UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Turtlebot 4 y ROS2: análisis de algoritmos SLAM y de navegación.

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Folch Company, Francesc

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO: 2022/2023

# Resumen

Los algoritmos de localización y mapeo simultáneos, más conocidos como algoritmos SLAM, permiten a los robots construir mapas de entornos desconocidos y localizarse sobre ellos al mismo tiempo, lo cual es esencial para la navegación precisa y la toma de decisiones informadas en tiempo real.

El presente proyecto ha puesto en marcha el robot móvil TurtleBot 4, y con el se han analizado los algoritmos SLAM más conocidos en la actualidad. TurtleBot 4 es una plataforma de la empresa canadiense Clearpath Robotics que utiliza ROS 2, la nueva versión del *Robot Operating System* que, pese a tener un periodo corto de vida y no tener tantas librerías desarrolladas como ROS 1, supone un gran avance respecto a la primera versión.

Utilizando estas últimas tecnologías, se ha realizado un estudio comparativo de varios algoritmos SLAM haciendo uso de las métricas y *benchmarks* que más se ajustaban al marco del proyecto. Este estudio tiene la intención de ver cuál tiene un mejor desempeño, en términos de eficiencia y precisión, sobre la plataforma TurtleBot 4. Finalmente se ha valorado el estado de madurez del *framework* ROS 2 y de la plataforma TurtleBot 4 como marco de desarrollo y ejecución de algoritmos SLAM.

**Palabras clave:** SLAM, TurtleBot, ROS, benchmarks

# Resum

Els algorismes de localització i mapeig simultanis, més coneguts com a algorismes SLAM, permeten als robots construir mapes d'entorns desconeguts i localitzar-se al mateix temps, cosa que és essencial per a la navegació precisa i la presa de decisions informades en temps real.

Aquest projecte ha posat en marxa el robot mòbil TurtleBot 4, i amb ell s'han analitzat els algorismes SLAM més coneguts actualment. TurtleBot 4 és una plataforma de l'empresa canadenca Clearpath Robotics que utilitza ROS 2, la nova versió del *Robot Operating System* que, malgrat tenir un període curt de vida i no tenir tantes llibreries desenvolupades com ROS 1, suposa un gran avenç respecte a la primera versió.

Utilitzant aquestes darreres tecnologies, s'ha realitzat un estudi comparatiu de diversos algorismes SLAM fent ús de les mètriques i *benchmarks* que més s'ajustaven al marc del projecte. Aquest estudi té la intenció de veure quin té un millor exercici, en termes d'eficiència i precisió, sobre la plataforma TurtleBot 4. Finalment s'ha valorat l'estat de maduresa del *framework* ROS 2 i de la plataforma TurtleBot 4 com a marc de desenvolupament i execució d'algorismes SLAM.

**Paraules clau:** SLAM, TurtleBot, ROS, benchmarks

# Abstract

Simultaneous localization and mapping algorithms, better known as SLAM algorithms, allow robots to construct maps of unknown environments and localize over them at the same time, which is essential for accurate navigation and informed real-time decision making.

The present project has implemented the mobile robot TurtleBot 4, and with it we have analyzed the most popular SLAM algorithms currently available. TurtleBot 4 is a platform of the Canadian company Clearpath Robotics that uses ROS 2, the new version of the Robot Operating System, which, despite having a short life span and not having as many libraries developed as ROS 1, is a great advance over the first version.

Using these latest technologies, a comparative study of several SLAM algorithms has been carried out using the metrics and benchmarks that best fit the project framework. This study is intended to see which one has a better performance, in terms of efficiency and accuracy, on the TurtleBot 4 platform. Finally, the maturity status of ROS 2 and the TurtleBot 4 platform as a framework for the development and execution of SLAM algorithms has been assessed.

**Key words:** SLAM, TurtleBot, ROS, benchmarks

# Índice general

# Índice de figuras



# Índice de tablas

# CAPÍTULO 1
# Introducción

## 1.1 Motivación

En los últimos años, el uso de robots industriales, y en concreto de robots móviles, ha crecido enormemente, y se espera que aumente cada año el número de instalaciones respecto al anterior (ver Fig. 1.1). Por lo que respecta al área de la robótica móvil, la capacidad de los robots para percibir y comprender su entorno ha aumentado enormemente, lo cual es esencial para su funcionamiento exitoso en una amplia gama de aplicaciones. En este contexto, los algoritmos SLAM (Simultaneous Localization and Mapping) han surgido como una pieza fundamental, permitiendo a los robots mapear entornos desconocidos mientras estiman su propia ubicación en tiempo real [1].

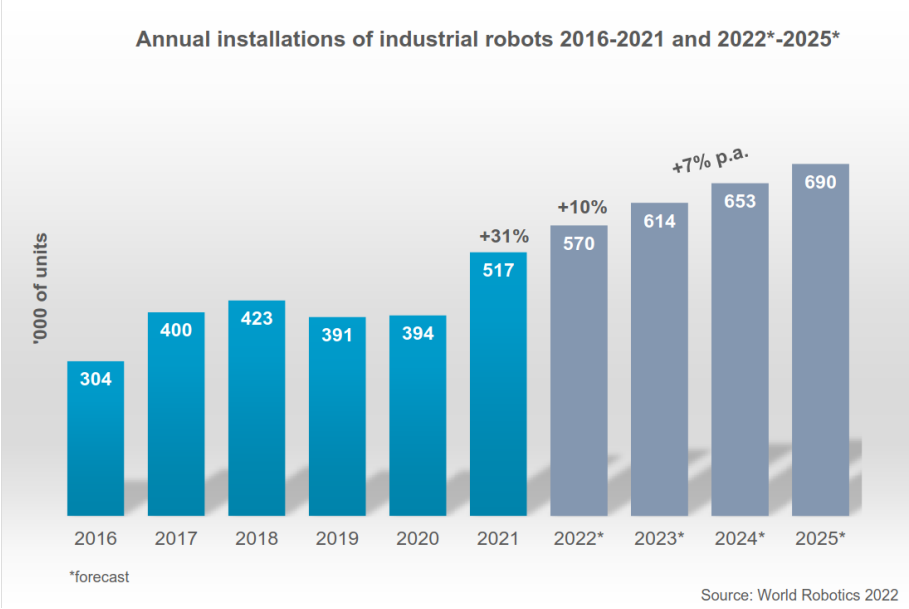


**Figura 1.1:** Proyección de crecimiento en la instalación anual de robots industriales hasta el año 2025.
Fuente: World Robotics. *Market presentation World Robotics 2022 extended version*. 2022.

La creación de mapas juega un papel importante en aspectos clave de la robótica:

- Muchas aplicaciones robóticas tienen como objetivo final la formación de un mapa.

- Los mapas son necesarios en la ejecución de algoritmos de navegación, concretamente para el *path planning*[2], la evasión de obstáculos, etc.

Se ha de tener en cuenta que la precisión del mapa generado influye en la robustez y en la certeza de la localización de un robot.

Sin embargo, la elección del algoritmo SLAM adecuado puede ser un desafío, ya que la eficiencia y precisión de estos algoritmos varían según el contexto y las condiciones del entorno.

En este escenario, ROS 2 (Robot Operating System 2) se ha establecido como una plataforma robusta y versátil para el desarrollo de sistemas robóticos [3]. Su enfoque en la comunicación distribuida, portabilidad y escalabilidad lo convierte en una opción atractiva para la implementación de algoritmos SLAM en diversas aplicaciones. A medida que ROS 2 evoluciona y madura, una variedad de algoritmos SLAM han sido implementados en esta plataforma, cada uno con sus propias características y enfoques. Debido a que muchas de las librerías disponibles en ROS 1 todavía no han sido migradas a ROS 2, es necesario también llevar a cabo un estudio sobre qué algoritmos ya han sido migrados con el objetivo de obtener un catálogo de algoritmos SLAM disponibles para ROS 2.

El análisis y benchmarking de los algoritmos SLAM disponibles en ROS 2 se presenta como un paso crucial para entender el rendimiento y las capacidades de estos algoritmos en condiciones del mundo real. Dado que diferentes aplicaciones requieren niveles variables de precisión, eficiencia computacional y adaptabilidad a diferentes tipos de sensores, se hace necesario evaluar y comparar detenidamente estos algoritmos para tomar decisiones informadas en el diseño y despliegue de sistemas robóticos autónomos.

Este proyecto se propone abordar este desafío al analizar exhaustivamente los algoritmos SLAM implementados en ROS 2 en el contexto de un TurtleBot 4. A través de un proceso de evaluación comparativa, se pretende identificar las fortalezas y debilidades de cada algoritmo en términos de precisión de la localización, calidad del mapeo y eficiencia computacional. Los resultados obtenidos tendrán implicaciones significativas en la elección y configuración adecuada de algoritmos SLAM en la plataforma TurtleBot 4, contribuyendo así al avance y mejora continua de la robótica autónoma en entornos diversos y desafiantes.

¿Y por qué la elección del TurtleBot 4? Originalmente la primera versión del TurtleBot fue diseñada por Willow Garage, los creadores de ROS. La filosofía del TurtleBot es clara: debe ser económicamente asequible y con una base de código abierto, con los sensores suficientes para realizar tareas de manera autónoma y con un carácter modular y expansible. Es por esto que, sobre el papel, el TurtleBot 4 es la mejor de las opciones para la realización de este estudio, con una capacidad de cómputo mejorada, mejores sensores y ROS 2 instalado como estándar.

## 1.2 Objetivos

Los objetivos del presente trabajo son los siguientes:

1. **Definir** las técnicas SLAM, sus principales características y sus diferencias.
2. **Identificar** los métodos de evaluación de algoritmos SLAM que más se ajusten al marco del trabajo.

3. **Listar** y **evaluar** los métodos SLAM implementados en librerías disponibles en ROS 2.
4. **Redactar** una guía de instalación y puesta en marcha del TurtleBot 4 para la creación y uso de mapas.
5. **Calificar** la capacidad del TurtleBot 4 como plataforma para la ejecución de algoritmos de mapeo, localización y navegación.
6. **Valorar** el estado de madurez y desarrollo de ROS 2 para el estudio y pruebas de algoritmos SLAM.

## 1.3 Estructura

La estructura del trabajo será clara y concisa: en primer lugar, se analizará el estado del arte de los algoritmos SLAM, sus diferentes componentes y utilidades, los algoritmos disponibles en ROS 2 y los métodos de evaluación de calidad. En segundo lugar, se repasarán las características principales de ROS 2 y sus diferencias respecto a ROS 1, así como la descripción y análisis del TurtleBot 4, la plataforma hardware en la que se realizará el trabajo. En tercer lugar, se analizarán las principales fallas en el estado del arte de los algoritmos SLAM en el contexto del TurtleBot 4.

Tras esto, se dará paso al desarrollo e implementación del proyecto, empezando por la instalación y puesta en marcha del software y hardware del TurtleBot 4 Standard, ya que no es un proceso trivial, puesto que se han tenido que tomar decisiones sobre la elección de versiones e implementaciones de ciertas librerías. Se proseguirá con la ejecución de los algoritmos y la obtención de los resultados. Finalmente se analizarán y representarán los resultados de las métricas de calidad de forma numérica y gráfica.

En última instancia, en las conclusiones, se valorará si se han cumplido o no los objetivos con su propia justificación. También se relacionará el trabajo con las materias cursadas en el máster, y finalmente se propondrán líneas de investigación y trabajo futuras para complementar y enriquecer el trabajo ya realizado.

# CAPÍTULO 2
# Estado del arte

## 2.1 Conceptos actuales de los algoritmos SLAM

### 2.1.1. Definición

Actualmente, los robots móviles se utilizan en una amplia variedad de situaciones, como en la industria, hogares, educación y atención médica. Cuando se trata de robots que pueden moverse de manera autónoma, se enfrentan al desafío de operar ubicarse y operar en entornos desconocidos. Por lo que deben construir un mapa de ese entorno (mapeo), saber dónde están (localización) y navegar hacia sus objetivos (navegación).

El problema de mapeo, localización y navegación es fundamental en la robótica móvil. Es un tema de investigación importante ya que resolver estos problemas de manera confiable es crucial para hacer que los robots móviles sean más autónomos y útiles en diversas aplicaciones.

Existen dos enfoques principales para representar el entorno,: los mapas métricos, representados con precisión geométrica, y los mapas topológicos, que son representados como una especie de grafo donde los nodos corresponden a características del entorno localizadas en el espacio. En lo que respecta a la localización, esto implica que el robot necesita saber dónde está dentro de ese mapa. Para hacerlo, primero debe construir un mapa y luego, usando ese mapa, puede estimar su ubicación. Sin embargo, también hay un enfoque llamado "Simultaneous Localization And Mapping"(SLAM) que combina estas dos tareas. El robot construye un mapa mientras se mueve y, al mismo tiempo, utiliza ese mapa en tiempo real para saber dónde está [4].

### 2.1.2. Uso de sensores

Hoy en día existen numerosas técnicas SLAM con diferentes enfoques y han sido desarrolladas utilizando todo tipo de sensores. Estos algoritmos pueden utilizar distintos tipos de sensores, como se puede observar en la tabla 2.1.

Además, la mayoría de algoritmos utilizan la odometría para aumentar la precisión en los resultados. La odometría es la estimación de la posición y velocidad del robot utilizando un modelo cinemático y los *encoders* de los motores. Por ejemplo, el TurtleBot 4 se compone de dos ruedas con motores independientes y una pequeña rueda tipo cas-

| Tipo de SLAM | Sensores Utilizados |
|---|---|
| SLAM Visual | Cámaras: Monocular, Estereoscópica, RGB-D (Cámara RGB + Sensor de Profundidad) |
| LIDAR SLAM | 2D LIDAR, 3D LIDAR |
| SLAM Inercial | IMU (Acelerómetros y Giroscopios) |
| SLAM de ultrasonidos | Sensor de ultrasonidos |
| Fusión de Sensores | Múltiples sensores combinados |

**Tabla 2.1:** Clasificación de Algoritmos SLAM según los sensores utilizados

tor que actúa como pivote. Esta configuración utiliza el modelo cinemático diferencial (ver figura 2.1). La odometría también puede hacer uso de otros sensores como la IMU (Unidad de medición inercial) para aumentar su precisión.

En ROS 2, la estimación de la velocidad y posición se suele publicar junto con un grado de incertidumbre. La plataforma Create 3, de la que se compone el TurtleBot 4, hace uso de un ratón óptico, además de la IMU y los *encoders* de los motores, para detectar el grado de deslizamiento de las ruedas con el suelo [5].

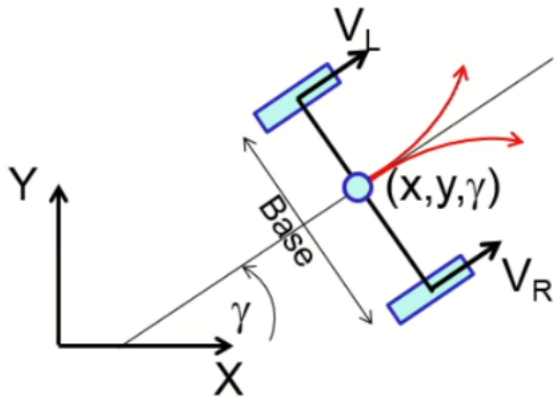


**Figura 2.1:** Modelo cinemático diferencial.
Fuente: DISA. ROBOTS MÓVILES. *Modelado Cinemático*.

Sin embargo, en la actualidad se suele hacer uso de una combinación de dos o más sensores para conseguir una mayor precisión en la localización del robot. La integración del componente inercial sobre otros sensores suele ser la más común. Por ejemplo, los algoritmos SLAM de tipo Visual-Inercial combinan información de cámaras e IMU; o los algoritmo SLAM de tipo 2D-Inercial fusionan LIDAR 2D e IMU.

### 2.1.3. Técnicas

El posicionamiento o localización es uno de los problemas esenciales del SLAM. Y diferentes algoritmos SLAM se pueden clasificar también según la técnica de posicionamiento utilizada, pudiendo tener un enfoque **probabilístico** o **no probabilístico**. El mejor enfoque y el utilizado hoy en día es el probabilístico, el cual hace uso de métodos estadísticos basados en el método de estimación Bayesiano. En este método se suelen aplicar filtros de partículas o filtros de Kalman Extendidos.

Estos dos filtros son idóneos en situaciones donde la relación entre la lectura de los sensores y la posición del robot y de su entorno no es lineal ni directa. Por ejemplo, cuando hay que combinar la información de ciertos sensores, o aparecen factores dinámicos del robot o reflexiones y distorsiones en las lecturas de algún sensor.

Sin embargo, un problema de los filtros es que los datos procesados se descartan una vez se han utilizado para estimar la porción del mapa y la posición actuales del robot. Esto significa que no se pueden volver a utilizar todos los datos cuando se construye el mapa final. Las técnicas más modernas utilizan grafos dispersos [6], donde los nodos representan las variables de interés, como la posición del robot y las ubicaciones de las características del entorno, mientras que las aristas entre los nodos representan las mediciones de distancia y orientación entre esas posiciones. El objetivo principal en SLAM basado en grafos es encontrar una estimación coherente de las variables del grafo que cumpla con todas las restricciones representadas por las aristas del grafo.

### 2.1.4. Scan Matching

El *scan matching* es una técnica iterativa para estimar la posición del robot dadas las lecturas de un LIDAR. Esencialmente consiste en comparar la medición actual con datos de referencia almacenados previamente en un mapa. El mapa contiene mediciones conocidas o características que se han adquirido durante la exploración anterior del robot. El objetivo es encontrar la mejor alineación entre las mediciones actuales y las del mapa de referencia. Esto implica ajustar la posición (traslación) y la orientación (rotación) del robot para minimizar las diferencias entre las mediciones actuales y las del mapa. La posición y la orientación resultantes se consideran la estimación de la ubicación del robot en ese momento (ver 2.2).

