_link	**Camera_rgb_frame**	Estática	Publicada al poner en marcha la cámara de profundidad
Camera_link	**Camera_depth_frame**	Estática	Publicada al poner en marcha la cámara de profundidad
Camera_rgb_frame	**Camera_rgb_optical_frame**	Estática	Publicada al poner en marcha la cámara de profundidad
Camera_depth_frame	**Camera_depth_optical_frame**	Estática	Publicada al poner en marcha la cámara de profundidad

4.6 Mapeo con SLAM dinámico

En esta sección se detalla el proceso de exploración autónoma en tiempo real implementado mediante los nodos RTAB-Map y explore_lite. El sistema fue diseñado para que el robot construya el mapa de su entorno mientras se desplaza, identificando áreas no exploradas y navegando hacia ellas de forma automática. Esta exploración dinámica se logró al combinar el mapeo visual con los algoritmos de navegación proporcionados por ROS.

RTAB-Map se encarga de la tarea de SLAM, integrando datos provenientes del LiDAR y la cámara Kinect v1 para crear mapas 2D de ocupación. Paralelamente, explore_lite genera objetivos

automáticos que guiaron al robot hacia las nuevas fronteras detectadas en el mapa en proceso de construcción. La coordinación entre ambos nodos facilitó una cobertura eficiente de los entornos seleccionados.

4.6.1 Algoritmo de mapeo

Para el mapeo en tiempo real, se ha empleado y adaptado un algoritmo SLAM del paquete RTAB-Map disponible en ROS Melodic, que se basa en SLAM visual. Este método utiliza imágenes para estimar la posición del robot y construir progresivamente un mapa del entorno. Gracias a la cámara de profundidad Kinect V1, que proporciona simultáneamente imágenes en color y nubes de puntos de profundidad, es posible reconocer características visuales y asociarlas con distancias reales, facilitando así la localización espacial del robot.

En este proyecto, se da prioridad a la nube de puntos generada por la cámara, procesada mediante el tópico '/camera/depth/image_raw' de RTAB-Map. Además, este nodo integra los datos de profundidad de Kinect con la información bidimensional del LiDAR, produciendo un mapa de ocupación 2D preciso. La configuración completa (ver Anexo A) fue optimizada para superar las limitaciones de rendimiento de la Jetson Nano durante la generación del mapa.

El sistema utiliza los tópicos '/scan' (LiDAR), '/camera/depth/image_raw' y '/camera/rgb/image_raw' (Kinect), publicando los mapas en '/grid_map'. Para asegurar compatibilidad con el algoritmo de navegación, este mapa fue remapeado a '/map' mediante 'topic_tools/relay'.

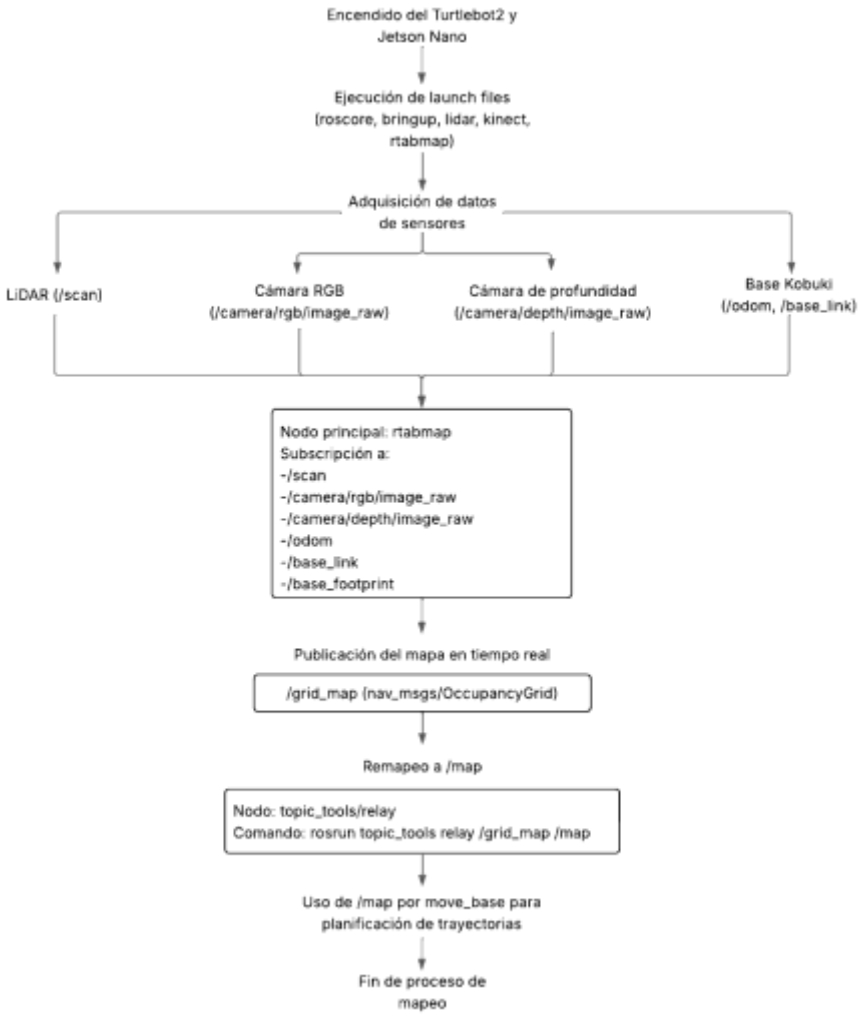


Fig. 4.13 Diagrama de flujo del sistema para mapeo

4.6.2 Integración de explore lite

Explore_lite ha sido integrado al sistema como una herramienta de navegación autónoma basada en fronteras. Este nodo analiza el mapa en construcción y, cuando detecta regiones limítrofes entre zonas mapeadas y desconocidas, genera metas automáticas que se publican en '/move_base/goal'. El nodo trabaja en conjunto con move_base y sus costmaps para calcular caminos seguros hacia dichas metas. (Ver Anexo B para parámetros usados en move_base y Anexo C para la configuración de explore_lite).

Se configuraron parámetros como 'planner_frequency', 'progress_timeout', 'visualize' y 'goal_blacklist' para adaptar el comportamiento del nodo a los tiempos de respuesta del sistema. Gracias a esto, el robot fue capaz de recorrer eficientemente el entorno sin intervención humana.

