Informe Técnico – Technical Report
DPTOIA-IT
julio, 2023

Integración de un robot móvil en el entorno ROS (Robot Operating System). Análisis de la implementación en diferentes versiones

Sergio García González

Departamento de Informática y Automática
Universidad de Salamanca

D. Vidal Moreno Rodilla y D. Francisco Javier Blanco Rodríguez, doctores de la Universidad de Salamanca, del Departamento de Informática y Automática.

Hacen constar:

Que el trabajo que se recoge en la presente memoria, titulado "Integración de un robot móvil en el entorno ROS (Robot Operating System). Análisis de la implementación en diferentes versiones" ha sido realizado por D. Sergio García González bajo su tutela y que dicho trabajo cuenta con su visto bueno para ser presentado ante la Comisión Evaluadora.

En Salamanca, a 13 de julio de 2023

MORENO RODILLA VIDAL - 07870365H
Firmado digitalmente por MORENO RODILLA VIDAL - 07870365H
Fecha: 2023.07.13 11:17:38 +02'00'

BLANCO RODRIGUEZ FRANCISCO JAVIER - 07965219C
Firmado digitalmente por BLANCO RODRIGUEZ FRANCISCO JAVIER - 07965219C
Fecha: 2023.07.13 11:28:01 +02'00'

Firmado: Vidal Moreno Rodilla

Firmado: Francisco Javier Blanco Rodríguez

D. Sergio García González, alumno del Máster en Sistemas Inteligentes de la Universidad de Salamanca.

Declara:

Que ha realizado y redactado el Trabajo de Fin de Máster titulado "Integración de un robot móvil en el entorno ROS (Robot Operating System). Análisis de la implementación en diferentes versiones" del Máster Universitario en Sistemas Inteligentes de la Universidad de Salamanca en el segundo semestre del curso académico 2020/21 de forma autónoma, con la ayuda de las fuentes y la literatura citadas en la bibliografía, y que ha identificado como tales todas las partes tomadas de las fuentes y de la literatura indicada, textualmente o conforme a su sentido.

Además, soy conocedor de que el citado TFM forma parte de los trabajos de investigación que lleva(n) a cabo mi(s) director(es) Dr. Vidal Moreno Rodilla y Dr. Francisco Javier Blanco Rodríguez dentro del grupo de investigación GROUSAL de la Universidad de Salamanca y, en consecuencia, comparto con el(los) la propiedad intelectual de los resultados alcanzados.

En Salamanca, a 13 de julio de 2023

GARCIA GONZALEZ, SERGIO (FIRMA)

Firmado digitalmente por GARCIA GONZALEZ, SERGIO (FIRMA)
Fecha: 2023.07.13 17:43:24 +02'00'

Firmado: Sergio García González

Resumen

En los últimos años, el campo de la robótica ha experimentado una evolución exponencial, y el uso de robots móviles se ha vuelto cada vez más común en diversos sectores de la sociedad, como la industria, la medicina, la agricultura y la logística. Por ello, cada vez más los desarrolladores optan por el uso de herramientas o recursos que les permitan un desarrollo de software ágil. Para poder desarrollar software en la robótica, se requiere de herramientas y plataformas de desarrollo software especializadas. Una de las más destacadas y ampliamente utilizadas es el Robot Operating System (ROS).

En esta memoria, se realizará un análisis detallado de las diferentes versiones de ROS, desde las primeras versiones hasta las más recientes. Se examinarán las características y mejoras introducidas en cada una de ellas, así como las posibles limitaciones y desafíos asociados durante la prueba de las mismas.

Además, en este proyecto se abordará el proceso de integración de un robot móvil en las versiones de ROS 1 y ROS 2. Indicando el proceso seguido desde la selección de componentes principales, como la percepción, el control, los recursos software y la planificación de movimiento. En este proceso se explorarán diferentes paquetes, repositorios y herramientas disponibles en ROS que facilitan esta integración.

El objetivo principal de esta memoria es proporcionar a los lectores una visión global de la integración de un robot móvil en el entorno de ROS, abarcando las diferentes versiones de esta plataforma y su evolución a lo largo del tiempo.

En conclusión, se espera que esta investigación fomente el uso y la investigación en el campo de la robótica, impulsando el avance de la tecnología y su aplicación en diversas áreas de la sociedad.

Abstract

In recent years, the field of robotics has undergone an exponential evolution, and the use of mobile robots has become increasingly common in various sectors of society, such as industry, medicine, agriculture and logistics. As a result, more and more developers are opting for the use of tools or resources that enable agile software development. In order to develop software in robotics, specialised software development tools and platforms are required. One of the most prominent and widely used is the Robot Operating System (ROS).

In this report, a detailed analysis of the different versions of ROS, from the earliest versions to the most recent ones, will be carried out. It will examine the features and improvements introduced in each of them, as well as the possible limitations and challenges associated with testing them.

In addition, this project will address the process of integrating a mobile robot into ROS 1 and ROS 2 versions, indicating the process followed from the selection of main components, such as perception, control, software resources and motion planning. In this process, different packages, repositories and tools available in ROS that facilitate this integration will be explored.

The main objective of this report is to provide readers with an overview of the integration of a mobile robot in the ROS environment, covering the different versions of this platform and its evolution over time.

In conclusion, it is hoped that this research will encourage the use of and research in the field of robotics, driving the advancement of the technology and its application in various areas of society.

Índice

Índice de figuras

1. Introducción

Es una realidad que la robótica se desarrolla actualmente a una velocidad exponencial, y cada vez más, se encuentra presente en nuestra realidad. Dentro de la robótica existen multitud de recursos para generar software, algunos de ellos creados por empresas y que son exclusivamente privados y otros libres. En concreto para el software libre una de las opciones principales usadas por los desarrolladores es el uso del marco de trabajo ROS, que facilita el desarrollo de todo tipo de robótica.

ROS o **Robot Operative System** no es considerado un sistema operativo como tal, pero ejerce en muchos aspectos de forma similar a como lo podría hacer un sistema operativo convencional. ROS no realiza una gestión de los recursos hardware, ni administra la memoria, ni se encarga de la gestión de interrupciones o el control de periféricos, sin embargo se encarga de facilitar las tareas más esenciales de la robótica como la comunicación y coordinación entre diferentes componentes de un sistema robótico, proporciona un conjunto de herramientas, bibliotecas y convenciones que facilitan la programación y el desarrollo de software para robots y permite la modularización y reutilización del código.

Es por ello que *ROS no es considerado un sistema operativo* en sí mismo, pero realiza acciones de comunicación y coordinación similares a las que realiza un sistema operativo tradicional.

En concreto este marco de trabajo ha tenido dos versiones principales, ROS 1 con la que se ha estado trabajando más de una década tanto a nivel educativo como a nivel empresarial y ROS 2 que es una nueva versión que intenta mejorar aspectos como la seguridad, el rendimiento y la escalabilidad de la robótica con el objetivo de hacer un "sistema operativo"mucho más robusto.

La integración de ROS ha sido fundamental en el desarrollo de la robótica ya que ha permitido que se hayan producido avances significativos en los últimos años. Con este proyecto, se quieren abordar los aspectos claves de la integración de ROS en la robótica, como es el uso de diferentes versiones, la construcción de prototipos, el uso de herramientas o recursos comerciales y la investigación actual en torno a ello.

Además de incidir en la integración de ROS 1 y ROS 2, en este proyecto se va a insistir en el uso de las técnicas de navegación como el **SLAM 2D** (Simultaneous Localization and Mapping) **y 3D**, así como la navegación autónoma y la localización precisa en entornos robóticos. Al igual que las técnicas son muy importantes, *el hardware desempeña una tarea crucial* a la hora de la elaboración de un prototipo robótico por lo que es importante resaltar las tareas de elección de recursos hardware del mismo modo.

El SLAM 2D y 3D permite a los robots construir mapas del entorno a medida que se mueven, al mismo tiempo que estiman su propia posición dentro de ese mapa. Esto es esencial para la navegación autónoma, ya que proporciona al robot una comprensión espacial y la capacidad de planificar rutas eficientes para alcanzar destinos específicos. Estas técnicas se basan en el uso de sensores como cámaras, LIDAR

(Light Detection and Ranging) y sistemas de odometría para recopilar información del entorno y determinar la posición relativa del robot en tiempo real.

