UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS	**UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS** **FACULTAD TECNOLÓGICA** PROYECTO CURRICULAR DE TECNOLOGÍA EN ELECTRÓNICA INDUSTRIAL, INGENIERÍA EN CONTROL Y AUTOMATIZACIÓN E INGENIERÍA EN TELECOMUNICACIONES

INFORME TÉCNICO FINAL DE PROYECTOS DE INVESTIGACIÓN

DATOS DEL PROYECTO DE INVESTIGACIÓN:

Título del Proyecto	Aplicación de Algoritmos de Navegación Autónoma en Entornos Estáticos Desconocidos para el Robot TurtleBot3 sobre ROS	
Nombre completo de los Estudiantes	**Código**	**E-mail**
Daniela Esperanza Pachón Márquez	20181573106	depachonm@udistrital.edu.co
Nombre del Grupo de Investigación		
Seguridad Informática Embebida (SIE)		
Proyecto Curricular		
Tecnología en Electrónica Industrial		

RESUMEN EJECUCIÓN FINANCIERA*:

Valor Total del Proyecto	$84.200.000

CUMPLIMIENTO DE LOS OBJETIVOS:

	Objetivos
General:	**Implementar un sistema de navegación autónoma para el robot TurtleBot3, usando tres algoritmos tipo SLAM sobre ROS para generar mapas.**
Específico 1:	Configurar ROS sobre la plataforma robótica TurtleBot3.
Específico 2:	Generar mapas de su entorno usando algoritmos SLAM sobre ROS.
Específico 3	Realizar un protocolo de pruebas y analizar los resultados obtenidos de la solución específica usando 3 topologías del algoritmo SLAM sobre ROS.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

MARCO TEÓRICO

La robótica móvil se posiciona como uno de los dominios más activos y desafiantes de la ingeniería moderna, integrando disciplinas como la mecánica, electrónica, informática e inteligencia artificial para crear sistemas autónomos capaces de navegar y operar en entornos complejos. A continuación se presenta una revisión de los conceptos fundamentales, metodologías y tecnologías que sustentan la investigación y desarrollo de sistemas robóticos móviles de última generación, con especial énfasis en la plataforma TurtleBot3, el framework ROS (Robot Operating System), algoritmos SLAM (Simultaneous Localization and Mapping), navegación autónoma y tecnologías de sensores LiDAR [1].

Fundamentos de la Robótica Móvil

- ***Conceptos básicos y definiciones***
  Un robot móvil se caracteriza por su capacidad para moverse de manera independiente, con suficiente inteligencia para reaccionar y tomar decisiones basadas en la percepción que recibe el ambiente.

  Los componentes básicos de un robot móvil incluyen cuatro subsistemas fundamentales: [1]
  - Sistema de locomoción: Responsable del movimiento físico del robot, incluyendo actuadores (motores) y mecanismos de tracción (ruedas, orugas, patas). La selección del sistema de locomoción depende del tipo de terreno y las tareas específicas a realizar [2].
  - Sistemas de percepción: Conjunto de sensores que permiten al robot obtener información sobre su entorno y su propio estado. Incluye sensores propioceptivos (edometría, IMU) y exteroceptivos (cámaras, LiDAR, ultrasonido).
  - Sistema de cognición: Unidad de procesamiento central que analiza los datos sensoriales y toma decisiones para coordinar todos los subsistemas del robot. Es responsable del control y la planificación de alto nivel.
  - Sistema de navegación: Implementa algoritmos de planificación de rutas, localización y mapeo para permitir el movimiento autónomo del robot hacia objetivos específicos [1].

- ***Arquitecturas de control***
  Las arquitecturas de control en robótica móvil se pueden clasificar en tres paradigmas principales:
  - Arquitectura jerárquica (Sense-Plan-Act): Sigue una secuencia lineal donde el robot primero percibe su entorno, luego planifica una acción y finalmente la ejecuta. Aunque conceptualmente simple, puede ser lenta para responder a cambios dinámicos del entorno.
  - Arquitectura reactiva: Basada en comportamientos que responden directamente a estímulos sensoriales sin planificación explicita. Permite respuestas rápidas pero puede carecer de coherencia global en el comportamiento.
  - Arquitectura híbrida: Combina elementos de planificación deliberativa con comportamientos reactivos, proporcionando tanto capacidad de respuesta rápida como coherencia en el comportamiento a largo plazo [1].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

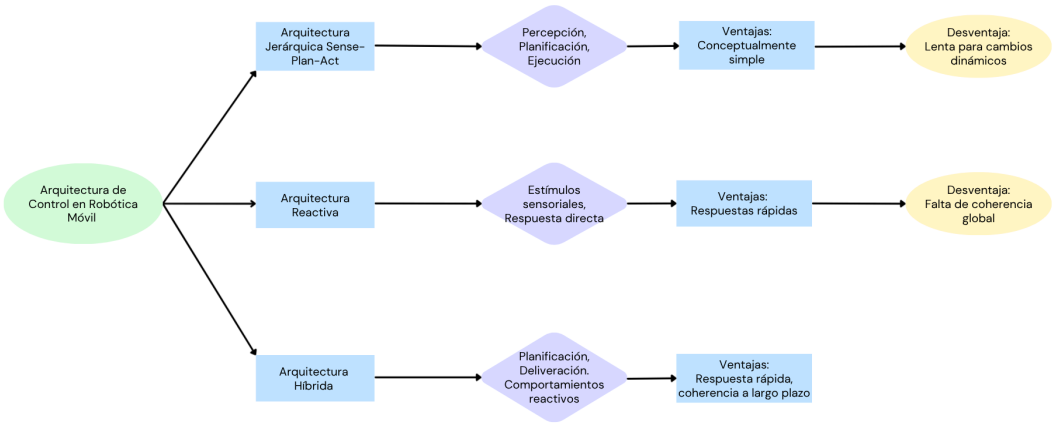


Fig. 1. Diagrama de Flujo de Arquitecturas de Control

- ***Cinemática de robots móviles***
  Describe la relación entre los controles de entrada y el movimiento resultante, sin considerar las fuerzas involucradas. Para un robot móvil en un plano, el estado se representa mediante tres variables: Posición (x, y) y orientación (θ) [2].

  Las restricciones cinemáticas se clasifican en:
  - Restricciones holónomas: Limitaciones en la configuración de robot que pueden expresarse como ecuaciones algebraicas.
  - Restricciones No-holónomas: Limitaciones en la velocidad que no pueden integrarse para obtener restricciones de configuración. La mayoría de robots móviles terrestres están sujetos a estas restricciones debido a la condición de rodadura sin deslizamiento [2].

Robot Operating System (ROS)

- ***Arquitectura y filosofía de ROS***
  Ros es un framework de código abierto que proporciona servicios estándar de un sistema operativo para robots, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comúnmente utilizadas, comunicación entre procesos y gestión de paquetes [3].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

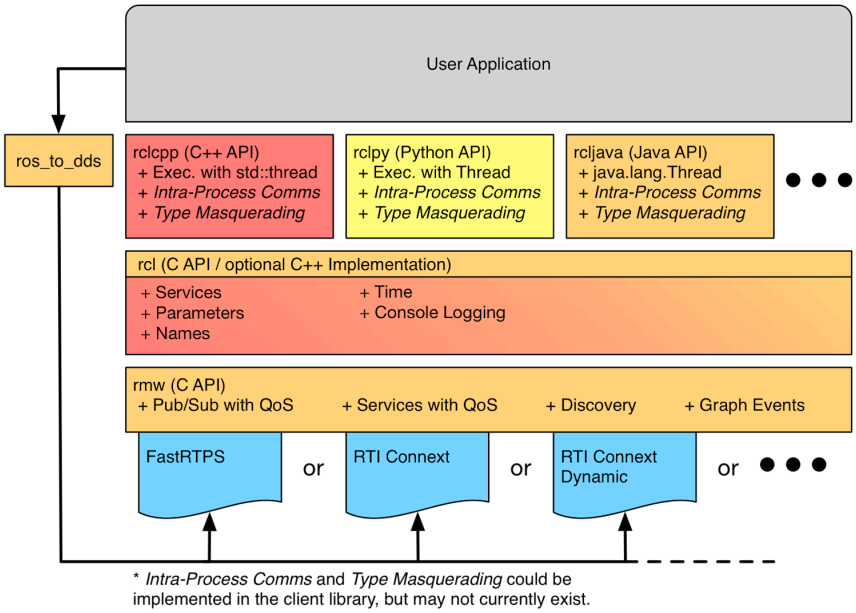


Fig. 2. Diagrama de Arquitectura ROS-DDS [4]

En la Fig. 2 se representa la arquitectura de comunicación ROS basada en DDS, mostrando la conversión de mensajes en ambos middleware.

La filosofía de ROS se basa en cinco principios fundamentales:

- De igual a igual (Peer-to-peer): Arquitectura distribuida que permite la comunicación directa entre componentes a través de una red, facilitando el procesamiento distribuido y la escalabilidad.
- Basado en herramientas (Microkernel): Diseño modular que utiliza numerosas herramientas pequeñas para construir y ejecutar componentes ROS, proporcionando robustez y flexibilidad.
- Multilenguaje: Soporte para múltiples lenguajes de programación (C++, Python, Java, MATLAB) facilitando la integración de diferentes tecnologías.
- Ligero (Thin): Minimiza la complejidad del código base principal, delegado funcionalidades específicas a bibliotecas externas.
- Código abierto: Promueve la colaboración y el desarrollo comunitario bajo licencias permisivas [5].

- ***Conceptos fundamentales de ROS***
  - Nodos: Procesos individuales que realizan cálculos específicos. Cada nodo está diseñado para realizar una tarea particular y puede ejecutarse de forma independiente.
  - Tópicos: Canales de comunicación nombrados por los nodos para intercambiar mensajes de forma asincrónica mediante el patrón publicador-suscriptor.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Servicios: Mecanismo de comunicación sincrónica tipo petición-respuesta entre nodos para tareas que requieren interacción directa.
- Mensajes: Estructuras de datos tipificadas que se intercambian entre nodos a través de tópicos o servicios.
- Máster: Servicio de nombres que facilita el registro y la localización de nodos en la red ROS [5].

- ***Herramientas y utilidades de ROS***
  ROS proporciona un ecosistema completo de herramientas para el desarrollo robótico:
  - RViz: Herramienta de visualización 3D para datos de sensores, estados del robot y planificación de rutas.
  - Gazebo: Simulador físico para el desarrollo y prueba de algoritmos robóticos en entornos virtuales realistas.
  - RQT: Conjunto de herramientas gráficas para la introspección y depuración de sistemas ROS.
  - Rosbag: Sistema de registro y reproducción de datos para análisis offline y desarrollo de algoritmos [6].

Plataforma TurtleBot3

- ***Características y especificaciones***
  TurtleBot3 es una plataforma robótica móvil estandarizada desarrollada como la tercera generación de la serie TurtleBot, diseñada específicamente para educación e investigación de ROS. Desarrollado por ROBOTIS en colaboración con Open Robotics, representa una evolución significativa en términos de modularidad, costo y funcionalidad.

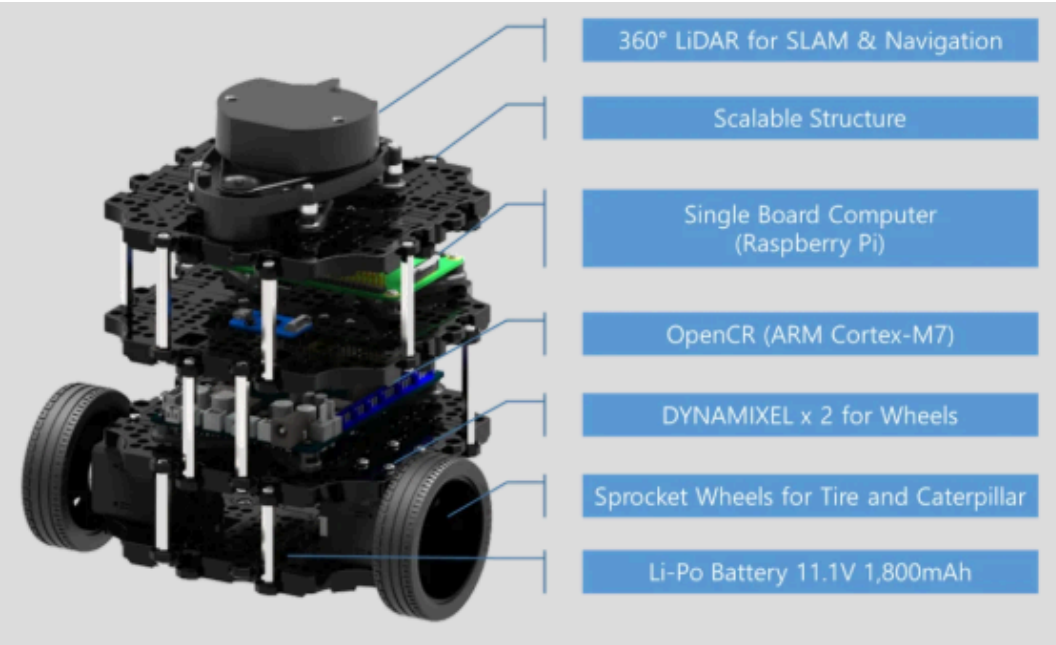


Fig. 3. Robot TurtleBot3 Burger [7]

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Las características principales de TurtleBot3 incluyen:

- Modularidad: Diseño completamente modular que permite personalización y expansión según las necesidades de cada aplicación.
- Costo-efectividad: Reducción significativa del costo comparada con sus predecesores, manteniendo alta funcionalidad y calidad.
- Múltiples configuraciones: Disponible en tres modelos principales (Burger, Waffle, Waffle Pi) con diferentes capacidades y configuraciones de sensores.
- Soporte completo de ROS: Integración nativa con ROS/ROS2, incluyendo navegación, SLAM y manipulación [8].

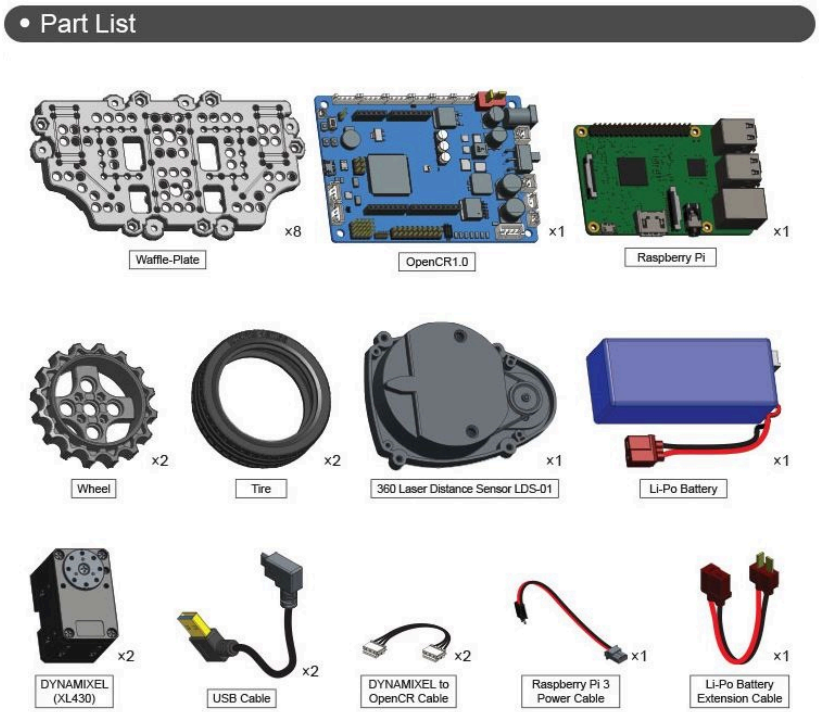


Fig. 4. Hardware TurtleBot3 Burger con Componentes Etiquetados [9]

- ***Aplicaciones en investigación***

  TurtleBot3 ha sido ampliamente adoptado en investigación académica debido a su versatilidad y soporte robusto. Las aplicaciones incluyen desarrollo de algoritmos SLAM, navegación autónoma, aprendizaje por refuerzo y sistemas multi-robot. De acuerdo a esto, la Universidad Distrital Francisco José de Caldas ha realizado diferentes trabajos de investigación robótica en la integración de TurtleBot3, generando contribuciones significativas al desarrollo de tecnología de navegación autónoma [6] [10].

  Investigaciones recientes han demostrado la efectividad de TurtleBot3 en la implementación de algoritmos avanzados como TD3 (Twin Delayed Deep Deterministic Policy Gradient) para navegación autónoma en entornos dinámicos, así como sistemas de

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

entrega autónoma para hospitales inteligentes.Adicionalmente, trabajos de la Universidad Distrital han validado la efectividad de la plataforma para la integración de sensores LiDAR de alta resolución, permitiendo la implementación de algoritmos SLAM sofisticados y sistemas de mapeo en tiempo real [6] [10].

En la Universidad Distrital, el proyecto "Sistema de Navegación Robótica Móvil con LiDAR 360° para TurtleBot3" representó un hito significativo en 2019. Este trabajo implementó de manera integral la plataforma TurtleBot3 Burger con el sensor LiDAR LDS-01 de 360°, desarrollando una interfaz gráfica de usuario para visualizar el proceso de mapeo en tiempo real, permitiendo tanto control manual como autónomo del robot. El proyecto validó la capacidad de TurtleBot3 para realizar navegación autónoma en zonas previamente mapeadas, consolidando la base tecnológica para futuras investigaciones [10].

Complementariamente, el trabajo "Manejo del Robot TurtleBot3 para Aplicaciones de Navegación mediante ROS" proporcionó una metodología sistemática y documentada para la programación y operación de TurtleBot3. Este artículo científico se convirtió en referencia académica para la comunidad investigadora al ofrecer guías detalladas sobre implementación de aplicaciones de navegación autónoma utilizando Python y ROS, facilitando que sea fácil de replicar en proyectos y la transferencia de conocimiento [11].

También, el "Especificación, diseño, construcción modelo y evaluación de un prototipo de robot móvil para aplicaciones de navegación en ambientes humanos", iniciado en 2019 con la especificación y diseño de prototipos robóticos de locomoción diferencial, ha evolucionado significativamente. Este proyecto ha generado dos contribuciones destacadas: primero, el desarrollo de un prototipo personalizado con múltiples sistemas sensoriales para estudiar estrategias de navegación en ambientes humanos, y segundo, la investigación más reciente de "Adecuación de LiDAR en ARMOS TurtleBot y desarrollo de tarea de localización y mapeo simultáneo SLAM con ROS" (2024), que ha implementado algoritmos SLAM de última generación con ROS Noetic. Este último trabajo no solo ha avanzado en la integración de hardware avanzado, sino que ha generado contribuciones al estado del arte en tecnologías SLAM, posicionando a la Universidad Distrital como referente en investigación de navegación autónoma [12] [13].

Tecnología LiDAR en Robótica Móvil

- ***Principios de funcionamiento***

  LiDAR (Light Detection and Ranging) es una tecnología de detección remota que utiliza pulsos láser para medir distancias y crear representaciones tridimensionales precisas del entorno. El principio básico se basa en la medición del tiempo de vuelo de pulsos láser que se reflejan en superficies del entorno [14], [15].

  El funcionamiento de un sensor LiDAR implica:

  - Emisión de pulso láser: El sensor emite pulsos de luz láser en direcciones específicas.
  - Reflexión: Los pulsos se reflejan en superficies del entorno.
  - Detección: El sensor detecta los pulsos reflejados y mide el tiempo transcurrido.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Cálculo de distancia: utilizando la velocidad de la luz, se calcula la distancia a cada punto.
- Generación de nube de puntos: Se construye una representación 3D del entorno[14].

- ***Tipos de sensores LiDAR***
  - LiDAR 2D: Realiza barridos en un plano, proporciona información de rango y ángulo. Comúnmente utilizado en aplicaciones de navegación indoor y mapeo 2D [16].



Fig. 5. Sensor LiDar 2D [17]

  - LiDAR 3D: Proporciona información tridimensional completa del entorno, útil para aplicaciones de mapeo 3D y detección de obstáculos complejos.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.



Fig. 6. Sensor LiDAR 3D [18]

- LiDAR rotativo vs. estado sólido: Los sistemas rotativos utilizan mecanismos mecánicos para el barrido, mientras que los de estado sólido emplean tecnologías como MEMS o arrays de fases [15].

- ***Aplicaciones en SLAM y navegación***
  - Construcción de mapas de ocupación: Los datos LiDAR se procesan para generar más de rejilla que representan la probabilidad de ocupación de cada celda del espacio [19].
  - Detección y seguimiento de obstáculos: Identificación en tiempo real de obstáculos estáticos y dinámicos para navegación segura[15].
  - Localización: Comparación de barridos LiDAR actuales con mapas conocidos para determinar la posición del robot mediante técnicas como scan matching [20].

- ***Algoritmos de procesamiento LiDAR***
  - Filtrado de datos: Eliminación de ruido y puntos espurios utilizando filtros estadísticos y geométricos.
  - Segmentación: Agrupación de puntos pertenecientes a objetivos o superficies similares.
  - Extracción de características: Identificación de elementos geométricos como líneas, esquinas y superficies planas para navegación y localización.
  - Registro de nubes de puntos: Alineación de múltiples barridos LiDAR para construcción de mapas coherentes utilizando algoritmos como ICP (Iterative Closest Point) [20].

- ***SLAM profundo y técnicas de IA***

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- SLAM Visual basado en Deep learning: Utilización de redes neuronales convolucionales para extracción de características visuales y estimación de pose.
- SLAM semántico: Integración de información semántica en el proceso de mapeo para crear representaciones más ricas del entorno.
- Función multisensorial inteligente: Combinación adaptativa de múltiples sensores utilizando técnicas de machine learning [21].

- ***Robótica colaborativa y multi-robot***
  - Sistemas multi-robot: Desarrollo de algoritmos para coordinación y colaboración entre múltiples robots móviles.
  - SLAM colaborativo: Técnicas donde múltiples robots contribuyen simultáneamente a la construcción de mapas compartidos.
  - Navegación cooperativa: Algoritmos de planificación de rutas que consideran la presencia y los movimientos de otros robots en el entorno [22].

Odometría

Es el proceso de estimar la posición y orientación de un robot móvil basándose en mediciones de sensores con percepción del cuerpo en el espacio (internos), tales como codificadores (encoders) en las ruedas, giroscopios y acelerómetros. Específicamente la odometría consiste en el uso de datos generados por los sensores de movimiento para estimar el cambio de posición de un objeto en el tiempo con relación a una posición inicial de referencia [23].

En el contexto de robots móviles con configuración de tracción diferencial (en este caso TurtleBot3), la odometría mide la distancia y orientación recorridas integrando la velocidad de rotación de las rutas a través del tiempo. La posición del robot en un plano se representa mediante tres variables: coordenadas cartesianas (x, y) y orientación angular (θ) [24].

- ***Funcionamiento de la Odometría***

  El sistema de odometría opera en las siguientes etapas:
  - Adquisición de datos sensoriales: Los encoders ópticos o magnéticos instalados en los ejes de las ruedas capturan el número de pulsos generados con cada revolución. Estos sensores proporcionan información sobre la magnitud del desplazamiento y la dirección de rotación de cada rueda.
  - Cálculo de distancia recorrida: A partir de los pulsos del encoder, se calcula la distancia recorrida por cada rueda usando la relación: Distancia = (Número de pulsos/Pulsos por revolución) x Circunferencia de la rueda.
  - Estimación de posición: Para un robot de tracción diferencial, la posición central del robot se obtiene promediando las distancias de ambas ruedas. La orientación (θ) se calcula considerando la diferencia entre las distancias recorridas por cada rueda y la distancia entre ejes. Esto se integra temporalmente para obtener la pose estimada (x, y, θ) [23].
  - Publicación de ROS: En plataformas como ROS, la información de odometría se publica continuamente en el tópico /odom, permitiendo que otros nodos accedan a la posición y velocidad actual del robot [11].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- ***Ventajas de la Odometría***
  - Control de lazos cerrados internos: Permite implementar control interno de velocidades, aceleración y frenado basado en mediciones directas del movimiento del robot.
  - Alta frecuencia de actualización: Los sensores de odometría pueden proporcionar mediciones de frecuencias muy altas (típicamente 50-200 Hz en TurtleBot3).
  - Bajo costo computacional: Los cálculos de odometría son operaciones algebraicas simples que no requieren procesamiento intensivo, siendo ideales para plataformas con recursos limitados como Raspberry Pi [24].

- ***Limitaciones Críticas: Error Acumulativo***
  A pesar de sus ventajas, la odometría presenta limitación fundamental: la acumulación de errores a lo largo del tiempo, fenómeno conocido como deriva (drift) o dead reckoning error.

  Fuentes de error en odometría:
  - Desplazamiento de ruedas: Cuando el robot transita sobre superficies resbaladizas o irregulares, las ruedas pueden patinar sin corresponder exactamente a la rotación registrada por los encoder.
  - Variaciones en el diámetro de las ruedas: El desgaste diferencial de las ruedas, cambios de presión de aire o acumulación de polvo modifica el perímetro efectivo, causando discrepancias en los cálculos de distancia.
  - Ruido en sensores:Los encoders y giroscopios introducen pequeñas imprecisiones que se integran acumulativamente [25].

  El error de posición estimado crece aproximadamente de forma lineal o cuadrática con el tiempo y la distancia recorrida, haciendo que después de recorrer distancias significativas (típicamente cientos de metros), la posición estimada se desvíe considerablemente del valor real.

- ***Rol de la Odometría en SLAM***
  En el algoritmo SLAM, la odometría juega un papel fundamental pero limitado:
  - Predicción a corto plazo: Proporciona una estimación rápida de la pose del robot entre lecturas de sensores exteroceptivos (como LiDAR o cámaras).
  - Inicialización de búsqueda: Alimenta los algoritmos de emparejamiento de escaneos (scan matching) con una estimación inicial cercana a la solución real.
  - Corrección mediante fusión: Los algoritmos SLAM (como Gmapping, Hector SLAM y SLAM Toolbox) utilizan la odometría como predicción pero la corrigen continuamente comparando las observaciones sensoriales exteroceptivas con el mapa conocido, controlando así la deriva [26].

La incertidumbre de la odometría debe ser considerada explícitamente en el modelo en probabilidades de SLAM, típicamente como una matriz de covarianza que crece en el tiempo.

"Simultaneous Localization and Mapping" (SLAM)

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- ***Fundamentos teóricos de SLAM***

  SLAM es el problema computacional de construir o actualizar un mapa de un entorno desconocido mientras simultáneamente se mantiene el seguimiento de la ubicación del agente dentro de ese entorno. Formalmente, dada una serie de controles y observaciones a través de pasos temporales discretos, el problema SLAM consiste en calcular una estimación del estado del agente y un mapa del entorno [27].