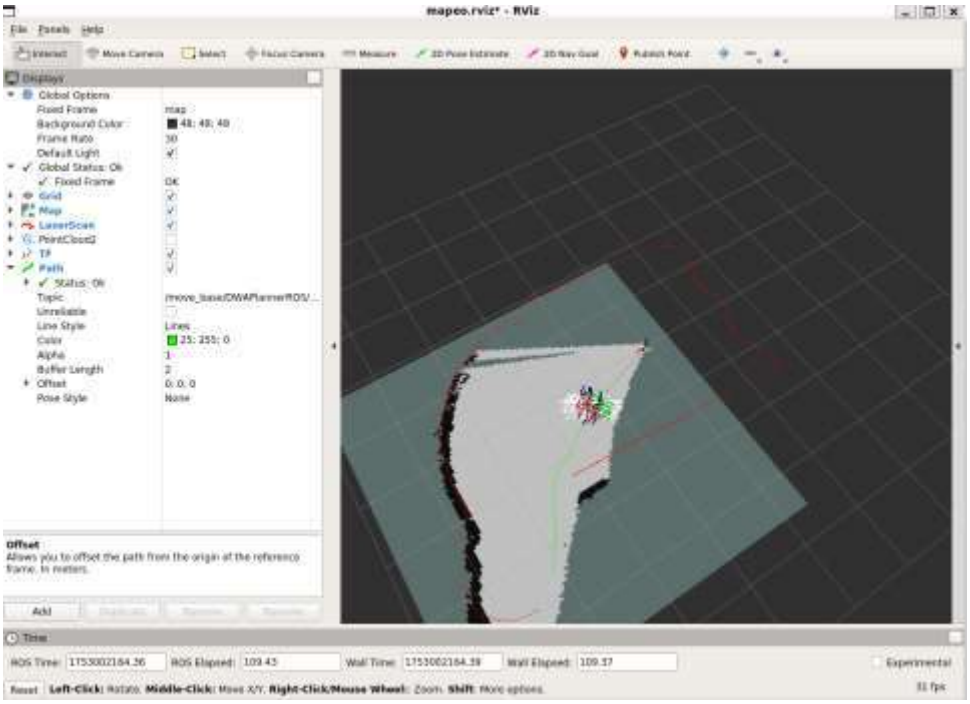


Fig. 4.14 Robot explorando sus fronteras con trayectoria automática (línea verde)

4.7 Escenarios experimentales

4.7.1 Escenario 1: Entorno simple sin obstáculos

El primer escenario de pruebas consistió en un entorno sencillo y libre de obstáculos móviles. El objetivo fue validar el funcionamiento básico del sistema SLAM dinámico y su capacidad para mapear un entorno sin complicaciones geométricas. El robot inició desde un punto fijo y fue generando el mapa a medida que se desplazaba automáticamente.

Durante esta prueba, el robot logró cubrir todo el entorno sin interrupciones y generó un mapa uniforme. Este escenario sirvió como referencia base para contrastar con entornos más complejos.

Fig. 4.15 Posición inicial del robot en el entorno 1

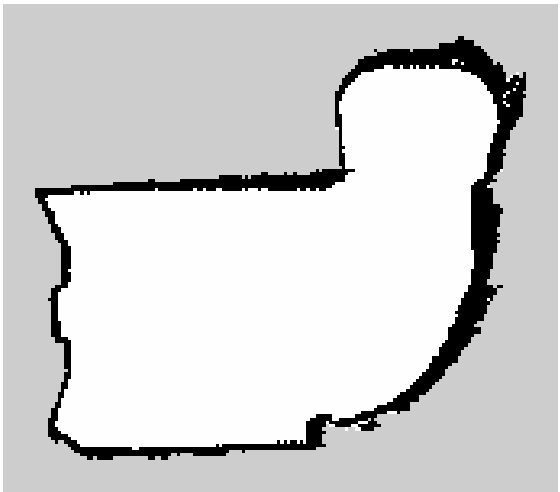


Fig. 4.16 Mapa final generado del entorno 1

4.7.2 Escenario 2: Entorno con pasillo y habitaciones

El segundo escenario incluyó una configuración con pasillos y habitaciones. Esto permitió observar el comportamiento del sistema ante decisiones de navegación complejas, como giros cerrados o zonas sin visibilidad directa. Explore_lite logró generar metas estratégicas que dirigieron al robot por cada sección del entorno.

Aunque el tiempo total de mapeo aumentó ligeramente, el robot fue capaz de recorrer y mapear toda el área, demostrando buena capacidad para adaptarse a bifurcaciones y estrechamientos.

Fig. 4.17 Robot en exploración del entorno 2

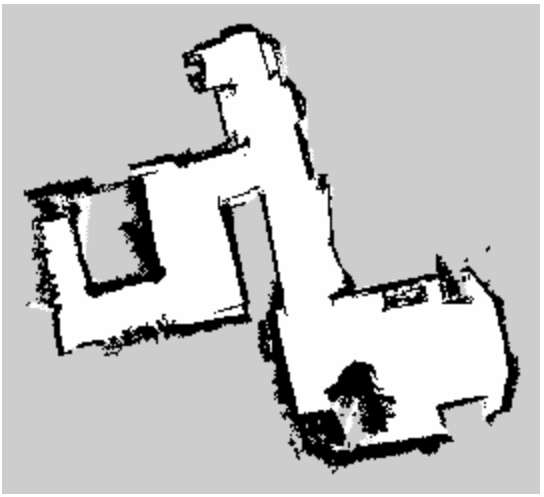


Fig. 4.18 Mapa final generado del entorno 2

4.7.3 Escenario 3: Entorno con obstáculos añadidos

El tercer escenario se enfoca en probar la robustez del sistema frente a obstáculos nuevos introducidos durante la navegación. Estos objetos no estaban presentes en el mapa original y fueron colocados de forma deliberada mientras el robot se desplazaba. La idea fue observar si el sistema era capaz de evitar colisiones en tiempo real.

El robot, gracias al uso del LiDAR en los costmaps locales, fue capaz de identificar estos obstáculos y ajustar su trayectoria en consecuencia. En algunos casos se notó una leve distorsión en el mapa debido al movimiento brusco, pero el sistema se recuperó adecuadamente.

Fig. 4.19 Incorporación de obstáculos durante la exploración del entorno 3

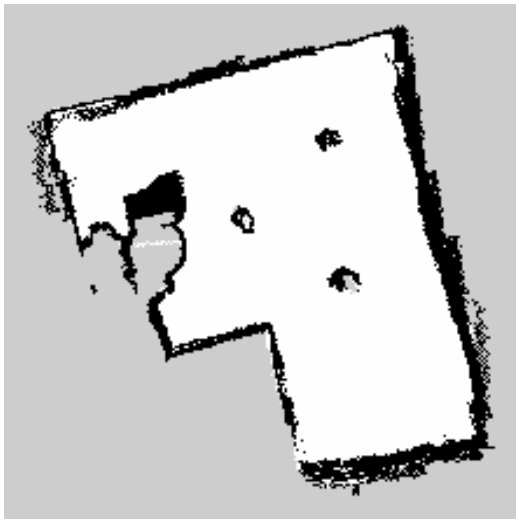


Fig. 4.20 Mapa final generado del entorno 3

4.8 Navegación con AMCL sobre los mapas guardados

4.8.1 Proceso de guardado del mapa generado

Una vez finalizado el proceso de exploración y mapeo autónomo mediante los nodos RTAB-Map y explore_lite, fue necesario almacenar el mapa generado para su posterior uso en tareas de

navegación con localización basada en AMCL. Para ello, se utilizó un comando personalizado desde la consola mediante el uso del paquete map_server:

- rosrun map_server map_saver -f ~/catkin_ws/src/turtlebot2/mapas/ejemplo.

Este comando guarda el mapa con el nombre ejemplo.pgm y el archivo ejemplo.yaml en la ruta especificada, siempre que el tópico '/map' esté siendo publicado correctamente por RTAB-Map. Es importante asegurarse de que el robot haya recorrido completamente el entorno antes de guardar el mapa, ya que este reflejará la información capturada en ese momento.