La **construcción de prototipos funcionales** desempeña un papel crucial en la integración de ROS en la robótica. Los prototipos permiten a los desarrolladores probar y validar las funcionalidades implementadas, así como identificar posibles limitaciones o mejoras requeridas. Mediante la construcción de prototipos, se pueden realizar pruebas exhaustivas de la integración de los diferentes componentes de un sistema robótico, como la percepción, la planificación de movimientos, la navegación y la interacción con el entorno.

En conclusión, la integración de ROS 1 y ROS 2 en la robótica impulsa la necesidad de utilizar técnicas avanzadas de navegación, como el SLAM 2D y 3D, y aborda el desafío de la localización precisa. Estos aspectos son esenciales para lograr la navegación autónoma de los robots en entornos reales. Además, la construcción de prototipos funcionales permite a los desarrolladores probar y mejorar las funcionalidades implementadas, garantizando un desarrollo robusto y confiable de los sistemas robóticos. En conjunto, estos avances contribuyen al rompecabezas de implementar una integración exitosa de ROS en la robótica.

2. Objetivos

Con este proyecto de investigación se quiere lograr como objetivo principal la integración de un robot móvil en el entorno de ROS, en sus dos versiones, ya se de manera real o de manera simulada. Con este objetivo se busca poder mostrar una visión global del desarrollo actual y futuro de la robótica enmarcada en el entorno de ROS. Del mismo modo de este objetivo principal surgen los objetivos secundarios mediantes los cuales se ha podido trabajar poco a poco hasta llegar al objetivo global. Estos objetivos son los siguientes:

- Realizar un estudio en forma de revisión de la literatura de las diferentes implementaciones de robots móviles en ROS.
- Realizar un estudio sobre las principales técnicas de escaneo en interiores y navegación.
- Realizar un estudio de las diferentes tecnologías y hardware asociado al escaneo de interiores.
- Selección de los recursos hardware para la construcción de los prototipos reales.
- Investigación de diferentes técnicas de SLAM para la compresión de sus principios y el estudio de sus ventajas.
- Realización de pruebas con diferentes robots móviles en entornos reales y simulados, utilizando diferentes versiones de ROS 1 y ROS 2.
- Identificación de limitaciones para integrar un robot móvil en ROS 1 y ROS 2.

3. Estado del arte

Para el desarrollo del estado del arte, se presentará el estado de las investigaciones actuales en cuanto a la integración de ROS en robótica móvil, así como de su implantación en el sector servicios como parte de la continuación del trabajo de fin de grado. Se ha recurrido a uno de los trabajos desarrollados en este máster en Sistemas Inteligentes, en el que se realiza una revisión de la literatura mediante un Mapping en el que se recogen de manera rigurosa y transparente las investigaciones de un campo de forma más concreta y que ayudarán a la elaboración de un estado del arte más completo.

3.1. Revisión sistemática de la literatura sobre la robótica móvil asociada al sector servicios

La robótica móvil se ha convertido en un campo de investigación y de rápido crecimiento en los últimos años. Con avances tecnológicos cada vez más sorprendentes, los robots móviles se están convirtiendo en una presencia común en diversas industrias, especialmente en el sector servicios. Pero antes de sumergirse en la teoría y los conceptos fundamentales de la robótica móvil para poder integrar un robot, es crucial comprender por qué es importante conocer ejemplos de investigaciones en este ámbito, específicamente en el sector servicios, que es el campo que ya he tratado.

El sector servicios abarca una amplia gama de áreas, como la atención médica, la logística, la hostelería, la seguridad y muchas más. En cada una de estas áreas, los robots móviles están desempeñando un papel cada vez más importante, ya sea realizando tareas repetitivas, ayudando en la atención al cliente o incluso tomando decisiones complejas basadas en algoritmos avanzados. Estos avances en la robótica móvil están transformando la forma en que se brindan los servicios y tienen un impacto significativo en la eficiencia, la calidad y la experiencia del cliente.

Al conocer ejemplos de investigaciones en robótica móvil aplicada al sector servicios, se pueden comprender mejor los desafíos y las posibilidades al integrar un robot en un marco de trabajo. Estos ejemplos prácticos permiten visualizar cómo los robots móviles están siendo implementados con éxito en situaciones del mundo real, superando obstáculos y mejorando los procesos existentes.

Para ello se ha realizado una revisión del arte inicial cercana al trabajo de integración ya realizado durante el trabajo de fin de grado.

3.1.1. Metodología utilizada

Para poder llevar a cabo un mapping sistemático se han hecho usos de diferentes herramientas y metodologías. Para gestionar la bibliografía se ha usado la herramienta Parfisal, que nos permite realizar por nosotros todo el trabajo duro de las

búsquedas, indicándole las palabras clave, los criterios de exclusión e inclusión y utilizando la metodología PICO. En esta herramienta online podemos realizar por completo todos los procesos necesarios de un mapping. Mediante PICO podemos formular preguntas de investigación más específicas y estructuradas, lo que facilita la búsqueda y el análisis de los artículos posteriormente. Además, se hará uso de la metodología PRISMA que se ha visto en la asignatura para la selección, filtrado y cribado de artículos.

El campo de la robótica es un campo muy amplio y por lo general en el desarrollo de software para componentes o plataformas robóticas suele ser lento y tedioso. Por lo tanto, la mayoría de ellos suelen optar por metodologías ágiles como ya he mencionado antes con ROS.

El mapeo sistemático de literatura sobre el uso de ROS, como marco de trabajo, en robots del sector servicios es una investigación que busca recopilar, analizar y sintetizar de manera sistemática los estudios existentes sobre la aplicación de ROS en robots utilizados en la industria del sector servicios. Este enfoque permitirá identificar las tendencias, las características principales, las ventajas y las limitaciones del uso de ROS en este ámbito específico, brindando una visión general y actualizada de las investigaciones relacionadas. El mapeo sistemático de literatura ayudará a comprender el estado actual de la aplicación de ROS en robots para la interacción con las personas, y proporcionará una base sólida para futuras investigaciones y desarrollos en este campo.

En general, la creación de ROS es relativamente cercana, porque fue lanzado en 2010, por lo que el número de investigaciones, aunque sea numeroso, cuando hagamos una búsqueda más concreta, lo más probable es que el número de artículos relacionados sea un número pequeño. Por ello, este trabajo se centrará en los avances y aplicaciones de los últimos 10 años, poniendo atención en el uso de ROS. Para este proceso de identificación de artículos científicos vamos a usar las webs: IEEE Xplore, Scopus, Arxiv Y Google Scholar, ya que se tratan de revistas / webs de propósito general con un gran número de artículos que están disponibles gratuitamente o a través de las licencias que posee la Universidad de Salamanca.

Por otra parte, este trabajo se centrará en tratar de resolver las siguientes cuestiones:

- ¿Qué estudios existen que utilicen ROS para el desarrollo de robots de servicios?
- ¿Qué versiones de ROS se están usando para desarrollar estos robots?
- ¿En que ámbitos se está aplicando ROS en la robótica orientada al sector servicios?
- ¿Qué éxito o que problemas presentan el uso de ROS en la robótica?

Una vez se han establecido las preguntas a las que queremos dar respuesta buscando artículos científicos que hayan tratado estos temas, debemos definir los siguientes criterios de inclusión y exclusión:

Criterios de inclusión:

- Los artículos están disponibles para descargar o ver completamente gratuitos AND
- Los artículos están disponibles para descargar o ver completamente con la licencia de la Universidad de Salamanca AND
- Los artículos se encuentran en una Revista o Libro AND
- Los artículos se encuentran en una conferencia o congreso, pero son relevantes AND
- Los artículos están escritos en inglés o español AND
- Los artículos tratan sobre la implantación de ROS en alguna de sus versiones AND
- Los artículos aplican técnicas de escaneo en profundidad AND
- Los artículos aplican la robótica en el sector servicios o similar

Criterios de exclusión:

- Los artículos no están disponibles para descargar o ver completamente gratuitos OR
- Los artículos no están disponibles para descargar o ver completamente con la licencia de la Universidad de Salamanca OR
- Los artículos no se encuentran en una Revista o Libro OR
- Los artículos se encuentran en una conferencia o congreso, pero no son relevantes OR
- Los artículos no están escritos en inglés o español OR
- Los artículos no tratan sobre la implantación de ROS en alguna de sus versiones AND
- Los artículos no aplican técnicas de escaneo en profundidad AND
- Los artículos no aplican la robótica en el sector servicios o similar