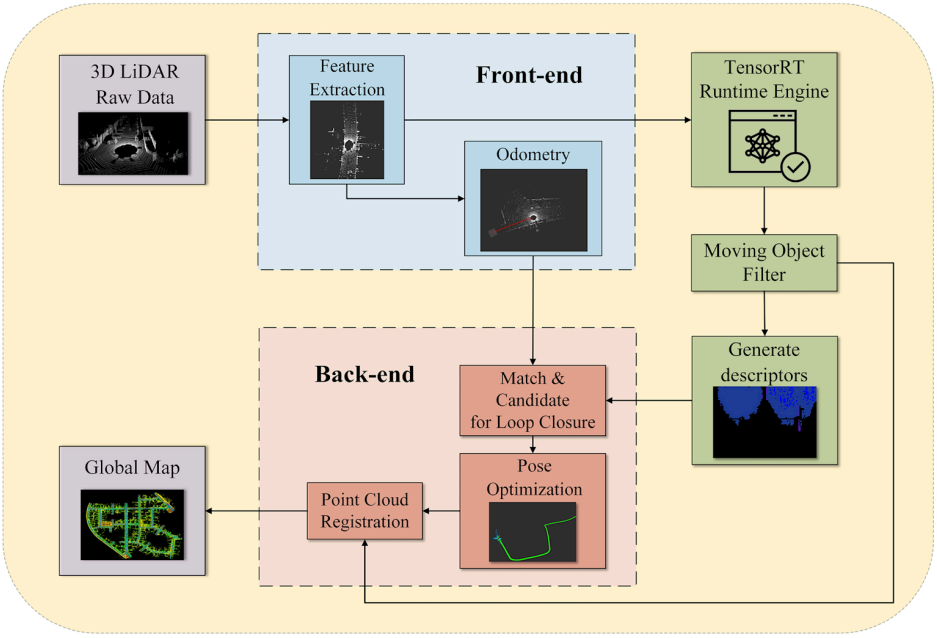


Fig. 7. Visualización de Salida SLAM [28]

Matemáticamente, SLAM busca computar la distribución de probabilidad conjunta:

$P(m_{t+1}, x_{t+1} | o_{1:t+1}, u_{1:t})$

Donde:

- m_t representa el mapa en el tiempo t
- x_t representa el estado del robot en el tiempo t
- o_t son las observaciones en el tiempo t
- u_t son los controles en el tiempo t

- ***Paradigmas principales de SLAM***
  - SLAM basado en filtros: Utiliza filtros de probabilidad como el filtro de Kalman Extendido (EKF) o filtros de partículas para mantener una estimación continua del estado del robot y del mapa.
  - SLAM basado en grafos: Representa el problema SLAM como un grafo donde los nodos representan poses del robot o puntos de referencia, y las aristas representan restricciones espaciales obtenidas de mediciones de sensores.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- SLAM visual: Utiliza información visual de cámaras para realizar localización y mapeo, incluyendo técnicas como visual SLAM (VSLAM) y Visual-Inertial SLAM [21].

- ***Algoritmos SLAM específicos***

Gmapping

Gmapping es un algoritmo SLAM basado en filtros de partículas que permite a un robot móvil construir mapas de ocupación 2D a partir de datos de sensores láser y odometría. Implementa el enfoque conocido como FastSLAM 1.0, en el que la trayectoria del robot se estima mediante una familia de partículas, y para cada partícula se mantiene un mapa con base en la información sensorial recibida [29].

Sus principales características son:

- Efectividad con bajo costo computacional: Gmapping es eficiente para plataformas con recursos limitados, como la TurtleBot3 Burger [29].
- Dependencia de la odometría: La precisión del mapa generado está condicionada a la calidad de la información de odometría y de sensores láser. Errores acumulados pueden afectar la precisión global del sistema.[30]
- Implementación en ROS: La disponibilidad de Gmapping como paquete ROS (slam_gmapping) lo convierte en uno de los algoritmos más utilizados para mapeo en tiempo real en entornos indoor [31].

Principio de funcionamiento: Gmapping utiliza el algoritmo Rao-Blackwellized Particle Filter (RBPF) que factoriza el problema SLAM en dos partes: la estimación de la trayectoria del robot mediante un filtro de partículas, y la construcción del mapa condicionado a cada trayectoria estimada. Esta factorización reduce significativamente la complejidad computacional permitiendo el mapeo en tiempo real [29].

Hector SLAM

Hector SLAM es un algoritmo que se distingue por prescindir de la odometría clásica del robot, empleando únicamente datos de sensores láser de alta tasa de actualización para estimar la pose y construir el mapa en tiempo real mediante técnicas de scan matching [32].

Característica principales:

- Ideal para robots sin buena odometría: adecuado para robots ligeros, drones y aquellos en los que la odometría es poco confiable o presenta grandes errores acumulados [32].
- Scan Matching multinivel: Utiliza el algoritmo Gauss-Newton y un enfoque jerárquico para optimizar la alineación de los datos láser, permitiendo robustez en escenarios dinámicos y rápidos [33].
- Disponibilidad en ROS: Su integración nativa facilita su uso en simuladores como Gazebo, así como en plataformas reales equipadas con sensores LiDAR de alta frecuencia [34].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Ventajas y limitaciones: Hector SLAM no requiere odometría, lo que elimina errores por deslizamiento de ruedas o terreno irregular. Sin embargo, su precisión depende críticamente de la tasa de actualización del sensor LiDAR (típicamente ≥40Hz) y puede presentar deriva acumulativa en ausencia de cierres de bucle explícitos [32] [33].

SLAM Toolbox

SLAM Toolbox es una colección moderna y robusta de herramientas y algoritmos SLAM para ROS2 (con soporte parcial para ROS1), diseñada para aplicaciones de robótica en tiempo real y escenarios multisesión en entornos dinámicos [35].

Características distintivas:

- SLAM síncrono y asíncrono: Ofrece capacidades para mapeo online y offline, incluyendo la optimización del mapa tras finalizar sesiones de exploración.
- Loop closure robusto: Emplea técnicas avanzadas para la detección de cierres de bucle, mejorando la coherencia y calidad de los mapas generados[35].
- Mapas persistentes y multisesión: Permite la fusión de múltiples sesiones de mapeo, una característica crucial en aplicaciones de cartografiado progresivo de entornos extensos[16].
- Modos de operación múltiples: Incluye modo de mapeo continuo, modo de localización pura, modo asíncrono para entornos de gran escala (hasta 24,000 m²), y herramientas de manipulación manual del grafo de poses.
- Adopción oficial: Seleccionado como el proveedor SLAM por defecto en ROS 2, reemplazando a Gmapping, e integrado en el proyecto Navigation2 [35].

SLAM Toolbox en MATLAB

MATLAB Navigation Toolbox proporciona un conjunto integrado de herramientas y algoritmos para implementar SLAM utilizando escaneos LiDAR 2D. El componente central es el objeto lidarSLAM, que correlaciona escaneos LiDAR y optimiza la trayectoria del robot mediante un grafo de poses [36].

Características principales:

- Procesamiento de escaneos LiDAR: El algoritmo agrega sucesivamente escaneos LiDAR, en correlación con escaneos previos para estimar la pose relativa del robot. Mantiene un grafo de poses donde los nodos representan posiciones del robot y las aristas representan restricciones de transformación [37].
- Detección de cierres de bucle: La función detectLoopClosure identifica automáticamente cuando el robot regresa a regiones previamente exploradas, comparando el escaneo actual con escaneos históricos dentro de un radio de búsqueda definido. Los cierres de bucle detectados se agregan con addLoopClosure para reducir la deriva acumulativa [38].
- Optimización del grafo de poses: Cuando se detectan cierres de bucle, el algoritmo optimiza el grafo completo utilizando solvers de optimización no lineal, distribuyendo los errores acumulados a través de toda la trayectoria [39].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Construcción de mapas de ocupación: Tras completar el mapeo, la función buildMpap genera un mapa de ocupación 2D a partir del grafo de poses optimizado y los escaneos LiDAR [37].
- Parámetros configurables: Incluye MapResolution (resolución de la cuadrícula), MaxLidarRange (rango máximo del sensor), y LoopClosureSearchRadius (radio de búsqueda de cierres) [36].
- Integracion con ROS: Permite procesar datos de sensor LiDAR en tiempo real desde robots que publican escaneos en tópicos ROS estándar [39].

Aplicaciones verificadas: Ha sido desplegado exitosamente en robots comerciales y de investigación, incluyendo TurtleBot3, Simbe Robotics Tally, robots de Samsung Research, y plataformas de Pal Robotics [35].

Tabla 2. Comparación de Algoritmos SLAM

Algoritmo	***Complejidad Computacional***	***Adecuación para TurtleBot3***
Hector SLAM	Aproximadamente O(pixels_per_scan·log pixels) por escaneo.	Alta: optimizado para LiDAR 2D y CPU limitada, genera mapas en tiempo real sin odometría.
Gmapping	O(M·log n) similar a FastSLAM (Rao-Blackwellized PF).	Alta: ampliamente utilizado en TurtleBot3, buena calidad de mapeo y eficiencia en Raspberry Pi.
SLAM Toolbox	Aproximadamente O(n) $O(n)$ por posición, con optimizaciones de gráficos y bucles de cierre globales; procesamiento intensivo en optimización y corrección de mapas grandes.	Alta: ideal para TurtleBot3 con ROS 2 y sensores láser; soporta modos síncrono y asíncrono, robusto frente a bucles de cierre y adecuado para plataformas con capacidad moderada de CPU.

- ***Sensores para SLAM***
  - LiDAR SLAM: Utiliza sensores de rango láser para obtener mediciones precisas de distancia y construir mapas de ocupación detallados. Los algoritmos incluyen Gmapping, Hector SLAM y Cartographer [10].
  - Visual SLAM: Emplea cámaras monoculares, estéreo o RGB-D para extraer características visuales y estimar movimiento y estructura del entorno
  - SLAM multi-sensorial: Combina múltiples tipos de sensores (LiDAR, cámaras, IMU) para mejorar robustez y precisión en la estimación [21].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Navegación Autónoma

- ***Arquitectura de navegación***

  La navegación autónoma en robótica móvil comprende la capacidad de un robot para moverse de un punto inicial a un destino objetivo de manera independiente, evitando obstáculos y optimizando la trayectoria. La arquitectura típica de navegación incluye varios componentes integrados: [14]

  - Planificación global: Genera una ruta óptima desde la posición inicial hasta el objetivo utilizando un mapa conocido del entorno. Algoritmos comunes incluyen A*, Dijkstra y RRT (Rapidly-exploring Random Trees) [40].
  - Planificación local: modifica la ruta global en tiempo real para evitar obstáculos dinámicos y no mapeados. Implementa algoritmos como Dynamic Window Approach (DWA) y campos potenciales artificiales.
  - Localización: Determina la posición actual del robot en el mapa utilizando técnicas como Monte Carlo Localization (MCL) o filtros de partículas.
  - Control de movimiento: Traduce los comandos de planificación en señales de control para los actuadores del robot, implementando controladores PID, controladores de seguimiento de trayectoria o técnicas de control no lineal [41].

- ***Algoritmos de planificación de rutas***

  Algoritmos basados en grafos:

  - A*: Algoritmo de búsqueda estratégica que encuentra la ruta óptima minimizando una función de costo que combina distancia recorrida y estimación del método práctico al objetivo [42].
  - Dijkstra: Encuentra la ruta de menor costo sin utilizar estrategia, garantizando optimalidad pero con mayor costo computacional.
  - D y D* Lite: Variantes dinámicas que permiten replaneación eficiente cuando se detectan cambios en el entorno [40].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

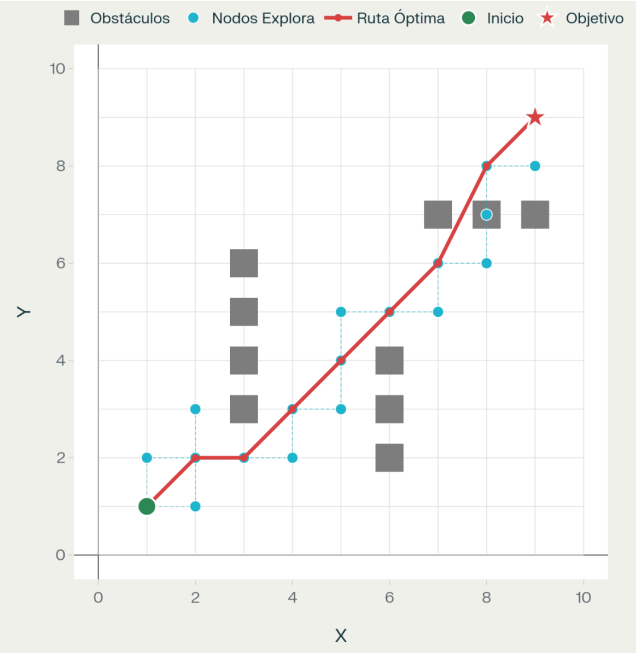


Fig. 8. Algoritmos de Grafos (A, Dijkstra, D*)*

Algoritmos basados en muestreo:

- RRT (Rapidly-exploring Random Trees): Construye un árbol de configuraciones válidas mediante muestreo aleatorio del espacio de configuraciones [43].
- RRT*: Versión asintóticamente óptima de RRT que refina continuamente la ruta para converger hacia la solución óptima [40].
- PRM (Probabilistic Roadmaps): Precomputa una red de rutas válidas que puede ser consultada eficientemente para múltiples consultas de planificación [43].

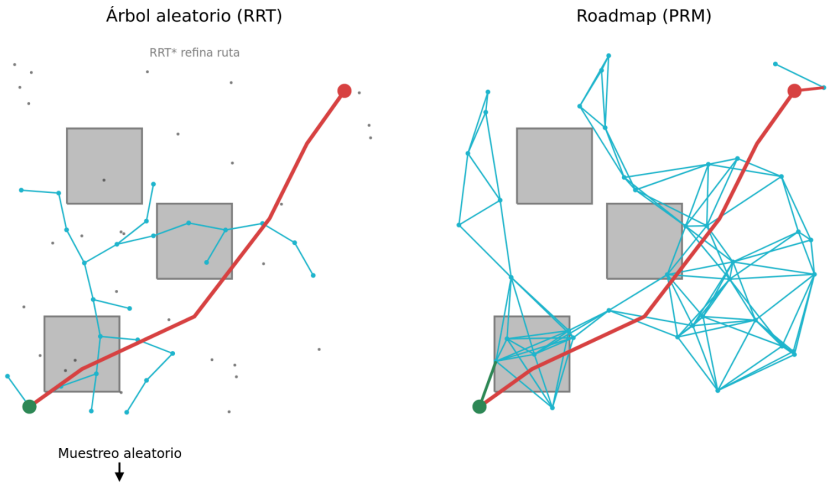


Fig. 9. Algoritmos de Muestreo

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Algoritmos basados en campos potenciales:
Utilizan funciones de potencial artificial donde el objetivo genera un campo atractivo y los obstáculos generan campos repulsivos. Aunque computacionalmente eficientes, pueden sufrir del problema de mínimos locales [40].

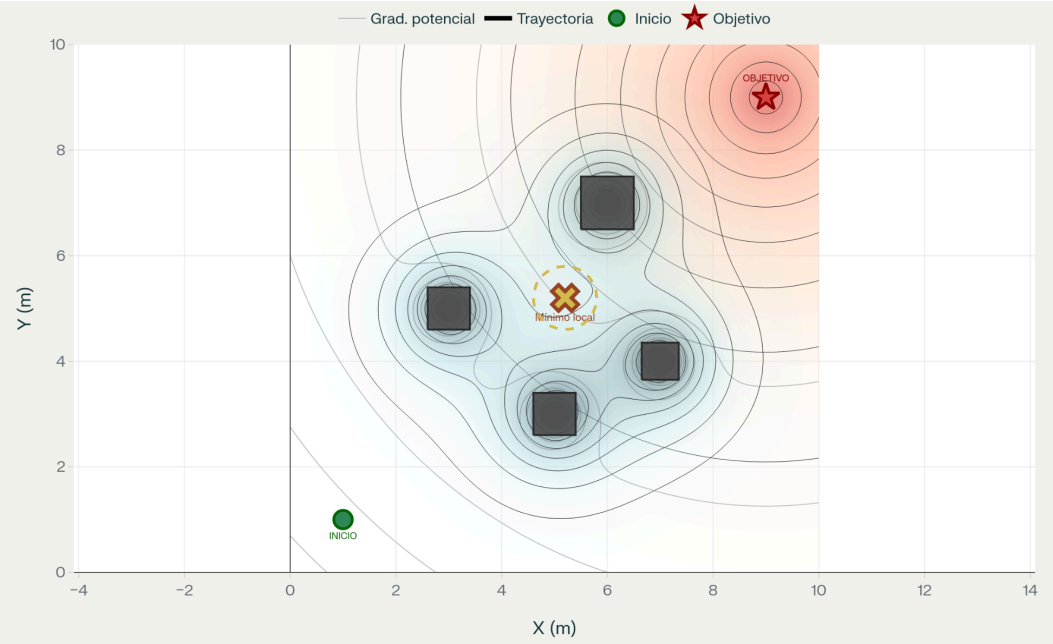


Fig. 10. Campos Potenciales para Planificación

- ***Navegación en entornos dinámicos***
  La navegación en entornos con obstáculos móviles presenta desafíos adicionales que requieren algoritmos especializados:
  - Predicción de movimiento: Estimación de trayectorias futuras de obstáculos dinámicos utilizando filtros Kalman o modelos de movimiento.
  - Planificación espacio-temporal: Incorpora la dimensión temporal en la planificación de rutas, considerando el estado futuro del entorno.
  - Navegación reactiva: Implementa comportamientos de evasión inmediata basados en información sensorial local [22].

- ***Técnicas de control para seguimiento de trayectoria***
  - Controladores cinemáticos: Basados en el modelo cinemático del robot, incluyen controladores de seguimiento de trayectoria como el controlador de Lyapunov y control por linealización.
  - Controladores dinámicos: Consideran las dinámicas del robot incluyendo efectos inerciales y fuerzas de fricción [41].
  - Control predictivo por modelo (MPC): Optimiza la trayectoria futura dentro de un horizonte de predicción, permitiendo manejo explícito de restricciones [44].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Métricas de Evaluación para Algoritmos SLAM

La evaluación sistemática del desempeño de los algoritmos SLAM es crítica tanto en investigación académica como en aplicaciones prácticas. Las métricas más empleadas incluyen:

- **Métricas de exactitud de trayectoria**
  - Error Absoluto de Trayectoria (ATE - Absolute Trajectory Error): Mide la diferencia entre la trayectoria estimada por el algoritmo y la trayectoria real del robot (*ground truth*), usualmente mediante la distancia raíz cuadrática media (RMSE). Se calcula alineando las trayectorias mediante transformación de similaridad y computando la norma euclidiana de las diferencias.
  - Error Relativo de Pose (RPE - Relative Pose Error): Evalúa la consistencia en la estimación de la trayectoria sobre pequeños intervalos de tiempo o distancia recorrida, proporcionando un análisis detallado de la deriva local. Es particularmente útil para evaluar la precisión de odometría visual [45]
- **Métricas de Calidad del Mapa**
  - Similitud estructural del mapa: Comparación del mapa generado con un mapa de referencia (*ground truth*) mediante métricas de superposición como IoU (Intersection over Union), correlación espacial, o análisis de ocupación por celdas [46].
  - Consistencia topológica: Verificación de que el mapa generado mantiene la topología correcta del entorno, incluyendo la correcta representación de cierres de bucle [47].
  - Resolución y densidad: Evaluación de la granularidad del mapa y la completitud de la información capturada en áreas exploradas [46].

- **Métricas de Robustez y Desempeño**
  - Detección de cierres de bucle: Capacidad del algoritmo para reconocer y corregir cuando el robot regresa a una ubicación previamente visitada, reduciendo la deriva acumulativa [46].
  - Robustez a ruido sensorial: Desempeño del algoritmo ante datos ruidosos, mediciones espurias o condiciones adversas del sensor [30] [47].
  - Tiempo de procesamiento y eficiencia computacional: Importante para aplicaciones en tiempo real sobre plataformas con recursos limitados. Se registran frecuencias de actualización (fps), latencia y uso de memoria RAM y CPU [47].

Herramientas de Visualización

En el contexto de robótica con TurtleBot3 y ROS, la visualización es central tanto para el desarrollo como para la evaluación experimental de algoritmos de mapeo y navegación.

- **RViz (ROS Visualization)**
  RViz es la herramienta principal para la visualización de datos en tiempo real en ROS. Sus capacidades incluyen:
  - Visualización de mapas de ocupación: Renderizado de mapas 2D generados por algoritmos SLAM como Gmapping, Hector SLAM y SLAM Toolbox [29] [46].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Representación de trayectorias: Visualización de la trayectoria estimada del robot y comparación con ground truth.
- Datos sensoriales múltiples: Despliegue simultáneo de nubes de puntos LiDAR, imágenes de cámaras RGB-D, marcadores de transformadas (TF) y vectores de pose [48].
- Cierres de bucle: Visualización gráfica de restricciones de cierre de bucle detectadas durante el proceso SLAM[21].
- Configuración por tópicos: Sincronización flexible con tópicos ROS relevantes (/map, /odom, /scan, /tf, etc.) para visualización personalizada según la aplicación [48].

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

DESARROLLO DEL PROYECTO

Desde la selección y preparación del hardware hasta el análisis final de resultados, el proyecto siguió un recorrido riguroso y metódico para lograr la navegación autónoma del TurtleBot3 Burger sobre ROS Noetic. A continuación se describe cada paso del proceso con profundidad.

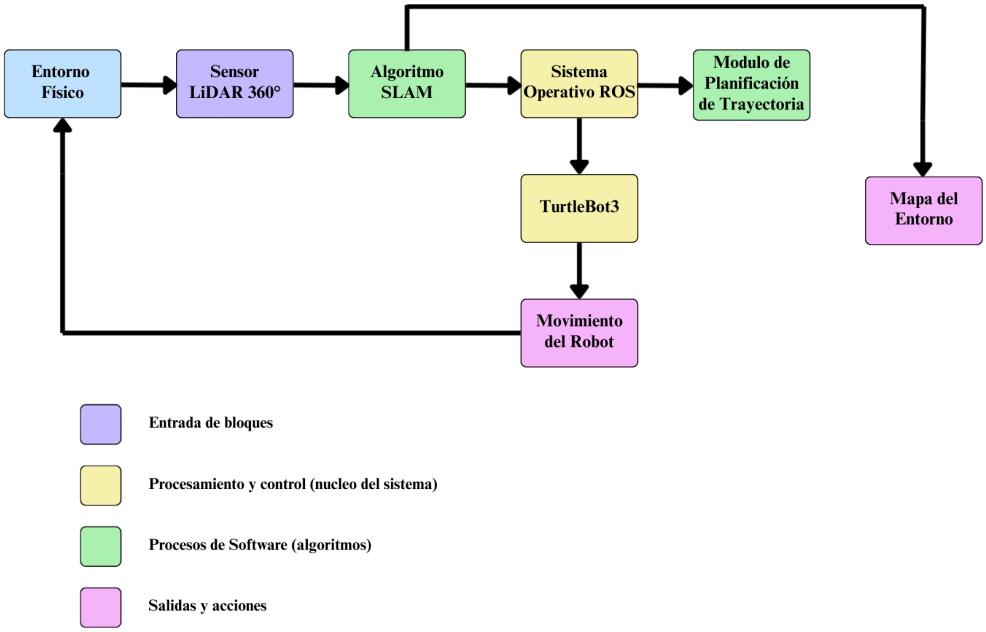


Fig. 11. Diagrama de bloques desarrollo de proyecto

Metodologia de Implementacion

Metodología objetivo 1: Configurar ROS sobre la plataforma robótica TurtleBot3

- ***Preparación de hardware y plataforma***
  El proyecto inició con la evaluación de la capacidad computacional del robot Turtlebot3 Burger original, el cual viene equipado con una raspberry pi 3 que cuenta con 1GB de RAM. Después de revisar y hacer algunas pruebas, se evidencio que no era suficiente para ejecutar simultáneamente ROS, el proceso del sensor, y los algoritmos de mapeo sin degradación significativa del desempeño.

  Por esta razón se realizó el cambio por la Raspberry Pi 4, incrementando la capacidad de RAM a 4GB. Este cambio fue fundamental para garantizar:
  - Suficiente espacio en memoria para mantener los mapas en construcción.
  - Capacidad de procesamiento para ejecutar los tres algoritmos sin colapso de sistema.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Mayor velocidad de compilación de paquetes ROS.
- Mejor rendimiento en la ejecución simultánea de múltiples nodos ROS.

Se verificó mecánicamente la integridad del robot: revisión de fijación de ruedas, anclaje del LiDAR y correcto acople de la placa de expansión OpenCR.

Instalación del Sistema Operativo

Con una tarjeta mocroSD de 32GB, se procedió a:

- Formatear la tarjeta en FAT32.
- Descargar la versión de Ubuntu que se va a implementar, en este caso la versión 20.04.5, seleccionada por su compatibilidad con ROS Noetic y estabilidad en sistemas embebidos. Esta versión se descarga directamente de la página web https://cdimage.ubuntu.com/releases/focal/release/?_gl=1*13skbvl*_gcl_au*ODIwOTcyNjcwLjE3NTg4NDMyNTA. en la opción que se verá a continuación en el recuadro verde.

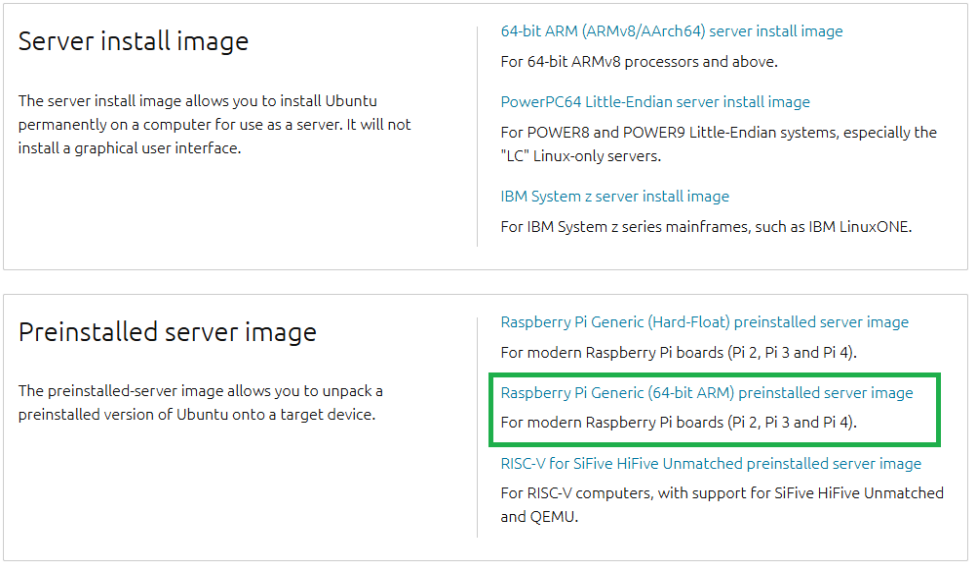


Fig. 12. Ventana para descargar imagen de Ubuntu.