4.8.2 Proceso de carga del mapa

Una vez que los mapas han sido generados y guardados mediante RTAB-Map, se utilizó el nodo map_server para cargar los archivos .pgm y .yaml correspondientes. Esta funcionalidad permitió ejecutar sesiones de navegación sobre mapas previamente creados, sin requerir nuevos escaneos. (Ver Anexo D para la configuración del nodo AMCL). Para ello se usó el siguiente comando personalizado:

- roslaunch turtlebot2 navegacion_amcl.launch map_file:=/home/jetson/catkin_ws/src/turtlebot2/mapas/ejemplo.yaml

Para la correcta visualización y uso del mapa, en el archivo global_costmap_params.yaml se activó el parámetro 'static_map: true'. De esta manera, move_base utilizó el mapa cargado como referencia para navegación.

4.8.3 Precisión de la localización

El nodo amcl fue responsable de la localización del robot sobre el mapa cargado. Este nodo utiliza un filtro de partículas y los datos del LiDAR para estimar la posición del robot en el espacio. La precisión fue verificada mediante la visualización del tópico '/amcl_pose', el cual mostró una

estimación coherente durante el desplazamiento. (Ver Anexo E y Anexo F para los parámetros de los costmaps global y local usados en navegación).

A pesar de pequeñas variaciones cuando el robot giraba bruscamente, la localización se mantuvo dentro de márgenes aceptables y permitió que el robot alcanzara con éxito los objetivos asignados.