Tras haber definido los criterios de inclusión y exclusión, se puede comenzar con el proceso de extracción de artículos. Para ello, voy a recopilar los artículos de los últimos 10 años, es decir, desde el año 2013 hasta el año 2023. Para proceder a la búsqueda se ha utilizado la técnica PICO como ya mencione, de esta forma basándonos en las preguntas iniciales definimos unas palabras clave que nos ayuden a construir nuestras cadenas de búsqueda. La lista de palabras clave es la siguiente:

service robot, delivery robot, catering robot, robot restoration, ROS, ROS 1, ROS 2, camera, turtlebot Una vez definidas las palabras clave se ha formado la siguiente cadena de búsqueda:

Cadena de búsqueda
(service robot **OR** delivery robot **OR** catering robot **OR** robot restoration) **AND** (ROS **OR** ROS 1 **OR** ROS 2) **AND** (CAMERA **OR** turtlebot)

Aunque esa cadena de búsqueda está bien formada, debemos realizar alguna modificación para poder buscarla en las diferentes webs que hemos indicado, de tal forma que obtenemos las siguientes cadenas de búsqueda:

Cadena de búsqueda para IEEE Xplore
(All Metadata:service robot OR All Metadata:delivery robot OR All Metadata:catering robot OR All Metadata:robot restoration) AND (All Metadata:ROS OR All Metadata:ROS 1 OR All Metadata:ROS 2) AND (All Metadata:camera OR All Metadata:turtlebot)

Cadena de búsqueda para Scopus
(TITLE-ABS-KEY(service robot OR delivery robot OR catering robot OR robot restoration) AND TITLE-ABS-KEY(ROS OR ROS 1 OR ROS 2) AND TITLE-ABS-KEY(camera OR turtlebot)) AND PUBYEAR >2013 AND PUBYEAR <2023 AND (LIMIT-TO (LANGUAGE,English) OR LIMIT-TO (LANGUAGE,Spanish))

Cadena de búsqueda para Arxiv
((service robot OR delivery robot OR catering robot OR robot restoration) AND (ROS OR ROS 1 OR ROS 2) AND (camera OR turtlebot))

Cadena de búsqueda para Google Scholar
((service robot OR delivery robot OR catering robot OR robot restoration) AND (ROS OR ROS 1 OR ROS 2) AND (camera OR turtlebot))

3.1.2. Artículos extraídos

Tras haber realizado las consultas anteriores y obtener toda la información, es decir, los artículos científicos, se almacenan en formato BibTex para poder usarlos posteriormente en la web de Parsifal. Alguna de las páginas web de donde se han extraído los artículos no permiten extraerlos en formato BibTex, como es el caso de la web Arxiv, para ello se han hecho uso de webs externas para obtener las referencias a los artículos en ese formato.

Después de haber realizado este proceso, obtenemos una cantidad de 96 artículos, distribuidos en las diferentes páginas de búsqueda que hemos utilizado. De esta forma hemos obtenido, 15 en IEEE Explores, 43 en Scopus, 5 en Arxiv y 33 en Google Scholar (Aunque en Google Scholar se ha limitado la búsqueda artículos solo en español, ya que sí no obteníamos demasiados resultados).

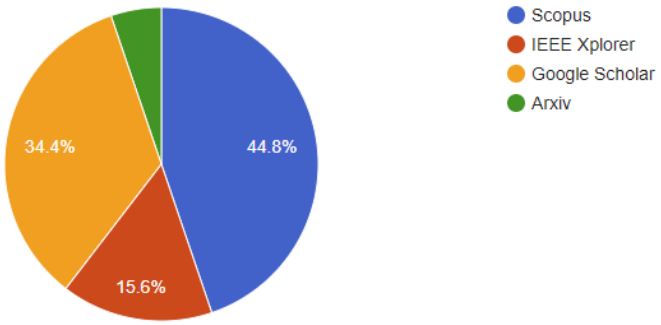

Figura 1: Artículos obtenidos

Por su puesto no todos los artículos van a ser útiles o se pueden usar, hay que realizar un cribado, para ello se ha usado la metodología PRISMA que se puede observar en la figura 2.

Mediante esta metodología lo primero que hacemos es seleccionar todo el conjunto de artículos que hemos obtenido de las diferentes webs, tras ello al haber realizado búsquedas similares en las diferentes webs es necesario eliminar todos aquellos artículos que se encuentren duplicados. Y tras eliminar los duplicados se han obtenido 68 artículos de los 96 iniciales. Para proseguir con el filtrado, lo siguiente que hemos hecho ha sido eliminar los artículos que pertenezcan a congresos y conferencias, pero sin ser muy relevantes sobre los temas que estamos tratando. Tras aplicar este filtro de los 68 artículos que disponíamos se han excluido 3. Tras ello se revisan los artículos restantes, leyendo con más detalle el título y su resumen, y en función de si presentan las palabras claves mencionadas anteriormente y si mediante el resumen parecen estudios relevantes sobre la robótica centrada en el sector servicios y ROS, pues los aceptamos y los que no los rechazamos. Pasando así de 65 artículos a tan solo 30. Finalmente, en base a unos criterios de calidad que establecemos en forma de preguntas, con los que daremos una puntuación a los artículos seleccionados an-

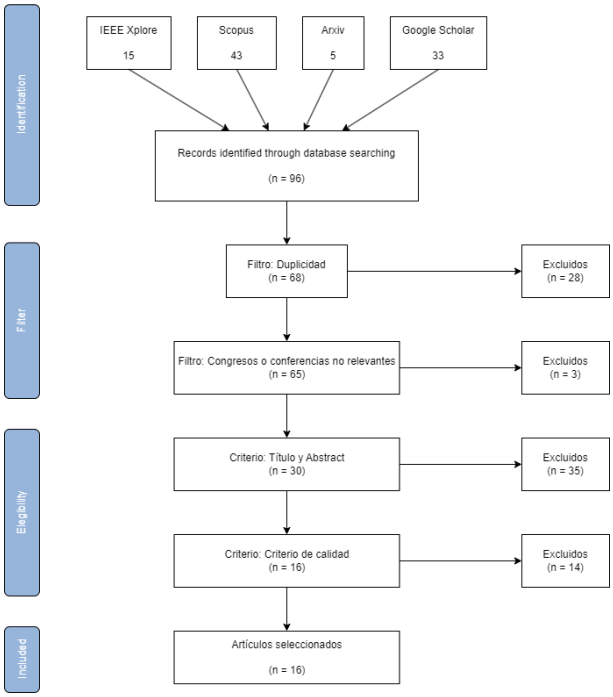

Figura 2: Metodología PRISMA

teriormente, si superan una puntuación los aceptaremos como artículos finales. Los criterios de calidad son los siguientes:

- ¿El artículo proporciona resultados precisos?
- ¿Realmente se hace uso de técnicas o modelos de reutilización del software?
- ¿El estudio presenta alguna de las técnicas mencionadas en las palabras clave?
- ¿Los objetivos del estudio son claros?

Aplicando dichos criterios y teniendo en cuenta tan solo aquellos que obtengan una puntuación >=4.0 se obtiene como resultado un conjunto de 16 artículos.

Además, la web de parsifal nos proporciona un gráfico con los artículos aceptados y rechazados de cada una de las fuentes que hemos utilizado, como se puede ver en la figura 3.