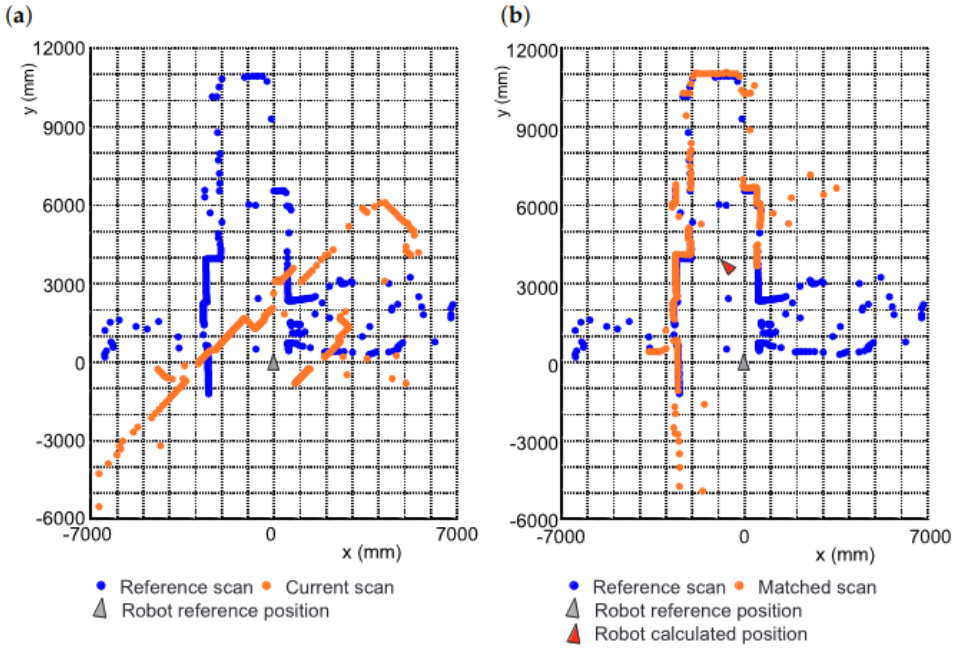


**Figura 2.2:** Scan matching: definición del problema. (a) Scans antes del alineamiento (b) Scans después del alineamiento.
Fuente: https://cutt.ly/TwzUIXP1 [consultado el 5 Sep, 2023]

El *scan matching* también se puede describir como una técnica para encontrar los parámetros de transformación de la lectura actual y el mapa de referencia.

Uno de los métodos más utilizados es *Iterative closest point* (ICP) [7], el cual intenta minimizar el RMSE (raíz de la desviación cuadrática media). Recientemente surgió un nuevo método (*Scan Matching by Cross-Correlation and Differential Evolution*) cuyo rendimiento es mayor cuando se comparan dos *scans* con mayor distancia entre ellos, este método combina *Cross-Correlation*, para medir la similitud entre dos *scans*, y *Differential Evolution*, para encontrar los parámetros de transformación del robot [8].

### 2.1.5. Loop Closing

Un concepto fundamental en las técnicas SLAM es el *loop closing* o cierre de bucle. Cuando un robot móvil se desplaza por un entorno, puede volver a visitar una ubicación previamente mapeada. La detección de bucles implica identificar cuándo el robot ha regresado a una ubicación que ya había registrado en su mapa anteriormente. Cuando se detecta un bucle, se realiza una corrección en la trayectoria del robot y en el mapa. Esto implica ajustar la estimación de la trayectoria para corregir la desviación acumulada y obtener un mapa coherente.

Los algoritmos actuales resuelven el cierre de bucle como problemas de optimización de grafos, utilizando los métodos SPA (*Sparse Pose Adjustment*) [9] o SBA (*Sparse Bundle Adjustment*) [10], ya que son eficientes y óptimos para problemas de tiempo real. Aunque en 2021 se ha demostrado en un estudio que el uso de redes neuronales puede mejorar el rendimiento del cierre de bucle respecto a los otros métodos modernos basados en optimización de grafos [11].

## 2.2 Métodos SLAM

Pese a que muchos algoritmos son clasificados según los sensores que utilizan [12], también se pueden clasificar los métodos SLAM según el enfoque y las técnicas utilizadas. Existen dos principales grupos, SLAM basados en filtros y SLAM basados en grafos.

### 2.2.1. HectorSLAM

**HectorSLAM** es un enfoque SLAM que puede utilizarse sin odometría y en plataformas que presentan movimientos de balanceo e inclinación (del sensor, de la plataforma o de ambos). Utiliza un tipo de **filtro de Kalman extendido** (EKF) para estimar la posición y construir el mapa. Aprovecha la alta frecuencia de actualización de los sistemas LIDAR modernos y proporciona estimaciones de pose 2D a la frecuencia de barrido de los sensores. Aunque el sistema no ofrece una capacidad explícita de cierre de bucle, es suficientemente preciso para muchos escenarios del mundo real. El sistema se ha utilizado con éxito en robots terrestres no tripulados, vehículos de superficie no tripulados, dispositivos cartográficos portátiles y datos registrados de vehículos aéreos no tripulados.

Por lo tanto, HectorSLAM se especializa en el uso de sensores LIDAR. Puede generar mapas con una alta frecuencia de actualización, sin embargo, no usa datos de odometría,

por lo que pueden surgir errores de precisión cuando los datos del LIDAR llegan a menor frecuencia o cuando el mapa es grande o escaso de objetos.

El repositorio oficial de HectorSLAM en GitHub no da soporte a ROS 2 y los creadores no planean migrar de ROS 1 a la nueva versión [1]. Pese a esto, desarrolladores independientes crearon un nuevo repositorio para ROS 2, pero ha dejado de mantenerse [2]. El archivo launch no llega a ejecutarse debido a múltiples errores, aunque todavía puede funcionar si se ejecuta el nodo directamente.

### 2.2.2. Gmapping

Presentado en 2007, **Gmapping** ha sido una de las librerías de SLAM más usadas. Utiliza un filtro de partículas, por lo que no escala bien en mapas grandes. El uso de filtros de partículas *Rao-Blackwellized*[13] son medios eficaces para resolver el problema de localización y mapeo simultáneos (SLAM). Este enfoque utiliza un filtro de partículas en el que cada partícula lleva un mapa individual del entorno. En consecuencia, una cuestión clave es cómo reducir el número de partículas. Gmapping presenta técnicas adaptativas para reducir el número de partículas en un filtro de partículas Rao-Blackwellized para el aprendizaje de mapas reticulares. También propone un enfoque para calcular una distribución de propuestas precisa teniendo en cuenta no sólo el movimiento del robot, sino también la observación más reciente. Esto disminuye drásticamente la incertidumbre sobre la posición del robot en el paso de predicción del filtro. Gmapping también aplica un enfoque para realizar selectivamente operaciones de remuestreo que reducen seriamente el problema del agotamiento por consumo de partículas.

El enfoque de este algoritmo toma datos de LIDAR 2D y odometría. Esta versión está optimizada para escáneres láser de largo alcance. Los LIDAR de corto alcance como el escáner Hokuyo no funcionarán tan bien con la configuración de parámetros estándar.

Al igual que en el caso de HectorSLAM, el repositorio de ROS de slam_gmapping solo da soporte a ROS 1 [3]. Por lo que se migró de forma independiente a ROS 2, pero dejó de mantenerse en 2021. Aunque todavía es una librería funcional que puede instalarse en la última distribución [4].

### 2.2.3. Cartographer

**Cartographer**, desarrollado por Google, ofrece una solución en tiempo real para generar mapas de cuadrícula 2D con una resolución de celda de 5 cm. Los escaneados láser se insertan en un submapa en la mejor posición estimada, que se supone suficientemente precisa para periodos cortos de tiempo. El *scan-matching* se realiza con un submapa reciente, por lo que sólo depende de los escaneos recientes, y se acumula el error de las estimaciones de la posición en el *frame* global.

Para lograr un buen rendimiento con hardware modesto, Cartographer no emplea un filtro de partículas, si no un grafo de posiciones y características. Para hacer frente a

[1] Ver repositorio en: https://github.com/tu-darmstadt-ros-pkg/hector_slam

[2] El repositorio de HectorSLAM en ROS 2 se encuentra en: https://github.com/RRL-ALeRT/hector_slam_ros2

[3] Repositorio oficial de gmapping en: https://github.com/ros-perception/slam_gmapping

[4] Nuevo repositorio de gmapping en: https://github.com/Project-MANAS/slam_gmapping

la acumulación de errores, se ejecuta periódicamente una optimización de la posición. Cuando se termina un submapa, es decir, ya no se insertan en él nuevas exploraciones, se ejecuta el cierre de bucle.

Este algoritmo tiene dos niveles de SLAM, el nivel local o *front-end*, que se encarga del *scan-matching* y de formar la trayectoria y los submapas; y el nivel global o *back-end*, el cual se ejecuta en paralelo y se encarga de encontrar cierres de bucle y de generar el mapa global (ver Fig. 2.3). A demás, es altamente configurable y parametrizable, con muchas variables para ajustar el funcionamiento del SLAM local y el global.

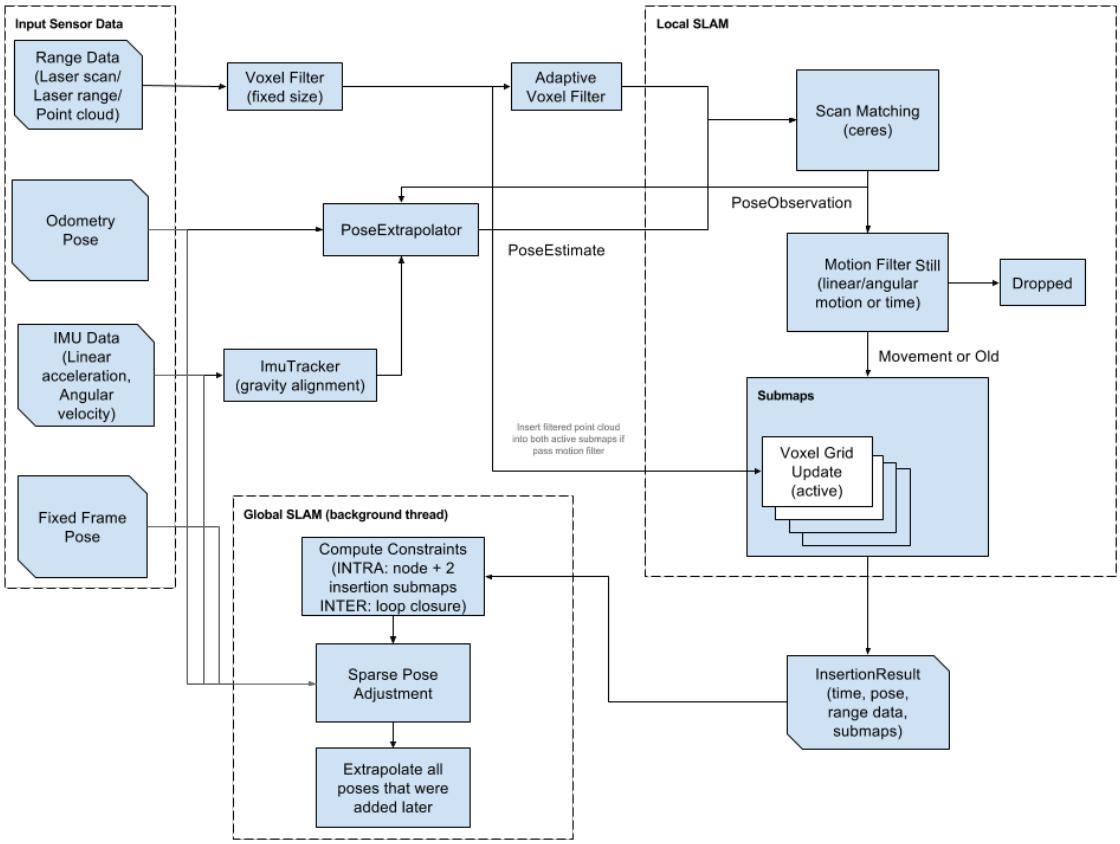


**Figura 2.3:** Esquema del funcionamiento de Cartographer
Fuente: https://google-cartographer.readthedocs.io/en/latest/ [consultado el 10 Sep, 2023]

El uso de la IMU en este algoritmo otorga capacidades de estimación más precisas al *scan matcher*.

### 2.2.4. SLAM Toolbox

**SLAM Toolbox**, que utiliza una modificación de la librería OpenKarto [14], es un conjunto de librerías y herramientas que se ha convertido en el estándar de ROS 2, por lo que está instalado por defecto en el TrutleBot 4. Dispone de *loop-closing* y está basado en optimización de grafos. Dispone de numerosas funcionalidades:

- SLAM 2D de fácil ejecución, con herramientas incluidas como guardar mapas.

- Ofrece la posibilidad de crear mapas en diferentes sesiones, con la capacidad de serializar y deserializar mapas o editar grafos de posiciones.
- Ofrece un modo de solo localización basado en grafos de posiciones, con la opción de utilizar un mapa o sin mapa (con odometría de LIDAR).
- Mapeo síncrono y asíncrono.
- Fusión de mapas basado en manipulación elástica de grafos.
- Posibilidad de insertar nuevos optimizadores en forma de *plug-in*. El optimizador por defecto es una versión de Ceres, el optimizador diseñado por Google.
- Ofrece un *plug-in* en RVIZ para interactuar con las herramientas de forma gráfica.

### 2.2.5. RTAB-Map

**RTAB-Map** es un algoritmo de SLAM-Visual y también está disponible en ROS 2. RTAB-Map es una herramienta de código abierto utilizada en una variedad de aplicaciones robóticas, que van desde la navegación autónoma de robots móviles hasta la construcción de mapas 3D en hogares y entornos industriales y de investigación [15].

A continuación, se describen los componentes y característicTFM-Turnas clave de RTAB-Map:

- **Sensores**: RTAB-Map es compatible con una variedad de sensores, como cámaras RGB-D , láseres (LIDAR), cámaras RGB convencionales y sensores inerciales (IMU). Los datos de estos sensores se utilizan para capturar información sobre el entorno, incluyendo la geometría y la apariencia visual.
- **Localización Simultánea y Mapeo (SLAM):** RTAB-Map utiliza el enfoque SLAM para estimar la posición del robot en tiempo real mientras crea un mapa del entorno. Esto implica una estimación continua de la posición y la orientación del robot, así como la construcción simultánea del mapa.
- **Construcción de Mapas 3D:** RTAB-Map se especializa en la construcción de mapas tridimensionales (3D) del entorno. Esto significa que no solo crea mapas 2D de las ubicaciones de los obstáculos, sino que también incorpora información sobre la elevación y la geometría 3D de los objetos y el terreno.
- **Detección de Características Visuales:** RTAB-Map utiliza técnicas de visión por computadora para detectar y reconocer características visuales en las imágenes capturadas por las cámaras. Esto permite la identificación de ubicaciones clave en el entorno, que luego se utilizan para la localización y la construcción del mapa.
- **Gestión de Grandes Conjuntos de Datos:** RTAB-Map ha sido diseñado para manejar grandes conjuntos de datos, lo que lo hace adecuado para aplicaciones en las que los robots exploran extensas áreas o recopilan una gran cantidad de datos.
- **Optimización y Corrección de Errores:** El sistema RTAB-Map incorpora algoritmos de optimización que ayudan a corregir errores acumulativos en la estimación de la posición y la construcción del mapa. Esto mejora la precisión a lo largo del tiempo.

- **Interfaz de Usuario y Visualización:** RTAB-Map suele estar acompañado de una interfaz de usuario y herramientas de visualización que permiten a los usuarios supervisar y analizar el proceso de mapeo y localización en tiempo real.

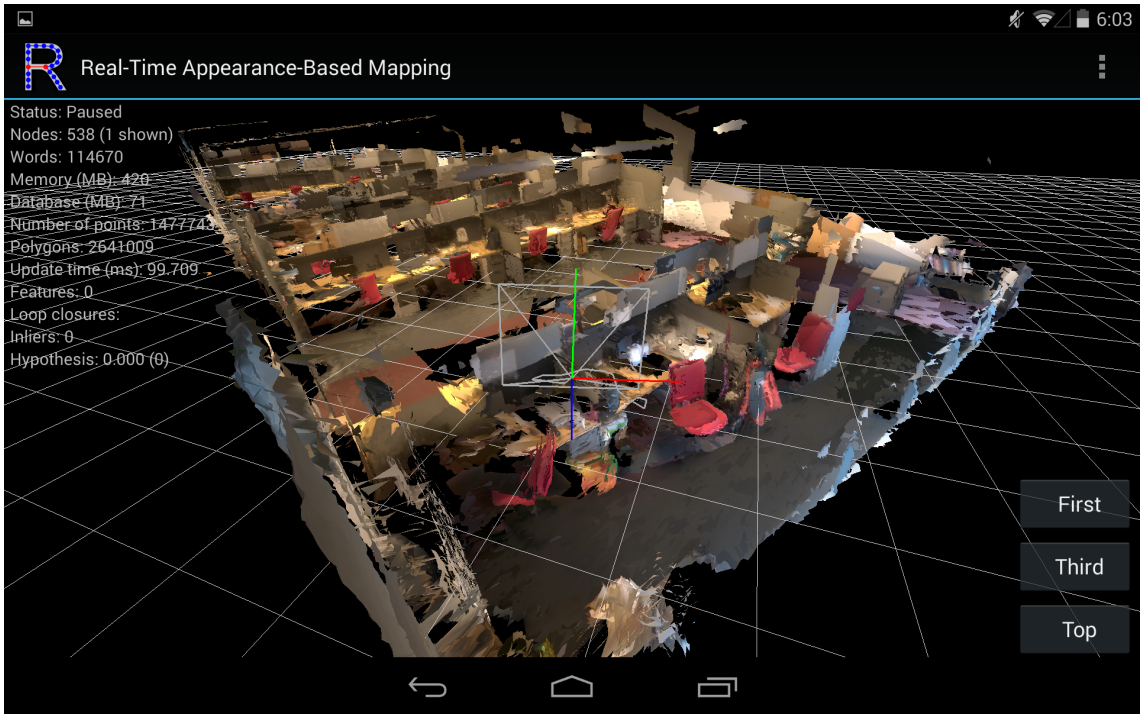


**Figura 2.4:** Oficinas mapeadas utilizando Tango, una aplicación de Android que ejecuta RTAB-Map.
Fuente: https://introlab.github.io/rtabmap/ [consultado el 10 Sep, 2023]

RTAB-Map es una herramienta muy extensa, y está disponible en muchas plataformas, incluyendo como paquete de ROS 2, aunque también está disponible en Linux como una librería independiente y como una aplicación de Android. En la figura 2.4 se puede observar una captura de pantalla de Tango, una aplicación de Android que ejecuta RTAB-Map.