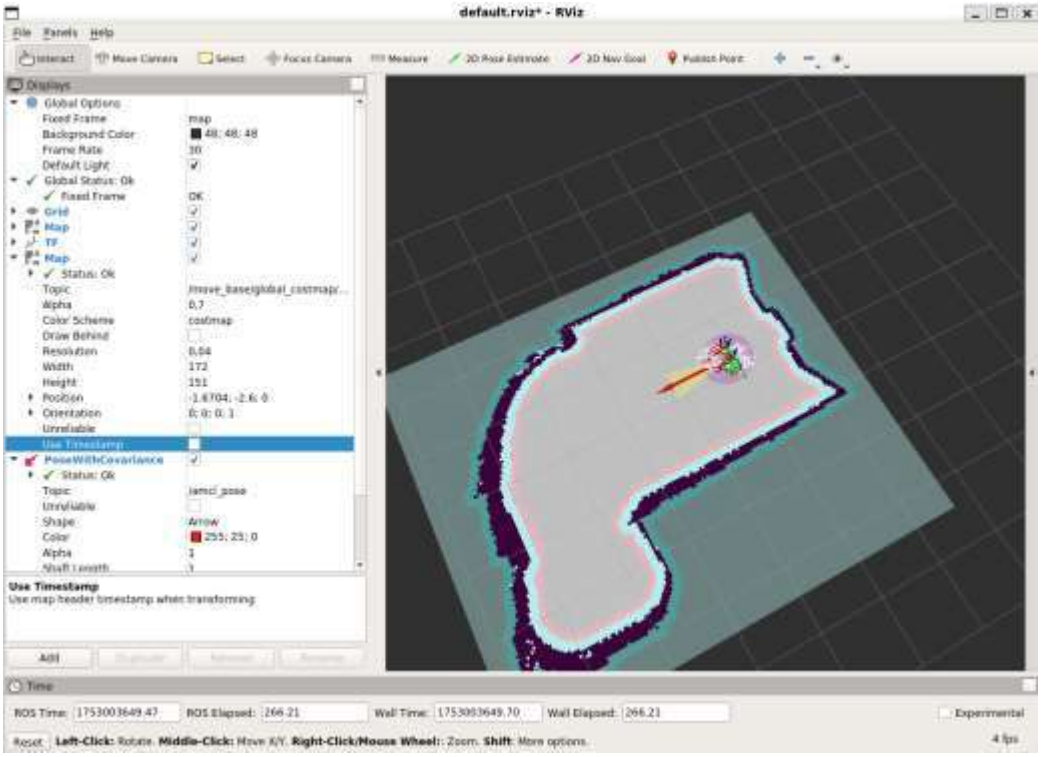


Fig. 4.21 Posición inicial estimada por amcl

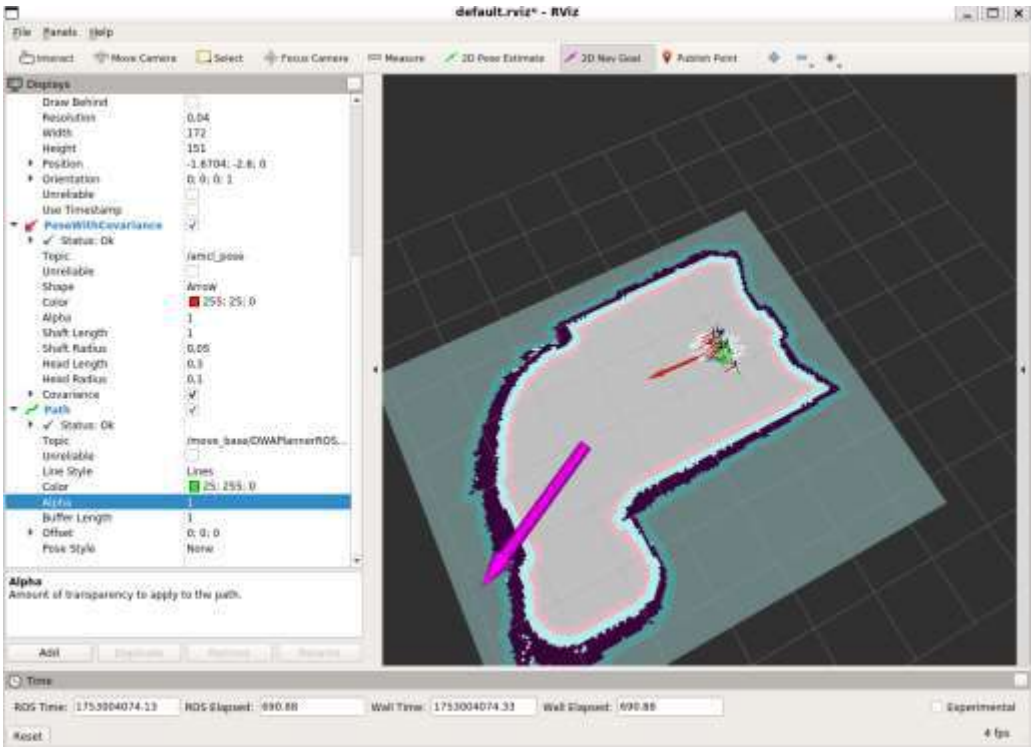


Fig. 4.22 Establecimiento del objetivo de navegación

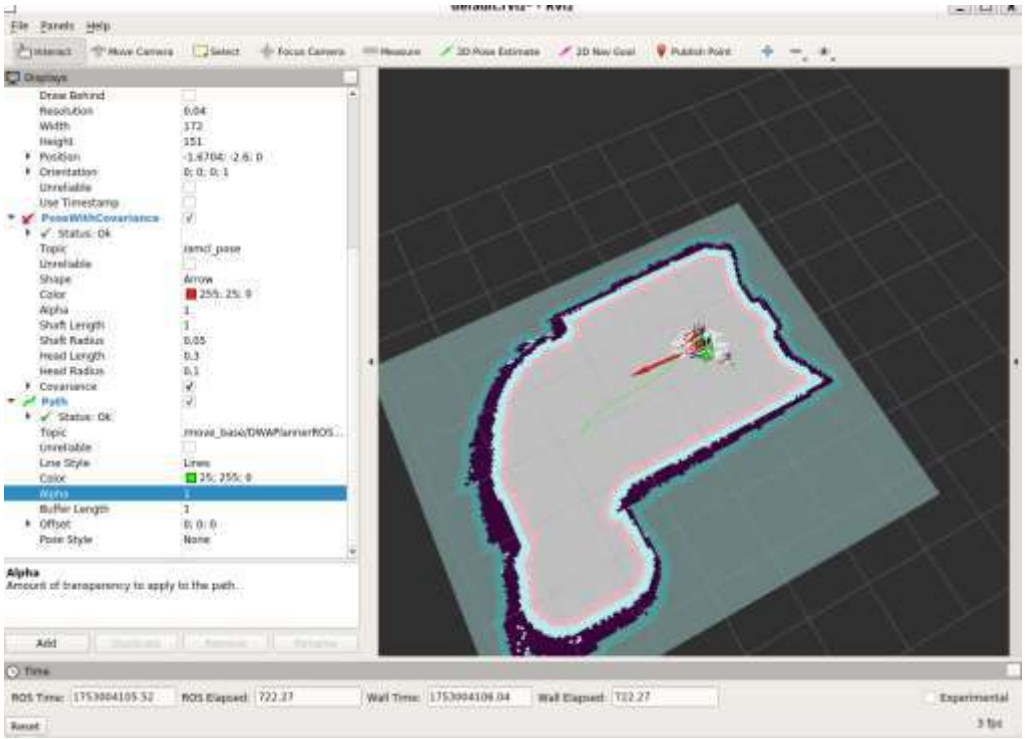


Fig. 4.23 Trayectoria generada hacia el objetivo

4.8.4 Evasión de obstáculos dinámicos

Durante la navegación sobre un mapa guardado previamente, se ha evaluado la capacidad del sistema para identificar y evitar obstáculos imprevistos que no formaban parte del mapeo inicial. Para ello, se colocó un objeto físico entre el robot y el objetivo definido mediante la herramienta 2D Nav Goal en RViz. Al inicio, el planificador global generó una trayectoria directa hacia la meta, ya que el obstáculo no había sido detectado aún por el robot.

A medida que el TurtleBot2 se acerca al obstáculo, el costmap local, actualizado con datos del LiDAR, detecta el nuevo objeto. Esto hizo que el DWAPlannerROS recalculara dinámicamente una ruta alternativa, desviando ligeramente el robot hacia un costado (la configuración detallada del planificador se encuentra en el Anexo G). Posteriormente, se crea una nueva trayectoria local que rodea completamente el obstáculo y se reconecta con el destino original, logrando una navegación segura y fluida.

Este comportamiento se refleja claramente en las capturas de pantalla de Rviz donde, inicialmente se observa la trayectoria recta, luego una desviación ligera y finalmente la ruta alternativa que evita el obstáculo antes de llegar al destino. Estas pruebas confirman la correcta integración de move_base con DWAPlannerROS y la efectividad del costmap local para adaptarse a obstáculos dinámicos o inesperados.