3.1.3. Análisis de la integración robótica en el sector servicios

En el ámbito de la robótica aplicada al sector hostelero y sanitario, se han realizado diversos estudios y proyectos relacionados con la implementación y diseño de robots, a menudo basados en plataformas como Turtlebot. Estos artículos exploran la integración de la robótica en entornos como restaurantes, hoteles, hospitales y centros de atención médica, con el objetivo de mejorar la eficiencia, la calidad de los servicios y la interacción con los usuarios. Mediante el análisis de estos artículos,

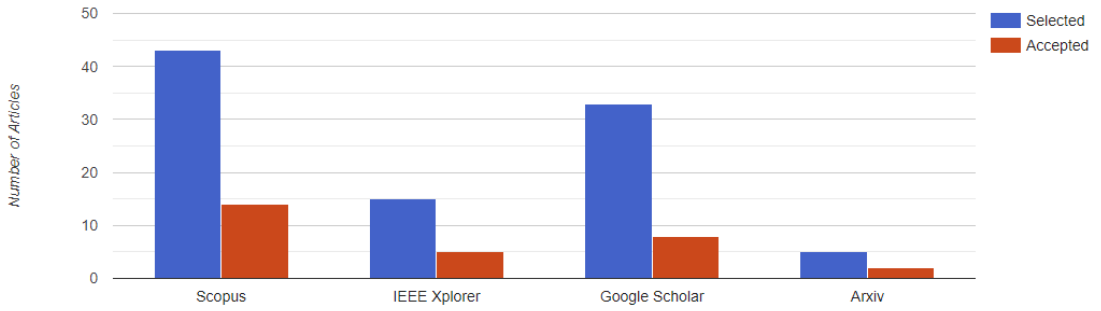

Figura 3: Artículos excluidos/incluidos según la fuente

se puede obtener una visión detallada de las tecnologías utilizadas, los desafíos enfrentados y los resultados obtenidos en la aplicación de robots en estos sectores, lo que contribuye a la comprensión y el avance de la robótica en contextos hostelería y sanitarios.

En el artículo *Inferring user intent to interact with a public service robot using bimodal information analysis* [1] se presenta un método novedoso y práctico para predecir las intenciones de interacción de las personas con un robot de servicio público en entornos poblados. A diferencia de las investigaciones tradicionales, que utilizan solo señales visuales para analizar la atención del usuario, este método combina la información de una cámara RGB-D y un láser para percibir al usuario, logrando una percepción de 360 grados y compensando la falta de perspectiva de la cámara RGB-D.

Se extrajeron siete tipos de características de intención interactiva y se entrenó un modelo de regresión de bosques aleatorios para puntuar las intenciones de interacción de las personas en el campo de visión. Además, se diseñó una regla de prioridad para la inferencia de intenciones considerando el orden de dos sensores diferentes. El algoritmo se implementó en un sistema operativo de robot (ROS) y se evaluó en un robot de servicio público. Los resultados experimentales extensos demuestran que el método propuesto permite que los robots de servicio público logren un nivel más alto de cortesía que el enfoque tradicional e interactivo pasivo en el que los robots esperan comandos de los usuarios. Aunque este primer artículo estaba más relacionado con la interacción de los humanos con el robot es importante tener esto en cuenta ya que el sector hostelero o de la restauración es un sector que está en alto contacto con personas.

Es por ello por lo que la investigación de Ruchik Mishra [2] presenta un **robot hostelero** inteligente basado en ROS que simplifica el proceso de registro, uno de los problemas más relevantes en este sector en el que tienen que recibir a multitud de personas. Además, el robot puede planificar rutas con precisión y llegar a las habitaciones en diferentes pisos. También, se agregó un sistema de voz inteligente que brinda asistencia a los clientes. De nuevo se muestra la importancia de la interacción humano máquina que es tan necesaria de potenciar últimamente en la robótica.

Finalmente decidí que uno de los artículos [3] que trataba sobre la aplicación de algoritmos de localización, era relevante ya que utilizaba el mismo robot que había usado para el desarrollo de mi TFG [4]. En el muestra resultados de la implementación de un sistema de localización en el que gracias a un escaneo realizado con una cámara RGB-D que utiliza el algoritmo Monte Carlo, consigue ofrecer unos resultados de localización excelentes.

En el caso de las investigaciones relacionadas con las **robótica y el sector sanitario** parece ser que he encontrado algunas más que para el sector hostelero que está todavía creciendo en este aspecto. Por lo general la mayoría de soluciones robóticas planteadas en los estudios han sido los AGVS. [5]. Los AGVS son la solución más usada ya que la mayoría de empresas que se dedican a la robótica orientada al transporte de productos se enfocan en el desarrollo de estos.

Tras leer algunos de los artículos de forma breve se puede concluir que los robots se utilizan en el transporte sanitario debido a varias razones. En primer lugar, el creciente número de personas mayores [6] y su tendencia a sufrir enfermedades crónicas requieren una atención médica adecuada. Los robots pueden desempeñar un papel fundamental en la entrega de medicamentos a tiempo y en el monitoreo de signos vitales en hogares de ancianos, lo que ayuda a garantizar el cuidado adecuado de los adultos mayores. Además, los robots pueden actuar como intermediarios en la entrega de alimentos, medicamentos y suministros médicos en entornos hospitalarios [7], lo que facilita la gestión logística y libera a los profesionales de la salud para que se centren en tareas más especializadas. Los robots también pueden proporcionar una navegación autónoma basada en inteligencia artificial, lo que les permite moverse de manera eficiente en entornos hospitalarios y cumplir con sus tareas asignadas de manera segura [7] [8] [9] [10], como se ha mencionado en otros artículos.

Para el caso de **robots orientados al sector doméstico**, pero basados en el marco de trabajo ROS y en relación con turtlebot también se han encontrado varios resultados. Encontrando diversos estudios que llevan a cabo una implantación similar a las anteriores en cuanto a hardware y software, pero con un punto de vista distinto [11].
Sin irse más lejos, el uso de robots domésticos como robots de servicio es cada vez más común debido a varias razones. En primer lugar, el envejecimiento de la población y la necesidad de asistencia para personas discapacitadas [12] [13] han generado problemas sociales importantes. Los robots de servicio en el hogar pueden ayudar a mitigar estos problemas al proporcionar asistencia en tareas diarias y cuidado personal.
Además, los robots domésticos están diseñados para interactuar de manera natural con los usuarios humanos, lo que incluye la comprensión y respuesta a comandos de voz, así como la capacidad de interpretar el contexto social y moverse de manera segura y socialmente apropiada en entornos domésticos. Estos robots también pueden brindar asistencia en la realización de tareas cotidianas y ofrecer seguridad, como la detección de fugas de gas [14] o alarmas de detección de movimiento. Aunque todavía existen desafíos y áreas de mejora en el desarrollo de robots domésticos, como la autonomía completa y la realización de inspecciones y servicios en el hogar de manera independiente, los avances en tecnología y la implementación de sistemas

basados en ROS han permitido el diseño de robots domésticos de bajo costo con funciones completas y buen rendimiento. [15] [16]

En conclusión, la implementación y el diseño de robots orientados al sector servicios y sanitario han experimentado avances significativos en los últimos años. Los estudios y proyectos llevados a cabo han demostrado el potencial de la robótica para mejorar la eficiencia operativa, la calidad de los servicios y la interacción con los usuarios en entornos como la hostelería y la atención médica. La utilización de plataformas como Turtlebot ha facilitado el desarrollo de aplicaciones robóticas versátiles y personalizables.

Sin embargo, a pesar de los avances realizados, aún existen desafíos y complicaciones a la hora de encontrar estudios relacionados. La implementación de robots en estos sectores conlleva desafíos técnicos, éticos y regulatorios. La adaptación de los robots a entornos complejos y cambiantes, la garantía de la seguridad y la privacidad de los datos, así como la aceptación y confianza de los usuarios son aspectos críticos a abordar.

Con el tiempo, se espera que la implementación de robots en estos sectores se convierta en una práctica más común, mejorando la calidad de vida de las personas y optimizando los procesos en la industria de servicios y sanitaria.

Por ultimo cabe destacar la falta de artículos relacionados con un tema tan concreto como es la robótica orientada a este tipo de sectores, en los cuales la investigación todavía es breve. También es importante resaltar las palabras claves que se han utilizado, ya que se han usado con el objetivo de acercarse tanto a la robótica desarrollada con el framework ROS como al sector servicios de forma muy concreta por lo que es posible mejorar estos resultados separando ambas búsquedas con una temática concreta y ampliando el rango de búsqueda con otros conceptos pertenecientes a dicha temática. Ya que aunque el diseño e implementación de robots de servicios sea pequeño, aunque en auge, el uso de ROS esta cada vez más extendido en el ámbito de la robótica.

3.2. Revisión sistemática de la literatura sobre la robótica móvil y su integración en ROS

Sin embargo, después de toda la búsqueda de investigación relacionada en torno a ROS, el sector servicios y turtlebot, se quería indagar algo más en la integración de la robótica móvil en el marco de trabajo de ROS, es por ello que el mapping fue de ayuda en los primeros pasos del estado del arte pero la investigación va más allá de este mapping inicial. Para ello se quería sentar unas bases en cuanto a las tecnologías de escaneo utilizadas, las técnicas y algoritmos de navegación autónoma y la localización interior de un robot móvil.