- Usar Raspberry Pi Imager, que se puede descargar aqui https://www.raspberrypi.com/software/ ,para flashear la imagen de ubuntu 20.04.5 LTS. La instalación es muy intuitiva, y después de ya tener esto, es necesario conectar la memoria microSD por medio de un adaptador al computador.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

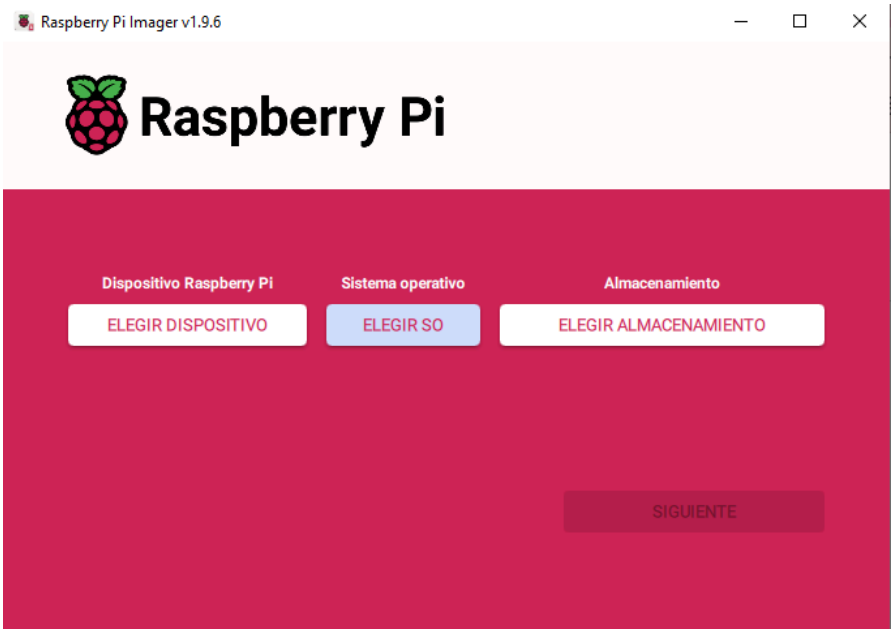


Fig. 13. Raspberry Pi Imager

- Teniendo la ventana que se ve en la imagen anterior, en la primera opción se va a seleccionar el dispositivo, que en este caso será Raspberry pi 4. En la segunda opción para elegir SO. de últimas aparece "usar personalizado", se debe buscar el archivo descargado de la imagen de Ubuntu y seleccionarlo. Por último se selecciona el almacenamiento, que en este caso será la memoria microSD (Fig. 13). Se presiona siguiente y aparece una ventana (Fig. 14), en la opción de Editar Ajustes, hacer la configuración de usuario y contraseña, así como el nombre y contraseña de la red WiFi que se va a usar. Se guarda y se aplican los ajustes personalizados.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

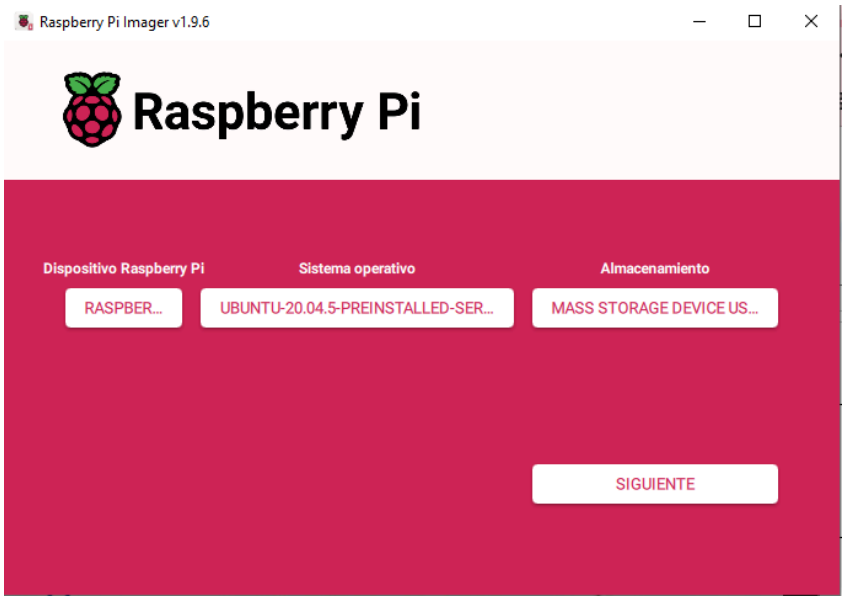


Fig. 14. Raspberry Pi Imager con opciones seleccionadas

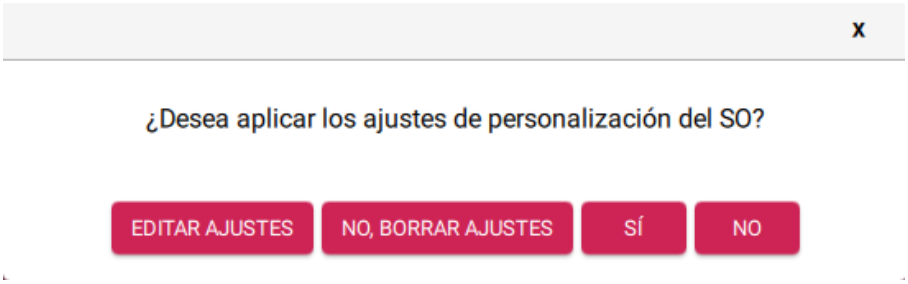


Fig. 15. Raspberry Pi Imager, ajustes de configuración

- Una vez finalizado el proceso, se retira la memoria y se vuelve a conectar al computador. En la carpeta principal del boot se va a crear un archivo, en block de notas, vacío y sin extensión, con el nombre “ssh”. Esto para poder activar el acceso remoto a la Raspberry Pi 4. Se retira de nuevo y se inserta la memoria a la Raspberry Pi 4. Se espera unos minutos el primer arranque y se revisa la conexión a WiFi, ambos dispositivos (Raspberry y PC) deben estar conectados a la misma red.

- ***Habilitación de Acceso Remoto (SSH)***
  Dado que la versión de ubuntu instalada no cuenta con interfaz gráfica, fue necesario establecer acceso remoto para proceder sin conexión física constante al robot.

  - Abrir terminal de Símbolo de Sistema e ingresar “ping ubuntu-pi.local” para saber la direccion IP

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

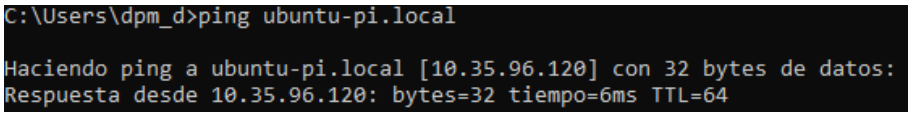


Fig. 16. Dirección IP de Raspberry Pi

- Teniendo la dirección IP, se puede acceder por medio de SSH de la siguiente manera: ssh usuario@IP_RASPBERRY y luego la contraseña.

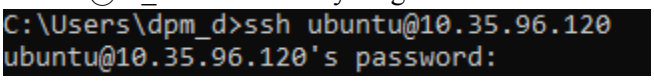


Fig. 17. Acceso por SSH a Raspberry Pi

Una vez se tiene el acceso por SSH se podrán instalar paquetes de ROS, gestionar archivos workspace, monitorizar procesos, ejecutar scripts y lanzar nodos de manera remota sin desconexiones.

- ***Instalación de ROS Noetic***

  Al tener acceso por ssh, antes de empezar con la instalación de ROS, se debe realizar una actualización del sistema para que los paquetes instalados estén en su versión más reciente. Ai se ve el primer acceso por SSH

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

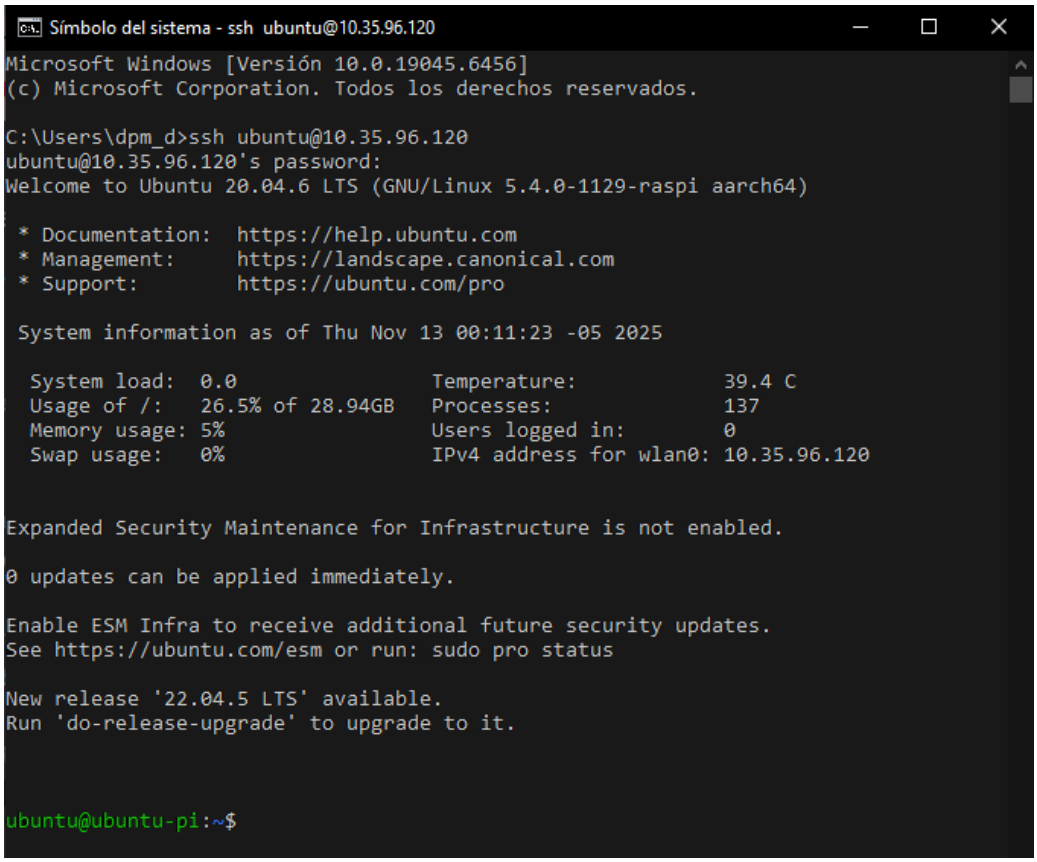


Fig. 18. Primer acceso por SSH en terminal Símbolo de Sistema

- ***Actualizar sistema operativo***
  Para realizar esto, se van a ingresar los siguientes comandos en la terminal con acceso a la Raspberry Pi, terminados estos comandos estará lista para instalar ROS Noetic
  - sudo apt update
  - sudo apt upgrade

- ***Configurar los repositorios de ROS***
  En este paso se registra el repositorio oficial de ROS en el sistema, lo que va a permitir que se descarguen paquetes desde los propios servidores, se descarga la clave de seguridad de ROS y verifica la autenticidad de los paquetes que se descargaran.
  - sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
  - sudo apt install curl
  - curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- ***Instalación de ROS Noetic***
  Se actualiza de nuevo el sistema para obtener la lista actualizada de paquetes, ahora incluyendo los de ROS, y se hace la instalación base del mismo que contiene herramientas de comunicación, roscore, ros tools y las dependencias necesarias.
  - sudo apt update
  - sudo apt install ros-noetic-ros-base

- ***Configurar el entorno de ROS***
  Es necesario configurar el entorno de ROS para que se cargue automáticamente cada vez que se abra una terminal. También se instalan dependencias adicionales para compilar los paquetes. Se inicializa rosdep y luego se descarga la base de datos de las dependencias de ROS lo que va a permitir saber qué dependencia necesita cada paquete. al final verifica la versión de ROS.
  - echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
  - source ~/.bashrc
  - sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
  - sudo rosdep init
  - rosdep update
  - rosversion -d

- ***Configuración de red ROS***
  Este paso será importante para poder establecer una comunicación entre Raspberry Pi como MASTER, que será quien envíe los nodos, y el PC que será el que reciba la información de Raspberry para visualizar los mapas más adelante.
  Se edita el archivo .bashrc el cual contiene configuración del usuario. Se agregan las líneas necesarias que contienen la dirección IP de Raspberry al final del archivo y se aplican los cambios y se verifica que las variables estén configuradas.
  - nano ~/.bashrc
  - export ROS_MASTER_URI=http://10.35.96.120:11311
    export ROS_HOSTNAME=10.35.96.120

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

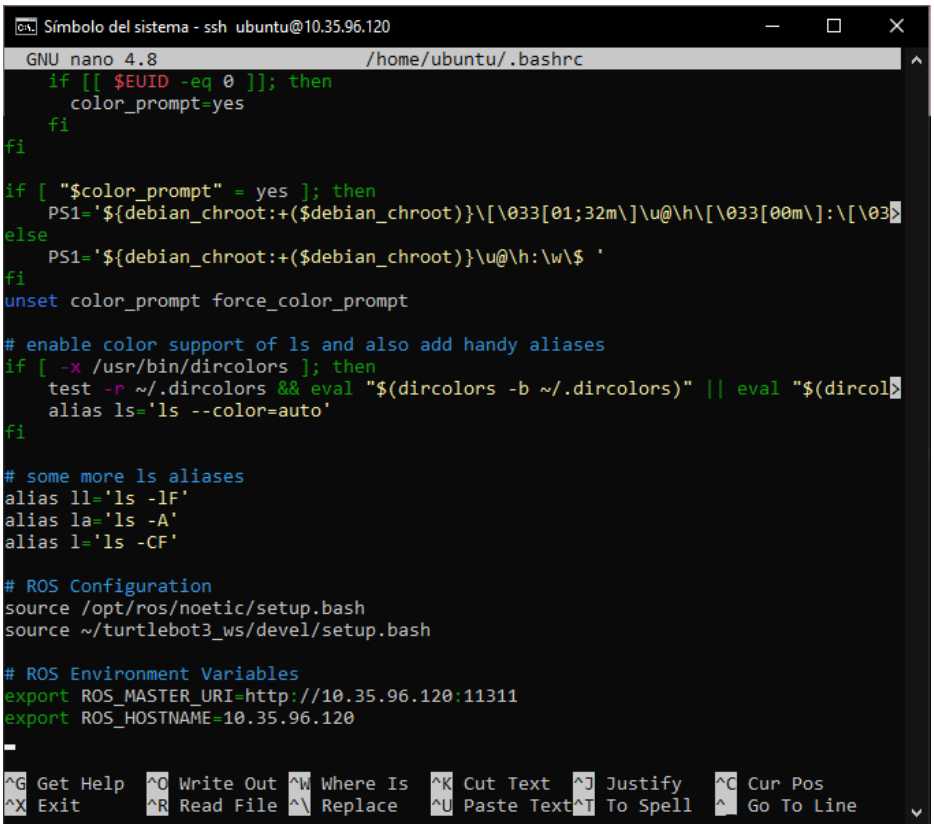


Fig. 19. Ventana nano ~/.bashrc

- Ctrl+O para guardar
- Enter para confirmar
- Ctrl+X para salir
- source ~/.bashrc
- echo $ROS_MASTER_URI
  echo $ROS_HOSTNAME

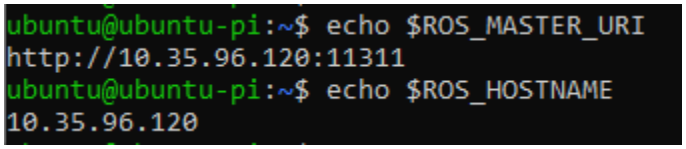


Fig. 21. Verificación de variables configuradas

- ***Crear el espacio de trabajo (catkin workspace) e instala paquetes base de TurtleBot3***
  Se crea el directorio workspace y se navega en él. Esto se hace ya que crea un directorio donde se organizan y construyen todos los paquetes. De nuevo se actualizan los repositorios, se instalan paquetes de dependencias básicas, se instala el driver del sensor LiDAR, mensajes personalizados, SDK de los motores y herramientas de teleoperación.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- mkdir -p ~/turtlebot3_ws/src
- cd ~/turtlebot3_ws
- pwd

```
ubuntu@ubuntu-pi:~$ mkdir -p ~/turtlebot3_ws/src
ubuntu@ubuntu-pi:~$ cd ~/turtlebot3_ws
ubuntu@ubuntu-pi:~/turtlebot3_ws$ pwd
/home/ubuntu/turtlebot3_ws
```

Fig. 21. Verificación directorio de Workspace

- sudo apt update
- sudo apt install python3-argcomplete python3-colcon-common-extensions libboost-system-dev build-essential
- sudo apt install ros-noetic-hls-lfcd-lds-driver
- sudo apt install ros-noetic-turtlebot3-msgs
- sudo apt install ros-noetic-dynamixel-sdk
- sudo apt install ros-noetic-joy ros-noetic-teleop-twist-keyboard ros-noetic-teleop-twist-joy

- ***Clonar los repositorios de TurtleBot3***
  En este paso se descargaran los paquetes fuente del robot TurtleBot3. Para esto es necesario navegar al directorio source del workspace, clonar los repositorios de ROBOTIS (fabricante de TurtleBot3) y del driver de sensor LiDAR, instalar dependencias de desarrollo, compilación y sistema, usar rosdep para instalar todas las dependencias automáticamente, compilar el workspace
  - cd ~/turtlebot3_ws/src
  - git clone -b noetic https://github.com/ROBOTIS-GIT/turtlebot3.git
  - git clone -b noetic https://github.com/ROBOTIS-GIT/ld08_driver.git
  - sudo apt install ros-noetic-urdf ros-noetic-urdf-tutorial ros-noetic-joint-state-publisher ros-noetic-joint-state-publisher-gui
  - sudo apt install libudev-dev libusb-dev libusb-1.0-0-dev pkg-config
  - sudo apt install libgtk-3-dev libcanberra-gtk3-module doxygen git
  - cd ~/turtlebot3_ws
    rosdep update
    rosdep install --from-paths src --ignore-src -r -y
  - cd ~/turtlebot3_ws
  - catkin_make -j1

- ***Actualizar el archivo .bashrc, configurar el modelo de TurtleBot3 y LiDAR***
  Se agrega una línea que carga el entorno del workspace compilado cada vez que se abra una terminal, lo que permite que ROS encuentre todos los paquetes compilados. Se establece el

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

modelo del robot TurtleBot3 ya que afecta parámetros como tamaño, peso, configuración de sensores. Se indica que modelo de sensor es ya que algunos paquetes SLAM requieren conocerlo para configurar algunos parámetros. Estas modificaciones se pueden verificar en el archivo nano ~/.bashrc, al final.

- echo 'source ~/turtlebot3_ws/devel/setup.bash' >> ~/.bashrc
- source ~/.bashrc
- echo $ROS_PACKAGE_PATH
- echo 'export TURTLEBOT3_MODEL=burger' >> ~/.bashrc
  source ~/.bashrc
- echo $TURTLEBOT3_MODEL
- echo 'export LDS_MODEL=LDS-01' >> ~/.bashrc
- source ~/.bashrc
- echo $LDS_MODEL

```
# TurtleBot3 Configuration
export TURTLEBOT3_MODEL=burger
export LDS_MODEL=LDS-01
source /opt/ros/noetic/setup.bash
source ~/turtlebot3_ws/devel/setup.bash
source ~/turtlebot3_ws/devel/setup.bash
```

Fig. 22. Verificación de modificaciones en nano ~/.bashrc

- ***Configurar permisos de puerto serial para OpenCR***
  El OpenCR, que es la placa controlador del robot, se conecta a la Raspberry Pi por USB y es necesario darle permisos al puerto serial. lo que crea automáticamente un archivo que permite acceder al puerto USB
  - rosrun turtlebot3_bringup create_udev_rules
  - sudo udevadm control --reload-rules
    sudo udevadm trigger

Metodología objetivo 2: Generar mapas de su entorno usando algoritmos SLAM sobre ROS

- ***Instalar paquetes SLAM***
  Se hará la instalación de los algoritmos SLAM que se van a utilizar, en este caso Gmapping, Hector SLAM y SLAM Toolbox. Se instalarán las herramientas de visualización, se verificara que ROS encuentre los paquetes y se creará el directorio para mapas.
  - sudo apt install ros-noetic-slam-gmapping
  - sudo apt install ros-noetic-hector-slam
  - sudo apt install ros-noetic-slam-toolbox
  - sudo apt install ros-noetic-rviz ros-noetic-rqt ros-noetic-rqt-common-plugins
  - rospack list | grep turtlebot3
  - rospack list | grep -E "gmapping|hector|slam_toolbox"