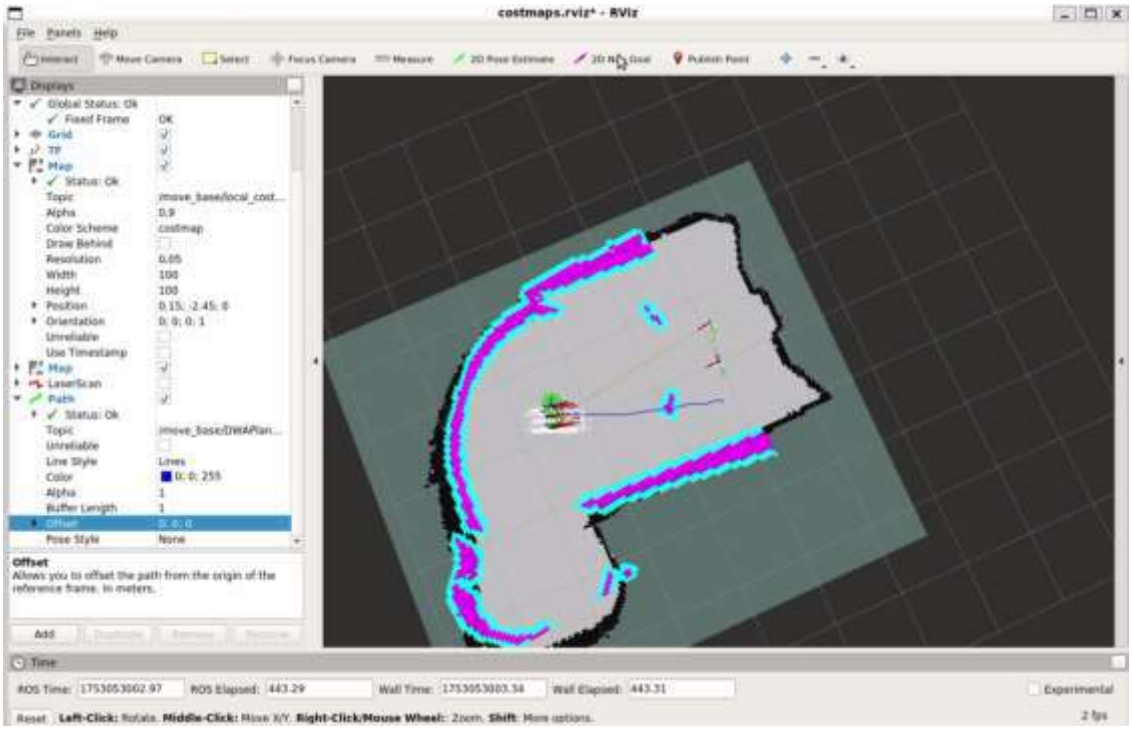


Fig. 4.24 Trayectoria global inicial

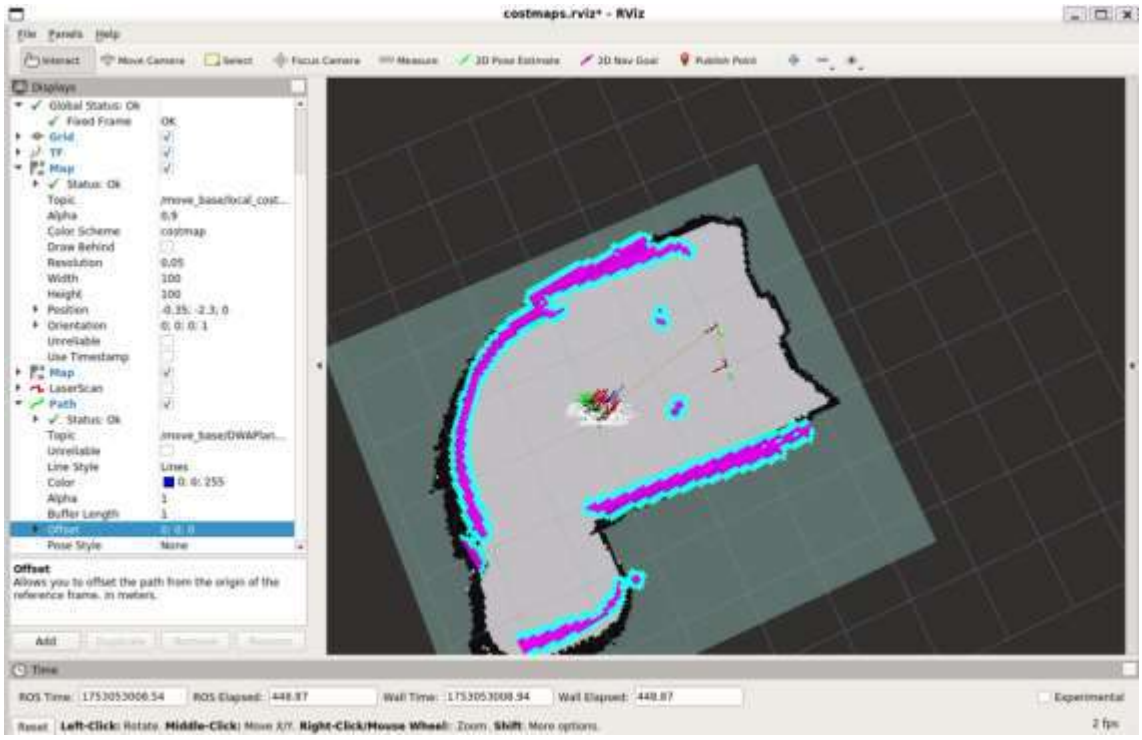


Fig. 4.25 Desvío inicial para rodear el obstáculo

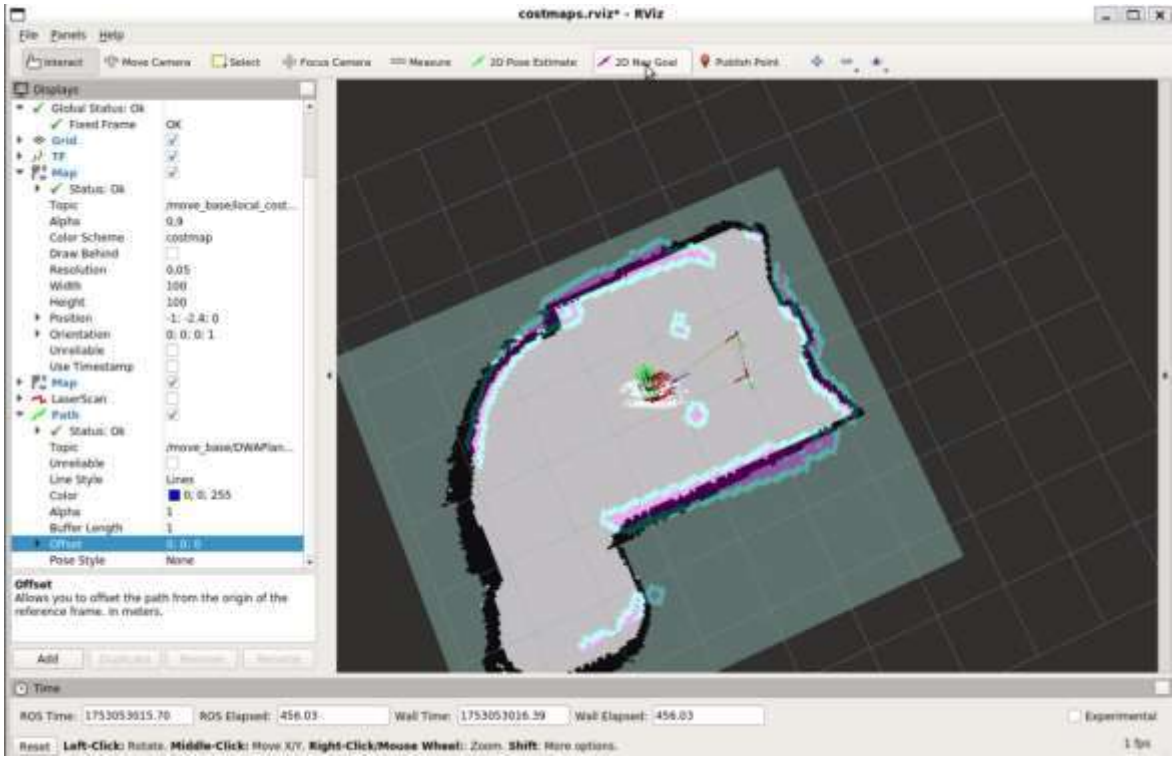


Fig. 4.26 Segundo desvío de trayectoria

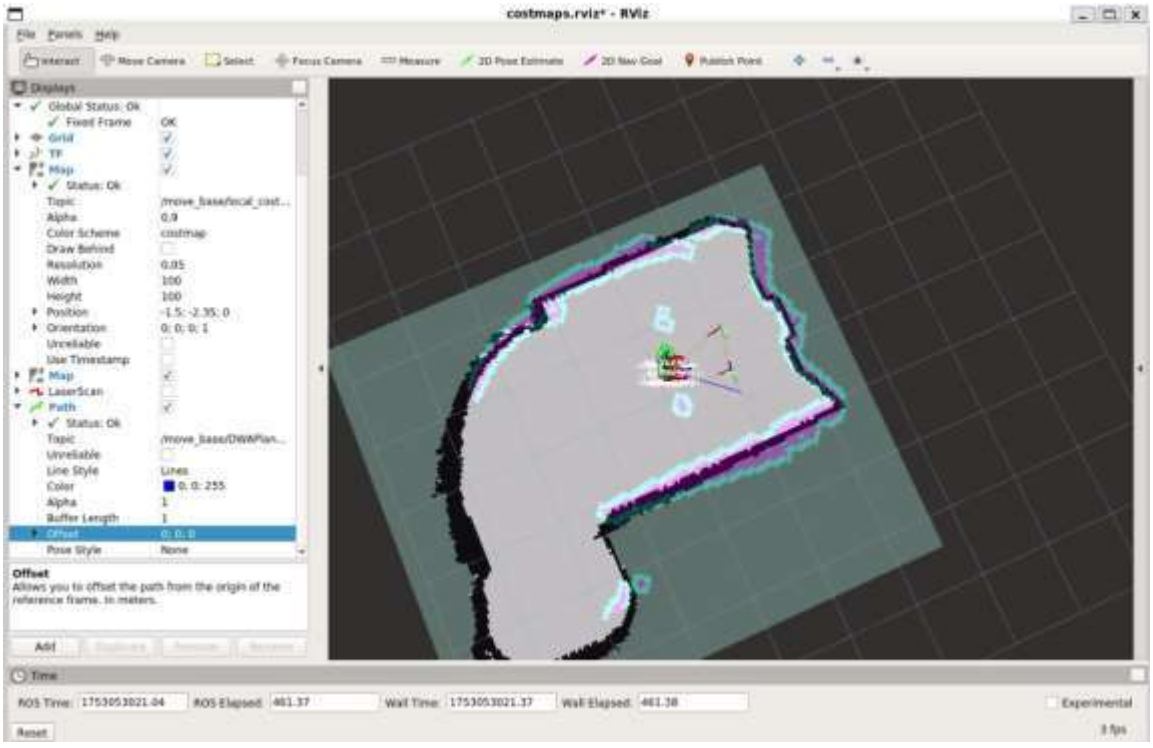


Fig. 4.27 Ruta final ajustada para alcanzar el objetivo inicial

4.9 Análisis de resultados

El análisis de los resultados obtenidos en los tres escenarios experimentales ha permitido evaluar el desempeño del sistema robótico en tareas de mapeo y navegación autónoma. Se confirma que la integración de RTAB-Map con explore_lite facilitó una exploración eficiente en entornos dinámicos, permitiendo la generación de mapas en tiempo real a partir de sensores como LiDAR y la cámara Kinect. Posteriormente, al utilizar AMCL junto con move_base, se alcanzó una navegación precisa sobre el mapa guardado, incluso frente a obstáculos nuevos no presentes durante el mapeo inicial.