3.2.1. Artículos extraídos

Al igual que se hizo en el caso anterior de Revisión sistemática sobre la robótica móvil asociada al sector servicios a continuación se va a utilizar la misma metodología para la recolección y análisis de los artículos. De la misma forma se utilizarán una serie de bases de datos, debido a la complicación anterior de trabajar con Google Scholar solo se utilizarán las siguientes: IEEE Xplore y Scopus. Y se especificarán a continuación una serie de preguntas o cuestiones que se intentaran resolver:

- ¿Cuáles son las diferencias clave entre ROS 1 y ROS 2 en términos de la integración de un robot?
- ¿Cómo se maneja la comunicación entre nodos en ROS 1 y en ROS 2, y qué mejoras ofrece ROS 2 en comparación con ROS 1?
- ¿Qué herramientas y recursos están disponibles para facilitar la integración de un robot en ROS 1 y en ROS 2, y cuáles son las mejores prácticas recomendadas en cada versión?
- ¿Cómo se realiza el mapeo del entorno en la navegación de robots y qué técnicas o algoritmos se utilizan para construir mapas precisos y actualizados?
- ¿Cuáles son los algoritmos de planificación de trayectorias más comunes utilizados en la navegación de robots y cómo se comparan en términos de eficiencia y precisión?

Teniendo en cuenta las cuestiones anteriores y la información que se quiere obtener de ellas, del mismo modo que con la anterior revisión de la literatura o mapping se elaboran unos criterios de inclusión y exclusión que son los siguientes:

Criterios de inclusión:

- Información relevante sobre las diferencias clave entre ROS 1 y ROS 2 en la integración de un robot AND
- Detalles sobre cómo se maneja la comunicación entre nodos en ROS 1 y en ROS 2, y las mejoras que ofrece ROS 2 en comparación con ROS 1 AND
- Descripción de herramientas y recursos disponibles para la integración de un robot en ROS 1 y en ROS 2, así como las mejores prácticas recomendadas en cada versión AND
- Explicación de técnicas y algoritmos utilizados en el mapeo del entorno en la navegación de robots, y cómo se construyen mapas precisos y actualizados AND
- Comparación de los algoritmos de planificación de trayectorias más comunes en términos de eficiencia y precisión en la navegación de robots AND

- Los artículos están disponibles para descargar o ver completamente gratuitos AND
- Los artículos están disponibles para descargar o ver completamente con la licencia de la Universidad de Salamanca AND

Criterios de exclusión:

- Información que no esté directamente relacionada con las diferencias entre ROS 1 y ROS 2 en la integración de robots OR
- Detalles sobre otras áreas de ROS que no estén específicamente relacionadas con la comunicación entre nodos OR
- Recursos y herramientas no relevantes para la integración de un robot en ROS 1 y ROS 2 OR
- Técnicas y algoritmos no relacionados con el mapeo del entorno en la navegación de robots OR
- Algoritmos de planificación de trayectorias que no sean comunes en la navegación de robots o que no se comparen en términos de eficiencia y precisión OR
- Los artículos no están disponibles para descargar o ver completamente gratuitos OR
- Los artículos no están disponibles para descargar o ver completamente con la licencia de la Universidad de Salamanca OR

Una vez definidos los criterios de inclusión y exclusión se define un intervalo de fechas para el cribado de los artículos que abarca desde el año 2005 al año 2022 para poder comprender sobre todas las investigaciones relacionadas con ROS 1. El ultimo paso es la selección de unas palabras claves que nos ayuden a realizar las búsquedas de artículos en las diferentes bases de datos, para ello se establecen las siguientes palabras: *ROS, ROS 1, ROS 2, middleware, mapping, LIDAR, camera, navigation, path plannig, SLAM, algorithm, planning* con las que elaboraremos las siguientes consultas:

- **IEEE Xplore:** (Abstract:ROS OR Abstract:ROS 1 OR Abstract:ROS 2 OR Abstract:middleware) AND (Abstract:mapping OR Abstract:LIDAR OR Abstract:camera) AND (Abstract:navigation OR Abstract:path planning ORSLAM) AND (Abstract:algorithm OR Abstract:planning)
- **Scopus:** TITLE-ABS-KEY(ROS OR ROS 1 OR ROS 2 OR middleware) AND TITLE-ABS-KEY(mapping OR LIDAR OR camera) AND TITLE-ABS-KEY(navigation OR path planning OR SLAM) AND TITLE-ABS-KEY(algorithm OR planning) AND PUBYEAR >2004 AND PUBYEAR <2024

Tras haber realizado las consultas anteriores y obtener todos los artículos en función de las palabras claves se han obtenido 798 artículos en total, 140 de IEEE Xplore y 658 artículos de Scopus.

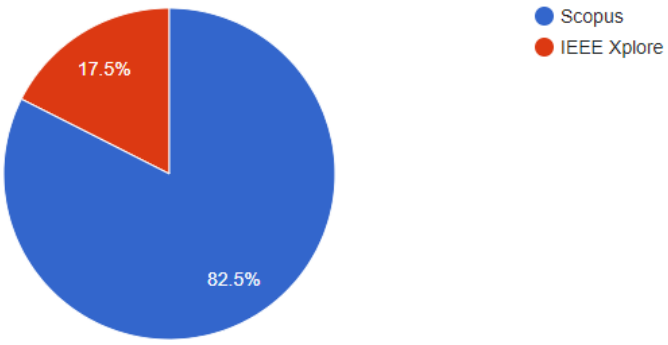

Figura 4: Artículos obtenidos en la segunda revisión de la literatura

Al igual que se realizo anteriormente no todos los artículos van a ser útiles, por lo tanto se debe hacer un cribado , para ello se ha usado la metodología PRISMA que se observa en la figura 5.

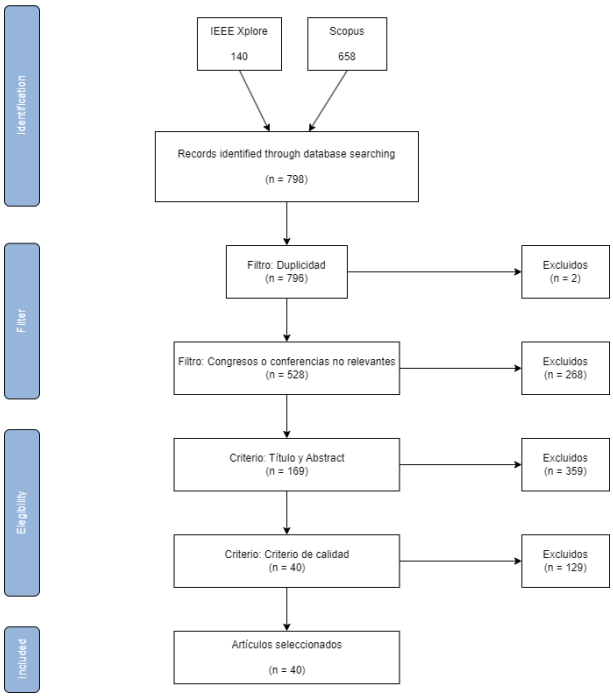

Figura 5: Metodología PRISMA utilizada en la segunda revisión de la literatura

Tras aplicar los mismos filtros que en la anterior revisión de la literatura, se pasa de 798 artículos en total a 40 artículos que han sido aceptados. Se han realizado los filtrados de duplicidad y los filtros que tienen que ver con los criterios de exclusión

e inclusión, además del criterio de calidad establecido en función de las mismas preguntas que se establecieron en la anterior revisión de la literatura.

3.2.2. Origen de ROS 1 y ROS 2

ROS (Robot Operating System) es un framework de código abierto diseñado para facilitar el desarrollo de software para robots. Aunque ROS es utilizado en numerosos proyectos robóticos en la actualidad, tiene dos versiones principales: ROS 1 y ROS 2 [17], cada una con su propio origen y enfoque.

ROS 1, la versión inicial de ROS, fue desarrollado por el laboratorio de inteligencia artificial de Stanford en 2007 como parte del proyecto Stanford AI Robot (STAIR). Inicialmente, ROS se centró en proporcionar una arquitectura flexible y modular para la investigación en robótica, con énfasis en la reutilización de código y la colaboración entre investigadores. A lo largo de los años, ROS 1 ganó popularidad en la comunidad robótica debido a su capacidad para integrar diferentes componentes y facilitar la programación y el control de robots.