### 2.2.6. ORB-SLAM3

**ORB-SLAM3** es un sistema de mapeo que utiliza información visual y visual-inercial para estimar la posición y crear mapas de entornos 3D. A continuación, se explican los puntos clave este nuevo algoritmo:

- **Novedades de ORB-SLAM3**: ORB-SLAM3 es el primer sistema visual y visual-inercial que puede aprovechar de manera efectiva las asociaciones de datos a corto plazo, a mediano plazo, a largo plazo y de múltiples mapas. Estas asociaciones se refieren a cómo el sistema relaciona la información de diferentes momentos y lugares en su entorno. ORB-SLAM3 ha alcanzado un nivel de precisión que supera a los sistemas existentes [16].
- **Comparación con Enfoques Alternativos**: En términos de precisión, la capacidad de ORB-SLAM3 para utilizar estas asociaciones de datos es superior a otras opciones, como el uso de métodos directos en lugar de características visuales o la

marginalización de fotogramas clave para la optimización local, en lugar de asumir un conjunto externo de fotogramas clave estáticos.

- **Desafíos en Entornos con Poca Textura**: El principal problema de ORB-SLAM3 es su rendimiento en entornos con poca textura, es decir, lugares donde no hay muchas características visuales distintivas para rastrear. Los métodos directos son más robustos en tales entornos, pero limitados a asociaciones de datos a corto y mediano plazo.

Pese a que este método es novedoso y presenta grandes ventajas y que existen varios repositorios que implementan la librería en ROS 2, estos no se encuentran en un estado óptimo de desarrollo, ya que presentan numerosos errores de compilación y ejecución y los mantienen una comunidad pequeña de desarrolladores [5]. Para empeorar la situación, el repositorio de la librería base de ORB-SLAM3 contiene 453 *issues* a fecha de Septiembre de 2023.

Se han invertido horas en intentar migrar y compilar el repositorio de ORB-SLAM3 de zang09. El repositorio originial había sido compilado en ROS 2 Foxy y Ubuntu 20.04, por lo que había que cambiar algunas dependencias que rompían la compilación:

- OpenCV: Se ha instalado y compilado la versión 4.5.4 de OpenCV, ya que habían conflictos con la versión anterior 4.2.0.
- CMakeList: Se han tenido que modificar varias líneas del archivo de compilación CMake para conseguir una compilación exitosa.

Tras aplicar y validar los cambios se solicita un *Pull Request* a una nueva rama nueva del repositorio para ROS 2 Humble. Aunque llega a compilar, al lanzar el nodo se produce un error de segmentación durante la ejecución y no muestra ningún mensaje adicional de error, por lo que la depuración es casi imposible y no se consigue proseguir con el desarrollo de ORB-SLAM3, ya que excede la cantidad de trabajo y tiempo disponible para este proyecto.

## 2.3 Métodos de evaluación

Para poder realizar un correcto estudio de los algoritmos SLAM también se han considerar los métodos de evaluación o medición de calidad conocidos, así como analizar cada cada uno de estos para valorar su viabilidad dados los recursos disponibles en el proyecto. Por ejemplo, existen métodos que requieren marcos de referencia, de los cuales no siempre se tiene acceso.

Para evaluar la calidad de un algoritmo se debe de calificar la precisión tanto del mapa, como de la ubicación y trayectoria del robot generados por el algoritmo SLAM:

- **Trayectoria:** No es posible evaluar la localización de un algoritmo SLAM sin la disposición de algún tipo de *ground truth*. Se evalúa comparándola con un marco de referencia que se puede obtener de las siguientes formas:

  1. Utilizando herramientas externas como un sistema *OptiTrack* que recolecte las posiciones del robot. Esta trayectoria puede compararse con la calculada por el

[5] Respositorio creado por el usuario alsora: https://github.com/alsora/ros2-ORB_SLAM2

algoritmo SLAM utilizando métricas precisas como la energía de deformación [17], la cual se utiliza ampliamente.

2. Utilizando una trayectoria marcada en el suelo si no se dispone de estas herramientas [18]. Esto tiene una limitación, y es que no es posible utilizar trayectorias curvas, por lo que solo es efectivo sobre rectas y giros estáticos, a demás, es un proceso manual, por lo que en trayectorias largas se pueden acumular errores.

- **Mapa:** Los mapas se pueden evaluar de varias formas:

1. Mediante inspección visual, que puede ser valiosa para detectar problemas obvios o errores en el mapa, como artefactos o características incorrectamente ubicadas.

2. Superponiendo el mapa generado con un *ground truth* (mapa de referencia) mediante el método de ICP, utilizado también en *scan matching*, y calculando el error medio, el cual es la media de las distancias de cada punto respecto al punto más cercano [19]. El mapa de referencia se genera utilizando láser de alta precisión y construyéndolo manualmente.

3. Utilizando métricas independientes a cualquier mapa de referencia, como el número de esquinas en un mapa, el número de áreas encerradas o la proporción de celdas ocupadas [20]. Estas métricas en sí no dicen nada, pero se pueden evaluar de forma comparativa si se contrastan entre los resultados de varios mapas del mismo entorno.

## 2.4 Plataforma de desarrollo

A continuación se analizará el contexto tecnológico: tanto la versión del *framework* ROS, el simulador disponible, como los diferentes componentes del robot y los sensores y actuadores disponibles. Esto es necesario ya que condicionarán los resultados de cada algoritmo y proporcionarán un marco de trabajo en el que llevar a cabo el proyecto.

### 2.4.1. ROS 2

Un *framework* de robótica, también conocido como marco de robótica o plataforma de robótica, es un conjunto de herramientas, bibliotecas y software diseñado para simplificar y agilizar el desarrollo de aplicaciones y sistemas robóticos. Estos *frameworks* proporcionan una estructura y una base sobre la cual los desarrolladores pueden construir, programar y controlar robots de manera más eficiente.

Uno de estos *frameworks* es ROS 2 (*Robot Operating System*), la versión más reciente y avanzada del sistema operativo de código abierto diseñado para la programación y control de robots [21]. Las ventajas respecto a la primera versión son numerosas y representan un gran avance en la industria de la robótica. Las principales son:

- **Arquitectura distribuida:** ROS 2 fue diseñado desde cero para admitir una arquitectura distribuida, lo que facilita la coordinación entre múltiples robots o entre múltiples unidades de procesamiento dentro de una misma plataforma. Esto también implica el soporte a sistemas de gran escala.

- **Soporte para Diversos Sistemas Operativos:** A diferencia de ROS 1, ROS 2 es compatible con una variedad de sistemas operativos, incluyendo Windows, macOS y varias distribuciones de Linux.
- **Mejor Gestión de Recursos:** Incluye una arquitectura más robusta para la administración eficiente de recursos y ciclos de vida de nodos y componentes.
- **Comunicación Robusta:** ROS 2 está basado en *Data Distribution System* (DDS), el cual es un estándar de comunicación abierto utilizado en numerosos sectores de la industria, y facilita la comunicación en entornos de red inestables, mayor seguridad y soporte para aplicaciones de tiempo real.
- **Soporte para Diferentes Lenguajes de Programación:** ROS 2 admite varios lenguajes de programación, incluyendo Python 3, C++, Java o Matlab.

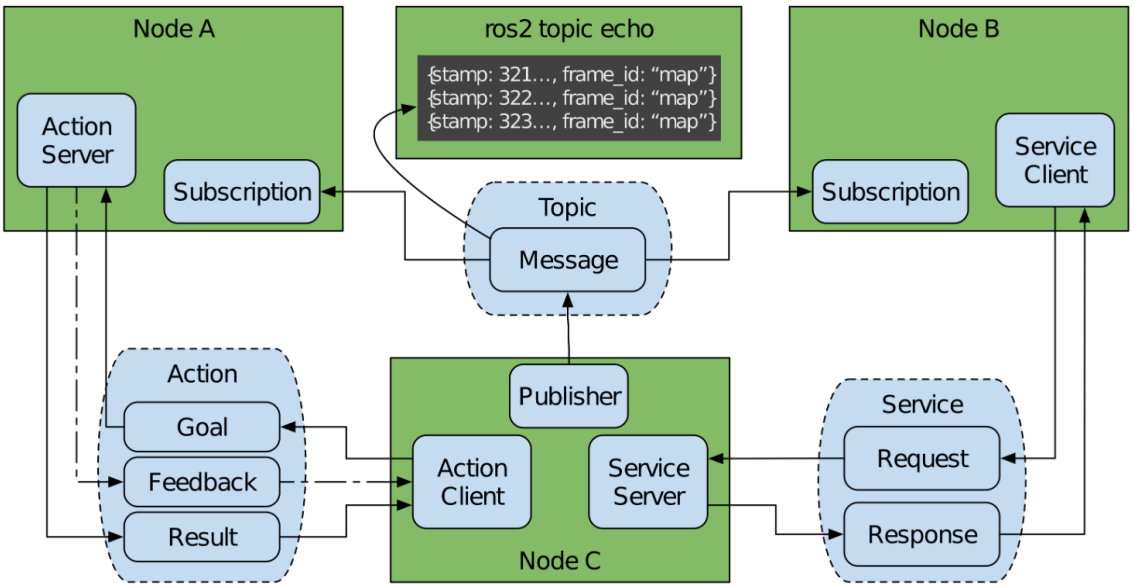


**Figura 2.5:** Patrones de comunicación en ROS 2.
Fuente: Robot Operating System 2: Design, Architecture, and Uses In The Wild [consultado el 6 Sep, 2023]

En ROS 2 las interfaces son componentes clave que permiten la comunicación y la interacción entre diferentes partes de un sistema robótico o entre múltiples nodos dentro de un sistema distribuido. Estos nodos son una forma de organizar las interfaces y funciones de un robot. Las interfaces en ROS 2 se utilizan para definir cómo los nodos pueden comunicarse entre sí de manera estándar y coherente. Las principales interfaces son:

- ***Topics*:** Utilizando el patrón PUB/SUB (Publicador/Suscriptor), permite la comunicación asíncrona entre nodos, donde un nodo puede publicar datos en un tema y otros nodos pueden suscribirse para recibir esos datos. Como se puede ver en la figura 2.5,
- **Servicios:** Facilita la comunicación síncrona entre nodos, donde un nodo proporciona un servicio con una petición y una respuesta, y otros nodos pueden enviar peticiones y esperar respuestas. Cuando se envía una petición a un servicio, el cliente queda bloqueado hasta recibir una respuesta por parte del servidor.

- **Acciones:** Similar a las interfaces de servicio, pero permite la comunicación asíncrona para tareas que pueden tomar un tiempo significativo. Los clientes pueden monitorizar el progreso y recibir resultados intermedios (*Feedback*). Viendo la figura 2.5 podemos observar que los resultados intermedios se envían a través del campo *Result*, mientras que la petición del cliente pasa a través del campo *Goal*.

ROS 2 también proporciona un servicio de parámetros, utilizados para configurar y ajustar los nodos y componentes del robot en tiempo de ejecución. Los nodos pueden leer y escribir parámetros compartidos y cambiar su comportamiento de manera dinámica.

Cabe aclarar que ROS 2, al igual que ROS 1, va evolucionando mediante el despliegue de distribuciones, cada distribución tiene un conjunto específico de características, mejoras, correcciones de errores y paquetes que se han incorporado desde la distribución anterior. La comunidad pone a disposición las distribuciones con mantenimiento a largo plazo (LTS), con cinco años de actualizaciones y corrección de errores, y las distribuciones *bleeding edge*, con solo un año de actualizaciones y que cada año se publica una nueva. Actualmente ROS 2 Humble es la distribución LTS, con soporte hasta Mayo de 2027.

### 2.4.2. TurtleBot 4

El TurtleBot 4 Standard (ver Fig. 2.6) representa la última iteración de una de las plataformas de robótica de código abierto más destacadas en el ámbito de la educación e investigación en robótica [22]. Esta plataforma ha evolucionado para ofrecer un conjunto de mejoras sustanciales que comprenden mayor capacidad de procesamiento, sensores de mayor calidad y una experiencia de usuario más refinada, todo ello manteniendo una accesibilidad económica.

**Figura 2.6:** TurtleBot 4 Standard (izquierda) y TurtleBot 4 Lite (derecha).
Fuente: https://clearpathrobotics.com/turtlebot-4/ [consultado el 6 Sep, 2023]

Entre las características destacadas del TurtleBot 4 Standard se encuentran:

1. **Configuración**: Este modelo se distribuye ensamblado y configurado con el entorno ROS 2 Humble preinstalado. Además, se proporciona documentación online y un modelo de simulación en Ignition Gazebo.

2. **Sensores**: Está equipado con: una cámara estéreo AI espacial OAK-D, un LiDAR 2D, una IMU, un sensor óptico de seguimiento del piso, codificadores de ruedas, un sensor infrarrojos y sensores de acantilados y golpes. También tiene la capacidad de detectar deslizamiento de las ruedas haciendo uso de estos sensores. Los sensores se integran como *topics* de ROS accesibles a través de la *API*.
3. **Movilidad**: Este robot se basa en la plataforma de robot educativo iRobot® Create 3 [5], que ofrece una base móvil con sensores inteligentes para la localización y posicionamiento. Además, presenta una capacidad de carga útil de 9 kilogramos, y alcanza una velocidad máxima de 0.306 m/s, todo esto con batería integrada para alimentar la plataforma y los periféricos externos.
4. **Modularidad**: Otra característica es la capacidad de expansión del TurtleBot 4. Los puertos USB accesibles y una placa superior facilitan la integración de hardware adicional.
5. **Código abierto**: Todo el software desarrollado por Clearpath Robotics es de código abierto, lo que facilita la documentación, la expansión y la colaboración con la comunidad *Open Source*. Sin embargo el *software* de la plataforma Create 3 no es de código abierto, por lo que dificulta la integración entre las dos partes y obstaculiza enormemente el proceso de desarrollo con este robot.

Con esta plataforma es posible ejecutar algoritmos SLAM utilizando el LIDAR 2D, así como de SLAM visual con cámara estereoscópica, cámara de profundidad o monocular. Todo esto haciendo uso de una precisa odometría, que integra un detector de deslizamiento de ruedas, la IMU y los *encoders* de los dos motores. El robot incluye un mando inalámbrico para manejarlo a distancia vía bluetooth, y sus controles con configurables mediante un archivo de configuración.

### 2.4.3. Simulador Ignition Gazebo

El uso de simuladores en robótica es esencial porque facilitan el desarrollo, prueba y depuración de algoritmos y controladores robóticos sin riesgo de dañar hardware costoso o causar accidentes. También permiten la creación de entornos virtuales variados y desafiantes para probar la capacidad de los robots en diversas situaciones.

Los simuladores ahorran tiempo y recursos al evitar la necesidad de configurar y operar robots físicos en todas las etapas de desarrollo. Y posibilitan la realización de experimentos repetibles y controlados para analizar el rendimiento de los robots en diferentes escenarios. Facilitan la investigación y la innovación al proporcionar un ambiente de trabajo accesible y versátil para la comunidad.

Ignition Gazebo es una plataforma de simulación de robots de código abierto que se integra con ROS. Ignition Gazebo proporciona un entorno de simulación 3D en el que los usuarios pueden modelar y simular robots, sensores y entornos virtuales. Esta simulación permite probar y depurar algoritmos de control, planificación de movimiento y percepción en un ambiente seguro y controlado. También incorpora un motor de física avanzado que simula con precisión la dinámica del mundo real. Esto incluye la simulación de la física de los objetos, la gravedad y la fricción, lo que permite una simulación realista de los movimientos y comportamientos de los robots.

Gazebo permite la simulación de una amplia gama de sensores, como cámaras, LIDAR, IMUs y más. Los usuarios pueden configurar estos sensores y simular cómo funcionarían en el entorno deseado. Y ofrece una interfaz gráfica fácil de usar que permite a los usuarios interactuar con la simulación, visualizar los datos del robot y ajustar los parámetros en tiempo real.


**Figura 2.7:** Captura de pantalla del entorno virtual del TurtleBot 4 simulado en Gazebo .

La empresa *Clearpath Robotics* ofrece una simulación del TurtleBot 4 (ver Fig. 2.7) , pero esta no es funcional y está llena de errores, como se puede ver en la sección *Issues* de su repositorio oficial [6]. Por lo que no han sido posibles el testeo y pruebas iniciales en este simulador. Para dar constancia de estos errores se escribió en uno de los *issues* titulado *Support for ROS 2 Humble?* y que parece que no ha sido solucionado todavía [7].

## 2.5 Crítica al estado del arte

Pese a que el campo de la robótica está sufriendo una revolución con grandes avances por múltiples frentes tecnológicos (la visión artificial, el control inteligente, la navegación autónoma o la asistencia sanitaria) todavía existe una falta de estandarización en el sector. Y es que, pese a que la primera distribución estable de ROS 2 se publicó en 2017, seis años después todavía no se han migrado muchas de las librerías de ROS 1, debido a que muchos de los desarrolladores que contribuyen código a ROS son empresas, las cuales no adoptan cambios tan a la ligera.

Esto se puede ver en la disponibilidad de muchos algoritmos SLAM en ROS 2. Durante el desarrollo del trabajo se han encontrado numerosas barreras de compatibilidad, ya que muchos de los algoritmos existentes en ROS 1 no habían sido migrados a ROS 2, bien por la falta de interés o por la falta de incentivos por parte de la comunidad. Como TinyS-

[6] Repositorio disponible en el enlace: https://github.com/turtlebot/turtlebot4_simulator/issues
[7] Issue disponible en este enlace: https://github.com/turtlebot/turtlebot4_simulator/issues/32

LAM, un algoritmo sencillo utilizado para aprender los conceptos básicos del SLAM, no es compatible con ROS 2 [8].

Para agravar el problema, muchos algoritmos que fueron migrados o desarrollados en ROS 2 han dejado de mantenerse, por lo que no es posible compilarlos en las últimas distribuciones y sufren errores de ejecución constantemente. Un ejemplo de esto es ORB-SLAM2, un algoritmo de SLAM visual que fue actualizado en 2021 por última vez y que tiene 51 *issues* abiertos en GitHub [9]. También ORB-SLAM3, cuyo repositorio ha sido abandonado y sufre de múltiples errores de ejecución y de problemas de compatibilidad con ROS 2 Humble.