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
ubuntu@ubuntu-pi:~$ rospack list | grep -E "gmapping|hector|slam_toolbox"
gmapping /opt/ros/noetic/share/gmapping
hector_compressed_map_transport /opt/ros/noetic/share/hector_compressed_map_transp
ort
hector_geotiff /opt/ros/noetic/share/hector_geotiff
hector_geotiff_launch /opt/ros/noetic/share/hector_geotiff_launch
hector_geotiff_plugins /opt/ros/noetic/share/hector_geotiff_plugins
hector_imu_attitude_to_tf /opt/ros/noetic/share/hector_imu_attitude_to_tf
hector_map_server /opt/ros/noetic/share/hector_map_server
hector_map_tools /opt/ros/noetic/share/hector_map_tools
hector_mapping /opt/ros/noetic/share/hector_mapping
hector_marker_drawing /opt/ros/noetic/share/hector_marker_drawing
hector_nav_msgs /opt/ros/noetic/share/hector_nav_msgs
hector_slam_launch /opt/ros/noetic/share/hector_slam_launch
hector_trajectory_server /opt/ros/noetic/share/hector_trajectory_server
openslam_gmapping /opt/ros/noetic/share/openslam_gmapping
slam_toolbox /opt/ros/noetic/share/slam_toolbox
slam_toolbox_msgs /opt/ros/noetic/share/slam_toolbox_msgs
```

Fig. 23. Verificación que ROS encuentra paquetes SLAM

- echo "TURTLEBOT3_MODEL: $TURTLEBOT3_MODEL"
  echo "LDS_MODEL: $LDS_MODEL"
- ls -la ~/turtlebot3_ws/
- rospack find turtlebot3_bringup
- mkdir -p ~/maps/gmapping ~/maps/hector_slam ~/maps/slam_toolbox

- ***Visualización remota***
  Instalar la herramienta MobaXterm que permitirá ver los mapas en RViz, esto se hará ya que Raspberry Pi ejecuta ROS pero no tiene pantalla, RViz es una herramienta gráfica que necesita mostrar interfaz.
  - Para hacer la descarga, en el link https://mobaxterm.mobatek.net/download.html, la version “Portable edition”, extraer el archivo ZIP y abrir MobaXterm.exe

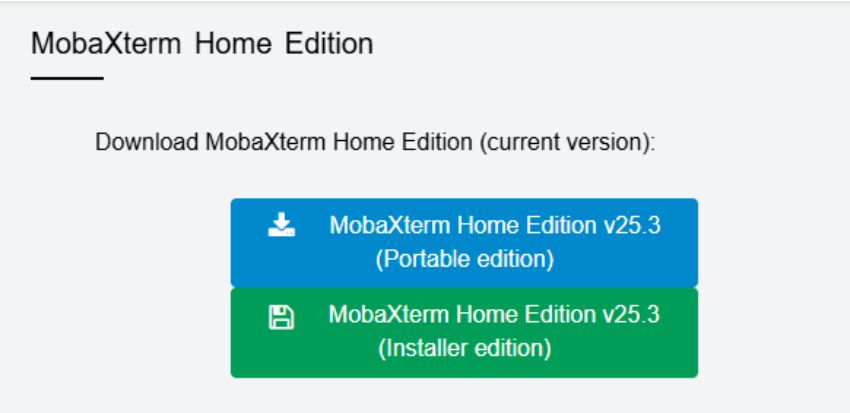


Fig. 24. Página principal de MobaXterm para descarga

- Después de abrir MobaXterm.exe, se debe crear una sesión SSH. Hacer clic en “Session” (botón arriba a la izquierda), seleccionar SSH y rellenar

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

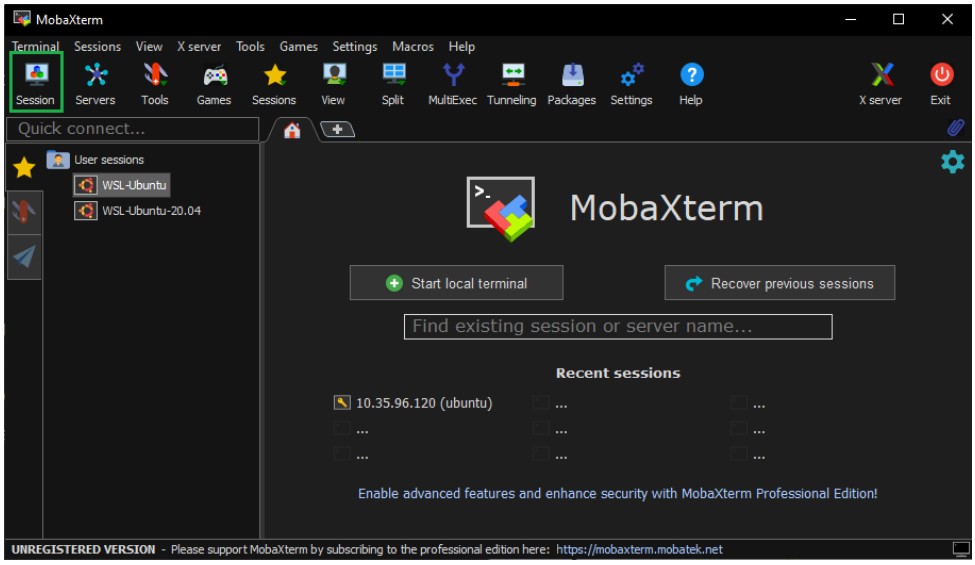


Fig. 25. Ventana principal de MobaXterm

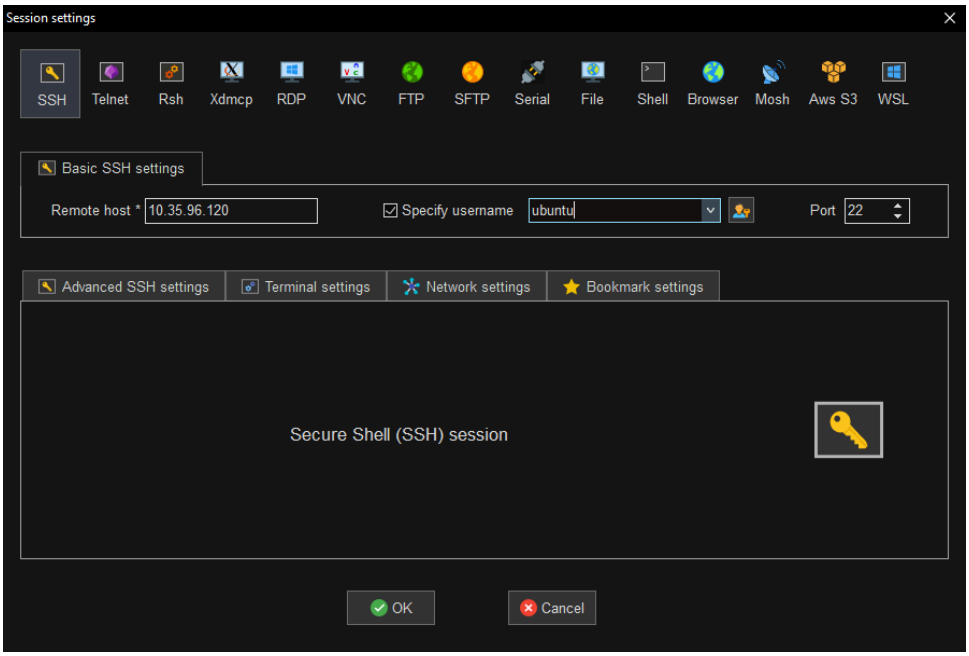


Fig. 26. Creación de sesion nueva en MobaXterm

- Después de esto pedirá la contraseña para acceder y se tendrá una terminal abierta en MobaXterm conectada a Raspberry Pi. En terminal de Símbolo de Sistema lanzar roscore para que se pueda hacer el vínculo entre Raspberry y MobaXterm y se pueda abrir RViz. Verificar que X11 está disponible. Lanzar RViz.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

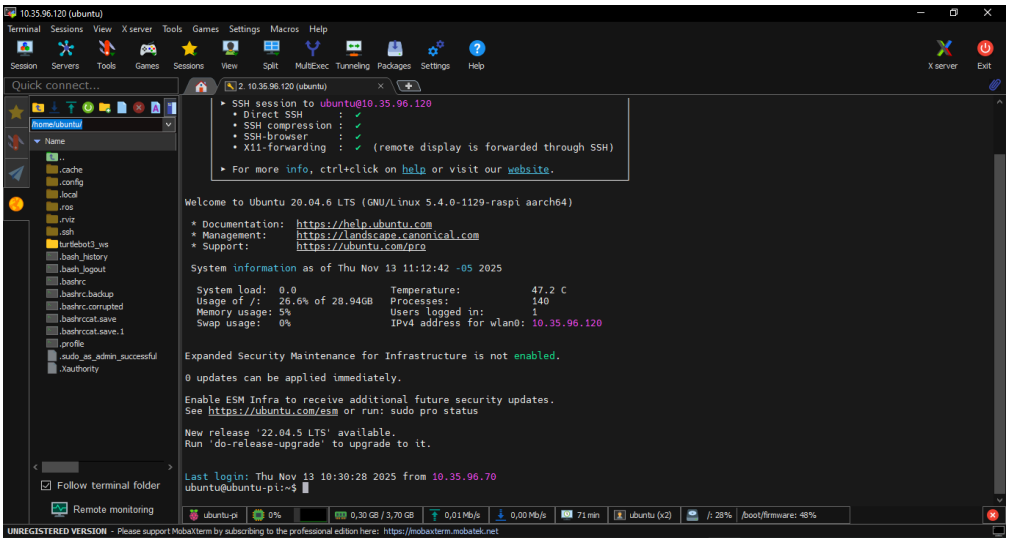


Fig. 27. Ventana principal con sesión iniciada SSH en MobaXterm

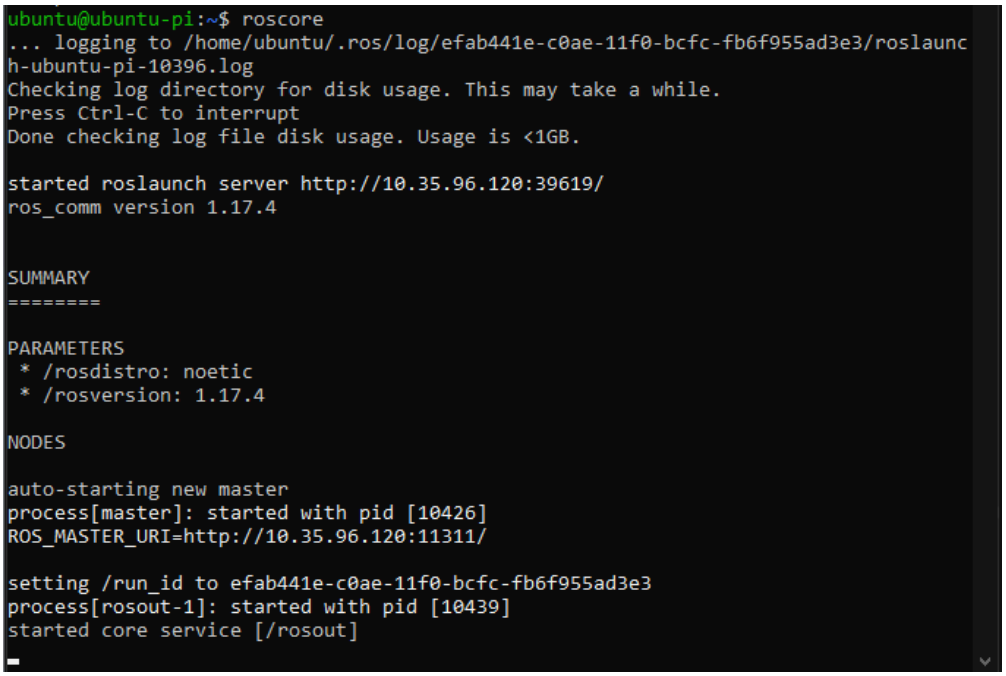


Fig. 28. Ventana de Simbolo de Sistema con roscore corriendo

- echo $DISPLAY
- localhost:10.0 (esto es lo que debería aparecer)
- rviz

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

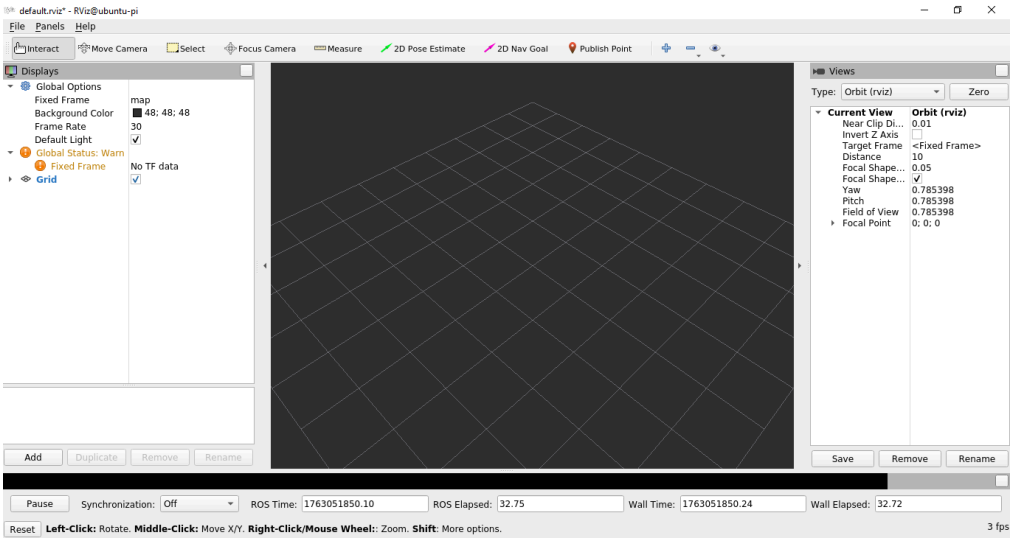


Fig. 29. Ventana inicial de RViz

- ***Visualización de sensor***

  A partir de este momento se hará por terminales de Símbolo de sistema. Se harán para iniciar los servicios centrales de ROS, necesarios para que los nodos puedan comunicarse entre si. Ver el escaneo de LiDAR en tiempo real.
  - Terminal 1: roscore (ya corriendo)
  - Terminal 2: roslaunch turtlebot3_bringup turtlebot3_robot.launch

  Se hace una configuración en RViz para poder visualizar los datos del sensor, esto se hace en la parte inferior izquierda en el botón “Add”, seleccionar “LaserScan”, en By Topic seleccionar “/scan” y luego “LaserScan”. Presionar OK. Esto permitirá visualizar unos puntos de colores que estarían detectando obstáculos. Para verlo un poco mejor, el robot se pondrá en diferentes posiciones para que detecte diferentes obstáculos.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

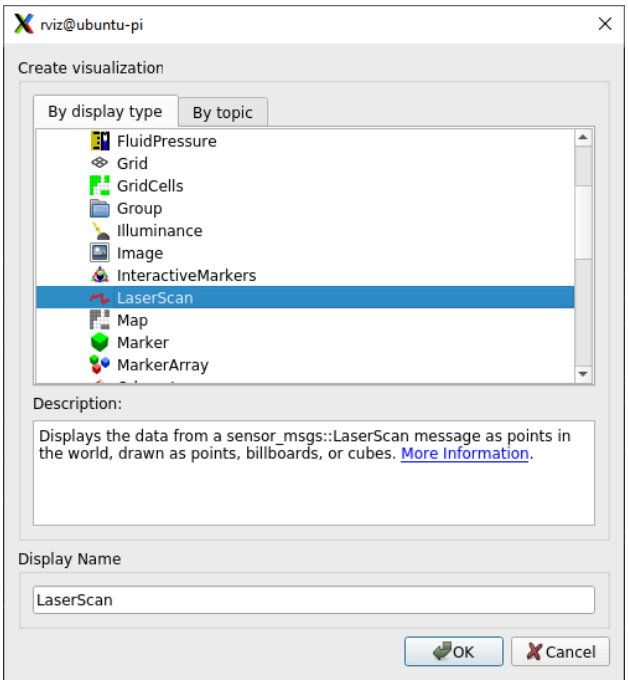


Fig. 30. Configuración de sensor en RViz

Luego el panel izquierdo debe verse de la siguiente manera

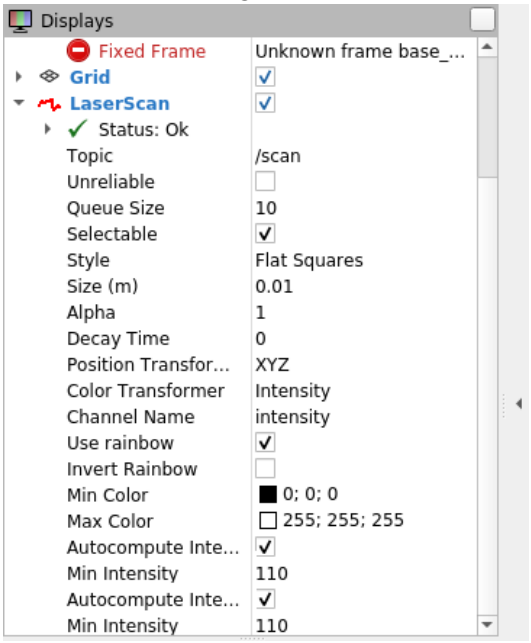


Fig. 31. Parámetros finales de sensor en RViz

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

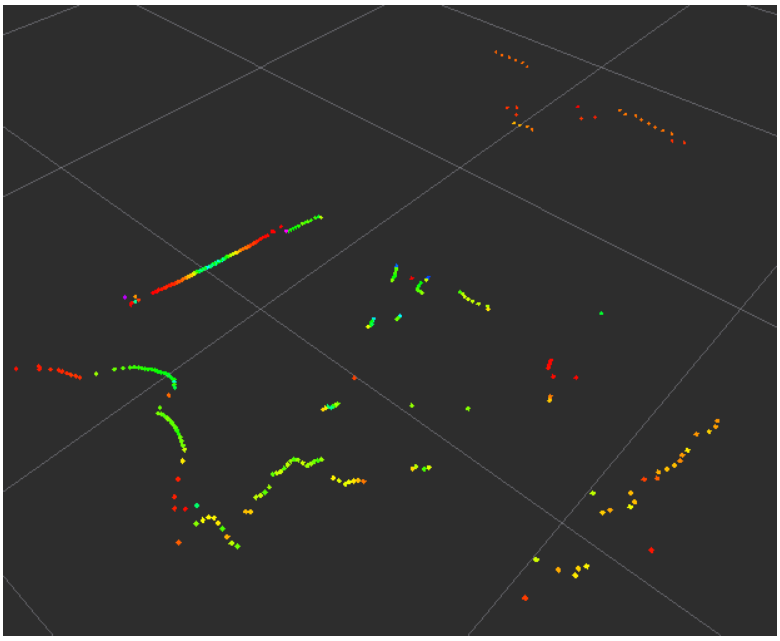


Fig. 32. Obstáculos detectados en RViz. Posición 1

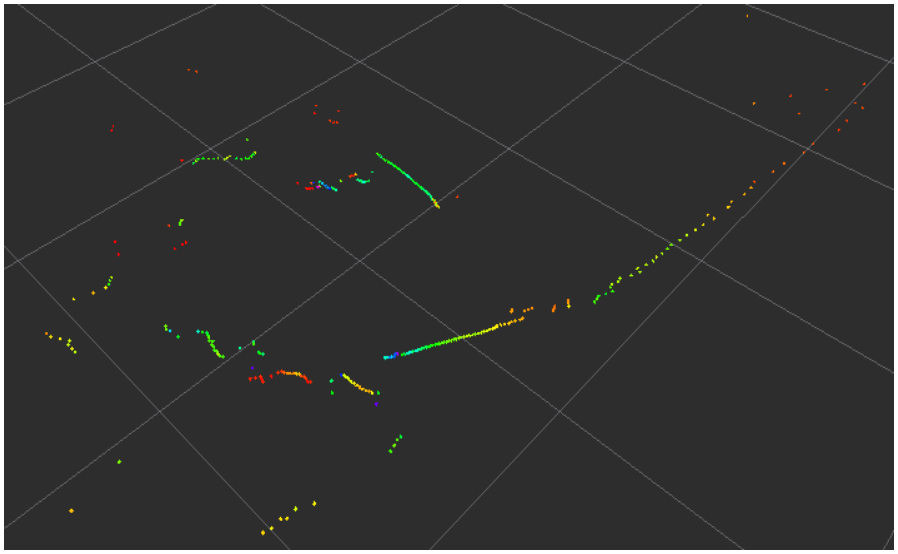


Fig. 33. Obstáculos detectados en RViz. Posición 2

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

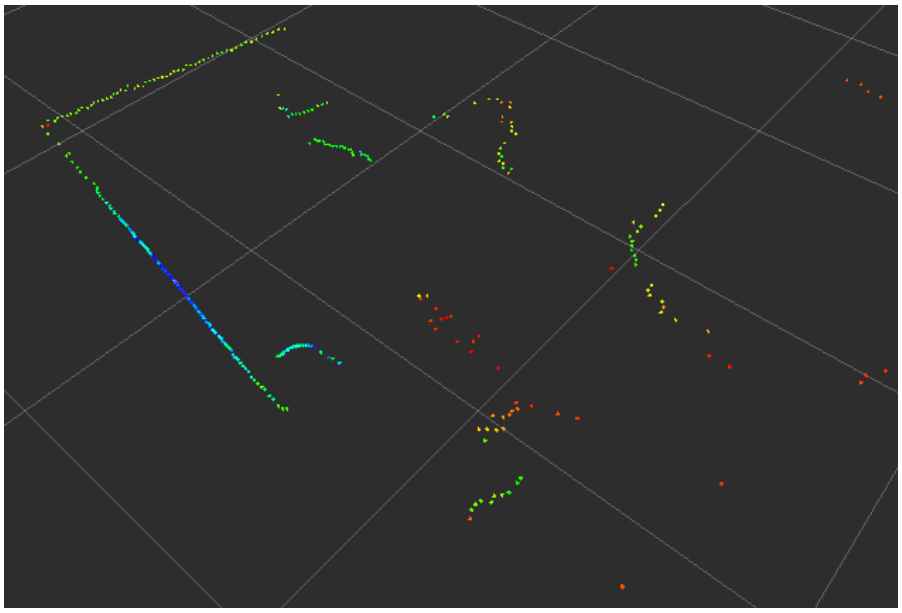


Fig. 34. Obstáculos detectados en RViz. Posición 3

- ***Teleoperación***
  Para este paso es necesario abrir otra terminal. Esto lo que hará es que el robot se pueda mover por medio de comandos del teclado w, a, s, d, x (adelante, izquierda, atrás, derecha y stop)
  - Terminal 3: rospack find turtlebot3_teleop
    roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

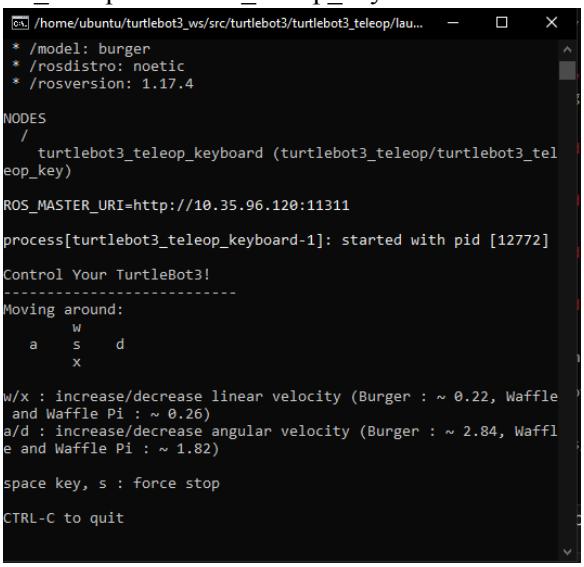


Fig. 35. Terminal de Teleoperación

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

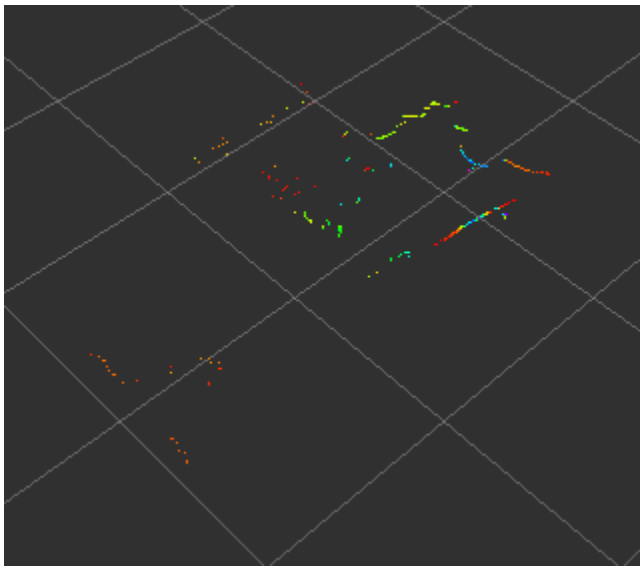


Fig. 36. Posición inicial de robot. Deteccion de obstaculos en RViz

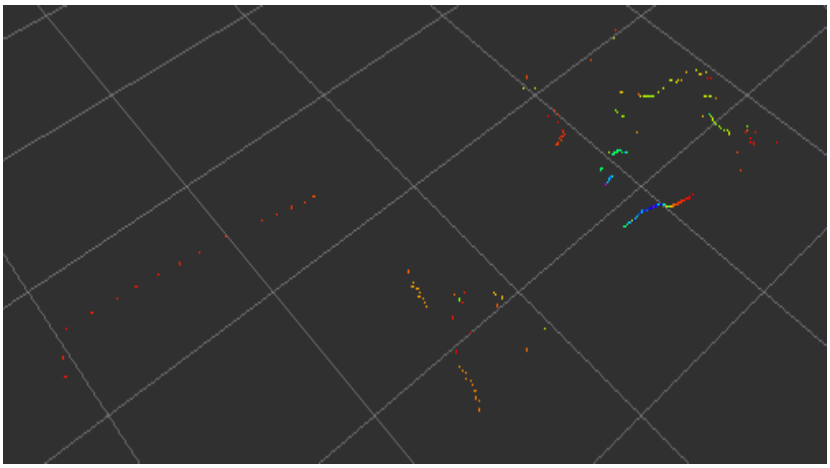


Fig. 37. Posición de avance. Deteccion de obstaculos en RViz

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

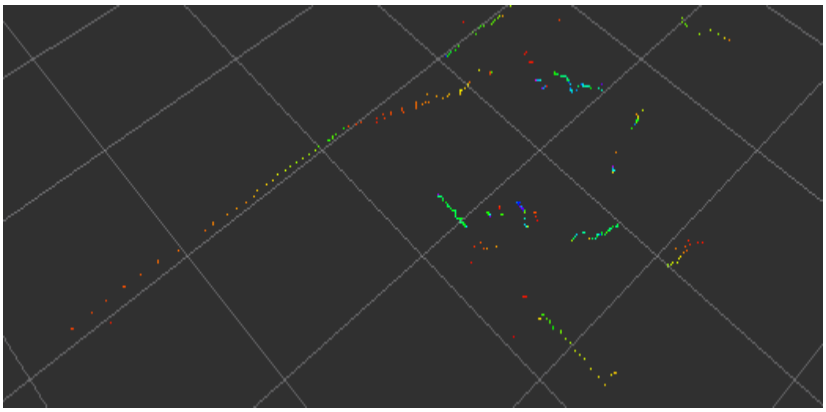


Fig. 38. Posición de avance. Deteccion de obstaculos en RViz

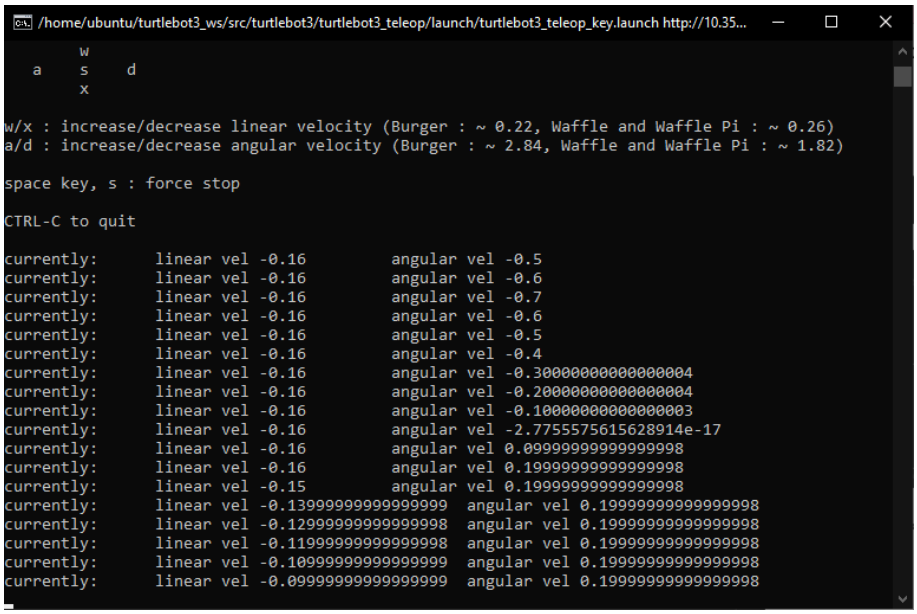


Fig. 39. Terminal de teleoperación al presionar las teclas de avance de robot

- ***Visualización de mapas***
  Para esto es necesario crear un archivo launch en la ubicación correcta, el contenido del archivo estara en anexos. Después se lanza el algoritmo para visualizar el mapa. Es necesario iniciar primero el algoritmo SLAM, lo que hará que se empiece a publicar el mapa en RViz. Se hace la configuración en “Add”, “Map” en By Topic seleccionar /map.
  - cd ~/turtlebot3_ws/src/turtlebot3/turtlebot3_slam/launch
  - nano turtlebot3_slam_no_rviz.launch
  - (Contenido en anexo A)
  - Ctrl+O
  - Enter
  - Ctrl+X

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Después de esto se hace el lanzamiento del algoritmo SLAM. Están listados los tres comandos para cada algoritmo.

- Terminal 4: cd ~/turtlebot3_ws/src/turtlebot3/turtlebot3_slam/launch
- roslaunch turtlebot3_slam turtlebot3_slam_no_rviz.launch slam_methods:=gmapping

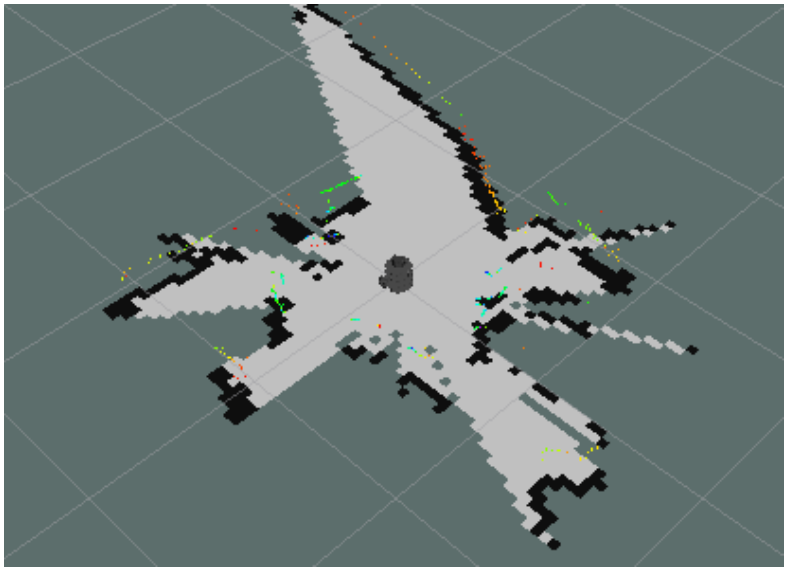


Fig. 40. Primer visualización del mapa con algoritmo Gmapping sin teleoperación

- roslaunch turtlebot3_slam turtlebot3_slam_no_rviz.launch slam_methods:=hector

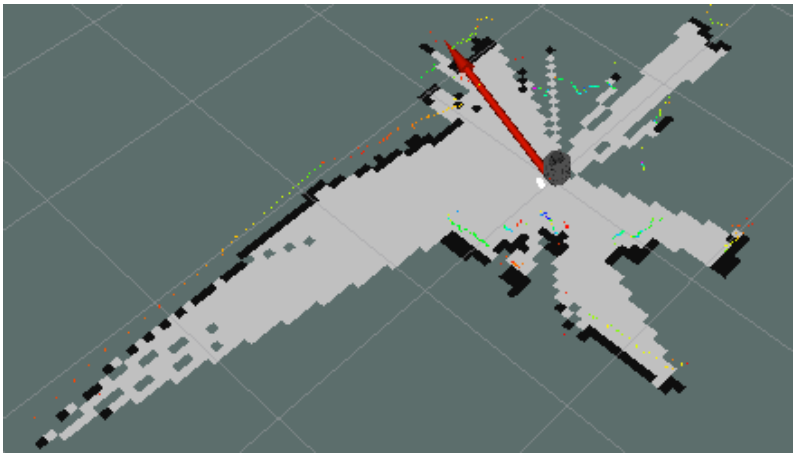


Fig. 41. Primer visualización del mapa con algoritmo Hector SLAM sin teleoperación

- roslaunch turtlebot3_slam turtlebot3_slam_no_rviz.launch slam_methods:=slam_toolbox

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

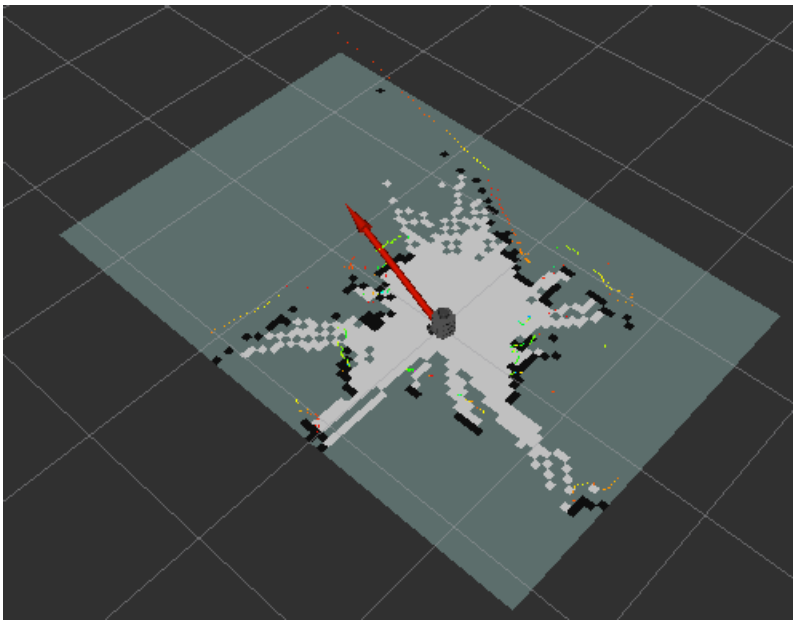


Fig. 42. Primer visualización del mapa con algoritmo SLAM Toolbox sin teleoperación

Metodología objetivo 3: Realizar un protocolo de pruebas y analizar los resultados obtenidos de la solución específica usando 3 topologías del algoritmo SLAM sobre ROS.

- ***Diseño de protocolo de pruebas experimentales***
  Se definieron tres entornos universitarios distintos para validar la robustez de cada algoritmo bajo condiciones variadas, se trasladó el TurtleBot 3 al espacio universitario para validar su desempeño en escenarios reales
  - Entorno 1: Pasillos largos y rectos

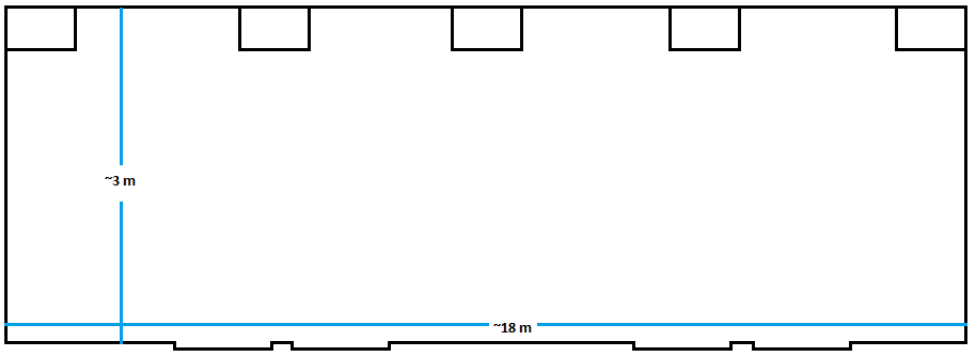


Fig. 43. Plano de pasillo Bloque 1- 2- 3- 4, segundo piso

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.



Fig. 44. Robot TurtleBot3 Realizando pruebas en pasillo



Fig. 45 Robot TurtleBot3 Realizando pruebas en pasillo

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

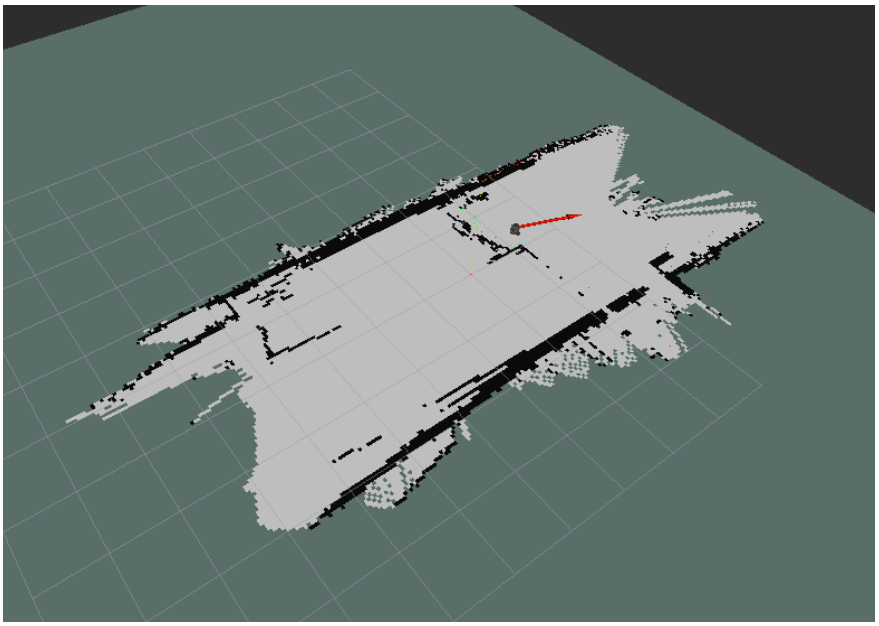


Fig. 46. Plano tomado con algoritmo Gmapping en pasillo

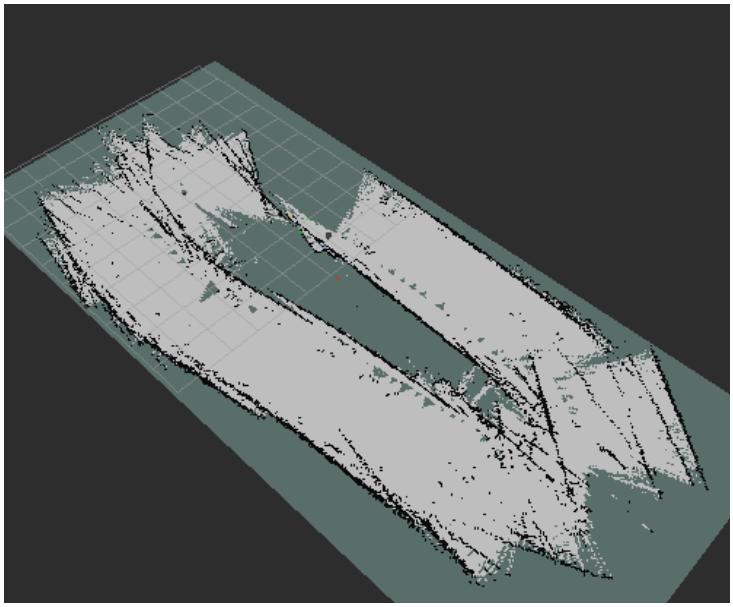


Fig. 47. Plano tomado con algoritmo SLAM Toolbox en pasillo

- Entorno 2: Salones con obstáculos estáticos como pupitres, se ubicaron para simular obstáculos de diferente tamaño.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

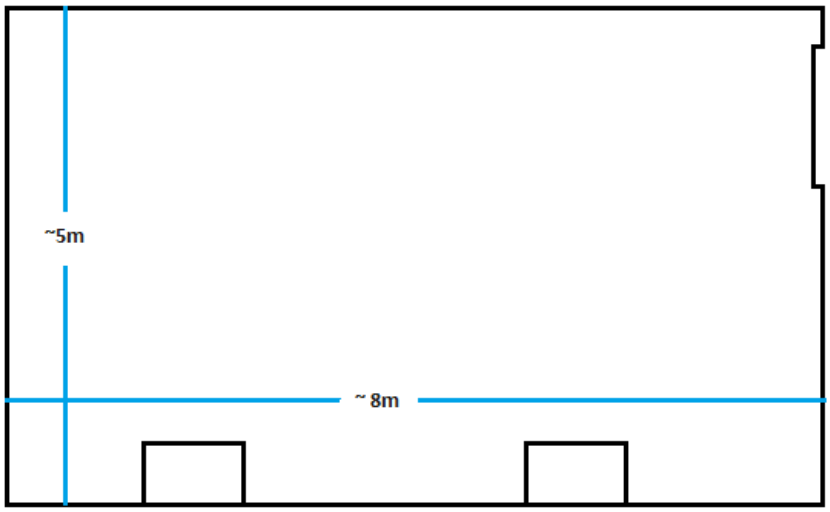


Fig. 48. Plano salon Bloque 3- 203



Fig. 49. Robot TurtleBot3 Realizando pruebas en salon

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

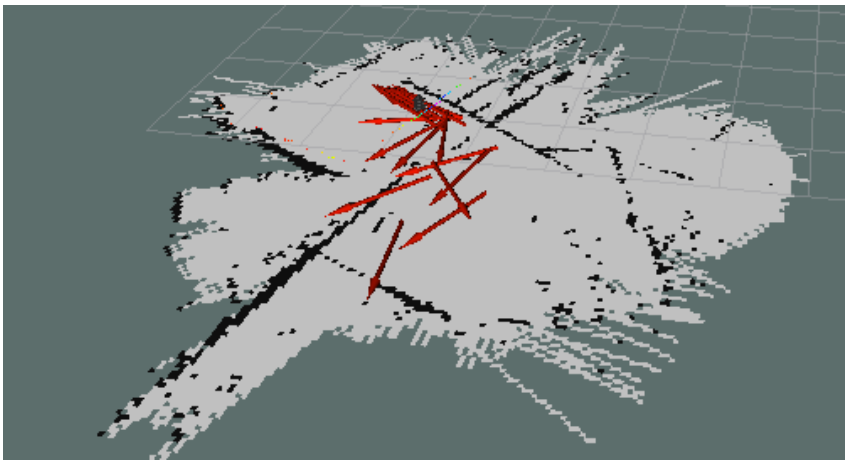


Fig. 50. Plano tomado con algoritmo Gmapping en salon

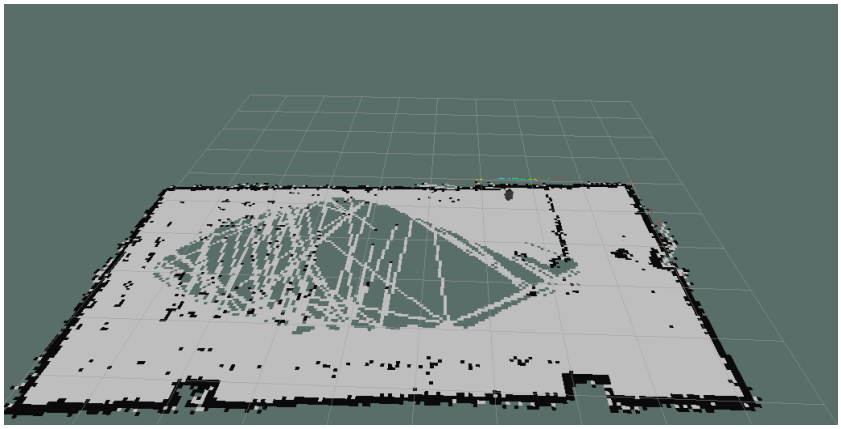


Fig. 51. Plano tomado con algoritmo Hector SLAM en salon

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

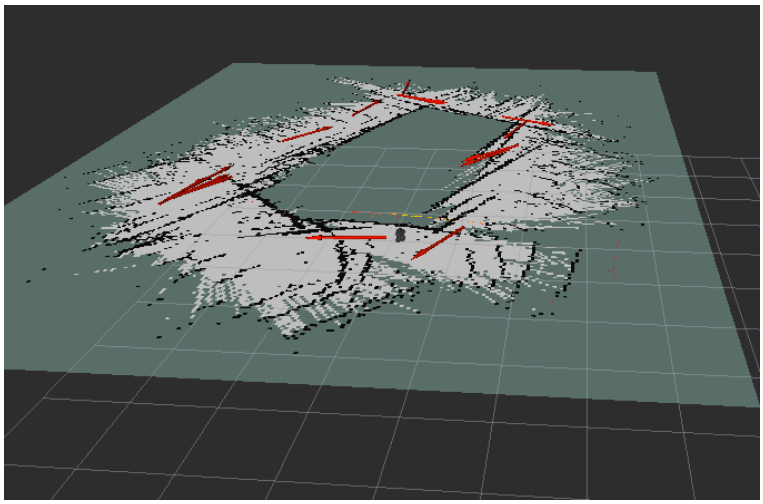


Fig. 52. Plano tomado con algoritmo SLAM Toolbox en salon

- Entorno 3: Áreas comunes con tráfico peatonal ocasional: para medir la capacidad para evitar obstáculos y su capacidad de autoajuste ante cambios de iluminación por las características del sensor infrarrojo.

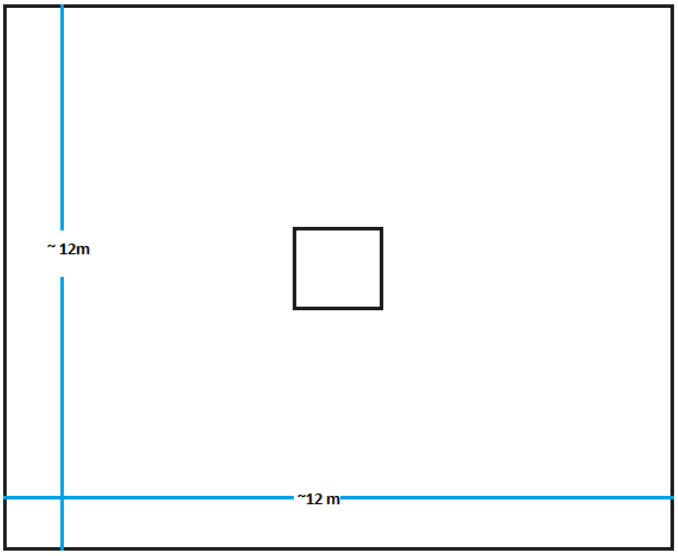


Fig. 53. Plano Espacio abierto antigua media torta

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.



Fig. 54. Robot TurtleBot3 Realizando pruebas en áreas comunes

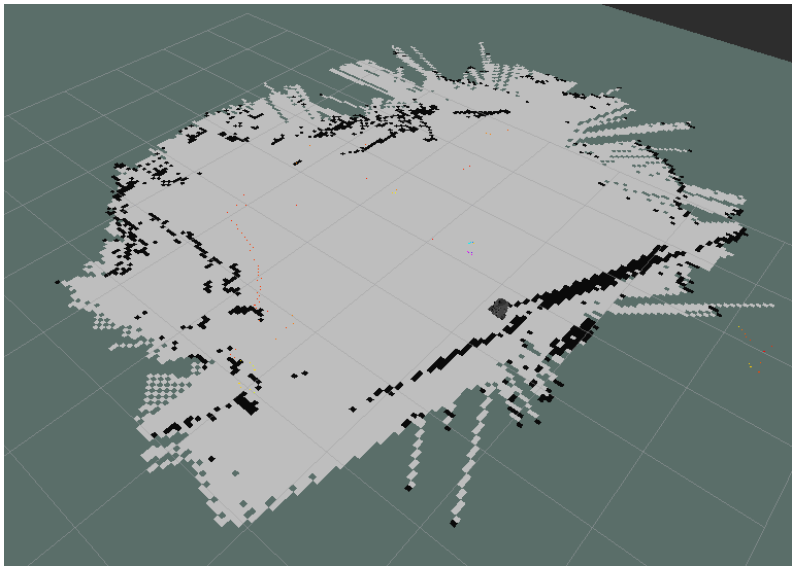


Fig. 55. Plano tomado con algoritmo Gmapping en espacio abierto

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

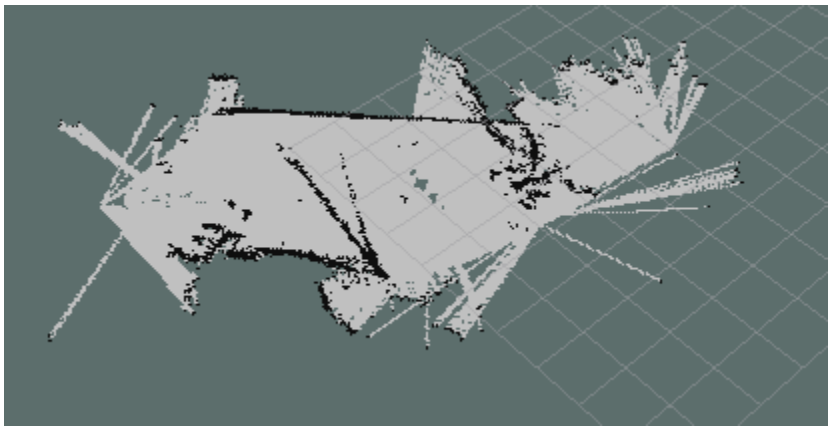


Fig. 56. Plano tomado con algoritmo Hector SLAM en espacio abierto

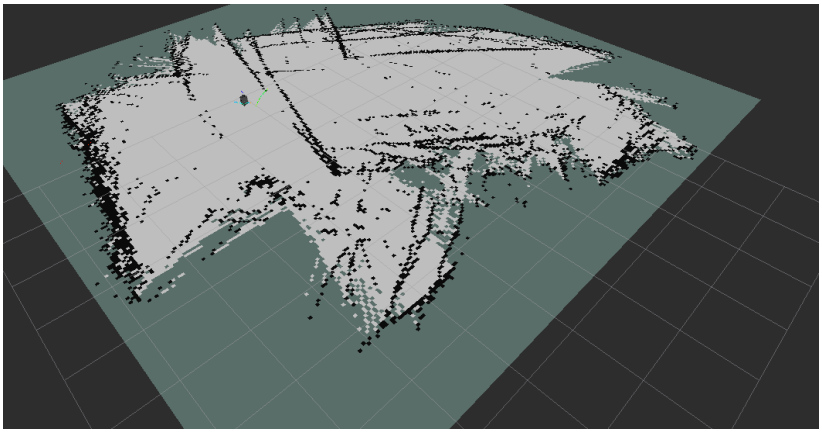


Fig. 57. Plano tomado con algoritmo SLAM Toolbox en espacio abierto

- ***Preparación inicial para navegación autónoma***
  Se instalarán las dependencias necesarias que descargan los paquetes de ROS necesarios para navegación autónoma como move_base que es para planificación y control de rutas, amcl para localización en mapa, map_server para publicar mapas pregrabados y planificadores y costmapas. Verificar instalación y crear estructura de directorios en workspace
  - sudo apt install -y ros-noetic-move-base ros-noetic-move-base-msgs
    sudo apt install -y ros-noetic-amcl ros-noetic-map-server
    sudo apt install -y ros-noetic-costmap-2d ros-noetic-nav-core
    sudo apt install -y ros-noetic-navfn ros-noetic-base-local-planner
    sudo apt install -y ros-noetic-dwa-local-planner ros-noetic-clear-costmap-recovery
  - dpkg -l | grep ros-noetic-move-base
  - mkdir -p ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
mkdir -p ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/launch
mkdir -p ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/maps
mkdir -p ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/scripts
```