Durante el tiempo que estuvo en uso, se encontraron ciertas limitaciones como problemas de rendimiento, escalabilidad y confiabilidad [18] [19]. Esto llevo a que surgiera el desarrollo de su segunda versión ROS 2, una nueva versión iniciada en 2014 con el objetivo de abordar esos problemas. ROS 2 nace como una reescritura completa [20] de ROS 1, por lo que rehace su arquitectura de una manera mucho más modular y distribuida de manera que ofrece un mayor rendimiento e interoperabilidad.

Actualmente, tanto ROS 1 como ROS 2 están activos y en desarrollo. ROS 1 sigue siendo ampliamente utilizado en numerosos proyectos robóticos, mientras que ROS 2 se está adoptando gradualmente para aplicaciones más exigentes en términos de rendimiento, confiabilidad y requisitos industriales. La transición de ROS 1 a ROS 2 es un proceso gradual, y se proporcionan herramientas y puentes para facilitar la migración de proyectos existentes.

3.2.3. Otras alternativas al framework ROS

Por supuesto ROS aunque se ha vuelto muy popular en los últimos años para el desarrollo en robótica no es el único framework para que permiten a los usuarios controlar y programar un robot móvil. Existen alternativas como Microsoft Robotics Developer Studio (MRDS), OROCOS (Open Robot Control Software), YARP (Yet Another Robot Platform), Player Project e incluso algunas más, todas ellas de una forma similar a ROS ofrecen herramientas, bibliotecas o paquetes que nos permiten controlar una plataforma robótica. A continuación describiré algunas de ellas para compararlas con el marco de trabajo ROS que estamos estudiando:

Microsoft Robotics Developer Studio (MRDS): MRDS [21] es un entorno de desarrollo de software basado en Windows para la creación de aplicaciones robóticas. Proporciona herramientas y bibliotecas para el control de robots, la simulación

y la programación visual. A diferencia de ROS, este framework está destinado a la creación de aplicaciones robóticas únicamente en el entorno de Windows, no como ROS que permite ser usado en diferentes sistemas operativos. No solo eso sino que estamos ante una herramienta de Microsoft por lo que puede estar sujeta a licencias y restricciones de un equipo Windows.

OROCOS (Open Robot Control Software): OROCOS [22] es un marco de trabajo de código abierto al igual que ROS, que esta destinado al control de máquinas y robots en tiempo real utilizando el lenguaje de C++. La principal diferencia que encontramos de OROCOS con ROS [23], es que está demasiado enfocado al control en tiempo real y ROS está destinado a proporcionar una arquitectura completa para el desarrollo de aplicaciones robóticas, abarcando no solo el control en tiempo real, sino también la percepción, la planificación y la comunicación entre componentes.

YARP (Yet Another Robot Platform): YARP [24] es un marco de trabajo que es bastante diferente a los anteriores que se acercaban en cierta medida a ROS, y es que este está especialmente diseñado para la construcción de sistemas robóticos distribuidos. YARP se encarga de realizar una comunicación entre componentes y de proporcionar una arquitectura flexible, mientras que ROS se encarga de proporcionar una arquitectura más completa y abarca una gama más amplia de funcionalidades.

3.2.4. Técnicas de escaneo

Las técnicas de escaneo en ROS, como SLAM [25], escaneo láser [26], escaneo RGB-D [27] y fusionado de nubes de puntos, son fundamentales en la robótica y la percepción del entorno. Estas técnicas permiten a los robots construir mapas tanto bidimensionales como tridimensionales precisos de los interiores, lo que es crucial para la navegación autónoma, la planificación de trayectorias y la interacción segura con el entorno. Los datos capturados por los sensores son procesados y fusionados para crear una representación espacial del entorno en tiempo real. Esto proporciona información vital para la toma de decisiones y la ejecución de tareas complejas en entornos dinámicos.

Las técnicas de escaneo en ROS son importantes porque permiten a los robots comprender su entorno, evitar obstáculos, planificar rutas óptimas y realizar tareas de manera autónoma y eficiente. Entre las técnicas más relevantes de escaneo encontramos las siguientes:

- **SLAM (Simultaneous Localization and Mapping):** SLAM es una técnica esencial en robótica y que permite a cualquier robot móvil construir un mapa del entorno a la vez que se localiza mediante los puntos que está escaneando. En ROS se puede realizar un escaneo 2D o 3D, todo dependerá de los recursos hardware que se utilicen para ello. Además, encontramos diferentes paquetes disponibles para SLAM como son GMapping, Hector SLAM, Cartographer, SLAM Toolbox y otros más. Estos paquetes utilizan sensores como láseres (LiDAR), cámaras y odometría para estimar la posición del robot y construir un mapa.

- **Escaneo 2D:** A diferencia de la técnica de escaneo anterior, podemos encontrar técnicas que nos permitan construir un mapa bidimensional sin necesidad de que el robot sea capaz de localizarse. Para el escaneo 2D de interiores es muy común el uso de sensores infrarrojos o láser como los populares LIDAR A1 o Hokuyo.

- **Escaneo 3D:** Los sensores de escaneo 3D también conocidos como lidars 3D, pueden proporcionar información tridimensional del entorno. Utilizando sensores como láseres 3D, se registran múltiples puntos en el espacio para obtener una nube de puntos tridimensional que representa la forma y la geometría del objeto o entorno escaneado. Los sensores láser 3D emiten haces de luz y miden el tiempo que tarda en reflejarse en las superficies, para luego procesar esos datos y poder crear modelos tridimensionales detallados y precisos.

- **Escaneo RGB-D:** Las técnicas de escaneo que se basan en sensores RGB-D, como la Kinect de Microsoft, el Intel RealSense o la Orbbec Astra [28] proporcionan información de color y profundidad. En ROS, existen paquetes como RGBDSLAM y RTABMap [29] que utilizan sensores para realizar el escaneo 3D del entorno. El escaneo RGB-D se basa en el uso de sensores que combinan una cámara RGB (que captura información de color) y un sensor de profundidad (generalmente un sensor infrarrojo). La información de profundidad capturada permite estimar la distancia de los objetos con respecto al sensor y construir un mapa tridimensional del entorno. Estás técnicas de escaneo no son iguales que las de escaneo 3D aunque puedan parecerse, la diferencia principal es que estás, utilizan la profundidad recogida por la cámara y las técnicas de escaneo 3D miden el tiempo que tarda la luz en reflejarse.

3.2.5. Visión, lidars y cámaras de profundidad

Anteriormente se han nombrado varias de las técnicas de escaneo más usadas en el marco de trabajo ROS y en la robótica móvil. Sin embargo, métricas como la precisión en un escaneo interior son muy dependientes de que tipo de sensores se utilizan. Es por ello que se va a describir algunos de los sensores necesarios para poder hacer un escaneo bidimensional o tridimensional, con el objetivo de tener claro que resultado nos pueden dar para los posteriores casos de estudio en el que se utilizarán algunos de ellos. A continuación, se describirán algunos de los sensores comúnmente utilizados para realizar escaneo bidimensional o tridimensional, incluyendo sus características y el tipo de información que pueden proporcionar:

Lidar A1: El lidar A1 es un sensor láser 2D que es muy utilizado en la robótica y en aplicaciones de escaneo. Este sensor utiliza un láser que emite haces de luz y mide el tiempo que tarda la luz en reflejarse en los objetos cercanos. Con la información que recibe el lidar es capaz de determinar la ubicación de planos y objetos en un espacio interior o exterior. No es uno de los lidars más preciso pero su precisión y rango de detección lo hacen perfecto para la generación de mapas y la detección de obstáculos.

Lidar Hokuyo: El lidar Hokuyo al igual que lidar A1 es un sensor que permite un escaneo 2D. La diferencia entre el lidar A1 y el lidar Hokuyo es que el primero trabaja en el rango del infrarrojo y este trabaja con luz láser entre el espectro visible e infrarrojo.Sin embargo, difieren en cuanto a las especificaciones técnicas, como el rango de detección, la resolución, la velocidad de escaneo y la precisión, lo que puede hacerlos más adecuados para diferentes aplicaciones.