TurtleBot 4 Standard, está construido sobre un Create 3, un robot que utiliza ROS 2 pero que no es de código abierto. Esto es algo que dificulta enormemente el desarrollo y ejecución de aplicaciones, ya que la depuración y configuración del robot se ven limitadas a los mensajes de *log* que ofrece el robot en su portal web. Un ejemplo de esto es que utilizando una configuración con FastDDS, una implementación de DDS, provoca que la CPU del Create 3 llegue al 100 % de utilización aun estando en reposo [10]. Debido a esto, se producen constantes errores de comunicación y sincronización entre el Create 3 y la Raspberry Pi del TurtleBot 4, lo que impide la ejecución de algoritmos de navegación y provoca fallos en algunos algoritmos SLAM.

A parte de esto, el soporte proporcionado por los desarrolladores del TurtleBot 4 no se puede considerar aceptable, ya que a fecha de la realización del trabajo existen 67 *issues* reportados sin ser tratados [11].

[8] El repositorio de ROS 1 dejó de mantenerse en 2017: https://github.com/OSLL/tiny-slam-ros-cpp

[9] A fecha 6 de Sep. de 2023, consultado en: https://github.com/appliedAI-Initiative/orb_slam_2_ros/tree/ros2

[10] Se pueden consultar los errores activos del Create 3 en: https://github.com/iRobotEducation/create3_docs/issues

[11] A fecha de 6 de Sep. de 2023, se puede comprobar en: https://github.com/turtlebot/turtlebot4/issues

# CAPÍTULO 3
# Desarrollo

## 3.1 Repositorio del proyecto

Para el desarrollo del presente Trabajo de Fin de Máster se ha creado un repositorio público (ver Fig. 3.1) donde se pueden encontrar la mayoría de recursos, como archivos *bag*, los mapas creados, algunos archivos de configuración del robot, la guía de instalación del robot, análisis superficial de los resultados de la evaluación de los algoritmos e incluso vídeos de la ejecución del SLAM.

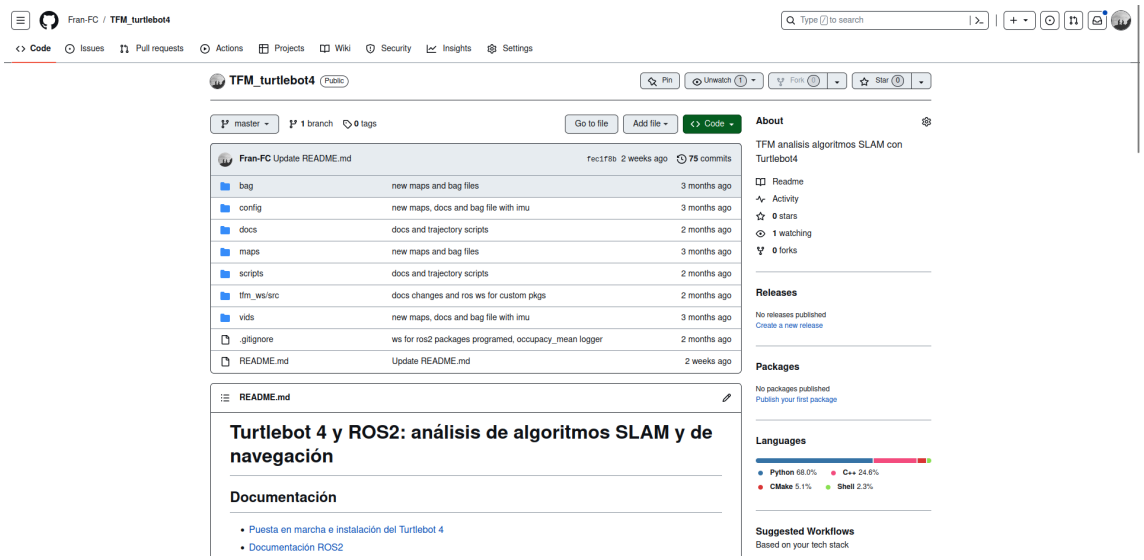


**Figura 3.1:** Captura de pantalla de la vista principal del repositorio del TFM en GitHub.

## 3.2 Instalación y puesta en marcha

Para la instalación del robot se ha seguido la guía oficial [1] [2]. Primero se describirán los componentes hardware y la interconexión entre si.

[1] Disponible en: https://turtlebot.github.io/turtlebot4-user-manual/overview/

[2] Se ha escrito una guía de instalación online en el repositorio del TFM: https://github.com/Fran-FC/TFM_turtlebot4

Se dispone de un PC con Ubuntu 22.04LTS instalado, este PC se encargará de ejecutar los algoritmos SLAM, ya que tiene mayor capacidad de cómputo y aliviará la carga computacional a la Raspberry Pi (RP) del TurtleBot 4. El PC también actuará como punto de acceso inalámbrico y como rúter para la RP, por lo que se dispondrá de un adaptador WiFi TP-Link Archer T3U Plus, el cual será el punto de acceso. La RP será el front-end del TurtleBot 4, y ofrecerá al PC todos los topics del sistema, incluyendo los del Create 3. El Create 3 se comunicará con la RP mediante una interfaz USB-C (ver Fig. 3.2).

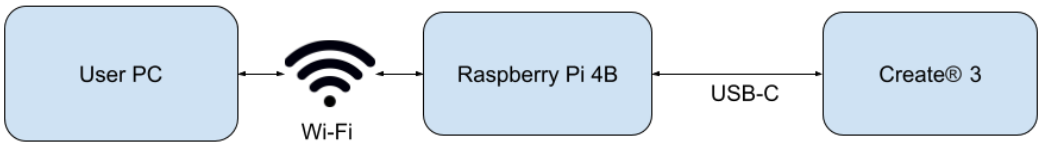


**Figura 3.2:** Comunicación tipo Discovery-Server entre los componentes.
Fuente https://turtlebot.github.io/turtlebot4-user-manual/setup/networking.html [consultado el 6 Sep, 2023].

### 3.2.1. Instalación del punto de acceso WiFi

A continuación se ejecutan una serie de comandos para instalar los paquetes necesarios para la instalación de los drivers del adaptador WiFi, se clona el repositorio de los drivers, se compila el código fuente y se instala y se carga el módulo del kernel recién instalado para que el sistema lo reconozca y lo utilice:

```
sudo apt update
sudo apt install git linux-headers-generic dkms dwarves
cp /sys/kernel/btf/vmlinux /usr/lib/modules/‘uname -r‘/build/
git clone https://github.com/RinCat/RTL88x2BU-Linux-Driver
cd RTL88x2BU-Linux-Driver
make
sudo make install
sudo modprobe 88x2bu
sudo reboot
```

Una vez instalados los drivers ha de activarse el modo punto de acceso en el PC, especificando el uso de banda 5GHz a través de un canal soportado, en este caso el 44. Se añade una regla *ip tables* para habilitar el tráfico en modo rúter NAT teniendo en cuenta la ip del adaptador y se guarda la regla de forma permanente. A continuación se muestra la sucesión de comandos:

```
nmcli d wifi hotspot ifname wlx30de4be0df50 ssid turtlebot4PC band a
    password 12345678 channel 44
sudo iptables -t nat -s 10.42.0.0/24 -A POSTROUTING -j MASQUERADE
sudo su
apt install iptables-persistent
iptables-save > /etc/iptables/rules.v4
```

A continuación se ha de configurar la Raspberry para conectarla al punto de acceso. Primero se conecta el PC a la Raspberry por medio de un cable Ethernet y utilizando una

sesión SSH [3]. Luego se ejecuta el comando `turtlebotsetup` y se introducen los ajustes como en la figura 3.3.

Tras guardar la configuración se desconecta el cable Ethernet y la RP se conecta automáticamente al punto de acceso.

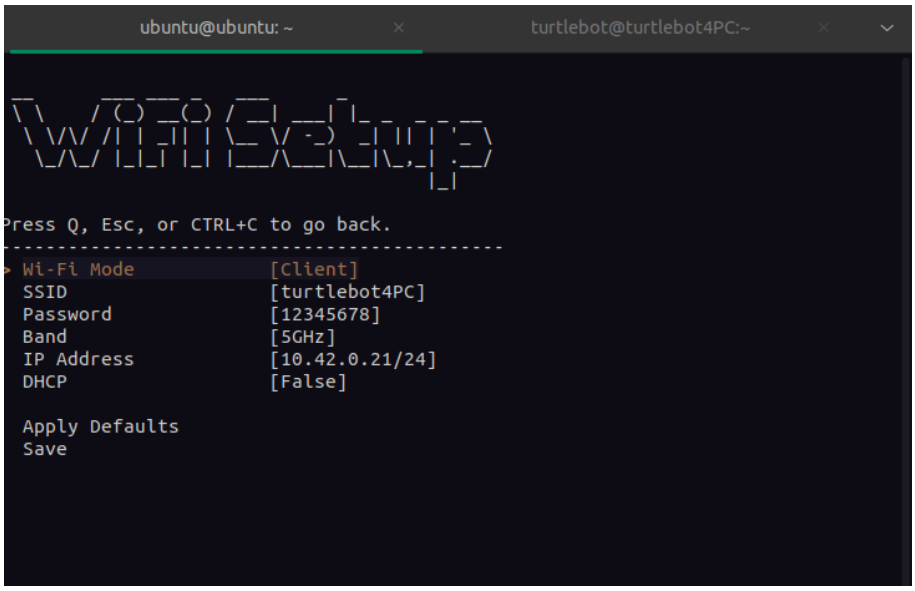


**Figura 3.3:** Configuración del WiFi de la Raspberry Pi del TurtleBot 4.

### 3.2.2. Configuración del DDS

En el siguiente paso es necesario elegir una de las dos configuraciones de *discovery* para la comunicación DDS entre los tres principales componentes (RP, PC y Create 3). El término *discovery* en el protocolo DDS se refiere al proceso de encontrar automáticamente a los participantes de una red dentro del mismo dominio. Existen dos fases principales en el proceso de *discovery* [23]:

1. **Fase de Descubrimiento de Participantes (PDP)**: Durante esta fase, los *DomainParticipants* (Participantes del Dominio) descubren de la existencia de los demás. Para lograrlo, cada *DomainParticipant* envía mensajes de anuncio periódicos que especifican, entre otras cosas, las direcciones *unicast* (IP y puerto) en las que el *DomainParticipant* está escuchando para recibir tráfico de meta-datos y datos de usuario entrantes. Dos *DomainParticipants* se consideran coincidentes cuando existen en el mismo Dominio DDS. Por defecto, los mensajes de anuncio se envían utilizando direcciones *multicast* conocidas y puertos (calculados usando el *DomainId*). Además, es posible especificar una lista de direcciones para enviar anuncios utilizando *unicast*.

2. **Fase de Descubrimiento de Extremos (EDP):** Durante esta fase, los *DataWriters* o escritores y los *DataReaders* o lectores se dan cuenta de la existencia mutua. Para lograrlo, los *DomainParticipants* comparten información sobre sus *DataWriters* y *DataReaders* entre sí, utilizando los canales de comunicación establecidos durante la

[3]Este proceso se describe más en detalle en el enlace: https://turtlebot.github.io/turtlebot4-user-manual/setup/basic.html#recovering-the-raspberry-pi

fase de PDP. Esta información contiene, entre otras cosas, el topic y el tipo de datos. Para que dos extremos coincidan, su topic y tipo de datos deben coincidir. Una vez que un *DataWriter* y un *DataReader* coinciden, están listos para enviar/recibir tráfico de datos de usuario.

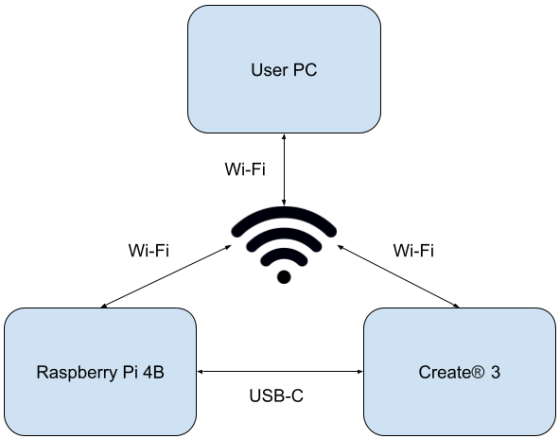


**Figura 3.4:** Topología del protocolo Simple-Discovery.
Fuente: https://turtlebot.github.io/turtlebot4-user-manual/setup/networking.html [consultado el 6 Sep, 2023].

Una vez conocido el funcionamiento del proceso de *discovery*, TurtleBot 4 ofrece dos configuraciones o implementaciones diferentes:

1. La primera opción es la llamada Discovery-Server (ver Fig. 3.2), donde la RP actúa como el componente central y como puente para la comunicación entre el PC y el Create 3. La ventaja principal es que es más escalable, con un número mayor de componentes conectados la red se ve menos saturada (ver Fig. 3.3). La principal desventaja es que se requiere mayor configuración en el proceso de instalación, además, hay que añadir una regla para direccionar el tráfico entrante y saliente del Create 3.

2. La otra opción es Simple-Discovery (ver Fig. 3.4), este usa multicasting para que un componente pueda distribuir la información al resto. Es de fácil configuración, todos los componentes se conectan a la misma red y pueden empezar a funcionar, aunque la mayor desventaja es que no es escalable a muchos componentes. además, como se verá más a delante, esta opción es totalmente incompatible con la configuración de hardware disponible para el proyecto.

Por lo tanto, una vez analizadas las dos configuraciones, se llega a la conclusión de que solo es posible utilizar la configuración de Discovery-Server. Debido a las siguientes razones:

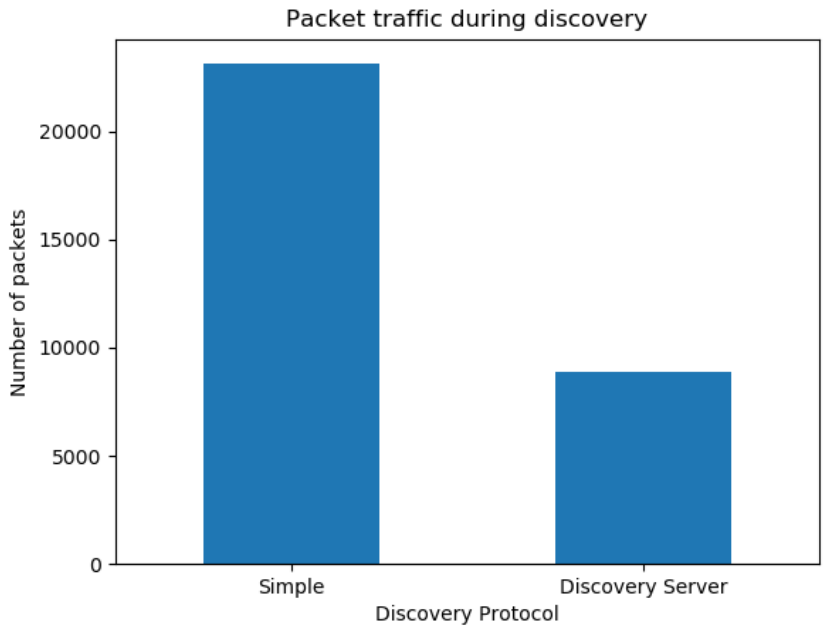


**Figura 3.5:** Uso de red de los protocolos Discovery-Server y Simple-Discovery. Fuente:https://fast-dds.docs.eprosima.com/en/latest/fastdds/ros2/discovery_server/ros2_discovery_server.html#compare-discovery-server-with-simple-discovery. [Consultado el 6 Sep, 2023].

1. El Create 3 solo se puede conectar a redes WiFi 2.4GHz, mientras que la Raspberry, debido a un error de software en la imagen del sistema operativo distribuida por Clearpath Robotics, solo puede conectarse a redes 5GHz.
2. El driver del adaptador de red WiFi TP-Link Archer T3U Plus soporta emitir en una única frecuencia, 5GHz o 2.4GHz.

Debido a estas dos razones, no es posible utilizar la configuración Simple-Discovery, ya que para ello es necesario que tanto la Raspberry como el Create 3 estén conectados a la misma red. Dicho esto, se sigue el manual de instalación del TurtleBot 4 para configurar el Discovery-Server [4].

### 3.2.3. Configuración del NTP

ROS 2 requiere que todos los componentes que se estén comunicando tengan los relojes sincronizados. Esto es debido a que cada mensaje que se envía lleva una marca de tiempo, y esta se utiliza en ciertos casos para sincronizar el árbol de transformadas o árbol de tf. El árbol de tf es un diagrama donde se muestra la relación de transformaciones espaciales entre las articulaciones o componentes de un robot, o de su entorno, para saber en todo momento la posición relativa de cada articulación o *joint* respecto a la articulación padre. Por ejemplo, en un brazo, el hombro es padre del codo, y la posición (traslación y rotación) del codo se expresa relativa al hombro. El árbol de tf también se utiliza para saber la posición del propio robot respecto al mapa, esto es especialmente útil en SLAM.

[4]Siguiendo los pasos descritos en este enlace: https://turtlebot.github.io/turtlebot4-user-manual/setup/discovery_server.html

Por lo tanto, para que los relojes de todos los componentes estén sincronizados uno deberá actuar como servidor NTP, en este caso el PC, ya que también actúa como rúter. Se utilizará la herramienta *chrony* para este fin, editando en el PC el archivo `/etc/chrony/chrony.conf` y añadiendo lo siguiente:

```
# cnfig ntp sync with turtlebot
server 10.42.0.21 presend 0 minpoll 0 maxpoll 0 iburst  prefer trust
# enable serving time to ntp clients on 10.42.0.0 subnet
allow 10.42.0.0/24
# serve time even offline
local stratum 10
```

Para aceptar peticiones NTP desde el PC hay que abrir el puerto 123 en el protocolo UDP con el siguiente comando:

```
sudo ufw allow from any to any port 123 proto udp
```

También habrá que editar el mismo archivo en la Raspberry y añadir la siguiente linea:

```
server 10.42.0.1 iburst prefer
```

Antes de proseguir se debe comprobar con el comando `date` que se muestra la misma fecha y hora que en el PC.

Adicionalmente se ha configurado la Raspberry Pi para que utilice el PC como servidor DNS, y así disponer de conexión a internet. Esto es recomendable para actualizar todos los paquetes del TurtleBot 4 antes de proceder con el proyecto. Las instrucciones se encuentran en el manual de instalación, en el repositorio de GitHub del Trabajo de Fin de Máster [5]. También es recomendable generar una imagen del sistema en la Raspberry Pi a modo de copia de seguridad una vez terminada toda la configuración.