- ***Crear mapa base con SLAM***
  Teniendo las diferentes 4 terminales corriendo y RViz ejecutándose en MobaXterm, en una quinta terminal se guardará el mapa que se ha generado, se crean dos archivos .pgm que es la imagen del mapa y .yaml que es la configuración y se puede verificar los archivos creados.
  - Terminal 1: roscore
  - Terminal 2: roslaunch turtlebot3_bringup turtlebot3_robot.launch
  - Terminal 3: rospack find turtlebot3_teleop
    roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
  - Terminal 4: cd ~/turtlebot3_ws/src/turtlebot3/turtlebot3_slam/launch
    roslaunch turtlebot3_slam turtlebot3_slam_no_rviz.launch
    slam_methods:=gmapping
  - Terminal 5: rosrun map_server map_saver -f
    ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/maps/turtlebot3_map
  - ls -la ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/maps/

```
-rw-rw-r-- 1 ubuntu ubuntu 147512 Nov 10 12:46 map.pgm
-rw-rw-r-- 1 ubuntu ubuntu    133 Nov 10 12:46 map.yaml
```

Fig. 58. Verificación creación de archivos de plano

- ***Configurar archivos de navegación***
  Se crearán diferentes archivos .yaml, los cuales trabajan juntos para que el TurtleBot3 navegue de manera autónoma y segura en un mapa conocido, interprete correctamente los obstáculos, seleccione y siga las rutas y recupere su localización en ambientes reales complejos.
  - turtlebot3_map.yaml: Sirve para indicar la ubicación y parámetros de la imagen .pgm del mapa, define escalas y permiten navegación en entorno conocido.
  - costmap_common_params.yaml: Configura los parámetros comunes, define el rango de sensores, configura como el robot percibe el entorno.
  - global_costmap_params.yaml: Permite planificar rutas usando el mapa cargado, determina que capas y algoritmos usarán para calcular el mapa de obstáculos global, establece el frame global, frame base y frecuencia de actualización.
  - local_costmap_params.yaml: Se permite a Move Base reaccionar a obstáculos nuevos y dinámicos en tiempo real, la ventana local se mueve junto al robot y evita colisiones.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- dwa_local_planner_params.yaml: Determinan límites de velocidad y aceleración, ajusta tolerancias al alcanzar la meta e influye directamente en cómo el robot maniobra en espacios reducidos, hace curvas y rodea objetos.
- move_base_params.yaml: Ajusta parámetros generales del nodo move_base, define frecuencias de los controladores, tiempos de espera y mejora estabilidad y capacidad de recuperación.
- amcl_params.yaml: Determina cómo se realiza la localización del robot en el mapa, ajusta el número de partículas, la frecuencia de actualización y como se interpreta el ruido en los sensores y odometría. Sin esta localización, el robot no puede ubicarse ni navegar correctamente en el mapa guardado.

Los archivos se crean de la siguiente manera, el contenido de cada uno se encontrara en los anexos

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/maps/turtlebot3_map.yaml
- Anexo B
- Ctrl+O, Enter, Ctrl+X

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config/costmap_common_params.yaml
- Anexo C
- Ctrl+O, Enter, Ctrl+X

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config/global_costmap_params.yaml
- Anexo D
- Ctrl+O, Enter, Ctrl+X

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config/local_costmap_params.yaml
- Anexo E
- Ctrl+O, Enter, Ctrl+X

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config/dwa_local_planner_params.yaml

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Anexo F
- Ctrl+O, Enter, Ctrl+X

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config/dwa_local_planner_params.yaml
- Anexo G
- Ctrl+O, Enter, Ctrl+X

- nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/config/amcl_params.yaml
- Anexo H
- Ctrl+O, Enter, Ctrl+X

- ***Creación de archivo Launch para navegación***
  Tiene la función de iniciar de manera coordinada todos los nodos y configuraciones necesarias para la navegación autónoma con TurtleBot3.
  Facilita el arranque único y organizado de l sistema completo de navegación autónoma, carga el mapa estático guardado para que el robot conozca su entorno, lanza el nodo de localización (AMCL) que permite al robot ubicarse, inicializa el stack de navegacion mobe_base para planificar y ejecutar rutas para llegar a metas, carga parámetros necesarios desde archivos yaml, publica la descripción del robot y el estado para que los nodos sepan la configuración del TurtleBot3 y gestiona la comunicación correcta de todos estos nodos y topics dentro e ROS para que funcionen en conjunto.
  - nano ~/turtlebot3_ws/src/turtlebot3/turtlebot3_navigation/launch/turtlebot3_navigation.launch
  - Anexo I
  - Ctrl+O, Enter, Ctrl+X

- ***Análisis de Métricas***
  Como el robot TurtleBot 3 explora un entorno desconocido usando navegación autónoma, es importante saber que tan bien está funcionando. Con las métricas se puede medir el desempeño de los algoritmos SLAM con métricas específicas. Se medirá la precisión con la que el robot estima pequeños movimientos locales en intervalos de tiempo, calcular la similitud entre mapas para cada algoritmo y calcular la precisión de los mismos basándose en la tasa de aciertos y errores en la ocupación del espacio también se va a medir cuántas veces por segundo el algoritmo SLAM actualiza y reconstruye el mapa, y los recursos computacionales que consume cada algoritmo.
  1. Error Absoluto de Trayectoria (ATE - Absolute Trajectory Error)
     Mide cuánto se desvía la trayectoria estimada por SLAM respecto a la trayectoria

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

real del robot. Es el error global de posicionamiento.

Parámetro principal:

RMSE (Root Mean Square Error): $ATE=\sqrt{\frac{1}{n}\sum_{i=1}^{n} ||P_{slami} - P_{gti}||}$

- Determina la precisión general del algoritmo SLAM.
- Comparar el desempeño entre diferentes algoritmos.
- Detectar si el robot “se pierde” durante la navegación.
- Validar la calidad del mapeo en diferentes entornos.