En términos generales, el robot mostró un comportamiento robusto en los tres escenarios, con buena detección de fronteras y generación de trayectorias factibles. En el escenario con obstáculos dinámicos, el sistema de localización y los costmaps adaptativos demostraron ser efectivos para evitar objetos inesperados sin desviarse del objetivo. Las diferencias en tiempo de mapeo, distorsión del mapa y cobertura se detallan a continuación en la Tabla 4.5, que resume las métricas más relevantes obtenidas en cada prueba.

Tabla 4.5: Comparativa de métricas en los distintos escenarios experimentales

Métrica	Escenario 1: Entorno simple	Escenario 2: Pasillo y habitaciones	Escenario 3: Obstáculos dinámicos
Área total mapeada (m^2)	8	24	20
Tiempo de mapeo (min)	3	6	5
Fronteras detectadas al inicio	3	2	3
Porcentaje de cobertura del entorno	100%	95%	98%
Evasión de obstáculos nuevos	Exitosa	No aplica	No aplica
Distorsión del mapa	Ninguna	Leve	Mínima en giros bruscos
Precisión de localización	Alta	Alta	Media-alta

CONCLUSIONES

- El sistema desarrollado integró algoritmos existentes como RTAB-Map, AMCL y DWAPlannerROS en una arquitectura funcional para SLAM dinámico, capaz de mapear y navegar en tiempo real con evasión de obstáculos. Gracias a su configuración específica, el robot logró explorar entornos desconocidos de forma autónoma, combinando sensores Kinect y LiDAR para una mayor precisión. Esto valida el desarrollo de un algoritmo eficiente basado en la integración y parametrización de módulos existentes, adaptado a una plataforma de cómputo limitada como la Jetson Nano.
- El análisis de diferentes técnicas de estimación y localización, mediante el uso de una cámara de profundidad y sensores LiDAR, evidenció que la integración de múltiples fuentes de datos optimiza la precisión del sistema. Se logró adaptar el algoritmo a las condiciones de un entorno no estructurado, permitiendo una navegación más estable y eficiente, lo que confirma la importancia de seleccionar e implementar tecnologías adecuadas para mejorar los sistemas SLAM en escenarios complejos.
- Las pruebas realizadas en entornos reales permitieron validar la efectividad del sistema desarrollado, demostrando que el algoritmo propuesto es capaz de ajustarse a variaciones en la escena y responder de manera eficiente a la presencia de obstáculos móviles. Si bien se observaron limitaciones en la detección de cambios abruptos en el entorno, los resultados obtenidos evidencian el potencial de esta tecnología para ser aplicada en diversos campos, como la automatización industrial y la robótica de servicio.

RECOMENDACIONES

- Para futuros desarrollos, se recomienda la implementación de técnicas de inteligencia artificial, como redes neuronales y aprendizaje profundo, para mejorar la precisión del algoritmo SLAM en entornos altamente dinámicos. La incorporación de estos enfoques permitirá una mejor adaptación a cambios en el entorno y una mayor estabilidad en la localización, optimizando la autonomía de los robots móviles en aplicaciones reales.
- Se sugiere la exploración y prueba de nuevas configuraciones de sensores, combinando tecnologías como LiDAR de mayor resolución, cámaras estereoscópicas y sensores inerciales, con el fin de mejorar la recolección de datos y la precisión del mapeo. Esto permitirá desarrollar sistemas más robustos que puedan operar en condiciones adversas y con menor margen de error, facilitando su aplicación en sectores industriales y de servicio.
- Es recomendable fomentar la colaboración entre instituciones académicas, sector privado y organismos gubernamentales del Ecuador para impulsar el desarrollo de sistemas autónomos basados en SLAM. La creación de proyectos interdisciplinarios y el fortalecimiento de laboratorios de investigación en robótica móvil permitirán la transferencia de conocimiento y el crecimiento de una comunidad científica enfocada en la automatización y la inteligencia artificial aplicada a la movilidad autónoma.

REFERENCIAS

[1] L. A. Góngora Velandia, “Localización y mapeo simultáneo basado en la librería de nube de puntos [PCL]: etapa de registro,” Universidad Piloto de Colombia, Bogotá D.C., 2015.

[2] Álvarez N and Segura M, “Implementación de Algoritmos Filtro de Kalman y Filtro de Partículas para Localización y Mapeo Simultáneo Aplicado a un Robot Móvil en Ambientes Interiores con Variaciones de Iluminación,” UNIVERSIDAD SANTO TOMÁS, Bogotá, 2020.

[3] M. GhaemiDizaji, C. Dadkhah, and H. Leung, “Efficient robot localization and SLAM algorithms using Opposition based High Dimensional optimization Algorithm,” *Eng Appl Artif Intell*, vol. 104, p. 104308, Sep. 2021, doi: 10.1016/j.engappai.2021.104308.

[4] I. Belkin, A. Abramenko, and D. Yudin, “Real-Time Lidar-based Localization of Mobile Ground Robot,” *Procedia Comput Sci*, vol. 186, pp. 440–448, 2021, doi: 10.1016/j.procs.2021.04.164.

[5] Z. Hu, H. Fang, R. Zhong, S. Wei, B. Xu, and L. Dou, “GMP-SLAM: A real-time RGB-D SLAM in Dynamic Environments using GPU Dynamic Points Detection Method,” *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 5033–5040, 2023, doi: 10.1016/j.ifacol.2023.10.1282.

[6] M. Chghaf, S. R. Flórez, and A. El Ouardi, “A multimodal loop closure fusion for autonomous vehicles SLAM,” *Rob Auton Syst*, vol. 165, p. 104446, Jul. 2023, doi: 10.1016/j.robot.2023.104446.

[7] H. Yang, J. Yuan, Y. Gao, X. Sun, and X. Zhang, “UPLP-SLAM: Unified point-line-plane feature fusion for RGB-D visual SLAM,” *Information Fusion*, vol. 96, pp. 51–65, Aug. 2023, doi: 10.1016/j.inffus.2023.03.006.

[8] N. Cisneros and G. Perugachi, “Mapeo y localización simultáneos (SLAM) en 3D mediante un sensor láser,” Escuela Politécnica Nacional, Quito , 2014.