## 3.3 Ejecución de SLAM

### 3.3.1. Navegación

Una vez configurado y en funcionamiento, ya es posible conducir el Turtlebot 4 mediante un controlador inalámbrico bluetooth. Como se ha mencionado en la Metodología, primero se ideará una ruta por la que se manejará el robot mientras registra los datos de sus sensores para posteriormente ejecutar los diferentes algoritmos SLAM.

Se pretende crear un mapa de la segunda planta del edificio 5C [6]. Como el objetivo no es crear un mapa completo cubriendo todas las salas (algunas de ellas despachos privados), se ha ideado un recorrido (ver Fig. 3.6) para crear un mapa parcial de la planta donde haya un cierre de bucle (pasar dos veces por el mismo punto) y así tener un mapa lo suficientemente grande y complejo como para ser evaluado.

Las primeras pruebas comportan la creación del mapa en tiempo real utilizando el paquete *slam_toolbox*, incorporado por defecto en el TurtleBot 4. Estas pruebas tendrán el objetivo de elegir la mejor configuración posible para la posterior ejecución y evaluación

[5] Disponible en el siguiente enlace: https://github.com/Fran-FC/TFM_turtlebot4/tree/master/docs
[6] Se puede consultar la vista de la planta en este enlace: https://openmaps.upv.es/?locate=V.5C.2.040

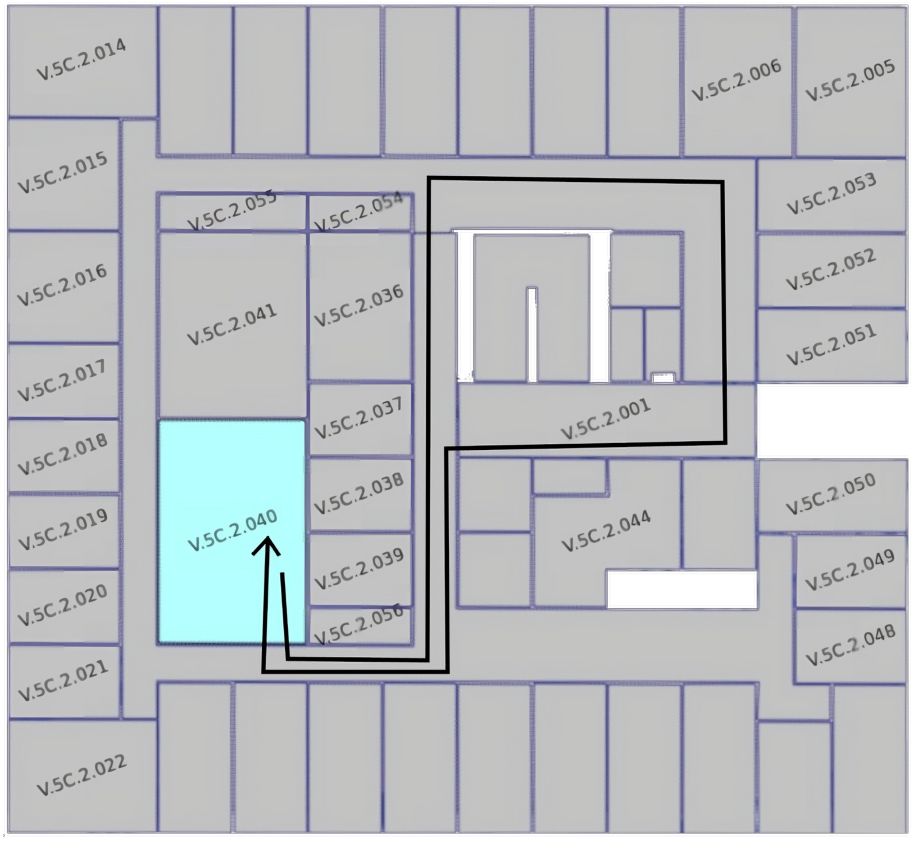


**Figura 3.6:** Recorrido del TurtleBot 4 para generar el mapa parcial de la planta 2 del edificio 5C. La flecha negra marca el sentido.

de todos los algoritmos SLAM disponibles. Existen cuatro escenarios o configuraciones disponibles:

1. **Escenario 1:** El PC ejecuta el SLAM en tiempo real, la Raspberry Pi comunica las lecturas de los sensores al PC vía WiFi.

   **Inconveniente:** La señal WiFi Se degrada a medida que se aleja el robot, el algoritmo deja de ejecutarse si la señal WiFi desaparece al estar demasiado lejos.

2. **Escenario 2:** La Raspberry ejecuta el SLAM en tiempo real, el PC no ejerce ningún papel en esta configuración.

   **Inconveniente:** La Raspberry tiene menos capacidad de procesamiento y puede provocar fallos en la detección de cierres de bucle, provocando fallos en forma de zonas redundantes, inconsistencias y exceso de ruido.

3. **Escenario 3:** Se utiliza un portátil para que ejecute el SLAM en tiempo real, apoyado en la base del TurtleBot 4, aumentando la capacidad de cómputo sin perder información debido a la degradación de la señal WiFi.

   **Inconveniente:** Se añade una carga extra al robot y puede ser inestable si se conduce a mayor velocidad.

4. **Escenario 4:** Se graba la ruta en un archivo y posteriormente se ejecuta el SLAM en un PC o portátil mientras se reproduce el archivo previamente grabado.

**Inconveniente:** El archivo ocupa muchos GB si se almacenan datos de las cámaras y la ruta es muy larga.

Tras probar y valorar los escenarios posibles (ver los mapas generados en cada escenario en la figura 3.7), se decide optar por el número cuatro, ya que permitirá utilizar una única grabación de la ruta para todos los algoritmos, siendo así una comparación independiente a los pequeños cambios de cada conducción sobre la ruta definida, también permitirá ahorrar una cantidad considerable de tiempo. Sin embargo, la ruta se grabará utilizando el portátil, ya que permitirá visualizar el estado de la grabación y reiniciar esta misma en el caso de que fallara.

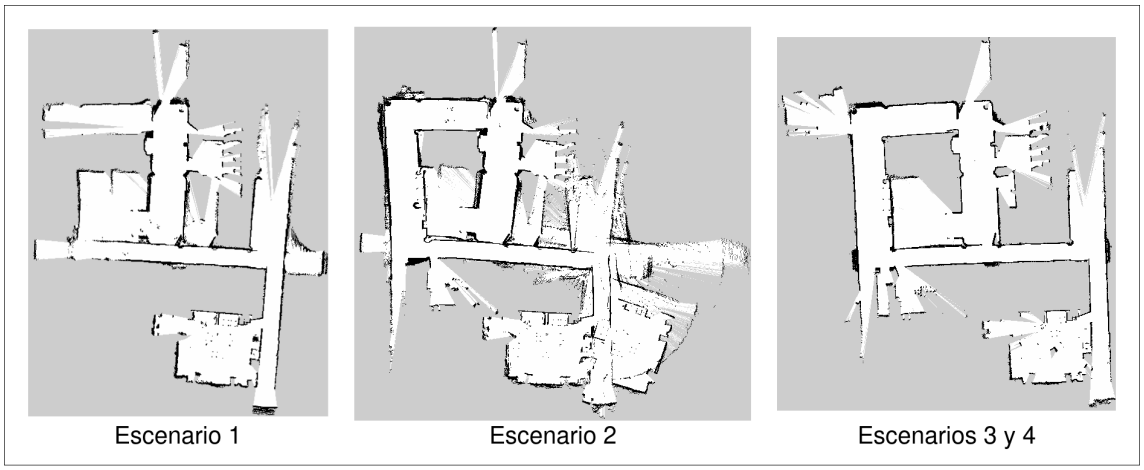


**Figura 3.7:** Comparación de los mapas según el escenario o configuración elegida. Los escenarios 3 y 4 llevan al mismo resultado, ya que son ejecutados por el portátil.

Una vez se ha ideado el recorrido para crear el mapa, se grabará en un archivo el itinerario del robot mientras se conduce para, posteriormente, reproducir ese fichero mientras se ejecuta cada algoritmo SLAM. De esta manera los datos de entrada siempre serán los mismos, independientemente del algoritmo ejecutado.

ROS 2 ofrece una funcionalidad para grabar los datos de todos los *topics* en tiempo real. La librería llamada *ros2bag* [7] tiene la capacidad de reproducir y analizar datos grabados, por lo que permite grabar datos del mundo real y luego reproducirlos en un entorno simulado, lo que facilita el desarrollo y la prueba de algoritmos y controladores en un entorno seguro.

Una vez obtenido el archivo *bag*, ya se pueden ejecutar los algoritmos SLAM disponibles.

### 3.3.2. SLAM 2D

Tal y como se menciona en la sección métodos SLAM del capítulo Estado del Arte, existen cuatro principales algoritmos disponibles en ROS 2. Se descargará el código fuente de cada algoritmo desde sus respectivos repositorios, se compilarán y se ejecutarán utilizando el archivo *bag* generado previamente.

El algoritmo ***slam_toolbox*** ya ha sido validado, por lo que el segundo algoritmo a instalar es ***Cartographer***, basado también en SLAM de grafos. Este paquete ha dejado de

[7] Los principios del diseño del *rosbag* pueden consultarse en: https://github.com/ros2/design/blob/ros2bags/articles/rosbags.md

mantenerse de manera oficial, por lo que es posible que se encuentren errores de ejecución. Para ejecutar el algoritmo se crea un nuevo archivo *launch* para lanzar todos los nodos necesarios:

1. **robot_state_publisher:** Esencial para la visualización del robot y para publicar las transformaciones espaciales del robot. Es necesario especificar el archivo *URDF* del TurtleBot. El URDF de un robot es un formato de archivo utilizado para describir la geometría, cinemática y otras propiedades de un robot en un formato XML legible por máquina.
2. **cartographer_ros:** El nodo principal que realiza el SLAM. Publica los sub-mapas.
3. **occupacy_grid:** Es el nodo que escucha los sub-mapas publicados por el nodo cartographer_ros y construye el mapa completo, detectando cierres de bucles y optimizando la matriz de ocupación.

Cartographer genera una matriz de ocupación, donde cada celda expresa la probabilidad entre 1 y 0 de que un espacio esté ocupado (1 = ocupado, 0 = libre). Si se desea conseguir una matriz de ocupación discreta, se puede calcular la media de los valores de todas las celdas y aplicar un límite para posteriormente cambiar el valor de la celda a 1 si su valor inicial es mayor que el límite o a 0 si es menor (ver Fig. 3.8). Para ello se programa un nodo de ROS 2 en Python 3 que se suscribe al *topic* del mapa para calcular y aplicar el *threshold* y guardar el mapa en un archivo.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$x_i = \begin{cases} 0 & \text{if } x_i \leq \mu\ , \\ 1 & \text{otherwise.} \end{cases} \quad , \forall x_i \in X$$

Donde $\mu$ es el límite calculado y $x_i$ el valor de cada celda de la matriz de ocupación $X$.

Por alguna razón, cartographer no logra fusionar correctamente los datos de la IMU con los del LIDAR 2D, por lo que se genera un mapa muy distorsionado. Para conseguir un resultado decente se deshabilita el uso de la IMU para estimar la posición del robot y así evitar este error. La figura 3.9 muestra el mapa distorsionado por el uso de la IMU, también se ha elegido un límite mas bajo a la hora de guardar el mapa.

El tercer algoritmo a instalar es ***HectorSLAM***, un algoritmo basado en filtro de partículas, que no realiza cierres de bucle y que es especialmente eficiente ejecutándose en tiempo real en espacios pequeños y con poco recorrido. Existe un repositorio para ROS 2 que dejó de ser mantenido por los creadores (la empresa RTT-ALeRT) pero que sigue siendo mantenido por desarrolladores independientes. El archivo *launch* no llega a ejecutarse debido a múltiples errores que no han sido corregidos, pero se puede ejecutar el nodo de mapeo directamente con el comando `ros2 run`. Debido al error de sincronización temporal del TurtleBot 4, muchas de las actualizaciones en la posición del robot se descartan, debido a que el programa cree que están desfasadas, aunque no sea así. Pese a estos errores se consigue crear un mapa, pero con una tasa de refresco muy baja y mostrando múltiples mensajes de error.

El cuarto y último algoritmo de SLAM 2D es ***gmapping***, un algoritmo muy eficiente basado en filtros que dispone de algunas funcionalidades avanzadas como el cierre de bucle.

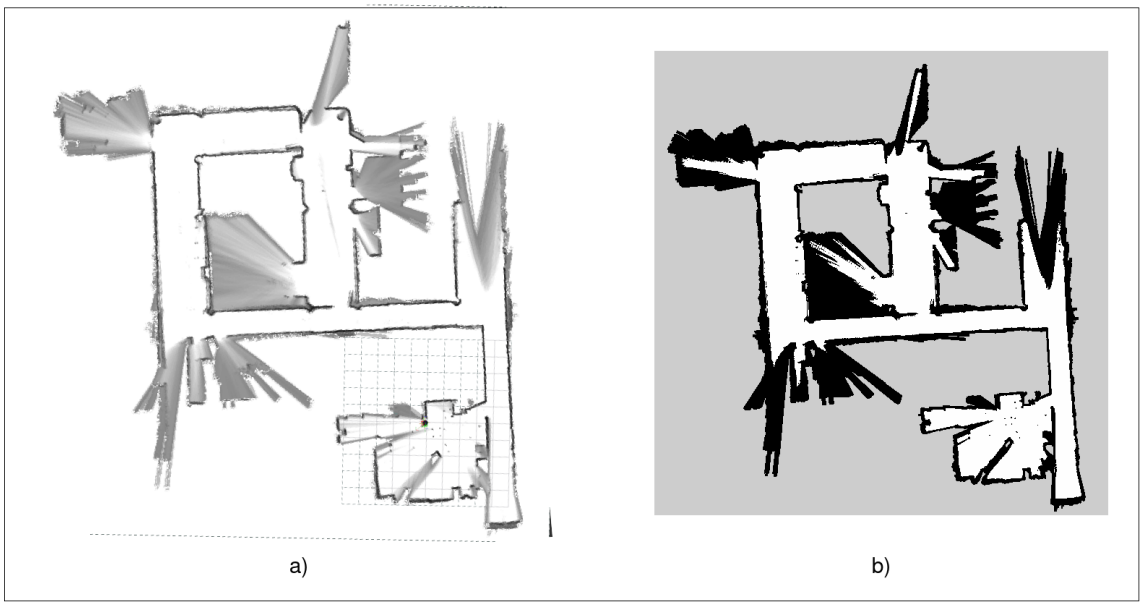


**Figura 3.8:** Resultado del algoritmo Cartographer: a) Matriz de ocupación expresada en probabilidades. b) Matriz de ocupación en valores discretos tras aplicar el límite.

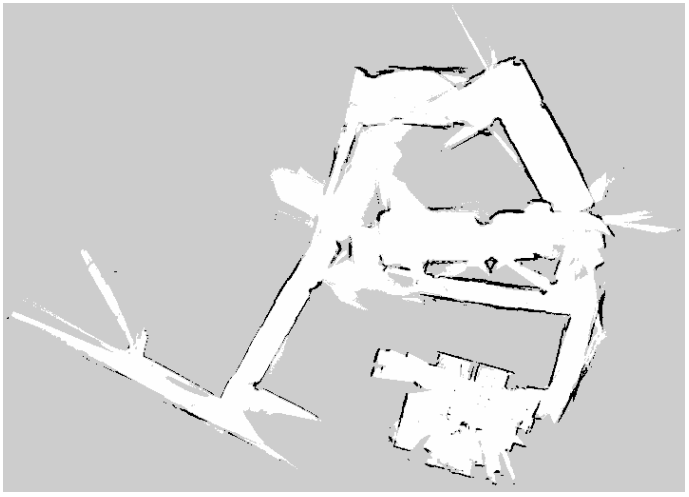

**Figura 3.9:** Mapa deformado por la mala fusión de la IMU con los demás sensores.

El repositorio ha dejado de mantenerse, y el último *commit* fue realizado en 2021 [8]. Pese a haber errores abiertos, nadie parece proporcionar soporte a los usuarios y desarrolladores.

Una vez ejecutados todos los algoritmos y generados los mapas se pueden observar las diferencias claras entre los resultados, así como las particularidades de cada uno de los mapas (ver Fig. 3.10).