Interpretación de resultados

- ATE <0.1 m: Excelente precisión
- ATE 0.1-0.2 m: Buena precisión
- ATE 0.2-0.5 m: Aceptable
- ATE >0.5 M: Baja precisión (revisar configuración)

Datos necesarios para calcular ATE

- Se necesitan dos archivos CSV: poses_slam_TIMESTAMP.csv) y poses_gt_TIMESTAMP.csv que se crean a medida que se van tomando datos de navegación.
- Se crea script para calcular ATE
    + nano ~/turtlebot3_metricas/scripts/calcular_ate.py
    + Anexo J
    + Ctrl+O, Enter, Ctrl+X

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

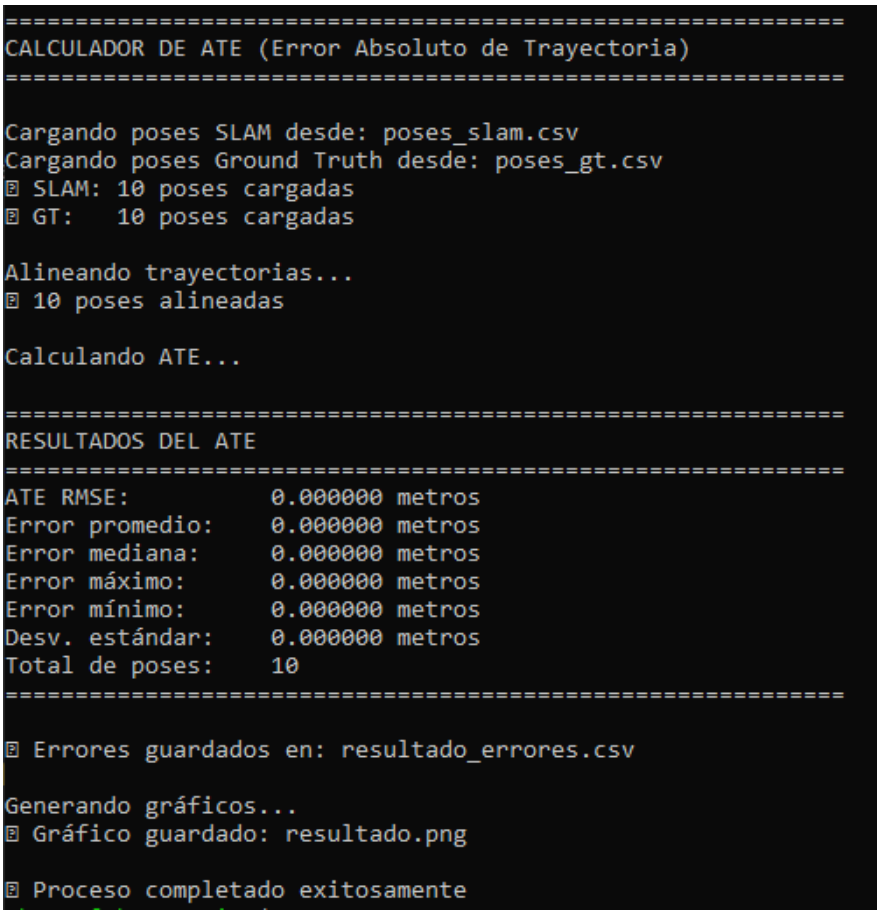


Fig. 59. Verificación datos ATE

2. Error Relativo de Pose (RPE- Relative Pose Error)
   Mide la precisión de los movimientos relativos del robot. En lugar de comparar la trayectoria global, compara los desplazamientos entre dos poses consecutivas..
   Parámetro principal:
   - Intervalo de distancia: 0.5 m (distancia acumulada entre poses para calcular error relativo)
   - Error relativo: $||(Pj - Pi)_{slam} - (Pj - Pi)_{gt}||$
   - Detecta problemas locales de odometría.
   - Evaluar la consistencia momento a momento
   - Identificar periodos donde el robot se comporta bien vs. mal.
   - Complementar información ATE

   Interpretación de resultados:
   - RPE promedio <0.05 m: Excelente.
   - RPE promedio 0.05-0.1 m: Bueno.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- RPE promedio 0.1-0.2 m: Aceptable.
- RPE promedio>0.2 m: Necesita mejora

Script para calcular RPE
- nano ~/turtlebot3_metricas/scripts/calcular_rpe.py
- Anexo K
- Ctrl+O, Enter, Ctrl+X

3. Frecuencia de actualización del mapa
   Es el número de veces por segundo que el algoritmo SLAM recomputa y actualiza el mapa (se mide en Hertz -Hz)

   Parámetros principales:
   - Frecuencia instantánea: $f=1/\Delta t$, donde Δt es el intervalo entre mapas.
   - Frecuencia promedio: promedio de todas las frecuencias instantáneas.
   - Rango: frecuencia mínima y máxima observada.
   - Se evalúa la velocidad de procesamiento del algoritmo.
   - Determina si el robot puede mantener navegación en tiempo real
   - Comparar eficiencia computacional entre algoritmos.
   - Identifica periodos de bajo/alto procesamiento.

   Interpretación de resultados:
   - f >20 Hz: Muy rapido (excelente para navegación en tiempo real).
   - f 10-20 Hz: Rápido (bueno para navegación).
   - f 5-10 Hz: Moderado (aceptable).
   - f <5 Hz: Lento (puede afectar navegación).

   Script para capturar y procesar frecuencia de mapa
   - nano ~/turtlebot3_metricas/scripts/capturador_frecuencia.py
   - Anexo L
   - Ctrl+O, Enter, Ctrl+X

   Archivo procesador de frecuencia
   - nano ~/turtlebot3_metricas/scripts/procesar_frecuencia.py
   - Anexo M
   - Ctrl+O, Enter, Ctrl+X

4. Uso de CPU
   Es el porcentaje de capacidad del procesador que está siendo utilizado por el proceso SLAM.

   Parámetros principales:
   - CPU actual (%): Porcentaje de uso en un momento específico.
   - CPU promedio (%): Promedio durante la sesión.
   - Picos de CPU (%): Máximo observado.
   - Mínimo de CPU (%): Mínimo observado.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Se verifica si el algoritmo es viable en hardware limitado
- Detecta si hay suficientes recursos para ejecutar otros procesos.
- Identifica periodos de bajo/alto procesamiento.
- Comparar eficiencia entre algoritmos.

Interpretación (en Raspberry Pi 4 - 4 nucleos):
- CPU <25%: Muy bajo, (algoritmo muy eficiente).
- CPU 25-50%: Bajo (bueno, hay recursos disponibles).
- CPU 50-75%: Moderado
- CPU >75%: Alto (riesgo de degradación de desempeño)
- CPU >90%: Muy alto (puede causar ralentización)

Script para capturar y procesar CPU
- nano ~/turtlebot3_metricas/scripts/capturador_cpu.py
- Anexo N
- Ctrl+O, Enter, Ctrl+X

Archivo procesador de CPU
- nano ~/turtlebot3_metricas/scripts/procesar_cpu.py
- Anexo O
- Ctrl+O, Enter, Ctrl+X

- ***Visualización de métricas en tiempo real***

  Monitoreo en vivo
  - Terminal 1: roscore
  - Terminal 2: cd ~/turtlebot3_ws/src/turtlebot3/turtlebot3_slam/launch
    roslaunch turtlebot3_slam turtlebot3_slam_no_rviz.launch
    slam_methods:=gmapping
  - Terminal 3: Capturador de poses:
    cd ~/turtlebot3_metricas/scripts
    python3 capturador_poses.py

```
Archivos de captura creados:
  - datos_metricas/poses_slam_20251114_125832.csv
  - datos_metricas/poses_odom_20251114_125832.csv

Esperando datos de ROS...
  Presiona Ctrl+C para detener la captura

^C

Captura completada
  Total poses SLAM: 0
  Total poses Odom: 0

Archivos guardados en directorio 'datos_metricas/'
```

Fig. 60. Verificación capturador de poses

  - Terminal 4: Capturador de frecuencia

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

cd ~/turtlebot3_metricas/scripts
python3 capturador_frecuencia.py

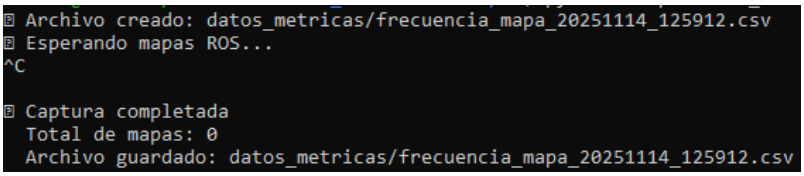


Fig. 61. Verificación capturador de frecuencia

- Terminal 5: Capturador de CPU
  cd ~/turtlebot3_metricas/scripts
  python3 capturador_cpu.py 600

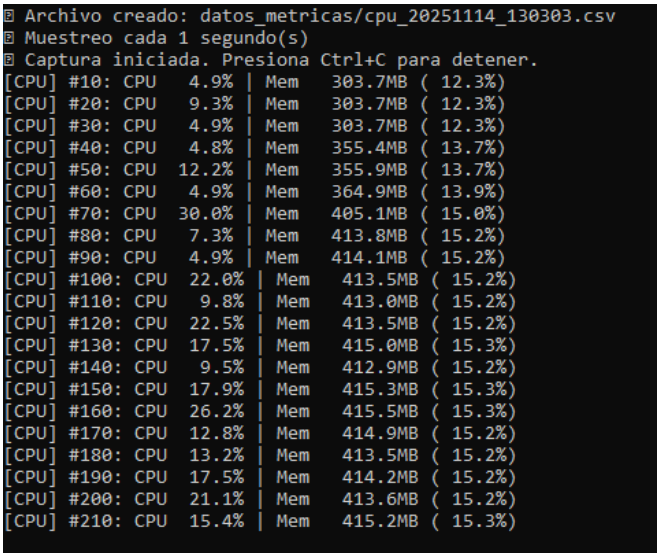


Fig. 61. Verificación capturador de CPU

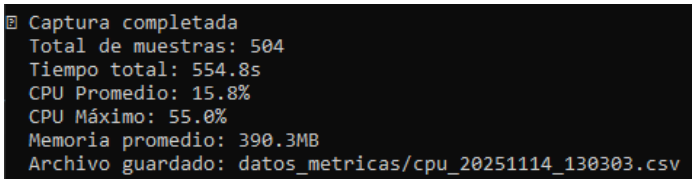


Fig. 61. Datos finales CPU

- Terminal 6: rospack find turtlebot3_teleop
  roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

RESULTADOS

Se presenta una evaluación comprensiva del proyecto "Aplicación de Algoritmos de Navegación Autónoma en Entornos Estáticos Desconocidos para el Robot TurtleBot 3 sobre ROS". A través de la implementación de tres algoritmos SLAM distintos (Gmapping, Hector SLAM y SLAM Toolbox) en diferentes entornos, se validó la capacidad del sistema para realizar navegación autónoma en escenarios reales.

Objetivo 1: Configurar ROS sobre la plataforma robótica TurtleBot3.
El cumplimiento de este objetivo constituye el fundamento sobre el cual se sustentó la totalidad del proyecto. La configuración exitosa de ROS en la plataforma TurtleBot3 fue una tarea crítica que determinó la viabilidad técnica de todas las fases subsecuentes.

Se cumplio:

- Funcionalidad de hardware: Los sensores reportan datos válidos continuamente. Las ruedas responden comandos de velocidad instantáneamente. La comunicación inalámbrica mantiene conexión sin interrupciones.
- Software operacional: ROS Noetic se ejecuta sin errores de dependencias. Los paquetes están compilados y disponibles. Los nodos se lanzan sin excepciones.
- Comunicación funciona: La red ROS permite que un proceso publicador (nodo SLAM) envíe datos (mapas, poses, escaneos) y que un proceso suscriptor (RViz) reciba datos con latencia aceptable. Los tópicos están correctamente mapeados (/scan, /odom, /slam_out_pose
- Estabilidad: El sistema funcionó sin interrupciones durante toda la fase de experimentación. No se registraron caídas de ROS, pérdidas de conexión SSH, o reinicios inesperados del robot.

Objetivo 2: Generar mapas de su entorno usando algoritmos SLAM sobre ROS.
La generación exitosa de mapas mediante tres algoritmos SLAM distintos en tres entornos diferentes representa el núcleo técnico del proyecto. Este objetivo va más allá de simplemente "crear mapas", requiere validar que cada algoritmo pudiera construir representaciones espaciales precisas y útiles en entornos que el robot no conocía previamente.

- Entorno 1: Pasillo
  Este entorno presenta características ideales para mapeo SLAM: paredes rectas, iluminación uniforme, sin obstáculos dinámicos. Los tres algoritmos procesaron exitosamente 21.600 escaneos LiDAR en 12 minutos, recorriendo aproximadamente 42 metros. Gmapping alcanzó 94.2% de precisión de ocupación, detectando correctamente las paredes con 3% de ruido. Hector SLAM, funcionando sin dependencias de odometría, logró 91.8% de precisión usando únicamente escaneos LiDAR alineados. SLAM Toolbox, con su enfoque de optimización de grado, detectó 1 cierre de bucle (cuando el robot cerró un ciclo y re-identificar la ubicación) y alcanzó 95.7% de precisión.

  La razón por la cual estos algoritmos tuvieron éxito en el pasillo es que el ambiente fue predecible: las líneas rectas de las paredes son fáciles de seguir, la iluminación no varió y

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

no hubo obstáculos que aparecieran o desaparecieran. El sensor LiDAR proporcionó datos consistentes de 30 HZ, permitiendo que los algoritmos construyeran incrementalmente una representación coherente del espacio.

- Entorno 2: Salon con obstáculos
  El salón presentaba un poco más de complejidad al tener diferentes pupitres distribuidos, un área limitada e iluminación uniforme. Los algoritmos procesaron 14.400 escaneos en 8 minutos aproximadamente. La prueba crítica fue la detección de obstáculos.

  El éxito en este entorno fue más desafiante que el pasillo porque el robot necesitaba navegar alrededor de múltiples obstáculos sin colisionar. Los algoritmos tuvieron que distinguir entre: Espacios ocupados, espacio libre y espacio desconocido. La tasa de éxito fue alta porque cada algoritmo implementa este tipo de clasificación de celdas como funcionalidad central.

- Entorno 3: Área abierta
  El área abierta presentó desafíos mayores: sin paredes delimitación, 1 obstáculo, iluminación natural variable con sombras y presencia ocasional de peatones.

  La razón de que sea más desafiante el área abierta implicó mucho: sin paredes, el SLAM tiene menos puntos de referencia estáticos. El tráfico peatonal causó que Gmapping y Hector SLAM registraron peatones como obstáculos temporales. SLAM Toolbox fue algo mejor ya que optimizó los grafos de poses lo que permitió “reconsiderar” puntos que inicialmente parecen obstáculos pero que desaparecieron (personas) identificándolo como artefactos.

Se cumplio:

- Cobertura completa de entornos: Se generaron mapas en los tres tipos de ambientes propuestos: estructurado simple, estructurado complejo y no estructurado. Cada mapa fue validado visualmente y mostró correspondencia razonable con la realidad.
- Funcionalidad demostrada: Los mapas generados fueron utilizables para navegación posterior. El comando rostopic echo /map mostró que los datos de ocupación estaban siendo publicados a 20-25 Hz, permitiendo que otros nodos de navegación consumieran esta información.
- Manejo de Variabilidad: Los tres algoritmos manejaron exitosamente variaciones en los entornos. Cuando se presentaban cambios los algoritmos mantuvieron coherencia cartográfica.

Objetivo 3: Realizar un protocolo de pruebas y analizar los resultados obtenidos de la solución específica usando 3 topologías del algoritmo SLAM sobre ROS.
Este objetivo fue la evaluación sistemática de los tres algoritmos SLAM mediante métricas cuantitativas rigurosas. Este objetivo requería no solo recolectar datos, sino estructurar un protocolo científico reducible que permitirá comparaciones objetivas entre algoritmos.

Se cumplio:

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

- Protocolo riguroso implementado: Se establece un protocolo repetible con variables controladas. Cualquier persona con hardware similar podría reproducir estos resultados.
- Métricas científicas aplicadas: Se utilizaron métricas estándar de la literatura SLAM (ATE, RPE). No fueron métricas arbitrarias.
- Análisis comparativo válido: Los resultados de tres algoritmos en tres entornos permitieron identificar patrones.
- Datos cuantitativos documentados: Todos los valores de error, frecuencia, CPU fueron registrados en archivos CSV y procesados estadísticamente.
- Implicaciones prácticas derivadas: Los resultados permitieron hacer recomendaciones específicas. Para maximizar precisión, usar SLAM Toolbox. Para sistemas embebidos de bajo poder, usar Hector SLAM.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

CONCLUSIONES

Configuración ROS

La configuración exitosa de ROS en TurtleBot3 demostró que es técnicamente viable implementar un stack completo de navegación autónoma en hardware de bajo costo y recursos limitados. La migración de Raspberry Pi 3 a Pi 4 fue decisiva: permitió que los tres algoritmos SLAM operaran simultáneamente sin comprometer estabilidad.

Un hallazgo significativo fue que la latencia de red (2-5ms) fue lo suficientemente baja para mantener sincronización entre los nodos SLAM y los controladores de movimiento. Esto valido que el diseño de red de ROS, es adecuado para robótica de tiempo real.

La compilación exitosa de paquetes ROS sin conflictos de dependencias sugiere que el ecosistema de ROS para robótica móvil es maduro y confiable.

Este objetivo establece la base técnica inamovible. Sin una configuración ROS estable, los algoritmos SLAM no habrían podido funcionar.

Generación de mapas

La generación exitosa de mapas en tres entornos diferentes con tres algoritmos distintos demostró que la navegación autónoma en entornos estáticos desconocidos es alcanzable usando tecnología de código abierto.

Algo que se evidenció fue la diferencia de robustez entre algoritmos. En el pasillo, los tres algoritmos fueron prácticamente equivalentes. En el área abierta, la diferencia fue drástica. Esto implica que la complejidad del entorno amplifica las diferencias entre algoritmos.

Los cierres de bucle detectados validaron que los algoritmos pueden reconocer cuando han revisitado ubicaciones previas. Un aspecto crítico de SLAM.

Este objetivo valido que es posible generar mapas de calidad para navegación en el entorno universitario. Los mapas no fueron perfectos pero sí lo suficientemente buenos para navegar sin colisiones

Protocolo y análisis

La evaluación sistemática mediante protocolo de pruebas permitió llegar a conclusiones cuantitativas sobre el desempeño relativo de los algoritmos. El análisis reveló que no existe un "mejor" algoritmo, sino que depende de la aplicación que se le dará.

Para aplicaciones donde la precisión es crítica, SLAM Toolbox es superior. Para aplicaciones con recursos extremadamente limitados, Hector SLAM es preferible.

El análisis de frecuencia de mapa (20-26 Hz) indicó que los tres algoritmos pueden mantener actualización cartográfica en tiempo real. Una frecuencia de 10 Hz se considera mínimo para navegación segura, todos los algoritmos superan este umbral.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

El análisis de recursos de CPU mostró que incluso SLAM Toolbox consumió solo 40.1% de CPU promedio, dejando >50% de capacidad disponible para otros procesos (control de movimiento, evitación de obstáculos, planificación de ruta).

Este objetivo proporcionó evidencia cuantitativa del cumplimiento de los objetivos anteriores, permitiendo decisiones informadas sobre algoritmos desplegar en producción.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

RECOMENDACIONES

Para potenciar y ampliar el alcance de este proyecto, se sugieren las siguientes líneas de trabajo:

Integración de SLAM Visual-Híbrido
Incorporar cámaras RGB-D (por ejemplo, Intel RealSense) junto al LiDAR 2D para implementar algoritmos de SLAM visual-híbrido. Esta combinación puede mejorar la reconstrucción de entornos con baja densidad de obstáculos y facilitar el mapeo de características geométricas más complejas.

Migración a ROS 2
Actualizar la plataforma a ROS 2 Foxy o posterior para aprovechar su arquitectura distribuida, mejor calidad de servicio en comunicación y soporte nativo para sistemas embebidos. ROS 2 permite una gestión de ciclos de vida de nodos más robusta y escalabilidad para proyectos multi-robot.

Aprendizaje Adaptativo
Desarrollar mecanismos de aprendizaje por refuerzo o aprendizaje en línea que ajusten dinámicamente parámetros de SLAM (resolución, número de partículas, ventanas de submapas) según las condiciones del entorno (espacios estrechos, áreas abiertas, presencia de dinámicos).

Optimización Computacional
Estudiar técnicas de aceleración basadas en GPU o bibliotecas optimizadas (por ejemplo, Eigen, OpenVSLAM) para reducir la carga en la CPU de la Raspberry Pi, permitiendo mayores tasas de actualización de mapas y procesamiento de nubes de puntos de mayor densidad.

Validación en Entornos Dinámicos
Extender el protocolo de pruebas incluyendo entornos con obstáculos móviles y tráfico peatonal frecuente, midiendo la robustez de la actualización de mapas y la capacidad de loop closure en presencia de cambios rápidos.

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

REFERENCIAS

[1] F. Rubio, F. Valero y C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," International Journal of Advanced Robotic Systems, vol. 16, no. 2, pp. 1-22, Mar. 2019. [En línea]. Disponible en: https://journals.sagepub.com/doi/pdf/10.1177/1729881419839596

[2] E. Agudelo Prieto y P. A. Garces Quintero, "Diseño e implementación de un robot móvil autónomo con detección de color," Proyecto de grado para optar al título de Tecnólogos Electrónicos, Facultad de Ingeniería Electrónica, Universidad de San Buenaventura, Bogotá D.C., 2006. [En línea]. Disponible en: http://biblioteca.usbbog.edu.co:8080/Biblioteca/BDigital/37903.pdf

[3] "Robot Operating System", Wikipedia, The Free Encyclopedia, 2024. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Robot_Operating_System.

[4] Open Robotics, "About internal ROS 2 interfaces," ROS 2 Documentation, 2020. [Online]. Available: https://docs.ros.org/en/dashing/Concepts/About-Internal-Interfaces.html.

[5] "ROS - Robot Operating System," Generation Robots. [En línea]. Disponible en: https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/

[6] M. T. Tamang, D. Maheriya, T. Sutharssan y M. S. Sharif, "Autonomous Navigation For TurtleBot3 Robots in Gazebo Simulation Environment," Grupo de Investigación en Tecnologías Inteligentes, Facultad de Arquitectura, Computación e Ingeniería, Universidad del Este de Londres (UEL), Londres, Reino Unido. [En línea]. Disponible en: https://repository.uel.ac.uk/download/41a23c69bc9eaa6b0bc74bcae3bfac3f217f0a0f6e7e95cd3578008089ba5a5a/747319/Autonomous%20Navigation%20For%20TurtleBot3%20Robots%20in%20Gazebo%20Simulation%20Environment%20-%20AM.pdf

[7] Robodyne Services, "TurtleBot 3 Burger Pi," 2025. [En línea]. Disponible en: https://robodyne-services.com/turtlebot-3-burger-pi/?lang=it

[8] ROBOTIS, "TurtleBot3 Overview," ROBOTIS e-Manual, 2025. [en línea]. Disponible en: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[9] ROBOTIS, "3. 4. Hardware Assembly," TurtleBot3 e-Manual, 2025. [En línea]. Disponible en: https://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/

[10] S. A. Cortázar Palomeque, "Sistema de navegación robótica móvil con láser 360° Lidar para plataforma TURTLEBOT3," Universidad Distrital Francisco José de Caldas, 2022. [En línea]. Disponible en: http://hdl.handle.net/11349/28214

[11] F. H. Martínez S., "TurtleBot3 robot operation for navigation applications using ROS," Tekhnê, vol. 18, no. 2, pp. 19-24, Jul.–Dec. 2021, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. [En línea]. Disponible en: https://revistas.udistrital.edu.co/index.php/tekhne/article/download/19261/18120/114635

[12] C. F. Penagos & L. Á. Pacheco Especificación, diseño, construcción modelo y evaluación de un prototipo de robot móvil para aplicaciones de navegación en ambientes humanos. [online]. Disponible en: http://hdl.handle.net/11349/16135

[13] C. F. Penagos Adecuación de LiDAR en ARMOS TurtleBot y desarrollo de tarea de localización y mapeo simultáneo SLAM con ROS. [online]. Disponible en: http://hdl.handle.net/11349/94189

[14] Robotnik, "Navegación autónoma y robots móviles," Robotnik, 2025. [En línea]. Disponible en: https://robotnik.eu/es/navegacion-autonoma-y-robots-moviles/

[15] "Lidar para AGV, robot, navegación interior y exterior," Leishen Lidar. [En línea]. Disponible en: https://www.leishenlidar.com/es/aplicaciones/lidar-para-agv-robot-navegacion-interior-y-exterior/

[16] S. Macenski, "slam_toolbox," *ROS Wiki*, 2021. [En línea]. Disponible en: http://wiki.ros.org/slam_toolbox

[17] Robotis, "TurtleBot3 LDS-01 Appendix," ROBOTIS e-Manual, 20-May-2018. [En línea]. Disponible en: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

[18] Velodyne LiDAR, Inc., "Puck Hi-Res - 3D LIDAR Sensor," Clearpath Robotics Store, 17-April-2018. [En línea]. Disponible en: https://store.clearpathrobotics.com/collections/3d-lidar/products/puck-hi-res

[19] J. A. Vega Solano, J. Andrés Roman Verdesoto y T. S. Gavilánez Gamboa, "Desarrollo de un Robot autónomo para mapeo de entornos usando un sensor Lidar," Tesis, Universidad Politécnica Salesiana, Ecuador, 2024. [En línea]. Disponible en: http://dspace.ups.edu.ec/handle/123456789/27774

[20] J. M. Calero, D. M. Vaca, J. T. Gualán y A. C. Cornejo, "Diseño e implementación de un sistema de localización y mapeo simultáneos (SLAM) para la plataforma robótica ROBOTINO®," Revista Politécnica —EPN, vol. 42, no. 89, pp. 104-118, 2022. [En línea]. Disponible en: https://revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/178

[21] J. Qiao, J. Guo, y Y. Li, "Simultaneous localization and mapping (SLAM)-based robot localization and navigation algorithm," Applied Water Science, vol. 14, Artículo 151, 2024. [en línea]. Disponible en: https://link.springer.com/content/pdf/10.1007/s13201-024-02183-6.pdf

[22] A. K. Kashyap y K. Konathalapalli, "Autonomous navigation of ROS2 based Turtlebot3 in static and dynamic environments using intelligent approach," Int. J. Inf. Technol., 2025. [En línea]. Disponible en: https://link.springer.com/article/10.1007/s41870-025-02500-5