[9] V. Narváez and F. Yandún, “Diseño e implementación de un sistema de localización y mapeo simultáneo (SLAM) para la plataforma robótica Robotino®,” Escuela Politécnica Nacional, Quito, 2013.

[10] E. Santillán, “Estudio, control e implementación de sistemas robóticos avanzados : Aplicación de algoritmos iterativos de planificación de rutas (path planning) en un sistema multi-agente dentro del ambiente de ROS Gazebo.,” Escuela Politécnica Nacional, Quito, 2022.

[11] J. Alvarado, “Exploración de fronteras para robots móviles en interiores,” Universidad Técnica del Norte, Ibarra, 2021.

[12] J. Quimbia, “Análisis comparativo de diferentes tecnologías de sensores para Slam sobre robots móviles,” Universidad Técnica del Norte, Ibarra, 2023.

[13] V. R. Barrientos, J. R. García Sánchez, and R. Silva Ortigoza, “Robots Móviles: Evolución y Estado del Arte,” *Polibits*, vol. 35, pp. 12–17, Jan. 2007, doi: 10.17562/PB-35-3.

[14] S. S. Ge and F. Lewis, *Autonomous Mobile Robots: Sensing, Control, Decision. Making and applications*, Taylor & Francis. Manchester, United Kingdom: University of Manchester Institute of Science and Technology, 2006.

[15] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. London, England: Massachusetts Institute of Technology, 2004.

[16] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, Second edition. New York: CAMBRIDGE UNIVERSITY PRESS, 2010.

[17] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Stanford: Stanford University, 2008.

[18] M. Mataric, "The Robotics Primer," *Genet Program Evolvable Mach*, vol. 9, no. 1, pp. 101–103, Mar. 2008, doi: 10.1007/s10710-007-9053-7.

[19] E. Mihret, "Robotics and Artificial Intelligence," *International Journal of Artificial Intelligence and Machine Learning*, vol. 10, no. 2, 2020.

[20] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2022.

[21] S. Thrun, W. Burgard, and D. Fox, *PROBABILISTIC ROBOTICS*. The MIT Press, 2005.

[22] C. Cadena *et al.*, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE TRANSACTIONS ON ROBOTICS*, vol. 32, Dec. 2016.

[23] J.-A. Fernández-Madrigal and J. L. Blanco Claraco, *Simultaneous Localization and Mapping for Mobile Robots*. IGI Global, 2013. doi: 10.4018/978-1-4666-2104-6.

[24] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robot Autom Mag*, vol. 13, no. 2, pp. 99–110, Jun. 2006, doi: 10.1109/MRA.2006.1638022.

[25] H. Choset *et al.*, *Principles of Robot Motion*. London, England: The MIT Press: Cambridge, Massachusetts, 2005.

[26] R. R. Murphy, *INTRODUCTION TO AI ROBOTICS*. London, England: The MIT Press: Cambridge, Massachusetts, 2000.

[27] D. Pérez, *SENSORES DE DISTANCIA POR ULTRASONIDOS*. 2019.

[28] P. Corke, *Robotics, Vision and Control*, vol. 73. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-20144-8.

[29] H. Fredriksson, *Navigation Sensors for Mobile Robots*. Luleå, Sweden: Luleå University of Technology: Department of Computer Science and Electrical Engineering EISLAB, 2007.

[30] K. Di, W. Wan, H. Zhao, Z. Liu, and R. Wang, "Progress and Applications of Visual SLAM," *Journal of Geodesy and Geoinformation*, 2019.

[31] S. Monk, *Programming the Raspberry Pi: Getting Started with Python*. 2016.

[32] M. Banzi and M. Shiloh, *Make: Getting Started with Arduino*, 3rd ed. MAKERMEDIA SEBASTOPOL, CA, 2015.

[33] J.-L. Ossohou, "NVIDIA Jetson Xavier AGX as multimedia broadcast system," Université de Liège, Liège, Belgique, 2022.

[34] Open Source Robotics Foundation, "ROS Wiki," wiki.ros.org. Disponible en https://wiki.ros.org/.

[35] H. Martínez Ruíz, *Metodología de la investigación*. México, D.F.: Cengage Learning, 2012.

[36] L. Blaxter, C. Hughes, and M. Tight, *Cómo se hace una investigación*. Editorial Gedisa, 1996.

[37] S. Ackerman, *Metodología de la investigación*. México D.F.: Ediciones del Aula Taller, 2013.

ANEXOS

Anexo A: Configuraciones y parámetros de RTAB-Map

```
<launch>

    <!-- ================  RTAB-Map SLAM Dinámico ============== -->

    <node pkg="rtabmap_ros" type="rtabmap" name="rtabmap" output="screen" clear_params="true">

                                <!-- BASE DE DATOS -->
    <param name="rtabmap/DatabasePath" value="~/.ros/rtabmap.db"/>

    <!-- ============= CONFIGURACIÓN DE CÁMARA KINECT ============== -->

    <param name="RGBD/Decimation" value="4"/>
    <param name="RGBD/OptimizeSlam2D" value="true"/>
    <param name="RGBD/Enabled" value="true"/>
    <param name="RGBD/MaxDepth" value="3.0"/>
    <param name="RGBD/MinDepth" value="0.2"/>

    <param name="Vis/MinInliers" value="10"/>
    <param name="RGBD/LoopClosureInlierDistance" value="0.15"/>
    <param name="Kp/DetectorStrategy" value="6"/>
    <param name="Grid/MaxHeight" value="0.4"/>
    <param name="Grid/MinHeight" value="-0.3"/>

    <!-- Parámetros para uniformidad y filtrado del mapa (Kinect + LiDAR) -->

    <param name="Grid/CellSize" value="0.04"/>
    <param name="Grid/MinClusterSize" value="5"/>
    <param name="Grid/NoiseFilteringRadius" value="0.15"/>
    <param name="Grid/NoiseFilteringMinNeighbors" value="2"/>
    <param name="Grid/RayTracing" value="true"/>
    <param name="Grid/VoxelSize" value="0.04"/>
    <param name="Grid/RangeMax" value="5.0"/>
    <param name="Grid/MaxObstacleHeight" value="0.95"/>
    <param name="Grid/OccupancyThr" value="0.25"/>
    <param name="Grid/ClusterRadius" value="0.10"/>

    <!-- ================== ESCANER LIDAR ================== -->
    <param name="subscribe_scan" value="true"/>
    <param name="scan_topic" value="/scan"/>
    <param name="Grid/FromScan" value="true"/>
    <param name="Grid/FromDepth" value="true"/>
    <param name="Grid/3D" value="false"/>

    <!-- ================== PUBLICACIONES ================== -->
    <param name="publish_tf" value="true"/>
    <param name="tf_map_to_odom" value="true"/>
    <param name="publish_map" value="true"/>

    <param name="grid_map" value="true"/>
    <param name="map_topic" value="/map"/>

    <param name="RGBD/LoopClosureHypotheses" value="1"/>
    <param name="RGBD/ProximityBySpace" value="false"/>

    <!-- ================== FRAMES ================== -->
    <param name="frame_id" value="base_link"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="map_frame_id" value="map"/>

    <!-- ================== OPTIMIZACIÓN Y MEMORIA ================== -->
    <param name="Reg/Force3DoF" value="true"/>
    <param name="Reg/Strategy" value="2"/>
    <param name="Optimizer/Strategy" value="2"/>
    <param name="Mem/IncrementalMemory" value="true"/>

    <!-- ================== REMAPS ================== -->
    <remap from="/rgb/image" to="/camera/rgb/image_raw"/>
    <remap from="/depth/image" to="/camera/depth/image_raw"/>
    <remap from="/rgb/camera_info" to="/camera/rgb/camera_info"/>

  </node>

</launch>
```