[8] El repositorio se puede visitar en este enlace: https://github.com/Project-MANAS/slam_gmapping

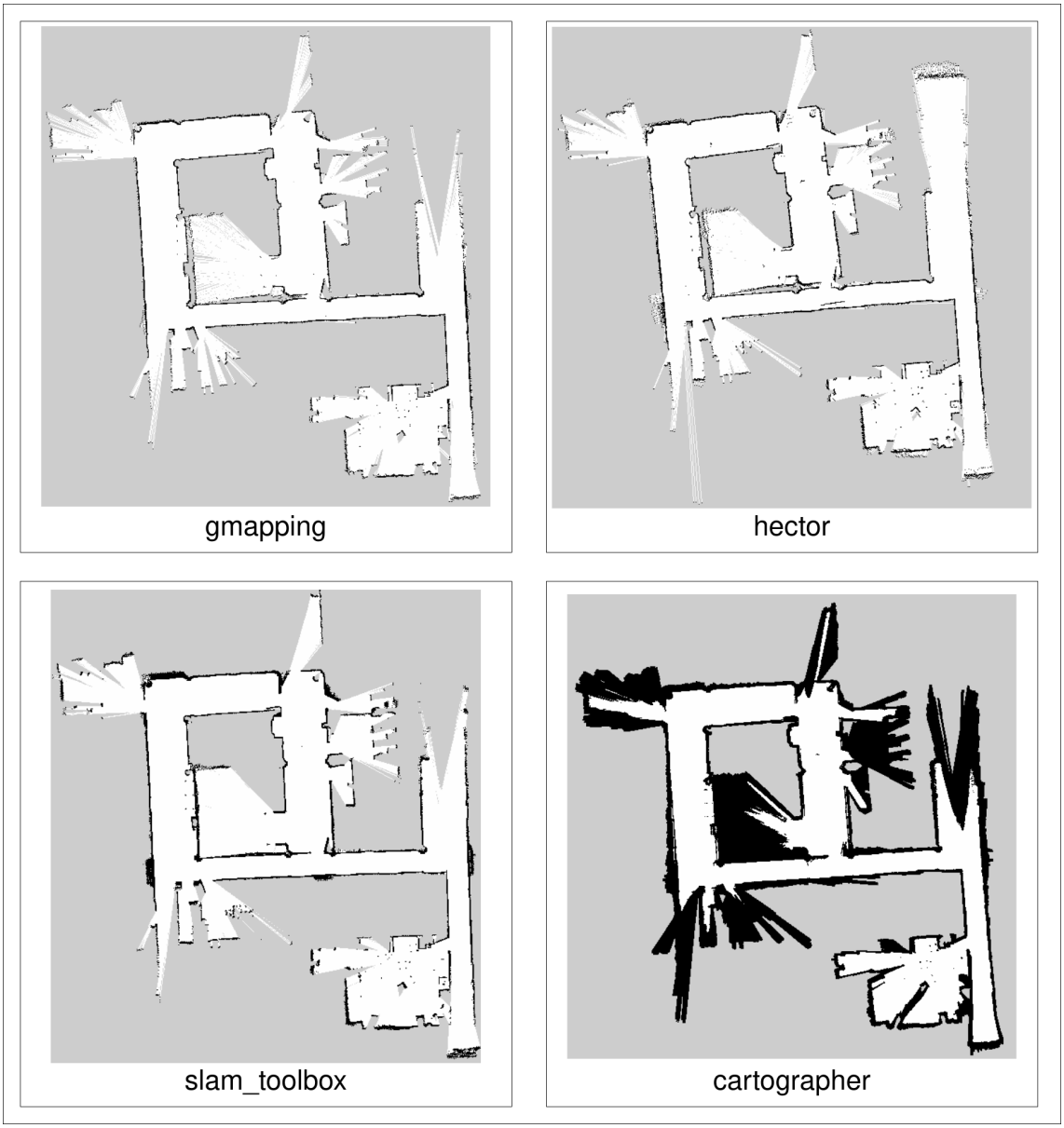


**Figura 3.10:** Mapas generados usando SLAM 2D.

### 3.3.3. SLAM Visual

Como se menciona en la sección Métodos SLAM del capítulo Estado del Arte, solo existe un algoritmo de SLAM Visual funcional para ROS 2, RTAB-Map. El resto de algoritmos como ORB-SLAM2 y ORB-SLAM3 no logran funcionar debido a numerosos errores de compilación y errores de ejecución por incompatibilidades con la distribución del sistema operativo Ubuntu 22.04LTS y la distribución Humble LTS de ROS 2.

RTAB-Map genera una matriz de ocupación al igual que los algoritmos de SLAM 2D, pero también una nube de puntos tridimensional (ver el proceso de mapeo en la figura 3.11). Como el objetivo de este trabajo es evaluar la calidad de los mapas y de la localización de los diferentes algoritmos SLAM, y el mapa 2D de RTAB-Map no tiene la suficiente

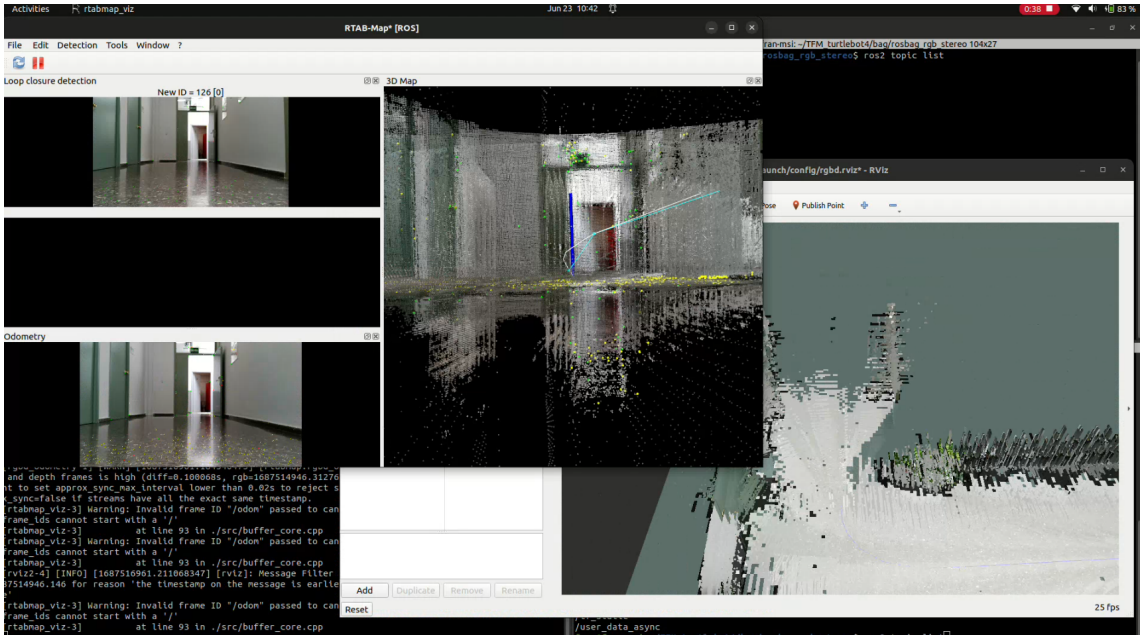


**Figura 3.11:** Captura de pantalla de todas las interfaces gráficas que muestra RTAB-Map.

calidad como para poder compararlo con el resto (ver Fig. 3.12), solamente se evaluará la precisión en la localización.

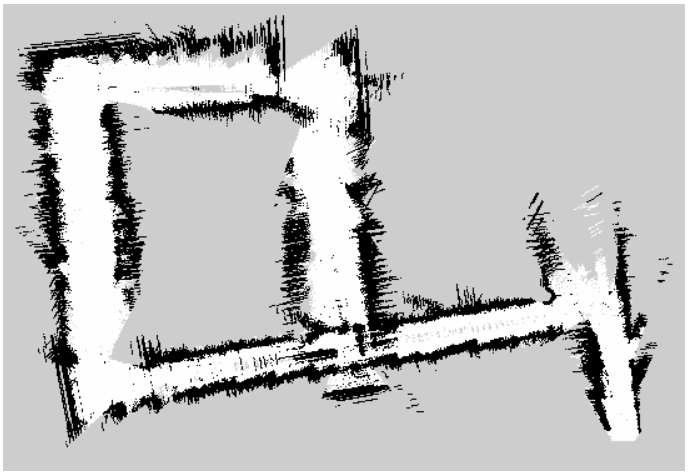

**Figura 3.12:** Matriz de ocupación generada por el algoritmo RTAB-Map utilizando los datos de la cámara RGBD.

## 3.4 Evaluación de mapas

### 3.4.1. Obtención de métricas

Para evaluar la calidad de un mapa se ha optado por utilizar métricas independientes a cualquier mapa de referencia o *ground truth*. Estas métricas mencionadas en la sección Métodos de Evaluación del capítulo Estado del Arte, se obtendrán mediante un *script* en Python 3.

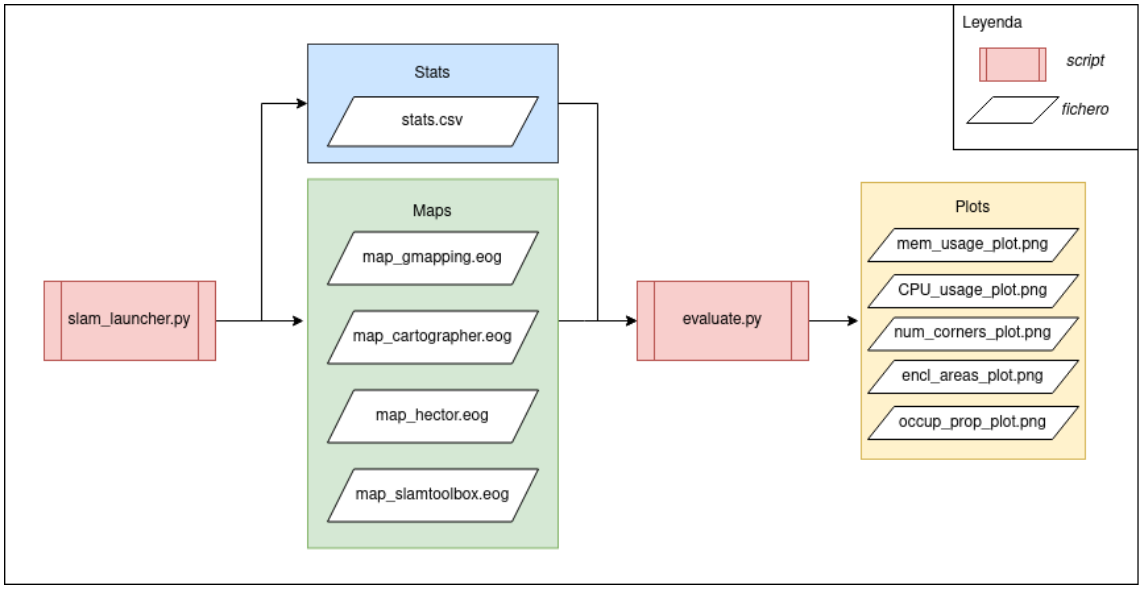


**Figura 3.13:** Proceso de evaluación de calidad de los mapas.

Dicho *script* ejecuta los algoritmos de SLAM 2D mientras reproduce a su vez el archivo *bag*. Mientras se está ejecutando el SLAM, el programa escribe en un archivo en formato CSV el porcentajes de uso de memoria y procesador. Cuando termina de ejecutarse un algoritmo, se guarda el mapa con el nombre del algoritmo, se reinicia el la reproducción del archivo *bag* y se lanza el siguiente algoritmo. Por lo tanto, el resultado final del *script* consiste en un archivo CSV con mediciones en el uso de memoria y procesador y un archivo conteniendo el mapa resultante para cada algoritmo.

Este archivo CSV y los mapas serán leídos por otro *script*, que analizará los resultados generará cinco gráficas en forma de imagen representando estos resultados (ver Fig. 3.13).

Para analizar las imágenes de los mapas se han utilizado principalmente las librerías *OpenCV* y *numpy*. A continuación se describen las tres principales métricas de calidad de mapas, así como su implementación en forma de funciones Python, todas reciben como argumento de entrada la imagen de tipo CVMat:

1. **Número de áreas encerradas:** son zonas de espacio libre rodeadas de espacio ocupado. Hay varias situaciones en las que la presencia de tales zonas indica un fallo. Por ejemplo, cuando una sala se escanea varias veces pero no se reconoce, de modo que el mapa final está compuesto por ligeras rotaciones de la sala superpuestas entre sí. Otra posibilidad es el fallo de cierre de bucle, cuando un algoritmo no reconoce que ha que ha vuelto al mismo lugar donde empezó. En ese caso solapamiento entre la primera y la última parte del mapa construido [20].

   La función `get_num_enclosed_areas` calcula el número de áreas cerradas en una imagen binaria. En primer lugar, se realiza una umbralización de la imagen de entrada, posteriormente se utiliza una función para etiquetar las regiones conectadas en la imagen binaria llamada **cv.connectedComponents**, uno de los valores de retorno es el total de estas.

2. **Número de esquinas:** se dice que cuantas más esquinas tenga un mapa, más probabilidades habrán de que sea inconsistente y contenga errores [20], como pasillos menos rectos y obstáculos representados de manera más irregular.

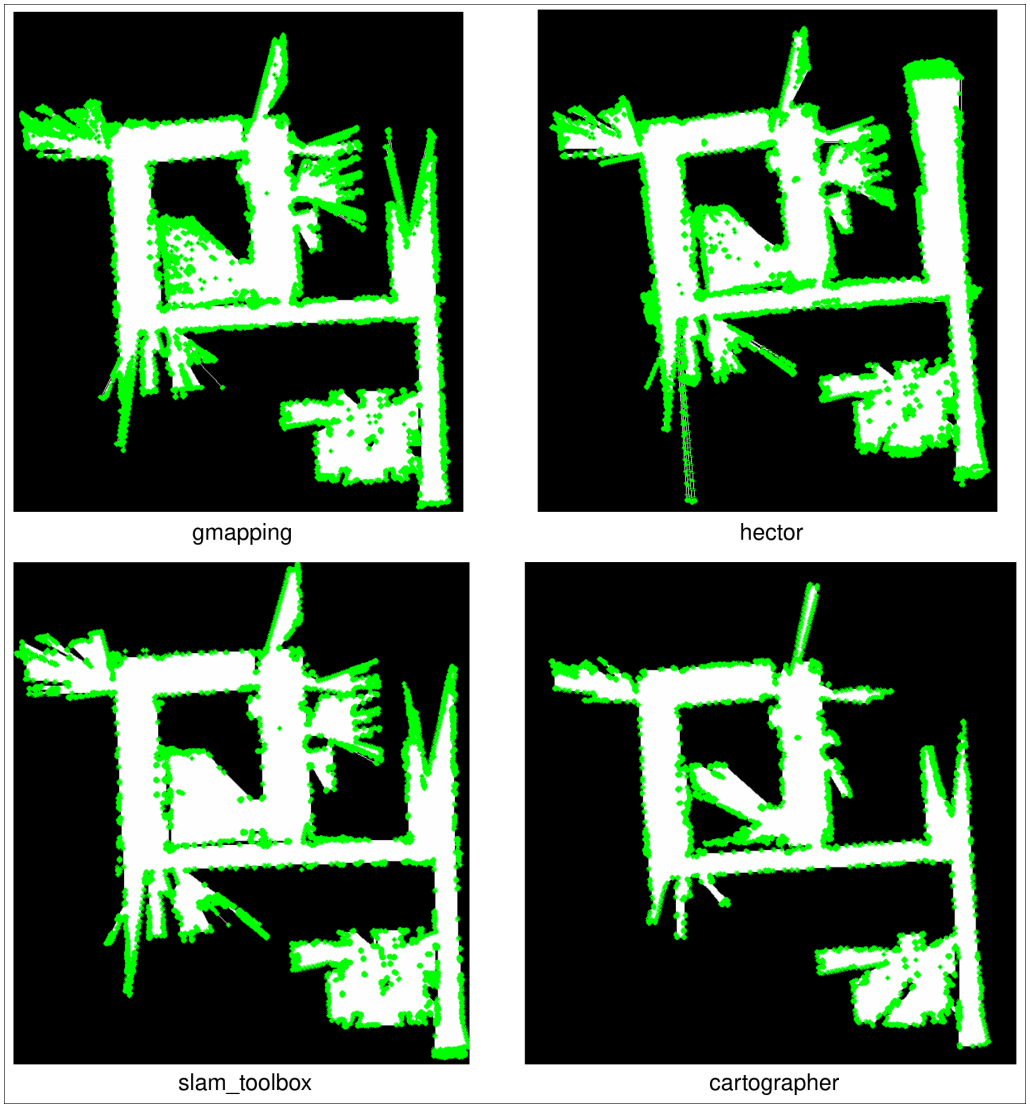


**Figura 3.14:** Esquinas marcadas en verde sobre los mapas binarizados.

La función *get_num_corners* realiza la detección de esquinas en una imagen utilizando el método de *Harris Corner Detection* [24]. Este método ya está implementado en la librería OpenCV, pero primero se realiza una umbralización de la imagen de entrada utilizando la función **cv.threshold** de OpenCV. Esta umbralización convierte la imagen en una imagen binaria, donde los píxeles que están por encima del valor de un umbral se establecen en 255 (blanco), y los píxeles que están por debajo del umbral se establecen en 0 (negro). La imagen binarizada se convierte a una matriz de *floats* y se pasa a la función **cv.cornerHarris** para obtener las esquinas marcadas en la matriz. Estas esquinas se pueden visualizar en forma de puntos verdes en la imagen (ver Fig. 3.14). El valor de retoTFM-Turnrno es el número total de estas.

3. **Proporción de celdas ocupadas:** representa la calidad del mapa: cuanto más grande sea esta proporción, peor será la calidad del mapa, representando zonas de incertidumbre que acaban etiquetándose como ocupadas y paredes u obstáculos duplicados, aumentando la cantidad de puntos ocupados [20].

La función *get_occupied_proportion* primero convierte la imagen de entrada en una matriz de tipo de datos de punto flotante. Esto se hace para asegurarse de que las divisiones posteriores produzcan resultados en punto flotante en lugar de truncar a enteros. Posteriormente se calcula la proporción de píxeles que tienen un valor de 0 dividiendo del número de píxeles con ese valor respecto al tamaño de la matriz.

A continuación se muestran las funciones descritas anteriormente, son un extracto del fichero `evaluate.py`:

```
def get_occupied_proportion(image):
    image = image.astype(float)

    occupied_proportion = np.sum(image == occupied_value) / image.size
    return occupied_proportion

def get_num_corners(image):
    _, binary_image = cv.threshold(image, unknown_value, free_value, cv.
        THRESH_BINARY)#remove unknown values
    image = np.float32(binary_image)
    # Apply corner detection
    dst = cv.cornerHarris(image, blockSize=3, ksize=3, k=0.04)

    # Threshold the corner response
    threshold = 0.01 * dst.max()
    corners = np.where(dst > threshold)

    return len(corners[0])

def get_num_enclosed_areas(image):
    _, binary_image = cv.threshold(image, unknown_value, free_value, cv.
        THRESH_BINARY)#remove unknown values
    num_labels, _ = cv.connectedComponents(binary_image)

    return num_labels - 1
```

### 3.4.2. Calidad de los mapas

Se pueden observar en la figura 3.15 los resultados de cada métrica.

En primer lugar se muestra el gráfico de barras naranja, que representa el **número de áreas encerradas**. Los resultados muestran que HectorSLAM tiene el mayor número de áreas encerradas, seguidos por gmapping y slam_toolbox, siendo cartographer el mapa con menor número. Dicho esto, es posible que cartographer muestre este número tan bajo debido a su mucho mayor número de píxeles ocupados. Por lo tanto, es seguro afirmar que HectorSLAM y gmapping son los algoritmos menos prometedores.

Los resultados mostrados por el número de esquinas en la gráfica de barras verde indican una situación similar a la anterior. HectorSLAM y gmapping presentan los peores datos, observando en los respectivos mapas pasillos más distorsionados y una mayor cantidad de errores. Cartographer muestra los mejores datos, seguido de slam_toolbox, ya que son los dos algoritmos basados en grafos y con mejor cierre de bucle.