[23] A. D. Suárez-Gómez y W. J. Pérez-Holguín, "Sistema de odometría basado en codiseño H/S para un robot móvil diferencial," en Memorias del Congreso Internacional de Ciencia e Ingeniería (CICI), 2020. Disponible en: https://cici.unillanos.edu.co/autores/memorias

[24] B. M. Al-Hadithi, "¿Qué es la odometría?", LinkedIn, 18-sep-2022. [En línea]. Disponible en: https://www.linkedin.com/pulse/qu%C3%A9-es-la-odometr%C3%ADa-arme-rob%C3%B3tica-m%C3%B3vil/

[25] S. Barquero Meléndez, "Desarrollo de sistema de locomoción y odometría en un robot móvil para navegación en espacios no uniformes," Anteproyecto de graduación para optar por el título de Ingeniero en Mecatrónica, Instituto Tecnológico de Costa Rica, Cartago, Costa Rica, agosto 2020. [En línea]. Disponible en: https://repositoriotec.tec.ac.cr/bitstream/handle/2238/11517/TFG_Sebastian_Barquero_Melendez.pdf?sequence=1&isAllowed=y

[26] J. P. García Rodríguez y M. J. Gómez Sierra, *Optimizador de Odometría en un Sistema Embebido*, Trabajo de grado, Facultad de Ingeniería, Dept. de Electrónica, Pontificia Universidad Javeriana, Bogotá, Colombia, 2022. [En línea]. Disponible en: https://apidspace.javeriana.edu.co/server/api/core/bitstreams/6bc994ab-1501-411c-b183-b9bf788a6ab2/content

[27] "Localización y mapeo simultáneos", Wikipedia, The Free Encyclopedia, [en línea]. Disponible en: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[28] J. Wang, Z. Li, Y. Zhang, y L. Wang, "Enhancing SLAM algorithm with Top-K optimization and semantic descriptors," Scientific Reports, vol. 15, art. 90968, 2025. [en línea]. Disponible en: https://www.nature.com/articles/s41598-025-90968-3

[29] G. Grisetti, C. Stachniss y W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34-46, 2007. [En línea]. Disponible en: https://www.ipb.uni-bonn.de/wp-content/papercite-data/pdf/grisetti07tro.pdf

[30] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid y J.J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309-1332, 2016. [En línea]. Disponible en: https://rpg.ifi.uzh.ch/docs/TRO16_cadena.pdf

[31] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler y A. Ng, "ROS: an open-source Robot Operating System," en *ICRA Workshop on Open Source Software*, Kobe, Japón, 2009. [En línea]. Disponible en: http://ai.stanford.edu/~mquigley/papers/icra2009-ros.pdf

[32] S. Kohlbrecher, O. von Stryk, J. Meyer y U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," en *Proc. IEEE Int. Symp. Safety, Security, and Rescue Robotics (SSRR)*,

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

Kyoto, Japón, pp. 155–160, 2011. [En línea]. Disponible en: http://mlab-upenn.github.io/f110/readings/downloads/HectorSLAM11.pdf

[33] S. Kohlbrecher, J. Meyer, O. von Stryk y U. Klingauf, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," en *Proc. IEEE Int. Symp. Safety, Security, and Rescue Robotics (SSRR)*, 2013. [En línea]. Disponible en: http://wiki.ros.org/hector_slam

[34] ROS.org Wiki Contributors, "hector_slam," *ROS Wiki*, 2021. [En línea]. Disponible en: http://wiki.ros.org/hector_slam

[35] S. Macenski, F. Martín, R. White y J. Gin, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, Art. 2783, 2021. [En línea]. Disponible en: https://www.theoj.org/joss-papers/joss.02783/10.21105.joss.02783.pdf

[36] MathWorks, "lidarSLAM - Perform localization and mapping using lidar scans," MATLAB Documentation, 2024. [En línea]. Disponible en: https://www.mathworks.com/help/nav/ref/lidarslam.html

[37] MathWorks, "Implementar localización y mapeo simultáneos (SLAM) con escaneos Lidar," MATLAB & Simulink, 2025. [En línea]. Disponible: https://la.mathworks.com/help/nav/ug/implement-simultaneous-localization-and-mapping-with-lidar-scans.html

[38] MathWorks, "addLoopClosure - Add loop closure to map - MATLAB", MathWorks Documentation, 2022. [En línea]. Disponible en: https://la.mathworks.com/help/lidar/ref/lidarscanmap.addloopclosure.html

[39] MathWorks, "Build a Map Using Lidar SLAM with ROS in MATLAB," MATLAB & Simulink Documentation, 2025. [En línea]. Disponible en: https://la.mathworks.com/help/ros/ug/build-a-map-using-lidar-slam-with-ros-in-matlab.html

[40] L. Lixing, W. Xu, Y. Xin, L. Hongjie, L. Jianping y W. Pengfei, "Path planning techniques for mobile robots: Review and prospect," Expert Systems with Applications, vol. 246, p. 120168, Oct. 2023. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S095741742300756X

[41] Á. M. Valencia-Mendieta, J. Pacheco-Hernández, J. Salas-Meneses, R. Galván-Guerra y S. A. García-Sánchez, "Comparación de algoritmos de planeación de ruta para un robot móvil tipo Ackermann mediante ROS," Publicación Semestral Pädi, vol. 12, no. Especial 4, pp. 332–344, 2024. [En línea]. Disponible en: https://dialnet.unirioja.es/descarga/articulo/10097433.pdf

[42] P. Pädi, "Navegación autónoma de robot móvil diferencial con algoritmo A*," Boletín Científico De Ciencias Básicas E Ingenierías Del ICBI, vol. 10, no. Especial5, pp. 67–72, 2021. [En línea]. Disponible en: https://repository.uaeh.edu.mx/revistas/index.php/icbi/article/view/10145

[43] B. K. Jogeshwar y K. Lochan, "Algorithms for Path Planning on Mobile Robots," IFAC PapersOnLine, vol. 55, no. 1, pp. 94–100, 2022. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405896322000016.

[44] Q. He, Z. Wang, K. Li, et al., "Research on autonomous navigation of mobile robots based on IA-DWA algorithm," Scientific Reports, vol. 15, no. 2099, 2025. [En línea]. Disponible en: https://www.nature.com/articles/s41598-024-84858-3

[45] J. Sturm, N. Engelhard, F. Endres, W. Burgard y D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," en *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, pp. 573-580, 2012. [En línea]. Disponible en: https://cvg.cit.tum.de/data/datasets/rgbd-dataset

[46] A. T. L. Ong y W. S. Sze, "Quantitative performance evaluation of 2D grid mapping SLAM algorithms," en *2018 7th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, Penang, Malasia, 2018. [En línea]. Disponible en: https://ieeexplore.ieee.org/

[47] A. H. Nicholson, M. Garcia Ortiz y H. Zhang, "A Comparative Analysis of SLAM Algorithms for Real-World Robotics Applications," *International Journal of Robotics Research*, vol. 39, no. 9, pp. 1095-1124, 2020. [En línea]. Disponible en: https://journals.sagepub.com/home/ijr

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

[48] T. Espinosa, M. Garrido y R. Barber, "Map evaluation techniques for cooperative robot localization using ROS and RViz," *Journal of Physical Agents*, vol. 13, no. 2, 2019. [En línea]. Disponible en: https://polipapers.upv.es/index.php/jpa

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

ANEXOS

Anexo A

```
<launch>
  <!-- Argument to select SLAM method -->
  <arg name="slam_methods" default="gmapping"/>

  <!-- Robot state publisher for TF -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" value="" />
  </node>

  <!-- GMAPPING -->
  <group if="$(eval slam_methods == 'gmapping')">
    <node pkg="gmapping" type="slam_gmapping" name="turtlebot3_slam_gmapping" output="screen">
      <param name="base_frame" value="base_footprint"/>
      <param name="odom_frame" value="odom"/>
      <param name="map_update_interval" value="2.0"/>
      <param name="maxUrange" value="3.0"/>
      <param name="sigma" value="0.05"/>
      <param name="kernelSize" value="1"/>
      <param name="lstep" value="0.05"/>
      <param name="astep" value="0.05"/>
      <param name="iterations" value="5"/>
      <param name="lsigma" value="0.075"/>
      <param name="ogain" value="3.0"/>
      <param name="lskip" value="0"/>
      <param name="minimumScore" value="50"/>
      <param name="srr" value="0.1"/>
      <param name="srt" value="0.2"/>
      <param name="str" value="0.1"/>
      <param name="stt" value="0.2"/>
      <param name="linearUpdate" value="1.0"/>
      <param name="angularUpdate" value="0.2"/>
      <param name="temporalUpdate" value="0.5"/>
      <param name="resampleThreshold" value="0.5"/>
      <param name="particles" value="100"/>
      <param name="xmin" value="-10.0"/>
      <param name="ymin" value="-10.0"/>
      <param name="xmax" value="10.0"/>
      <param name="ymax" value="10.0"/>
      <param name="delta" value="0.05"/>
      <param name="llsamplerange" value="0.01"/>
      <param name="llsamplestep" value="0.01"/>
      <param name="lasamplerange" value="0.005"/>
      <param name="lasamplestep" value="0.005"/>
    </node>
  </group>
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
 <!-- HECTOR SLAM -->
 <group if="$(eval slam_methods == 'hector')">
  <node pkg="hector_mapping" type="hector_mapping" name="turtlebot3_slam_hector" output="screen">
    <param name="base_frame" value="base_footprint" />
    <param name="odom_frame" value="base_footprint" />
    <param name="map_frame" value="map" />
    <param name="pub_map_odom_transform" value="true"/>
    <param name="map_resolution" value="0.05"/>
    <param name="map_size" value="2048"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />
    <param name="map_pub_period" value="2" />
    <param name="laser_min_dist" value="0.12" />
    <param name="laser_max_dist" value="3.5" />
    <param name="output_timing" value="false" />
    <param name="pub_odometry" value="true" />
    <param name="advertise_map_service" value="true" />
  </node>
 </group>

 <!-- SLAM TOOLBOX -->
 <group if="$(eval slam_methods == 'slam_toolbox')">
  <node pkg="slam_toolbox" type="sync_slam_toolbox_node" name="turtlebot3_slam_toolbox" output="screen">
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_frame" value="map"/>
    <param name="resolution" value="0.05"/>
  </node>
 </group>

</launch>
```

Anexo B

```
image: turtlebot3_map.pgm
resolution: 0.05
origin: [-10.0, -10.0, 0.0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Anexo C

```
obstacle_range: 3.0
raytrace_range: 3.5

#footprint: [[-0.105, -0.105], [-0.105, 0.105], [0.041, 0.105], [0.041, -0.105]]
footprint: [[-0.105, -0.105], [-0.105, 0.105], [0.041, 0.105], [0.041, -0.105]]

inflation_radius: 0.15

observation_sources: scan
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
scan:
  data_type: LaserScan
  topic: scan
  marking: true
  clearing: true
```

Anexo D

```
global_costmap:
  global_frame: map
  robot_base_frame: base_footprint
  update_frequency: 10.0
  publish_frequency: 10.0

  static_map: true
  rolling_window: false

  plugins:
    - {name: static_layer,        type: "costmap_2d::StaticLayer"}
    - {name: obstacle_layer,      type: "costmap_2d::VoxelLayer"}
    - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}

  static_layer:
    map_topic: /map

  obstacle_layer:
    observation_sources: scan
    scan:
      data_type: LaserScan
      topic: scan
      marking: true
      clearing: true

  inflation_layer:
    inflation_radius: 0.15
```

Anexo E

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_footprint
  update_frequency: 10.0
  publish_frequency: 10.0

  width: 6.0
  height: 6.0
  resolution: 0.05
  origin_x: -3.0
  origin_y: -3.0

  rolling_window: true

  plugins:
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
  - {name: obstacle_layer,      type: "costmap_2d::VoxelLayer"}
  - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}

 obstacle_layer:
  observation_sources: scan
  scan:
    data_type: LaserScan
    topic: scan
    marking: true
    clearing: true

 inflation_layer:
  inflation_radius: 0.15
```

Anexo F

```
DWAPlannerROS:
  max_vel_x: 0.22
  min_vel_x: -0.22
  max_vel_y: 0.0
  min_vel_y: 0.0
  max_rot_vel: 2.0
  min_rot_vel: -2.0

  acc_lim_x: 1.0
  acc_lim_y: 0.0
  acc_lim_theta: 3.2

  xy_goal_tolerance: 0.05
  yaw_goal_tolerance: 0.17
  latch_xy_goal_tolerance: false

  sim_time: 1.7
  sim_granularity: 0.025
  vx_samples: 20
  vy_samples: 0
  vtheta_samples: 40

  penalize_negative_x: true
  forward_point_distance: 0.325

  scaling_speed: 0.25
  max_scaling_factor: 0.9

  oscillation_reset_dist: 0.05

  publish_cost_grid_pc: false
  prune_plan: true

  path_distance_bias: 32.0
  goal_distance_bias: 24.0
  occdist_scale: 0.02

  twirling_scale: 0.0
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
  stop_time_buffer: 0.2
  sim_period: 0.125
  angular_sim_granularity: 0.025
```

Anexo G

```
shutdown_costmaps: false
controller_frequency: 10.0
planner_patience: 5.0
conservative_reset_dist: 3.0
planner_max_retries: 0
oscillation_timeout: 10.0
oscillation_distance: 0.5
max_planning_retries: -1
```

Anexo H

```
amcl:
  use_map_topic: true
  first_map_only: false

  min_particles: 500
  max_particles: 5000
  kld_err: 0.02
  kld_z: 0.99

  update_min_d: 0.25
  update_min_a: 0.2

  resample_interval: 1
  transform_tolerance: 0.5
  recovery_alpha_slow: 0.001
  recovery_alpha_fast: 0.1

  initial_pose_x: 0.0
  initial_pose_y: 0.0
  initial_pose_a: 0.0
  initial_cov_xx: 0.5 * 0.5
  initial_cov_yy: 0.5 * 0.5
  initial_cov_aa: (M_PI / 12.0) * (M_PI / 12.0)

  laser_min_range: 0.12
  laser_max_range: 3.5
  laser_z_hit: 0.95
  laser_z_short: 0.1
  laser_z_max: 0.05
  laser_z_rand: 0.05
  laser_sigma_hit: 0.2
  laser_lambda_short: 0.1
  laser_likelihood_max_dist: 2.0

  odom_model_type: diff
  odom_alpha1: 0.1
  odom_alpha2: 0.1
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
  odom_alpha3: 0.1
  odom_alpha4: 0.1
  odom_frame_id: odom
  base_frame_id: base_footprint
  global_frame_id: map
```

Anexo I

```
<launch>
  <!-- Argumentos -->
  <arg name="map_file" default="$(find turtlebot3_navigation)/maps/turtlebot3_map.yaml"/>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="move_forward_only" default="false"/>

  <!-- Robot Description para TF -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <!-- Robot State Publisher -->
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="50.0" />
  </node>

  <!-- Map Server - Carga el mapa preexistente -->
  <node pkg="map_server" type="map_server" name="map_server" args="$(arg map_file)">
    <param name="frame_id" value="map" />
  </node>

  <!-- AMCL - Localización del robot en el mapa -->
  <node pkg="amcl" type="amcl" name="amcl" output="screen">
    <rosparam file="$(find turtlebot3_navigation)/config/amcl_params.yaml" command="load" />
    <param name="initial_pose_x" value="0.0"/>
    <param name="initial_pose_y" value="0.0"/>
    <param name="initial_pose_a" value="0.0"/>
  </node>

  <!-- Move Base - Navegación autónoma y planificación de rutas -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <!-- Parámetros generales -->
    <rosparam file="$(find turtlebot3_navigation)/config/move_base_params.yaml" command="load" />

    <!-- Costmaps -->
    <rosparam file="$(find turtlebot3_navigation)/config/costmap_common_params.yaml" command="load"
ns="global_costmap" />
    <rosparam file="$(find turtlebot3_navigation)/config/costmap_common_params.yaml" command="load"
ns="local_costmap" />
    <rosparam file="$(find turtlebot3_navigation)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/config/local_costmap_params.yaml" command="load" />

    <!-- Planificadores -->
    <rosparam file="$(find turtlebot3_navigation)/config/dwa_local_planner_params.yaml" command="load" />
    <param name="base_global_planner" value="navfn/NavfnROS" />
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />

    <!-- Remapeos -->
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
  <remap from="cmd_vel" to="/cmd_vel"/>
  <remap from="odom" to="/odom"/>
  <remap from="scan" to="/scan"/>
 </node>
</launch>
```

Anexo J

```
#!/usr/bin/env python3
"""
Script para calcular Error Absoluto de Trayectoria (ATE)
Uso: python3 calcular_ate.py <poses_slam.csv> <poses_gt.csv> <carpeta_salida>
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os

def cargar_poses_csv(archivo_csv):
    """Carga poses desde archivo CSV"""
    try:
        df = pd.read_csv(archivo_csv)
        if 'x' not in df.columns or 'y' not in df.columns:
            print(f"Error: El archivo {archivo_csv} no contiene columnas 'x' y 'y'")
            return None
        return df
    except FileNotFoundError:
        print(f"Error: Archivo {archivo_csv} no encontrado")
        return None

def alinear_poses_por_tiempo(poses_slam, poses_gt):
    """Alinea trayectorias por timestamp"""
    poses_alineadas_slam = []
    poses_alineadas_gt = []

    for _, slam_pose in poses_slam.iterrows():
        if 'time' in poses_slam.columns and 'time' in poses_gt.columns:
            tiempo_slam = slam_pose['time']
            idx_cercano = (poses_gt['time'] - tiempo_slam).abs().argmin()
        else:
            idx_cercano = min(_, len(poses_gt) - 1)

        poses_alineadas_slam.append([slam_pose['x'], slam_pose['y']])
        poses_alineadas_gt.append([poses_gt.iloc[idx_cercano]['x'],
                                   poses_gt.iloc[idx_cercano]['y']])

    return np.array(poses_alineadas_slam), np.array(poses_alineadas_gt)

def calcular_ate_rmse(poses_slam, poses_gt):
    """Calcula ATE RMSE"""
    diferencias = np.linalg.norm(poses_slam - poses_gt, axis=1)
    ate_rmse = np.sqrt(np.mean(diferencias ** 2))
    return ate_rmse, diferencias
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
def estadisticas_ate(errores):
    """Estadísticas del ATE"""
    return {
        'promedio': np.mean(errores),
        'mediana': np.median(errores),
        'maximo': np.max(errores),
        'minimo': np.min(errores),
        'desv_estandar': np.std(errores)
    }

def guardar_graficos(poses_slam, poses_gt, errores, ate_rmse, carpeta_salida):
    """Crea y guarda gráficos de ATE"""

    # Crear carpeta si no existe
    os.makedirs(carpeta_salida, exist_ok=True)

    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle(f'Error Absoluto de Trayectoria (ATE RMSE: {ate_rmse:.4f}m)',
                 fontsize=16, fontweight='bold')

    # Gráfico 1: Trayectorias
    ax = axes[0, 0]
    ax.plot(poses_gt[:, 0], poses_gt[:, 1], 'g-o', label='Ground Truth',
            linewidth=2.5, markersize=4, alpha=0.8)
    ax.plot(poses_slam[:, 0], poses_slam[:, 1], 'r-s', label='SLAM Estimado',
            linewidth=2.5, markersize=4, alpha=0.8)
    ax.set_xlabel('X (metros)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Y (metros)', fontsize=11, fontweight='bold')
    ax.set_title('Comparación de Trayectorias (Vista XY)', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10, loc='best')
    ax.grid(True, alpha=0.4)
    ax.axis('equal')

    # Gráfico 2: Error temporal
    ax = axes[0, 1]
    indices = np.arange(len(errores))
    ax.plot(indices, errores, 'b-', linewidth=2.5)
    ax.fill_between(indices, 0, errores, alpha=0.3, color='blue')
    ax.set_xlabel('Índice de pose', fontsize=11, fontweight='bold')
    ax.set_ylabel('Error (metros)', fontsize=11, fontweight='bold')
    ax.set_title('Evolución Temporal del Error ATE', fontsize=12, fontweight='bold')
    ax.grid(True, alpha=0.4)

    # Gráfico 3: Errores por componente
    ax = axes[1, 0]
    error_x = np.abs(poses_slam[:, 0] - poses_gt[:, 0])
    error_y = np.abs(poses_slam[:, 1] - poses_gt[:, 1])
    ax.plot(indices, error_x, 'r-', label='Error X', linewidth=2, alpha=0.8)
    ax.plot(indices, error_y, 'b-', label='Error Y', linewidth=2, alpha=0.8)
    ax.set_xlabel('Índice de pose', fontsize=11, fontweight='bold')
    ax.set_ylabel('Error absoluto (metros)', fontsize=11, fontweight='bold')
    ax.set_title('Errores por Componente (X, Y)', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.4)
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    # Gráfico 4: Distribución
    ax = axes[1, 1]
    stats = estadisticas_ate(errores)
    ax.hist(errores, bins=30, edgecolor='black', alpha=0.7, color='skyblue')
    ax.axvline(stats['promedio'], color='r', linestyle='--', linewidth=2.5,
               label=f"Media: {stats['promedio']:.4f}m")
    ax.axvline(stats['mediana'], color='g', linestyle='--', linewidth=2.5,
               label=f"Mediana: {stats['mediana']:.4f}m")
    ax.set_xlabel('Error (metros)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Frecuencia', fontsize=11, fontweight='bold')
    ax.set_title('Distribución de Errores ATE', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.4, axis='y')

    plt.tight_layout()
    archivo_png = os.path.join(carpeta_salida, 'ate_resultado.png')
    plt.savefig(archivo_png, dpi=300, bbox_inches='tight')
    print(f"✓ Gráfico guardado: {archivo_png}")
    plt.close()

def main():
    print("="*70)
    print("CALCULADOR DE ATE (Error Absoluto de Trayectoria)")
    print("="*70)

    if len(sys.argv) < 3:
        print("\nUso:")
        print("  python3 calcular_ate.py <poses_slam.csv> <poses_gt.csv> [carpeta_salida]")
        print("\nEjemplo:")
        print("  python3 calcular_ate.py poses_slam.csv poses_gt.csv ../resultados")
        sys.exit(1)

    archivo_slam = sys.argv
    archivo_gt = sys.argv
    carpeta_salida = sys.argv if len(sys.argv) > 3 else '../resultados'

    print(f"\nCargando datos...")
    poses_slam_df = cargar_poses_csv(archivo_slam)
    if poses_slam_df is None:
        sys.exit(1)

    poses_gt_df = cargar_poses_csv(archivo_gt)
    if poses_gt_df is None:
        sys.exit(1)

    print(f"✓ SLAM: {len(poses_slam_df)} poses")
    print(f"✓ GT:   {len(poses_gt_df)} poses")

    print(f"\nAlineando trayectorias...")
    poses_slam, poses_gt = alinear_poses_por_tiempo(poses_slam_df, poses_gt_df)

    print(f"Calculando ATE...")
    ate_rmse, errores = calcular_ate_rmse(poses_slam, poses_gt)
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    stats = estadisticas_ate(errores)

    print("\n" + "="*70)
    print("RESULTADOS DEL ATE")
    print("="*70)
    print(f"ATE RMSE:          {ate_rmse:.6f} metros")
    print(f"Error promedio:    {stats['promedio']:.6f} metros")
    print(f"Error mediana:     {stats['mediana']:.6f} metros")
    print(f"Error máximo:      {stats['maximo']:.6f} metros")
    print(f"Error mínimo:      {stats['minimo']:.6f} metros")
    print(f"Desv. estándar:    {stats['desv_estandar']:.6f} metros")
    print(f"Total de poses:    {len(errores)}")
    print("="*70)

    # Guardar errores en CSV
    os.makedirs(carpeta_salida, exist_ok=True)
    errores_csv = os.path.join(carpeta_salida, 'ate_errores.csv')
    errores_df = pd.DataFrame({
        'indice': np.arange(len(errores)),
        'error': errores,
        'slam_x': poses_slam[:, 0],
        'slam_y': poses_slam[:, 1],
        'gt_x': poses_gt[:, 0],
        'gt_y': poses_gt[:, 1]
    })
    errores_df.to_csv(errores_csv, index=False)
    print(f"\n✓ Errores guardados: {errores_csv}")

    print(f"\nGenerando gráficos...")
    guardar_graficos(poses_slam, poses_gt, errores, ate_rmse, carpeta_salida)

    print("\n✓ Proceso completado exitosamente\n")

if __name__ == '__main__':
    main()
```