Anexo B: Configuración del nodo move_base para navegación

```
<launch>
  <node pkg="move_base" type="move_base" name="move_base" output="screen">

    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS"/>
    <param name="controller_frequency" value="3.0"/>
    <param name="planner_frequency" value="0.5"/>
    <param name="transform_tolerance" value="0.8"/>
    <param name="planner_patience" value="3.0"/>
    <param name="controller_patience" value="2.0"/>

    <param name="conservative_reset_dist" value="0.5"/>
    <param name="recovery_behavior_enabled" value="true"/>
    <param name="clearing_rotation_allowed" value="true"/>
    <param name="oscillation_timeout" value="5.0"/>
    <param name="oscillation_distance" value="0.1"/>

    <!-- Cargar los parámetros del Global Costmap -->
    <rosparam file="$(find turtlebot2)/config/move_base/global_costmap_params.yaml" command="load"/>

    <!-- Cargar los parámetros del Local Costmap -->
    <rosparam file="$(find turtlebot2)/config/move_base/local_costmap_params.yaml" command="load"/>

    <!-- Cargar los parámetros del Planner Local -->
    <rosparam file="$(find turtlebot2)/config/move_base/base_local_planner_params.yaml" command="load"/>

    <!-- Remapear el comando de velocidad al topic de la Kobuki -->
    <remap from="/cmd_vel" to="/mobile_base/commands/velocity"/>
    <remap from="scan" to="/scan"/>
  </node>
</launch>
```

Anexo C: Configuración del nodo explore_lite para planificación de trayectorias

```
<launch>
  <node pkg="explore_lite" type="explore" name="explore" output="screen">
    <param name="explore_costmap/robot_base_frame" value="base_link"/>
    <param name="explore_costmap/global_frame" value="map"/>

    <param name="planner_frequency" value="1.0"/>
    <param name="explore_rate" value="1.0"/>

    <param name="goal_distance_tolerance" value="0.5"/>
    <param name="progress_timeout" value="20.0"/>
    <param name="goal_blacklist" value="true"/>
    <param name="visualize" value="true"/>

    <param name="potential_scale" value="1.0"/>
    <param name="orientation_scale" value="0.0"/>
    <param name="gain_scale" value="1.0"/>

    <remap from="costmap" to="move_base/global_costmap/costmap"/>

  </node>
</launch>
```

Anexo D: Configuración del nodo AMCL para localización

```
<launch>
  <!-- Argumento para el archivo de mapa -->
  <arg name="map_file" default="$(find turtlebot2)/mapas/mapa_final.yaml" />

  <!-- Nodo de map server para cargar mapa -->
  <node pkg="map_server" type="map_server" name="map_server" args="$(arg map_file)">
  <param name="frame_id" value="map"/>
  <param name="map_update_interval" value="0.0"/>    <!-- Desactiva publicación continua -->
  </node>

  <!-- Parámetros para nodo AMCL  -->
  <node pkg="amcl" type="amcl" name="amcl" output="screen">

            <!-- Frames  -->
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_link"/>
    <param name="global_frame_id" value="map"/>

        <!-- Tópico del LiDAR -->
    <param name="scan_topic" value="/scan"/>

        <!-- Parámetros para Jetson Nano -->
    <param name="min_particles" value="300"/>
    <param name="max_particles" value="1000"/>
    <param name="update_min_d" value="0.01"/>
    <param name="update_min_a" value="0.01"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>

    <param name="odom_model_type" value="diff"/>
    <param name="resample_interval" value="1"/>
    <param name="transform_tolerance" value="0.2"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
    <param name="laser_max_beams" value="30"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>

    <param name="tf_broadcast" value="true"/>
  </node>

  <include file="$(find turtlebot2)/launch/move_base.launch" />
</launch>
```

Anexo E: Configuración de parámetros del costmap global

```
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 1.0
  publish_frequency: 1.0
  width: 30.0
  height: 30.0
  static_map: true
  resolution: 0.05
  transform_tolerance: 0.8
  robot_radius: 0.15
  plugins:
    - {name: static_layer, type: "costmap_2d::StaticLayer"}
    - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
  static_layer:
    enabled: true
    map_topic: /map
    track_unknown_space: true
    use_maximum: false
  obstacle_layer:
    enabled: false
    debug: true
    observation_sources: laser_scan
    laser_scan:
      topic: /scan
      sensor_frame: laser
      data_type: LaserScan
      marking: true
      clearing: true
      obstacle_range: 2.0
      raytrace_range: 3.5
      min_obstacle_height: 0.0
      max_obstacle_height: 2.0
  inflation_layer:
    enabled: true
    inflation_radius: 0.30
    cost_scaling_factor: 3.0
```

Anexo F: Configuración de parámetros del costmap local

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 5.0
  rolling_window: true
  width: 5.0
  height: 5.0
  resolution: 0.05
  transform_tolerance: 0.8
  robot_radius: 0.15
  plugins:
    - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
  obstacle_layer:
    enabled: true
    observation_sources: laser_scan
    combination_method: 1
    laser_scan:
      topic: /scan
      sensor_frame: laser
      data_type: LaserScan
      marking: true
      clearing: true
      obstacle_range: 3.5
      raytrace_range: 4.0
  inflation_layer:
    enabled: true
    inflation_radius: 0.10
    cost_scaling_factor: 5.0
```

Anexo G: Parámetros para el planeador de trayectorias DWAPlannerROS

```
DWAPlannerROS:
  # Velocidades máximas y mínimas
  max_vel_x: 0.15
  max_vel_theta: 1.0
  min_vel_theta: -1.0
  min_in_place_vel_theta: 0.3   # Evita giros muy lentos
  acc_lim_x: 0.3                # Aceleración lineal máxima
  acc_lim_theta: 1.0            # Aceleración angular máxima
  acc_lim_y: 0.0
  min_vel_y: 0.0
  max_vel_y: 0.0
  # Tolerancias para alcanzar el objetivo
  xy_goal_tolerance: 0.15
  yaw_goal_tolerance: 0.15
  latch_xy_goal_tolerance: false
  # Simulación y muestreo
  sim_time: 1.0
  vx_samples: 25
  vtheta_samples: 50
  sim_granularity: 0.025        # Resolución de simulación
  # Pesos de optimización
  path_distance_bias: 32.0      # Peso para seguir el camino global
  goal_distance_bias: 16.0      # Peso para llegar al objetivo
  occdist_scale: 0.3           # Peso para evitar obstáculos
  stop_time_buffer: 0.2
  scaling_speed: 0.25
  max_scaling_factor: 0.2
  forward_point_distance: 0.25
  oscillation_reset_dist: 0.05
  prune_plan: true
  # Frecuencia de control
  controller_frequency: 3.0
  holonomic_robot: false
```