Finalmente se muestra en el gráfico de barras azul la proporción de píxeles ocupados. En el presente gráfico se puede observar claramente que cartographer tiene la mayor

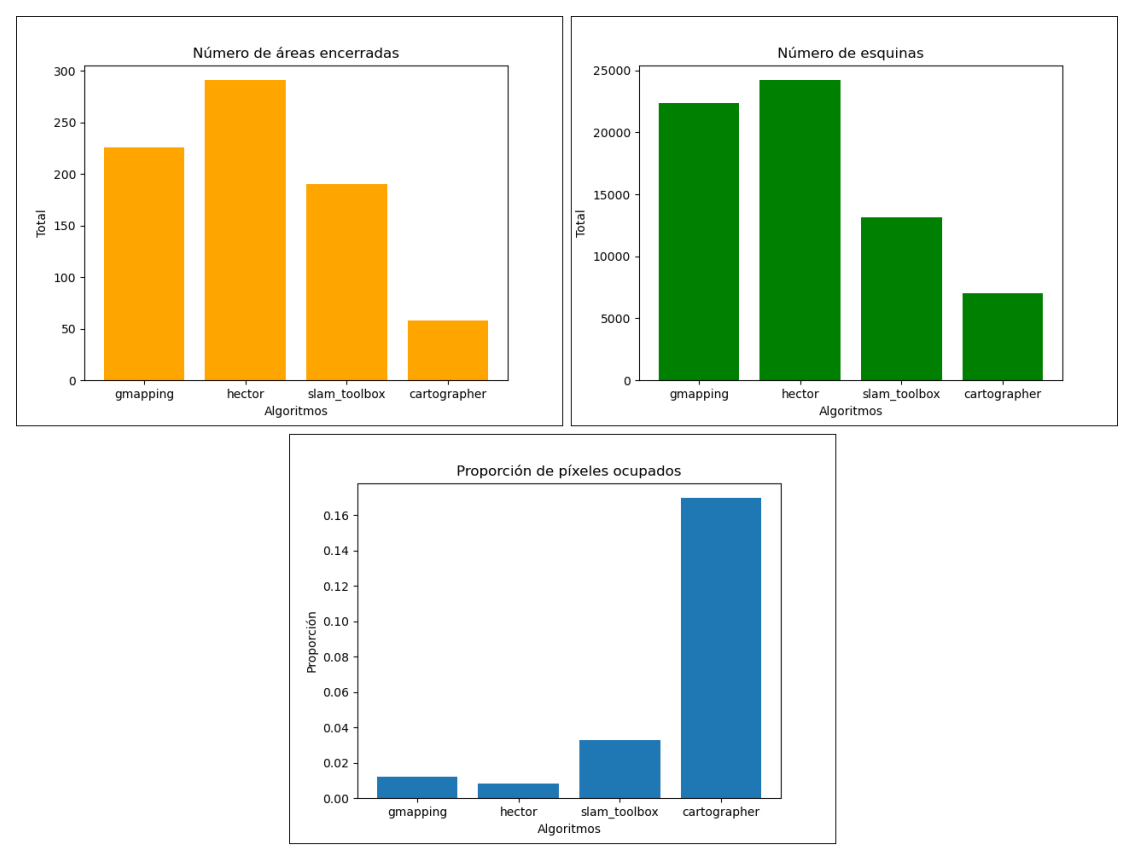


**Figura 3.15:** Resultado de la medición de las tres principales métricas de calidad en los mapas 2D.

proporción, debido a la forma que tiene de generar y guardar los mapas. HectorSLAM y gmapping generan los dos mapas con menor proporción de área ocupada.

Como reflexión general, es seguro afirmar que los dos peores algoritmos son gmapping y HectorSLAM, ya que gmapping no realiza a penas correcciones mediante cierres de bucle y HectorSLAM no detecta cierres de bucles. Esto provoca que el mapa se distorsione a medida que aumenta de tamaño, y se generen zonas repetidas, inconsistencias y deformaciones. Se podría argumentar que son algoritmos convenientes para mapear zonas pequeñas en tiempo real, ya que estos algoritmos ofrecen menos parámetros y su ejecución es bastante directa y simple, aunque esto último se ve entorpecido por la falta de soporte oficial del *software*, lo que puede provocar errores y falta de compatibilidad en distribuciones de ROS futuras.

### 3.4.3. Recursos computacionales

A parte de analizar las métricas de calidad, también es de gran interés para este estudio analizar el consumo de recursos computacionales de cada algoritmo, para así estimar los casos de uso de cada uno de ellos. En la figura 3.16 se observa el porcentaje de uso de memoria RAM (izquierda) y de CPU (derecha) durante la ejecución de cada algoritmo. Cada línea representa la media de consumo en un momento de tiempo, mientras que la zona sombreada que rodea cada línea representa la desviación estándar. Las gráficas se han obtenido ejecutando veinte veces el mismo algoritmo con el mismo archivo *bag*,

midiendo el uso de CPU y memoria en intervalos de tiempo para cada iteración. Posteriormente se agrupan los datos por tiempo y etiqueta (el nombre del algoritmo) y se obtienen la media y la desviación estándar.

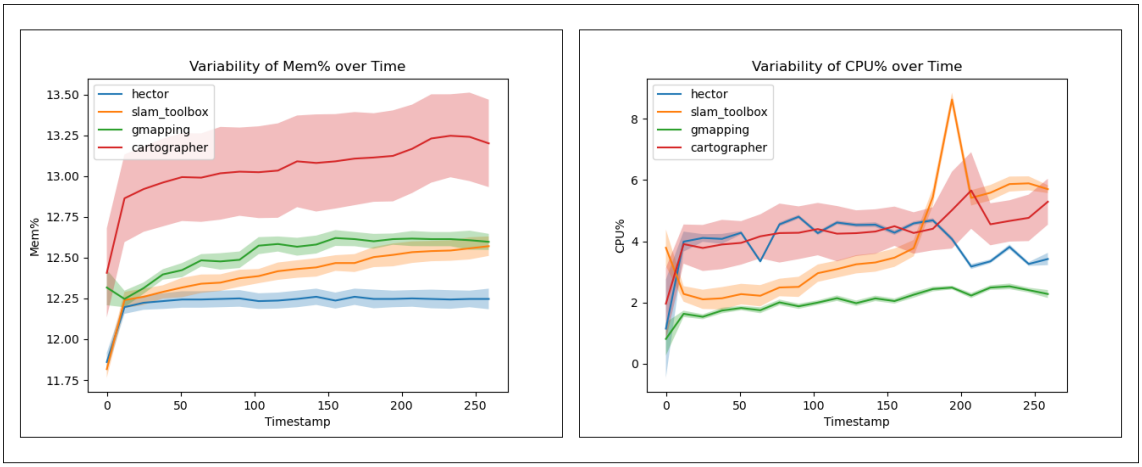


**Figura 3.16:** Uso de memoria y CPU durante la ejecución de cada algoritmo.

A simple vista se observa que cartographer es el algoritmo con más consumo medio de memoria y procesador, siendo también el algoritmo que presenta más dispersión en las medidas. El segundo algoritmo que más memoria consume es gmapping, pero a su vez es el que menos recursos de CPU utiliza. En cuanto a slam_toolbox, este tiene consumos moderados de memoria y CPU, que van aumentando conforme aumenta el tamaño del mapa, aunque es notable el pico de consumo de procesador que sufre en cierto punto del mapeo, este se debe a la detección del cierre de bucle y a la corrección del mapa global, ya que es un proceso con mayor demanda computacional. Este fenómeno también está presente en cartographer, aunque de manera menos pronunciada. Finalmente HectorSLAM no utiliza a penas memoria, y algo más de CPU.

En resumen, ningún algoritmo presenta consumos descabellados de recursos, ya que ninguno de ellos supera el 14 % de uso de memoria ni el 10 % de uso de procesador. Por lo que todos son buenas opciones para ser ejecutados en ordenadores personales con una potencia moderada. Si que es cierto que slam_toolbox y cartographer pueden presentar limitaciones en procesadores de bajo consumo como ARM, ya que para detectar y corregir cierres de bucle se requiere de una mayor capacidad de cómputo.

## 3.5 Evaluación de la localización

### 3.5.1. Obtención de la trayectoria real

Se pretende evaluar la calidad de la localización de varios algoritmos SLAM utilizando una trayectoria marcada como referencia y comparándola con la trayectoria obtenida por cada algoritmo [18], en este análisis comparativo se incluirán las trayectorias de los algoritmos de SLAM 2D y de RTAB-Map, el único algoritmo de SLAM Visual.

Se conduce el robot en un espacio no muy extenso, en lineas rectas y parando para realizar giros en 4 puntos diferentes. Cuando se va a girar se marca en el suelo la posición del robot. Luego se miden los segmentos formados por los puntos. Tras conducir el robot

**Figura 3.17:** Trayectoria real del robot marcada en el suelo.

queda la trayectoria de la figura 3.17. En la figura 3.18 se puede observar como se ha conducido el robot y como se han medido los segmentos con una cinta métrica, marcando los puntos de giro con cinta adhesiva.

Al haber 4 puntos, existen 3 segmentos por los que pasa el robot. El objetivo es obtener las coordenadas cartesianas de los cuatro puntos dadas las medidas de los tres segmentos para así poder representar y comparar gráficamente la trayectoria real con las trayectorias calculadas por los diferentes algoritmos SLAM.

Solo con tres segmentos no es suficiente para obtener las coordenadas de los puntos, por lo que también hay que medir las distancias entre los puntos no consecutivos para así obtener dos triángulos con los cuales poder calcular los respectivos ángulos entre los

**Figura 3.18:** Fotos del robot junto con las mediciones de los puntos.

segmentos. Con tres segmentos, una coordenada inicial (0,0) y los ángulos que forman dichos segmentos ya se podrán obtener el resto de las coordenadas de los puntos.

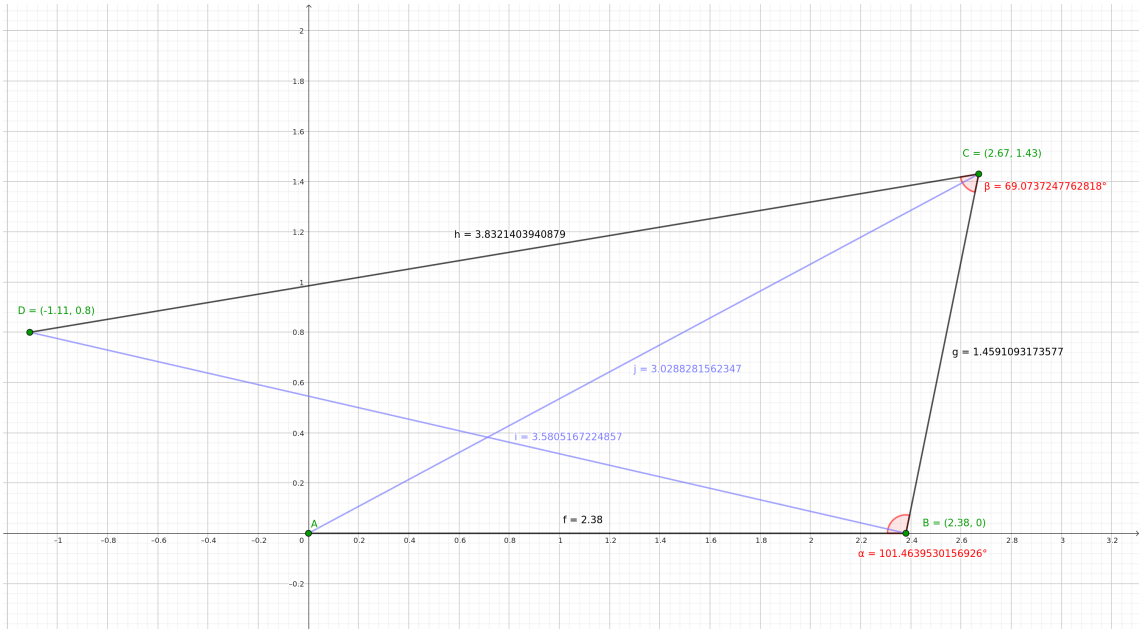


**Figura 3.19:** En negro la trayectoria real del robot. En azul los segmentos auxiliares para obtener los ángulos (en rojo), y posteriormente obtener las coordenadas de los puntos B, C y D.

En la figura 3.19 se marcan en azul los segmentos auxiliares que forman dos triángulos. Para saber el ángulo que forman los segmentos $f$ y $g$ se utiliza la ley del coseno:

$$\alpha = \frac{\arccos(f^2 + g^2 - j^2)}{2fg}$$

Donde $f$ es el segmento entre $A$ y $B$, $g$ es el segmento entre $B$ y $C$, y $j$ es el segmento entre $A$ y $C$. El resultado es el ángulo $\alpha$, entre $f$ y $g$, en este caso $\alpha = 101{,}46°$.

De igual manera, se utiliza la ley del coseno para obtener el ángulo $\beta$ entre los segmentos $h$ y $g$, utilizando el segmento auxiliar $i$. En este caso $\beta = 69{,}07°$.

Por lo tanto se dispone de una trayectoria con 4 puntos, sabiendo la longitud de los tres segmentos y sus dos ángulos de giro, suponiendo que el punto inicial es $A = (0,0)$ y que el desplazamiento al siguiente punto se realiza siguiendo el eje $X$, por lo que $B = (2{,}38, 0)$, se podrán calcular las coordenadas de los puntos restantes $C$ y $D$.

Para calcular $C$ se utilizan las siguientes expresiones:

$$Cx = Bx + g \cdot \cos(\alpha)$$

$$Cy = By + g \cdot \sin(\alpha)$$

Donde $Cx$ y $Cy$ son las coordenadas cartesianas de $C$, $Bx$ y $By$ las de $B$, $/alpha$ el ángulo entre $f$ y $g$, y $g$ el segmento entre $B$ y $C$. El resultado es $C = (2{,}67, 1{,}43)$. De igual forma se calcula $D$, cambiando los datos correspondientes y obteniendo $D = (-1{,}11, 0{,}8)$.

### 3.5.2. Obtención de las trayectorias SLAM

Cada algoritmo obtiene una estimación de la localización del robot en base a diferentes entradas de los sensores. El problema es que cada uno muestra la posición tomando como referencia *frames* o puntos diferentes, a parte de esto tampoco son consistentes entre si en el formato de mensaje en el que se publica la posición en cada instante.

El objetivo es obtener un archivo CSV para cada algoritmo con dos columnas: X e Y. Cada fila registra un punto diferente.

El primer método para obtener la lista de puntos que forman la trayectoria calculada por cada algoritmo es obtener los componentes de posición x e y de la transformación entre el *frame* m̈apÿ el *frame* b̈ase_link, esto es algo que ROS2 nos permite hacer con facilidad, programando un nodo que escuche el *topic* /tf y obtenga las posiciones.

El problema surge cuando los frames tienen una diferencia en sus marcas de tiempo, por lo que ROS2 no nos deja calcular la posición directamente. Esta diferencia de tiempo se debe al error de sincronización temporal previamente comentado en la sección Crítica al Estado del Arte del capítulo Estado del Arte. Este inconveniente surge en los algoritmos cartographer, gmapping y RTAB-Map.

Para estos casos especiales se han utilizado otros métodos alternativos para obtener las posiciones:

- **RTAB-Map**: el nodo publica un *topic* del tipo *PoseWithCovarianceStamped* [9], por lo tanto solo hay que programar un nodo que escuche en dicho *topic* y se guarde las coordenadas x e y de cada posición.
- **cartographer**: este nodo publicaba un topic de tipo *MarkerArray* [10], con una lista de posiciones que va aumentando conforme se registran nuevas. Se programa un nodo que escucha en este *topic* y se guarda el último elemento de la lista al terminar el mapeo, el cual contiene todas las posiciones, desde las más antiguas hasta las más recientes.
- **gmapping**: No publica ningún *topic*, pero imprime por salida estándar las posiciones que va registrando, por lo que solamente se redirecciona la salida estándar del nodo a un archivo y se le da formato CSV.

### 3.5.3. Comparación de trayectorias

Una vez obtenidas las trayectorias calculadas por los diferentes algoritmos SLAM, cada una en un archivo CSV, se deben de transformar para que todas tengan el mismo punto de referencia que la trayectoria real, es decir, el punto $A = (0, 0)$.

Se pone el ejemplo de la trayectoria calculada por RTAB-Map, aunque es igual para las trayectorias de slam_toolbox y gmapping. Se puede observar la trayectoria calculada comparada con la trayectoria real en la figura 3.20. El objetivo es alinear la trayectoria calculada con la real. Para ello, se aplica una primera transformación a todos los puntos

[9] Se puede consultar el formato del mensaje en: https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html

[10] Se puede consultar el formato del mensaje en: https://docs.ros.org/en/noetic/api/visualization_msgs/html/msg/MarkerArray.html

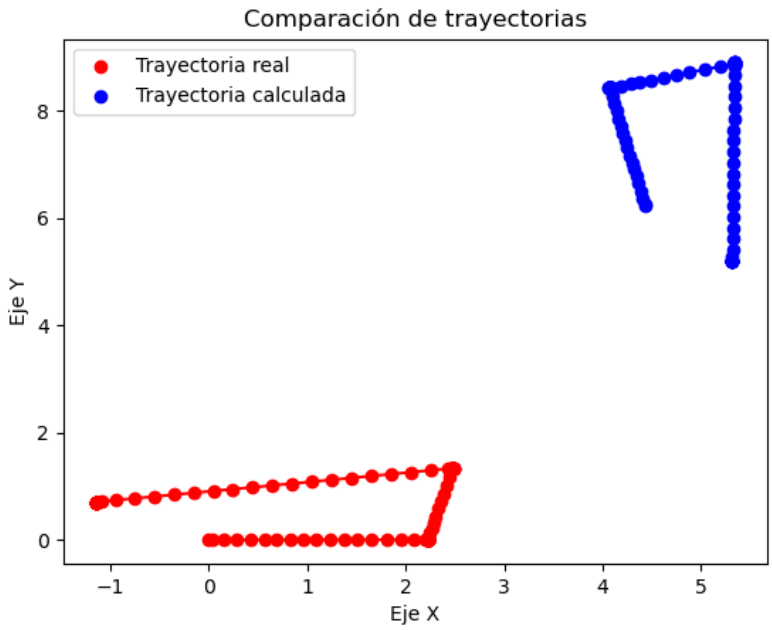


**Figura 3.20:** En azul, la trayectoria calculada sin transformar, en rojo, la trayectoria real.

de la trayectoria calculada, la cual será un desplazamiento equivalente a la distancia entre el primer punto obtenido y el origen de coordenadas. Posteriormente se rotará 180° respecto al eje Y y 90° respecto al eje Z.