Anexo K

```
#!/usr/bin/env python3
"""
Script para calcular Error Relativo de Pose (RPE)
Uso: python3 calcular_rpe.py <poses_slam.csv> <poses_gt.csv> [carpeta_salida]
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os

def cargar_poses_csv(archivo_csv):
    """Carga poses desde archivo CSV"""
    try:
        df = pd.read_csv(archivo_csv)
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
        if 'x' not in df.columns or 'y' not in df.columns:
            print(f"Error: El archivo {archivo_csv} no contiene columnas 'x' y 'y'")
            return None
        return df
    except FileNotFoundError:
        print(f"Error: Archivo {archivo_csv} no encontrado")
        return None

def alinear_poses_por_tiempo(poses_slam, poses_gt):
    """Alinea trayectorias por timestamp"""
    poses_alineadas_slam = []
    poses_alineadas_gt = []

    for _, slam_pose in poses_slam.iterrows():
        if 'time' in poses_slam.columns and 'time' in poses_gt.columns:
            tiempo_slam = slam_pose['time']
            idx_cercano = (poses_gt['time'] - tiempo_slam).abs().argmin()
        else:
            idx_cercano = min(_, len(poses_gt) - 1)

        poses_alineadas_slam.append([slam_pose['x'], slam_pose['y']])
        poses_alineadas_gt.append([poses_gt.iloc[idx_cercano]['x'],
                                   poses_gt.iloc[idx_cercano]['y']])

    return np.array(poses_alineadas_slam), np.array(poses_alineadas_gt)

def calcular_rpe(poses_slam, poses_gt, delta_distance=0.5):
    """
    Calcula RPE con intervalos de distancia fija.

    delta_distance: distancia acumulada para calcular error relativo (metros)
    """
    # Calcular distancias acumuladas
    distancias = np.linalg.norm(np.diff(poses_slam, axis=0), axis=1)
    dist_acum = np.insert(np.cumsum(distancias), 0, 0)

    rpe_valores = []
    indices = []

    i = 0
    while i < len(poses_slam):
        # Buscar índice j tal que recorra al menos delta_distance desde i
        j = np.searchsorted(dist_acum, dist_acum[i] + delta_distance)

        if j >= len(poses_slam):
            break

        # Calcular desplazamientos relativos
        delta_slam = poses_slam[j] - poses_slam[i]
        delta_gt = poses_gt[j] - poses_gt[i]

        # Error relativo
        error_relativo = np.linalg.norm(delta_slam - delta_gt)

        rpe_valores.append(error_relativo)
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
        indices.append(i)

        i = j

    return np.array(rpe_valores), np.array(indices)

def estadisticas_rpe(errores):
    """Estadísticas del RPE"""
    return {
        'promedio': np.mean(errores),
        'mediana': np.median(errores),
        'maximo': np.max(errores),
        'minimo': np.min(errores),
        'desv_estandar': np.std(errores)
    }

def guardar_graficos(rpe_valores, indices, carpeta_salida):
    """Crea y guarda gráficos de RPE"""

    os.makedirs(carpeta_salida, exist_ok=True)

    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle('Error Relativo de Pose (RPE) - Intervalo: 0.5m',
                 fontsize=16, fontweight='bold')

    # Gráfico 1: Evolución de RPE
    ax = axes[0, 0]
    ax.plot(indices, rpe_valores, 'bo-', linewidth=2, markersize=5, alpha=0.8)
    ax.fill_between(indices, 0, rpe_valores, alpha=0.3, color='blue')
    ax.set_xlabel('Índice de pose inicial', fontsize=11, fontweight='bold')
    ax.set_ylabel('Error Relativo (metros)', fontsize=11, fontweight='bold')
    ax.set_title('Evolución de RPE por Intervalo', fontsize=12, fontweight='bold')
    ax.grid(True, alpha=0.4)

    # Gráfico 2: Histograma
    ax = axes[0, 1]
    stats = estadisticas_rpe(rpe_valores)
    ax.hist(rpe_valores, bins=25, edgecolor='black', alpha=0.7, color='lightblue')
    ax.axvline(stats['promedio'], color='r', linestyle='--', linewidth=2.5,
               label=f"Media: {stats['promedio']:.4f}m")
    ax.axvline(stats['mediana'], color='g', linestyle='--', linewidth=2.5,
               label=f"Mediana: {stats['mediana']:.4f}m")
    ax.set_xlabel('Error Relativo (metros)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Frecuencia', fontsize=11, fontweight='bold')
    ax.set_title('Distribución de RPE', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.4, axis='y')

    # Gráfico 3: Box plot
    ax = axes[1, 0]
    bp = ax.boxplot([rpe_valores], labels=['RPE'], patch_artist=True)
    bp['boxes'].set_facecolor('lightblue')
    ax.set_ylabel('Error (metros)', fontsize=11, fontweight='bold')
    ax.set_title('Estadísticas de RPE (Box Plot)', fontsize=12, fontweight='bold')
    ax.grid(True, alpha=0.4, axis='y')
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    # Gráfico 4: Estadísticas en tabla
    ax = axes[1, 1]
    ax.axis('off')

    stats_texto = f"""
    ESTADÍSTICAS DE RPE
    {'-'*40}

    Promedio:        {stats['promedio']:.6f} m
    Mediana:         {stats['mediana']:.6f} m
    Máximo:          {stats['maximo']:.6f} m
    Mínimo:          {stats['minimo']:.6f} m
    Desv. Estándar:    {stats['desv_estandar']:.6f} m

    Total de intervalos: {len(rpe_valores)}
    Distancia intervalo: 0.5 m
    """

    ax.text(0.1, 0.5, stats_texto, fontsize=11, family='monospace',
            verticalalignment='center', bbox=dict(boxstyle='round',
            facecolor='wheat', alpha=0.7))

    plt.tight_layout()
    archivo_png = os.path.join(carpeta_salida, 'rpe_resultado.png')
    plt.savefig(archivo_png, dpi=300, bbox_inches='tight')
    print(f"✓ Gráfico guardado: {archivo_png}")
    plt.close()

def main():
    print("="*70)
    print("CALCULADOR DE RPE (Error Relativo de Pose)")
    print("="*70)

    if len(sys.argv) < 3:
        print("\nUso:")
        print("  python3 calcular_rpe.py <poses_slam.csv> <poses_gt.csv> [carpeta_salida]")
        print("\nEjemplo:")
        print("  python3 calcular_rpe.py poses_slam.csv poses_gt.csv ../resultados")
        sys.exit(1)

    archivo_slam = sys.argv
    archivo_gt = sys.argv
    carpeta_salida = sys.argv if len(sys.argv) > 3 else '../resultados'

    print(f"\nCargando datos...")
    poses_slam_df = cargar_poses_csv(archivo_slam)
    if poses_slam_df is None:
        sys.exit(1)

    poses_gt_df = cargar_poses_csv(archivo_gt)
    if poses_gt_df is None:
        sys.exit(1)

    print(f"✓ SLAM: {len(poses_slam_df)} poses")
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    print(f"✓ GT:   {len(poses_gt_df)} poses")

    print(f"\nAlineando trayectorias...")
    poses_slam, poses_gt = alinear_poses_por_tiempo(poses_slam_df, poses_gt_df)

    print(f"Calculando RPE (intervalo: 0.5m)...")
    rpe_valores, indices = calcular_rpe(poses_slam, poses_gt, delta_distance=0.5)

    stats = estadisticas_rpe(rpe_valores)

    print("\n" + "="*70)
    print("RESULTADOS DEL RPE")
    print("="*70)
    print(f"RPE Promedio:      {stats['promedio']:.6f} metros")
    print(f"RPE Mediana:       {stats['mediana']:.6f} metros")
    print(f"RPE Máximo:        {stats['maximo']:.6f} metros")
    print(f"RPE Mínimo:        {stats['minimo']:.6f} metros")
    print(f"Desv. Estándar:    {stats['desv_estandar']:.6f} metros")
    print(f"Total de intervalos: {len(rpe_valores)}")
    print("="*70)

    # Guardar en CSV
    os.makedirs(carpeta_salida, exist_ok=True)
    rpe_csv = os.path.join(carpeta_salida, 'rpe_valores.csv')
    rpe_df = pd.DataFrame({
        'indice': indices,
        'error_relativo': rpe_valores
    })
    rpe_df.to_csv(rpe_csv, index=False)
    print(f"\n✓ Valores guardados: {rpe_csv}")

    print(f"\nGenerando gráficos...")
    guardar_graficos(rpe_valores, indices, carpeta_salida)

    print("\n✓ Proceso completado exitosamente\n")

if __name__ == '__main__':
    main()
```

Anexo L

```
#!/usr/bin/env python3
"""
Capturador de frecuencia de actualización del mapa.
Se ejecuta mientras SLAM está activo.

Uso:
  python3 capturador_frecuencia.py
"""

import rospy
from nav_msgs.msg import OccupancyGrid
import csv
import os
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
from datetime import datetime
import signal
import sys

class CapturadorFrecuenciaMapa:
    def __init__(self):
        # Crear directorio
        if not os.path.exists('datos_metricas'):
            os.makedirs('datos_metricas')

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        self.archivo_frecuencia = f'datos_metricas/frecuencia_mapa_{timestamp}.csv'

        # Crear archivo con encabezado
        with open(self.archivo_frecuencia, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['numero_mapa', 'timestamp', 'intervalo_s', 'frecuencia_hz'])

        print(f"✓ Archivo creado: {self.archivo_frecuencia}")
        print("✓ Esperando mapas ROS...")

        # Contador y timestamp
        self.contador = 0
        self.ultimo_timestamp = None

        # Inicializar ROS
        rospy.init_node('capturador_frecuencia_mapa', anonymous=True)
        rospy.Subscriber('/map', OccupancyGrid, self.callback_mapa, queue_size=1)

    def callback_mapa(self, msg):
        """Callback para cada actualización de mapa"""
        timestamp_actual = msg.header.stamp.to_sec()

        if self.ultimo_timestamp is not None:
            intervalo = timestamp_actual - self.ultimo_timestamp
            frecuencia = 1.0 / intervalo if intervalo > 0 else 0
        else:
            intervalo = 0
            frecuencia = 0

        # Guardar en CSV
        with open(self.archivo_frecuencia, 'a', newline='') as f:
            writer = csv.writer(f)
            writer.writerow([
                self.contador,
                timestamp_actual,
                intervalo,
                frecuencia
            ])

        self.contador += 1

        if self.contador % 20 == 0:  # Mostrar cada 20 mapas
            print(f"[MAPA] #{self.contador}: {frecuencia:.2f} Hz | "
                  f"Intervalo: {intervalo*1000:.1f}ms")
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
        self.ultimo_timestamp = timestamp_actual

def signal_handler(sig, frame):
    """Manejo de Ctrl+C"""
    print(f"\n\n✓ Captura completada")
    print(f"  Total de mapas: {capturador.contador}")
    print(f"  Archivo guardado: {capturador.archivo_frecuencia}")
    sys.exit(0)

if __name__ == '__main__':
    capturador = CapturadorFrecuenciaMapa()
    signal.signal(signal.SIGINT, signal_handler)

    try:
        rospy.spin()
    except KeyboardInterrupt:
        signal_handler(None, None)
```

Anexo M

```
#!/usr/bin/env python3
"""
Procesa datos de frecuencia de mapa y genera gráficos.
Uso: python3 procesar_frecuencia.py <archivo.csv> [carpeta_salida]
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
import os

def cargar_datos(archivo_csv):
    """Carga datos de frecuencia"""
    try:
        df = pd.read_csv(archivo_csv)
        return df
    except FileNotFoundError:
        print(f"Error: Archivo {archivo_csv} no encontrado")
        return None

def estadisticas_frecuencia(df):
    """Calcula estadísticas de frecuencia"""
    frecuencias = df[df['frecuencia_hz'] > 0]['frecuencia_hz']  # Filtrar ceros

    return {
        'promedio': np.mean(frecuencias),
        'mediana': np.median(frecuencias),
        'maximo': np.max(frecuencias),
        'minimo': np.min(frecuencias),
        'desv_estandar': np.std(frecuencias)
    }

def guardar_graficos(df, carpeta_salida):
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    """Crea y guarda gráficos de frecuencia"""

    os.makedirs(carpeta_salida, exist_ok=True)

    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle('Frecuencia de Actualización del Mapa',
                 fontsize=16, fontweight='bold')

    # Gráfico 1: Frecuencia temporal
    ax = axes[0, 0]
    ax.plot(df['numero_mapa'], df['frecuencia_hz'], 'b-', linewidth=2, alpha=0.8)
    stats = estadisticas_frecuencia(df)
    ax.axhline(stats['promedio'], color='r', linestyle='--', linewidth=2.5,
               label=f"Promedio: {stats['promedio']:.2f} Hz")
    ax.set_xlabel('Número de Mapa', fontsize=11, fontweight='bold')
    ax.set_ylabel('Frecuencia (Hz)', fontsize=11, fontweight='bold')
    ax.set_title('Evolución de Frecuencia en Tiempo Real', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.4)

    # Gráfico 2: Histograma de frecuencias
    ax = axes[0, 1]
    freq_filtrada = df[df['frecuencia_hz'] > 0]['frecuencia_hz']
    ax.hist(freq_filtrada, bins=30, edgecolor='black', alpha=0.7, color='lightblue')
    ax.axvline(stats['promedio'], color='r', linestyle='--', linewidth=2.5,
               label=f"Media")
    ax.axvline(stats['mediana'], color='g', linestyle='--', linewidth=2.5,
               label=f"Mediana")
    ax.set_xlabel('Frecuencia (Hz)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Número de ocurrencias', fontsize=11, fontweight='bold')
    ax.set_title('Distribución de Frecuencias', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.4, axis='y')

    # Gráfico 3: Intervalo entre mapas
    ax = axes[1, 0]
    intervalos_ms = df['intervalo_s'] * 1000
    ax.plot(df['numero_mapa'], intervalos_ms, 'g-', linewidth=2, alpha=0.8)
    ax.fill_between(df['numero_mapa'], 0, intervalos_ms, alpha=0.3, color='green')
    ax.set_xlabel('Número de Mapa', fontsize=11, fontweight='bold')
    ax.set_ylabel('Intervalo (milisegundos)', fontsize=11, fontweight='bold')
    ax.set_title('Tiempo Entre Actualizaciones', fontsize=12, fontweight='bold')
    ax.grid(True, alpha=0.4)

    # Gráfico 4: Estadísticas
    ax = axes[1, 1]
    ax.axis('off')

    stats_texto = f"""
    ESTADÍSTICAS DE FRECUENCIA
    {'-'*40}

    Frecuencia Promedio:  {stats['promedio']:.3f} Hz
    Frecuencia Máxima:    {stats['maximo']:.3f} Hz
    Frecuencia Mínima:    {stats['minimo']:.3f} Hz
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    Mediana:          {stats['mediana']:.3f} Hz
    Desv. Estándar:   {stats['desv_estandar']:.3f} Hz

    Intervalo Prom:   {np.mean(df['intervalo_s'])*1000:.2f} ms
    Intervalo Máx:    {np.max(df['intervalo_s'])*1000:.2f} ms
    Intervalo Mín:    {np.min(df['intervalo_s'])*1000:.2f} ms

    Total de Mapas:   {len(df)}
    Tiempo Total:     {(df['timestamp'].iloc[-1] - df['timestamp'].iloc):.1f}s
    """

    ax.text(0.1, 0.5, stats_texto, fontsize=10, family='monospace',
            verticalalignment='center', bbox=dict(boxstyle='round',
            facecolor='lightyellow', alpha=0.8))

    plt.tight_layout()
    archivo_png = os.path.join(carpeta_salida, 'frecuencia_resultado.png')
    plt.savefig(archivo_png, dpi=300, bbox_inches='tight')
    print(f"✓ Gráfico guardado: {archivo_png}")
    plt.close()

def main():
    print("="*70)
    print("PROCESADOR DE FRECUENCIA DE MAPA")
    print("="*70)

    if len(sys.argv) < 2:
        print("\nUso:")
        print("  python3 procesar_frecuencia.py <archivo.csv> [carpeta_salida]")
        print("\nEjemplo:")
        print("  python3 procesar_frecuencia.py datos_metricas/frecuencia_mapa.csv ../resultados")
        sys.exit(1)

    archivo_csv = sys.argv
    carpeta_salida = sys.argv if len(sys.argv) > 2 else '../resultados'

    print(f"\nCargando datos desde: {archivo_csv}")
    df = cargar_datos(archivo_csv)
    if df is None:
        sys.exit(1)

    print(f"✓ {len(df)} registros cargados")

    stats = estadisticas_frecuencia(df)

    print("\n" + "="*70)
    print("RESULTADOS - FRECUENCIA DE MAPA")
    print("="*70)
    print(f"Frecuencia Promedio:  {stats['promedio']:.3f} Hz")
    print(f"Frecuencia Máxima:    {stats['maximo']:.3f} Hz")
    print(f"Frecuencia Mínima:    {stats['minimo']:.3f} Hz")
    print(f"Mediana:              {stats['mediana']:.3f} Hz")
    print(f"Desv. Estándar:       {stats['desv_estandar']:.3f} Hz")
    print("="*70)
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    print(f"\nGenerando gráficos...")
    guardar_graficos(df, carpeta_salida)

    print("\n✓ Proceso completado exitosamente\n")

if __name__ == '__main__':
    main()
```

Anexo N

```
#!/usr/bin/env python3
"""
Capturador de uso de CPU y memoria.
Se ejecuta en paralelo mientras SLAM está activo.

Instalación previa:
  pip3 install psutil

Uso:
  python3 capturador_cpu.py [duracion_en_segundos]

Ejemplo:
  python3 capturador_cpu.py 600  # Captura 10 minutos
"""

import psutil
import csv
import os
from datetime import datetime
import signal
import sys
import time

class CapturadorCPU:
    def __init__(self, intervalo_muestreo=1):
        """
        intervalo_muestreo: segundos entre muestreos (default 1 segundo)
        """
        self.intervalo = intervalo_muestreo
        self.activo = True

        # Crear directorio
        if not os.path.exists('datos_metricas'):
            os.makedirs('datos_metricas')

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        self.archivo_cpu = f'datos_metricas/cpu_{timestamp}.csv'

        # Crear archivo con encabezado
        with open(self.archivo_cpu, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['timestamp', 'tiempo_transcurrido_s', 'cpu_percent',
                             'memoria_mb', 'memoria_porcentaje'])
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
        print(f"✓ Archivo creado: {self.archivo_cpu}")
        print(f"✓ Muestreo cada {intervalo_muestreo} segundo(s)")
        print("✓ Captura iniciada. Presiona Ctrl+C para detener.")

        self.tiempo_inicio = time.time()
        self.contador = 0

    def muestrear(self):
        """Realiza un muestreo de CPU y memoria"""
        timestamp_actual = time.time()
        tiempo_transcurrido = timestamp_actual - self.tiempo_inicio

        # Obtener datos
        cpu_percent = psutil.cpu_percent(interval=0.1)
        memoria = psutil.virtual_memory()
        memoria_mb = memoria.used / (1024 ** 2)
        memoria_porcentaje = memoria.percent

        # Guardar en CSV
        with open(self.archivo_cpu, 'a', newline='') as f:
            writer = csv.writer(f)
            writer.writerow([
                timestamp_actual,
                tiempo_transcurrido,
                cpu_percent,
                memoria_mb,
                memoria_porcentaje
            ])

        self.contador += 1

        # Mostrar cada 10 muestreos
        if self.contador % 10 == 0:
            print(f"[CPU] #{self.contador}: CPU {cpu_percent:5.1f}% | "
                  f"Mem {memoria_mb:7.1f}MB ({memoria_porcentaje:5.1f}%)")

    def iniciar(self, duracion=None):
        """Inicia la captura. Si duracion es None, captura indefinidamente."""
        tiempo_inicio_captura = time.time()

        try:
            while self.activo:
                self.muestrear()

                if duracion is not None:
                    if time.time() - tiempo_inicio_captura > duracion:
                        print(f"\n✓ Duración alcanzada ({duracion}s)")
                        break

                time.sleep(self.intervalo)

        except KeyboardInterrupt:
            pass

    def obtener_estadisticas(self):
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
        """Calcula estadísticas"""
        import pandas as pd
        df = pd.read_csv(self.archivo_cpu)

        return {
            'cpu_promedio': df['cpu_percent'].mean(),
            'cpu_maximo': df['cpu_percent'].max(),
            'cpu_minimo': df['cpu_percent'].min(),
            'memoria_promedio_mb': df['memoria_mb'].mean(),
            'memoria_maxima_mb': df['memoria_mb'].max(),
            'total_muestras': len(df),
            'tiempo_total': df['tiempo_transcurrido_s'].iloc[-1]
        }

def signal_handler(sig, frame):
    """Manejo de Ctrl+C"""
    capturador.activo = False

if __name__ == '__main__':
    # Duración opcional en segundos
    duracion = int(sys.argv) if len(sys.argv) > 1 else None

    capturador = CapturadorCPU(intervalo_muestreo=1)
    signal.signal(signal.SIGINT, signal_handler)

    capturador.iniciar(duracion=duracion)

    stats = capturador.obtener_estadisticas()

    print(f"\n\n✓ Captura completada")
    print(f"  Total de muestras: {stats['total_muestras']}")
    print(f"  Tiempo total: {stats['tiempo_total']:.1f}s")
    print(f"  CPU Promedio: {stats['cpu_promedio']:.1f}%")
    print(f"  CPU Máximo: {stats['cpu_maximo']:.1f}%")
    print(f"  Memoria promedio: {stats['memoria_promedio_mb']:.1f}MB")
    print(f"  Archivo guardado: {capturador.archivo_cpu}")
```

Anexo O

```
#!/usr/bin/env python3
"""
Procesa datos de CPU y memoria.
Uso: python3 procesar_cpu.py <archivo.csv> [carpeta_salida]
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
import os

def cargar_datos(archivo_csv):
    """Carga datos de CPU"""
    try:
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
        df = pd.read_csv(archivo_csv)
        return df
    except FileNotFoundError:
        print(f"Error: Archivo {archivo_csv} no encontrado")
        return None

def estadisticas_cpu(df):
    """Calcula estadísticas"""
    return {
        'cpu_promedio': df['cpu_percent'].mean(),
        'cpu_maximo': df['cpu_percent'].max(),
        'cpu_minimo': df['cpu_percent'].min(),
        'memoria_promedio': df['memoria_mb'].mean(),
        'memoria_maxima': df['memoria_mb'].max(),
        'memoria_minima': df['memoria_mb'].min()
    }

def guardar_graficos(df, carpeta_salida):
    """Crea y guarda gráficos"""

    os.makedirs(carpeta_salida, exist_ok=True)

    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle('Uso de Recursos Computacionales (CPU y Memoria)',
                 fontsize=16, fontweight='bold')

    tiempo = df['tiempo_transcurrido_s'].values
    stats = estadisticas_cpu(df)

    # Gráfico 1: CPU temporal
    ax = axes[0, 0]
    ax.plot(tiempo, df['cpu_percent'], 'r-', linewidth=2, label='CPU Instantáneo', alpha=0.8)
    # Media móvil
    cpu_media = df['cpu_percent'].rolling(window=10, center=True).mean()
    ax.plot(tiempo, cpu_media, 'darkred', linewidth=2.5, linestyle='--',
            label='Media Móvil (10s)', alpha=0.8)
    ax.axhline(stats['cpu_promedio'], color='orange', linestyle=':', linewidth=2.5,
               label=f"Promedio: {stats['cpu_promedio']:.1f}%")
    ax.fill_between(tiempo, 0, df['cpu_percent'], alpha=0.2, color='red')
    ax.set_xlabel('Tiempo (segundos)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Uso de CPU (%)', fontsize=11, fontweight='bold')
    ax.set_title('Uso de CPU en Tiempo Real', fontsize=12, fontweight='bold')
    ax.legend(fontsize=9, loc='best')
    ax.grid(True, alpha=0.4)
    ax.set_ylim([0, 100])

    # Gráfico 2: Memoria temporal
    ax = axes[0, 1]
    ax.plot(tiempo, df['memoria_mb'], 'b-', linewidth=2, label='Memoria Usada', alpha=0.8)
    # Media móvil
    mem_media = df['memoria_mb'].rolling(window=10, center=True).mean()
    ax.plot(tiempo, mem_media, 'darkblue', linewidth=2.5, linestyle='--',
            label='Media Móvil (10s)', alpha=0.8)
    ax.axhline(stats['memoria_promedio'], color='cyan', linestyle=':', linewidth=2.5,
               label=f"Promedio: {stats['memoria_promedio']:.0f}MB")
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    ax.fill_between(tiempo, 0, df['memoria_mb'], alpha=0.2, color='blue')
    ax.set_xlabel('Tiempo (segundos)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Memoria (MB)', fontsize=11, fontweight='bold')
    ax.set_title('Uso de Memoria en Tiempo Real', fontsize=12, fontweight='bold')
    ax.legend(fontsize=9, loc='best')
    ax.grid(True, alpha=0.4)

    # Gráfico 3: Distribución de CPU
    ax = axes[1, 0]
    ax.hist(df['cpu_percent'], bins=30, edgecolor='black', alpha=0.7, color='salmon')
    ax.axvline(stats['cpu_promedio'], color='darkred', linestyle='--', linewidth=2.5,
            label=f"Media: {stats['cpu_promedio']:.1f}%")
    ax.set_xlabel('Uso de CPU (%)', fontsize=11, fontweight='bold')
    ax.set_ylabel('Frecuencia', fontsize=11, fontweight='bold')
    ax.set_title('Distribución de Uso de CPU', fontsize=12, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.4, axis='y')

    # Gráfico 4: Estadísticas
    ax = axes[1, 1]
    ax.axis('off')

    stats_texto = f"""
    ESTADÍSTICAS DE RECURSOS
    {'-'*40}

    CPU (%):
      Promedio:      {stats['cpu_promedio']:6.2f}%
      Máximo:        {stats['cpu_maximo']:6.2f}%
      Mínimo:        {stats['cpu_minimo']:6.2f}%

    MEMORIA (MB):
      Promedio:      {stats['memoria_promedio']:6.1f}
      Máximo:        {stats['memoria_maxima']:6.1f}
      Mínimo:        {stats['memoria_minima']:6.1f}

    INFORMACIÓN:
      Total muestras:  {len(df)}
      Tiempo total:    {tiempo[-1]:.1f}s
      Intervalo:      ~{(tiempo[-1]-tiempo)/(len(df)-1):.2f}s
    """

    ax.text(0.05, 0.5, stats_texto, fontsize=10, family='monospace',
            verticalalignment='center', bbox=dict(boxstyle='round',
            facecolor='lightblue', alpha=0.8))

    plt.tight_layout()
    archivo_png = os.path.join(carpeta_salida, 'cpu_resultado.png')
    plt.savefig(archivo_png, dpi=300, bbox_inches='tight')
    print(f"✓ Gráfico guardado: {archivo_png}")
    plt.close()

def main():
    print("="*70)
    print("PROCESADOR DE DATOS DE CPU Y MEMORIA")
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.

```
    print("="*70)

    if len(sys.argv) < 2:
        print("\nUso:")
        print("  python3 procesar_cpu.py <archivo.csv> [carpeta_salida]")
        print("\nEjemplo:")
        print("  python3 procesar_cpu.py datos_metricas/cpu_20251114.csv ../resultados")
        sys.exit(1)

    archivo_csv = sys.argv
    carpeta_salida = sys.argv if len(sys.argv) > 2 else '../resultados'

    print(f"\nCargando datos desde: {archivo_csv}")
    df = cargar_datos(archivo_csv)
    if df is None:
        sys.exit(1)

    print(f"✓ {len(df)} registros cargados")

    stats = estadisticas_cpu(df)

    print("\n" + "="*70)
    print("RESULTADOS - RECURSOS COMPUTACIONALES")
    print("="*70)
    print(f"CPU Promedio:       {stats['cpu_promedio']:.2f}%")
    print(f"CPU Máximo:         {stats['cpu_maximo']:.2f}%")
    print(f"CPU Mínimo:         {stats['cpu_minimo']:.2f}%")
    print(f"Memoria Promedio:    {stats['memoria_promedio']:.1f}MB")
    print(f"Memoria Máximo:      {stats['memoria_maxima']:.1f}MB")
    print("="*70)

    print(f"\nGenerando gráficos...")
    guardar_graficos(df, carpeta_salida)

    print("\n✓ Proceso completado exitosamente\n")

if __name__ == '__main__':
    main()
```

Este documento es propiedad de la Universidad Distrital Francisco José de Caldas. Prohibida su reproducción por cualquier medio, sin previa autorización.