Cámara Orbbec Astra: Este tipo de cámaras RGB-D combinan una cámara de color RGB con un sensor de profundidad. Este sensor de profundidad nos permite medir la distancia a los objetos y en función de ello generar un mapa de profundidad. La combinación de los dos sensores permite tomar capturas de imágenes en color y obtener información tridimensional del entorno, dando lugar así a aplicaciones como el reconocimiento de gestos, la realidad aumentada y la navegación autónoma como mencionaremos más tarde.

Cámara Intel RealSense: La cámara Intel RealSense [30] no es más que otra de las cámaras RGB-D que ofrece capacidades de escaneo tridimensional, que al igual que la Orbbec Astra [30] solo se diferencias en cuanto a sus especificaciones técnicas, pero ambas se pueden aplicar a contextos similares.

En concreto la tecnología de escaneo SLAM [31] es una de las más utilizadas y usadas en el entorno de ROS debido a que proporciona un marco de trabajo ideal para su implementación gracias a su flexibilidad, compatibilidad con diferentes sensores, comunidad activa y recursos disponibles. Esto ha llevado a que SLAM sea ampliamente utilizado y desarrollado en el ecosistema de ROS.

3.2.6. Navegación autónoma

La navegación autónoma en ROS es la capacidad fundamental para permitir que cualquier tipo de robot móvil se desplace de forma libre, es decir sin ser teleoperado, en un entorno desconocido. La navegación no es una función de un robot móvil por si solo si no que se basa en la combinación de sensores como los nombrados anteriormente, así como de los algoritmos de percepción y planificación de rutas, y control de movimiento para permitir que los robots naveguen de manera segura y eficiente.

Es importante destacar que la navegación autónoma es un concepto general para cualquier robot móvil, sin embargo, si se habla de ROS, tenemos que tener en cuenta que las dos versiones presentan diferencias clave en cuanto a la navegación autónoma:

La primera diferencia, es la más clara cuando se trabaja con las diferentes versiones y es que ROS 1 y ROS 2 tiene distinta arquitectura de comunicación. ROS 1 utiliza un sistema de comunicación basado en mensajes llamado ROS Topic"que utiliza un patrón de publicación/suscripción. ROS 2, por otro lado, introduce una arquitectura de comunicación más flexible y escalable llamada ROS 2 Middleware"[32] que incluye tanto el patrón de publicación/suscripción como el de servicio/respuesta. Esto lo podemos ver más claramente en la figura 6.

No solo se diferencian en cuanto a comunicación estas dos versiones sino tam-

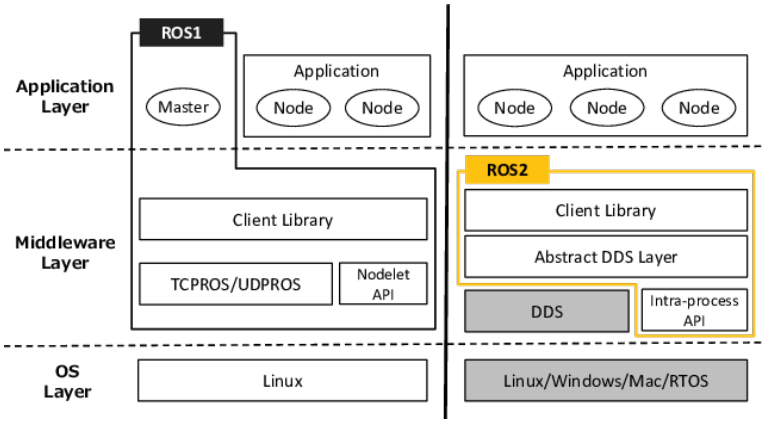

Figura 6: Arquitecturas de comunicación en las diferentes versiones de ROS

bién en cuanto a lenguajes de codificación, en ROS 1 se admite principalmente la programación en C++ y Python, mientras que ROS 2, se centra más en Python y otros lenguajes como Java. [33]

Al igual que ya he mencionado anteriormente ROS 2 esta más orientado a la escalabilidad por lo que permite que el rendimiento sea mayor y la gestión de recursos sea menor.

En resumen, se puede concluir que como se hace referencia en algunas investigaciones [34] [35], que ROS 2 ofrece una mejor comunicación, una mejor escalabilidad, un mayor rendimiento y una mejora de la confiabilidad que no puede ofrecer ROS 1.

3.2.7. Tipos de navegación

En la robótica y no solo en ROS, existen varios tipos de navegación que los robots móviles pueden utilizar para desplazarse de un punto a otro dependiendo de diversos factores, es por ello que pueden haber varios tipos.

La navegación de un robot móvil estima su complejidad en función de la cantidad de información de su entorno. Normalmente el entorno en el que se desplaza un robot móvil no es estático ya que siempre se produce algún cambio, por lo que consideramos que su entorno es dinámico, de esta forma podemos considerar dos tipos de navegación:

Navegación reactiva: La navegación reactiva como indica su nombre se basa en las respuesta directas de lo que percibe un robot en su entorno, sin tener un conocimiento previo como es un mapa. Este tipo de navegación consiste en la toma de decisiones de movimiento en función de la información que captan en tiempo real los sensores [36], por lo que permiten respuestas rápidas a cambios producidos en el entorno. Por tanto son robots que tienen una gran autonomía en un entorno pero que carecen de un plan establecido, se centran en lograr objetivos específicos, como alcanzar un punto de interés o explorar un área determinada. Su misión se enfoca en tomar decisiones inmediatas y adaptativas para lograr esos objetivos en función

de las condiciones presentes.

Navegación deliberativa: La navegación deliberativa a diferencia de la navegación reactiva se basa en la planificación de alto nivel y en la toma de decisiones anticipadas para lograr un objetivo de navegación. Esta se centra en una planificación mucho mayor a la anterior, una visión más amplía teniendo en cuenta restricciones, objetivos y preferencias.

Uno de los factores que hacen la navegación deliberativa totalmente distinta de la reactiva, es que está incorpora lo que se denomina planificación de trayectorias [37], que permiten la generación de una serie de acciones que permiten desplazarse al robot desde un punto de origen hasta su destino. Estos algoritmos pueden tener en cuenta factores como la eficiencia de la ruta, la evitación de obstáculos, la optimización de recursos o cualquier otra restricción específica. La navegación deliberativa es un tipo de navegación que puede ser muy útil cuando la navegación reactiva no es suficiente en un entorno dinámico.

Si bien la negación reactiva o deliberativa son unos tipos muy generales que pueden englobar tipos de navegación más específicos en el ámbito de ROS y más concretamente en la librería de ROS *navigation* encontramos otros dos tipos de navegación muy usados en todos los desarrollos y referente a el coste de la navegación en un mapa. Estos dos tipos de navegación son la navegación global y la local [38] que están asociadas con algoritmos de planificación como los mapas de coste.

La navegación global, es también conocida con el nombre de planificación global, está navegación se encarga de trazar una ruta desde el origen del robot hasta el destino que se haya seleccionado y tiene en cuenta todo su entorno, por ello normalmente es un tipo de navegación que se utiliza cuando tenemos información de un entorno como puede ser un mapa. Para esta navegación se utilizan algoritmos de planificación de trayectorias como se comentará posteriormente como son el A* (A estrella) o Dijkistra, para calcular una ruta óptima en función de la información que tienen.

La navegación local en cambio no tiene en cuenta el todo sino una porción pequeña de ello, se centra en el control del movimiento y la evitación de obstáculos en un rango de conocimiento pequeño o cercano, es decir local.

3.2.8. Planificación de trayectorias

La planificación de trayectorias tiene que ver con la capacidad que tiene un robot móvil de trazar una trayectoria desde un origen a un destino, evitando los obstáculos que interfieren la llegada a ese destino y en la que se dispone de una información adicional del entorno y de donde se encuentran algunos de los obstáculos estáticos [39].

En el contexto de ROS (Robot Operating System), la planificación de trayectorias se refiere a la generación de una secuencia de movimientos que permiten al robot desplazarse desde su posición actual hacia un destino deseado, evitando obstáculos

en su entorno.

Al generar una trayectoria, los algoritmos de planificación de trayectorias en ROS pueden utilizar diferentes enfoques, como algoritmos basados en métodos de búsqueda (como Dijkstra y A*), grafos (como RRT y PRM) u optimización (como CHOMP). A continuación, en el siguiente apartado se detallarán los métodos o algoritmos de planificación de trayectorias mencionados.