Para las trayectorias de HectorSLAM y cartographer solo es necesario realizar la rotación de 180° respecto al eje Y y la traslación.

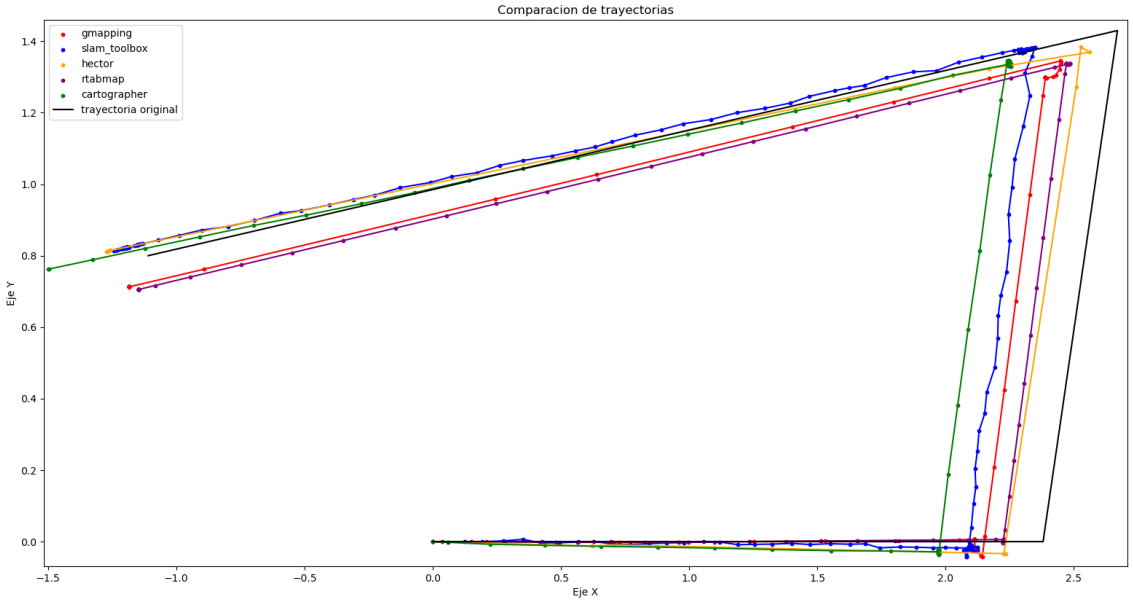


**Figura 3.21:** Trayectorias representadas gráficamente tras aplicarles las transformaciones convenientes.

Una vez se transforman todas las trayectorias se pueden representar gráficamente para compararlas visualmente con la real (ver Fig. 3.21).

También se puede obtener una estimación numérica del error de cada trayectoria. Se han utilizado dos métodos para estimar la precisión de las trayectorias. A continuación se describirán estos dos métodos y se elegirá el que mejor represente la realidad de las gráficas:

1. **Distancias entre puntos más cercanos:** el error se calcula obteniendo la distancia de cada punto de la trayectoria calculada respecto al punto más cercano de la trayectoria real. Como la trayectoria real solo está formada por cuatro puntos y tres segmentos que los conectan, se dividen estos segmentos cogiendo tantos puntos igualmente separados dentro estos como puntos hay en el segmento calculado. Por ejemplo, HectorSLAM calcula diez puntos en el primer tramo, por lo que se dividirá el tramo real en diez puntos diferentes de igual distancia entre si. Luego se medirá la distancia entre los puntos calculados por HectorSLAM y los puntos más cercanos de la trayectoria

   Esta no es una métrica ideal, ya que cada trayectoria dispone de una cantidad diferente de puntos. En la tabla 3.1 se muestra el error medio y el error máximo de cada algoritmo utilizando el método mencionado anteriormente, pero, como se puede observar, no se corresponde con la realidad que muestra la figura 3.21, ya que a simple vista se puede observar que HectorSLAM es de las trayectoria que más se asemejan a la real, y tampoco tiene ningún punto especialmente desviado, mientras que la tabla indica que es el peor de los algoritmos, teniendo el mayor error promedio y error máximo.

| Método | Error Promedio | Error Máximo |
|---|---|---|
| gmapping | 0.592273 | 1.877629 |
| slam_toolbox | 0.413419 | 1.560959 |
| HectorSLAM | 0.696634 | 2.543998 |
| RTAB-Map | 0.585274 | 1.745001 |
| cartographer | 0.435842 | 0.806478 |

**Tabla 3.1:** Errores en las trayectorias basados en la distancia entre los puntos más cercanos.

2. **Distancias punto-segmento:** el error es la media de las distancias entre los puntos comprendidos de un segmento de la trayectoria calculada y el propio segmento de la trayectoria real. Esta distancia es la proyección ortogonal del punto sobre el segmento. Si el punto carece de proyección ortogonal a su correspondiente segmento entonces se utilizará la distancia al extremo del segmento más cercano. Este es un método más simple en su implementación y no hace falta modificar la trayectoria real ni crear puntos ïmaginarios.

   Los resultados que se pueden observar en la tabla 3.2 se asemejan mucho más a la realidad, teniendo HectorSLAM la mejor trayectoria junto con RTAB-Map. Pero si se observa la cantidad de puntos calculados se discierne una clara diferencia en las resoluciones de las trayectorias, y es que este es un importante factor a tener en cuenta. Debido a errores de software, HectorSLAM solo es capaz de generar 16 puntos, mientras que slam_toolbox genera 453, pese a tener un mayor error promedio.

| Método | Error Promedio | Error Máximo | Cantidad de Puntos |
|---|---|---|---|
| gmapping | 0.254641 | 1.763977 | 113 |
| slam_toolbox | 0.237127 | 1.836548 | 453 |
| HectorSLAM | 0.141364 | 0.224882 | 16 |
| RTAB-Map | 0.216667 | 1.768230 | 188 |
| cartographer | 0.444757 | 1.914462 | 101 |

**Tabla 3.2:** Errores en las trayectorias basados en la distancia euclídea entre punto y segmento.

Habiendo tomado la figura 3.21 y la tabla 3.2 como referencia, se puede llegar a la conclusión de que HectorSLAM tiene la mayor precisión en este tipo de trayectorias más cortas, aunque tiene la menor resolución. El algoritmo RTAB-Map tiene el segundo error medio más bajo, y una buena resolución con un total de 188 puntos. Si lo que se desea es una tasa de actualización mayor, slam_toolbox ofrece gran resolución con un error aceptable. Finalmente cartographer obtiene los peores resultados en este escenario con el mayor error promedio y con 101 puntos totales, comportando una opción no muy aceptable para recorridos pequeños.

# CAPÍTULO 4
# Conclusiones

Como conclusión del presente trabajo caben destacar que se han cumplido todos los objetivos planteados este proyecto. En primer lugar, se han definido las principales técnicas SLAM, describiendo las principales características y métodos matemáticos y destacando los métodos SLAM basados en filtros de partículas o en grafos al ser los más comunes.

Se ha redactado una guía básica de puesta en marcha del robot Turtlebot4 y se ha justificado la elección de Discovery Server como configuración del DDS.

Se han listado y descrito los algoritmos disponibles en ROS 2 y se han evaluado utilizando métricas ya estudiadas, que se adaptan a las circunstancias y recursos del proyecto, llegando a la conclusión de que slam_toolbox es uno de los algoritmos más convenientes y con mejores resultados. También se han mostrando los puntos débiles y fuertes de cada uno.

También se ha llegado a la conclusión de que ROS 2, pese a ser un *framework* con muchas herramientas y librerías, no dispone de la suficiente madurez y soporte por parte de los desarrolladores opensource en el ámbito del SLAM, ya que muchos de los repositorios han dejado de mantenerse y muchos otros algoritmos no han sido implementados.

De igual manera, el robot TurtleBot 4 pese a ser un robot comercial resulta ser un producto con varias necesidades: mayor mantenimiento y soporte por parte de los desarrolladores y mayor colaboración con iRobot para una integración más robusta y completa con el Create 3. Mientras tanto será un robot poco recomendable, ya que existen errores que r̈ompenc̈ompletamente algunas funcionalidades básicas del robot, como la navegación autónoma y la evasión de obstáculos. Esto provoca que el robot no pueda ser puesto en producción ni siquiera para investigación.

## 4.1 Trabajos futuros

Con todos estos objetivos cumplidos, se da por concluido el proyecto, pero no sin antes mencionar algunas posibles futuras líneas de trabajo para complementar el presente trabajo:

- **Implementación y evaluación de algoritmos de SLAM Visual en ROS 2**: Debido a que solamente se ha conseguido trabajar con un algoritmo de SLAM Visual (RTAB-Map), se propone un trabajo donde se implementen o migren librerías a ROS 2

para poder realizar un análisis comparativo entre algunos otros algoritmos, como el conocido ORB-SLAM3 u OpenVSLAM.

- **Evaluación de SLAM 2D mediante *ground truth***: Como en este proyecto no se disponía de un mapa de referencia creado con herramientas externas al robot, es interesante un trabajo donde se genere un mapa de la planta del edificio y posteriormente compararlo con los generados por los algoritmos utilizando métodos de evaluación que no se han podido aplicar en este proyecto.
- **Optimización automática de algoritmos SLAM utilizando técnicas de inteligencia artificial:** debido a que este trabajo se ha utilizado la configuración por defecto con pequeñas variaciones manuales para hacer funcionar los algoritmos, sería interesante idear un proceso automático, inteligente e iterativo de optimización de los parámetros de cada algoritmo.

## 4.2 Relación del trabajo desarrollado con los estudios cursados

Para la realización de este trabajo se han puesto en práctica muchos conocimientos adquiridos en el Máster de Ingeniería Informática. La asignatura más útil ha resultado ser Automatización y Robótica, adquiriendo conocimientos de cinemática de los robots y de ROS. También han sido de gran ayuda las asignaturas de Configuración y Optimización de Sistemas de Cómputo, para la puesta en marcha del robot, y Servicios y Aplicaciones Distribuidas, para la comprensión y utilización de los patrones de comunicación utilizados en ROS 2.

A parte de estos ejemplos concretos, todas las asignaturas del máster han jugado un papel clave en el desarrollo del proyecto, ya que han proporcionado diferentes perspectivas y puntos de vista de la informática que han permitido aplicar un enfoque más amplio y completo al trabajo.

# Bibliografía


[1] A Alan B Pritsker. *Introduction to Simulation and SLAM II*. Halsted Press, 1984.

[2] Steven M La Valle. "Motion planning". En: *IEEE Robotics & Automation Magazine* 18.2 (2011), págs. 108-118.

[3] Yuya Maruyama, Shinpei Kato y Takuya Azumi. "Exploring the performance of ROS2". En: *Proceedings of the 13th International Conference on Embedded Software*. 2016, págs. 1-10.

[4] Sergio Cebollada et al. "A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data". En: *Expert Systems with Applications* 167 (2021), pág. 114195. URL: https://www.sciencedirect.com/science/article/pii/S095741742030926X.

[5] iRobot. *iRobot® Create® 3 Educational Robot*. 2023. URL: https://iroboteducation.github.io/create3_docs/.

[6] Sebastian Thrun y Michael Montemerlo. "The graph SLAM algorithm with applications to large-scale mapping of urban structures". En: *The International Journal of Robotics Research* 25.5-6 (2006), págs. 403-429.

[7] Jaromir Konecny, Michal Prauzek y Jakub Hlavica. "ICP Algorithm in Mobile Robot Navigation: Analysis of Computational Demands in Embedded Solutions." En: *IFAC-PapersOnLine* 49.25 (2016), págs. 396-400. URL: https://www.sciencedirect.com/science/article/pii/S2405896316327173.

[8] Jaromir Konecny et al. "Scan Matching by Cross-Correlation and Differential Evolution". En: *Electronics* 8 (ago. de 2019), pág. 856.

[9] Kurt Konolige et al. "Efficient sparse pose adjustment for 2D mapping". En: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, págs. 22-29.

[10] Kurt Konolige y Willow Garage. "Sparse Sparse Bundle Adjustment." En: *BMVC*. Vol. 10. 2010, págs. 102-1.

[11] Xieyuanli Chen et al. "OverlapNet: Loop closing for LiDAR-based SLAM". En: *arXiv preprint arXiv:2105.11344* (2021).

[12] Yujiao Jia, Xinying Yan y Yihan Xu. "A Survey of simultaneous localization and mapping for robot". En: *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. Vol. 1. IEEE. 2019, págs. 857-861.

[13] Giorgio Grisetti, Cyrill Stachniss y Wolfram Burgard. "Improved techniques for grid mapping with rao-blackwellized particle filters". En: *IEEE transactions on Robotics* 23.1 (2007), págs. 34-46.

[14] Steve Macenski e Ivona Jambrecic. "SLAM Toolbox: SLAM for the dynamic world". En: *Journal of Open Source Software* 6.61 (2021), pág. 2783.

[15] Mathieu Labbé y François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". En: *Journal of field robotics* 36.2 (2019), págs. 416-446.

[16] Carlos Campos et al. “Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam”. En: *IEEE Transactions on Robotics* 37.6 (2021), págs. 1874-1890.

[17] Rainer Kümmerle et al. “On measuring the accuracy of SLAM algorithms”. En: *Autonomous Robots* 27 (2009), págs. 387-407.

[18] Ilmir Z Ibragimov e Ilya M Afanasyev. “Comparison of ROS-based visual SLAM methods in homogeneous indoor environment”. En: *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*. IEEE. 2017, págs. 1-6.

[19] Rauf Yagfarov, Mikhail Ivanou e Ilya Afanasyev. “Map comparison of lidar-based 2d slam algorithms using precise ground truth”. En: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE. 2018, págs. 1979-1983.

[20] Anton Filatov et al. “2d slam quality evaluation methods”. En: *2017 21st Conference of Open Innovations Association (FRUCT)*. IEEE. 2017, págs. 120-126.

[21] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. En: *Science Robotics* 7.66 (2022), eabm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[22] Clearpath Robotics. *Turtlebot4 User Manual*. 2023. URL: https://turtlebot.github.io/turtlebot4-user-manual/.

[23] eProsima. *Fast DDS Documentation*. 2023. URL: https://fast-dds.docs.eprosima.com/_/downloads/en/latest/pdf/.

[24] Konstantinos G Derpanis. “The harris corner detector”. En: *York University* 2 (2004), págs. 1-2.

# ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

| **Objetivos de Desarrollo Sostenible** | **Alto** | **Medio** | **Bajo** | **No procede** |
|---|---|---|---|---|
| ODS 1. **Fin de la pobreza.** | | | X | |
| ODS 2. **Hambre cero.** | | | | X |
| ODS 3. **Salud y bienestar.** | | X | | |
| ODS 4. **Educación de calidad.** | | | | X |
| ODS 5. **Igualdad de género.** | | | | X |
| ODS 6. **Agua limpia y saneamiento.** | | | | X |
| ODS 7. **Energía asequible y no contaminante.** | | | | X |
| ODS 8. **Trabajo decente y crecimiento económico.** | X | | | |
| ODS 9. **Industria, innovación e infraestructuras.** | X | | | |
| ODS 10. **Reducción de las desigualdades.** | | | | X |
| ODS 11. **Ciudades y comunidades sostenibles.** | | | | X |
| ODS 12. **Producción y consumo responsables.** | | | | X |
| ODS 13. **Acción por el clima.** | | | | X |
| ODS 14. **Vida submarina.** | | | | X |
| ODS 15. **Vida de ecosistemas terrestres.** | | | | X |
| ODS 16. **Paz, justicia e instituciones sólidas.** | | | | X |
| ODS 17. **Alianzas para lograr objetivos.** | | | | X |

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El presente trabajo pretende mejorar el desarrollo de aplicaciones de robótica móvil, concretamente en el ámbito de los algoritmos SLAM. Si se consigue mejorar y optimizar la creación de mapas, se permitirá desarrollar novedosas e innovadoras aplicaciones para la realización de tareas autónomas por parte de los robots móviles, como la asistencia sanitaria en los hospitales y centros de mayores, la asistencia en entornos de producción para la mejora de la eficiencia y productividad de los robots en las plantas industriales y grandes almacenes. Todas estas innovaciones provocarán un mayor bienestar en la población

Para entrar en más detalle, los algoritmos SLAM permiten a los robots autónomos y drones navegar y trabajar de manera autónoma en entornos industriales. Esto aumenta la eficiencia de la producción, reduce los costos de mano de obra y mejora la calidad del trabajo realizado, lo que se traduce en un aumento de la productividad y, por lo tanto, en un crecimiento económico. Estos algoritmos también son esenciales en la automatización de almacenes y la gestión de la cadena de suministro. Los robots móviles y drones pueden realizar tareas de recolección, empaque y entrega de manera más eficiente y precisa, lo que mejora la logística y reduce los tiempos de entrega, lo que es crucial para el comercio electrónico y la distribución.

La robótica y los algoritmos SLAM se utilizan en la construcción para la inspección de obras, la topografía y la operación de maquinaria pesada. Esto acelera la construcción, reduce los errores y disminuye los riesgos laborales, lo que contribuye al crecimiento de la industria de la construcción.

En resumen, estos algoritmos tienen un impacto positivo en el crecimiento económico y la mejora de la industria al aumentar la eficiencia, reducir los costos, impulsar la innovación y crear nuevas oportunidades en diversos sectores. Estas tecnologías son clave para abordar los desafíos económicos y de productividad en un mundo cada vez más automatizado y orientado hacia la tecnología.

Los robots autónomos también pueden actual como robots asistenciales, estos pueden ayudar a las personas mayores o con diversidad funcional en tareas diarias, como la movilidad y la atención médica básica. Los robots autónomos también pueden entregar suministros médicos a áreas remotas o en situaciones de emergencia, lo que garantiza el acceso a tratamientos y medicamentos críticos.

En conjunto, la robótica y los algoritmos SLAM tienen un impacto significativo en la mejora de la salud de las personas al proporcionar un acceso más rápido a la atención médica, mejorar la precisión de los procedimientos médicos y reducir los riesgos laborales. Estas tecnologías continúan evolucionando y desempeñan un papel importante en la atención médica del futuro.

Por lo tanto, es seguro afirmar que el desarrollo de este trabajo puede contribuir a conseguir y mejorar varios ODS tanto en la mejora de las condiciones laborales, como en la asistencia a personas con necesidades especiales.