3.2.9. Algoritmos de planificación

Para realizar una planificación tenemos distintos tipos de algoritmos, algoritmos basados en grafos, algoritmos basados en búsquedas u algoritmos de optimización. Dependiendo de la complejidad del entorno, el conocimiento que tengamos de él y las prioridades y restricciones especificas será más oportuno utilizar unos u otros.

Algoritmos basados en métodos de búsqueda

Entre los algoritmos de planificación basados en grafos más comunes encontramos los algoritmos A* (A estrella) y Dijkstra [40]. El algoritmo A* es un algoritmo de búsqueda que trata de buscar la ruta más corta entre el nodo inicial (origen del robot móvil) y el nodo objetivo en un grafo ponderado. Este algoritmo utiliza una estrategia o heurística de estimación del coste, de tal forma que trabaja con dos lista, una en la que guarda los nodos ya explorados y otra en la que guarda los nodos candidatos a ser la ruta con menor coste y de esta forma construye el grafo o recorrido. A diferencia del algoritmo Dijkstra que es un algoritmo de búsqueda no informada, por lo que no utiliza una función heurística sino que explora todos los nodos y tras haberlos explorado todos es capaz de encontrar la ruta más corta a cada nodo.

Algoritmos basados en grafos

Los algoritmos basados en grafos tienen bastante relación con los anteriores, entre ellos encontramos el algoritmo PRM (Probabilistic Roadmap) [41] es un algoritmo de planificación de trayectorias que construye un grafo de probabilidades basado en el espacio de configuración del robot móvil [42]. Este algoritmo genera nodos aleatorios en el espacio, se crean más o menos dependiendo de la complejidad de la trayectoria y de la cantidad de obstáculos, y en función de la viabilidad de las transiciones entre nodos genera un grafo. Este tipo de algoritmos pueden ser optimizados utilizando algoritmos como A* o Dijkstra.

Por otro lado, el algoritmo RRT (Rapidly-Exploring Random Tree) es un enfoque de planificación de trayectorias basado en árboles que construye un árbol creciendo aleatoriamente en el espacio de configuración del robot. Comienza con un nodo inicial y, en cada iteración, genera una muestra aleatoria en el espacio y expande el árbol hacia la muestra más cercana. El algoritmo RRT busca rápidamente explorar el espacio de configuración [43] [41] y se detiene cuando se alcanza un nodo objetivo o se cumple algún criterio de finalización. Las trayectorias pueden ser obtenidas recorriendo el árbol desde el nodo objetivo hasta el nodo inicial. El algoritmo RRT es especialmente útil para espacios de configuración de alta dimensionalidad y entornos

con obstáculos complejos.

Algoritmos basados en la optimización

El algoritmo CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [44] consiste en encontrar una ruta con una trayectoria sin demasiados cambios y libre de colisiones que minimice la función de coste global. Para lograr este objetivo se divide la trayectoria del robot móvil en puntos de control, a los que se les añade un campo potencial de colisión que evalúa la proximidad del punto con las colisiones cercanas [45]. De tal manera que con varias iteraciones basadas en gradientes (medida de dirección y magnitud) se ajustan las posiciones y los tiempos de los puntos de control intentando reducir al máximo el coste de la trayectoria y evitando las colisiones.

3.2.10. Planificadores

En la robótica existen una gran cantidad de algoritmos que nos permiten planificar la trayectoria de un robot móvil o de cualquier otro tipo de robots como los articulados. Al haber una gran cantidad de algoritmos de planificación no siempre se sabe con certeza cual se debe utilizar en cada situación, es por ello que existen herramientas denominadas planificadores que permiten realizar pruebas simuladas de estos algoritmos antes de reproducirlo en la realidad. Estas plataformas brindan una interfaz amigable y una amplia gama de funciones para utilizar y combinar los algoritmos de planificación de manera eficiente, simplificando así el proceso de desarrollo de aplicaciones robóticas y permitiendo a los robots navegar de manera autónoma y precisa en entornos diversos. Algunas de ellas son:

MoveIt!: Es un paquete de ROS diseñado específicamente para la planificación de movimiento de brazos robóticos y manipuladores [46]. Esta herramienta permite la planificación de trayectorias tanto en manipuladores como en robótica móvil e incluso permite la creación de modelos robóticos propios para realizar su planificación de trayectorias. Además, permite la reproducción de esa simulación en la realidad y un control de colisiones.

OMPL (Open Motion Planning Library): Es una biblioteca de planificación [47] de movimiento de código abierto ampliamente utilizada en ROS. A diferencia de la anterior no es considerada una herramienta como tal, pero es igual o incluso más potente que MoveIt, además de que está integrada en la herramienta MoveIt [48] como tal. Asimismo, OMPL no solo se enfoca en encontrar una trayectoria válida, sino que también ofrece capacidades de optimización. Los usuarios pueden especificar funciones objetivo y restricciones adicionales para refinar las trayectorias generadas. Esto permite la búsqueda de soluciones óptimas o la adaptación en tiempo real a cambios en el entorno.

3.2.11. Localización y escaneo (SLAM)

Anteriormente se ha mencionado algunas técnicas de mapeado que además integran métodos de localización que permiten al robot móvil en todo momento conocer su posición. SLAM [31] es un de los métodos más utilizados y es por ello que se ha usado para el desarrollo de este trabajo fin de máster porque soluciona uno de los problemas más complicados en la robótica móvil que es la localización, problema que ha sido un rompecabezas desde los principios de la robótica.

Básicamente si se toma como ejemplo el movimiento de un roomba, si funcionará sin SLAM se movería de forma descontrolada y sin ningún objetivo, lo que haría ineficiente el objetivo del robot móvil que es limpiar. La tecnología de SLAM en este caso permite al robot móvil conocer información de su entorno como donde se encuentra, por donde ha pasado, las revoluciones de sus ruedas, que no dejan de ser la propia odometría del robot, los obstáculos que hay haciendo que por ejemplo no pase dos veces por una zona por donde ya ha pasado haciendo que su objetivo de limpieza ahora si que sea eficiente. Este ejemplo podemos verlo representado en la siguiente figura:

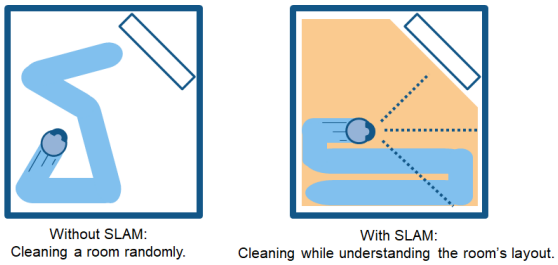

Figura 7: Ejemplo de uso de la tecnología SLAM

Sin embargo, este tipo de tecnología de localización simultanea y mapeado no es perfecta y presenta algunos inconvenientes. El primero de ellos es que por lo general los errores de localización se acumulan y es algo inevitable que produce una desviación de los valores reales, es decir que los puntos recogidos por el robot no coinciden. Este tipo de errores se pueden mitigar utilizando soluciones que optimicen los datos y minimicen los errores.

El segundo problema que nos encontramos cuando realizamos un escaneo que utiliza una técnica de localización simultanea es que puede fallar momentáneamente la localización ya que el mapeo puedo no tener en cuenta las características de movimiento de un robot móvil dando lugar a estimaciones de posición discontinuas.

Para poder realizar un escaneo ROS tiene varios paquetes o herramientas para poder implementar este tipo de métodos. Algunos de los paquetes más usados son Gmapping que es el más usual, Hector SLAM, SLAM ToolBox [49] y Google Cartographer.

SLAM ToolBox es una herramienta de código abierto desarrollada en ROS que proporciona una amplia gama de algoritmos y funciones para la implementación de SLAM en robots. Permite la construcción de mapas 2D y 3D, la estimación

de la posición del robot y la generación de trayectorias. Por otro lado, Gmapping es una librería popular en ROS que se enfoca en el mapeo 2D utilizando el filtro de partículas para la localización y actualización del mapa. Hector SLAM es otra opción para el mapeo 2D que utiliza una representación basada en rejillas para la generación de mapas. Finalmente, Google Cartographer es una herramienta de SLAM desarrollada por Google que se destaca por su capacidad para crear mapas 2D y 3D de alta calidad [50], además de ofrecer soporte para localización y navegación simultáneas. Estas herramientas son ampliamente utilizadas en la comunidad de robótica y proporcionan opciones flexibles y potentes para implementar SLAM en una variedad de aplicaciones robóticas.

3.2.12. Métricas para los algoritmos

En la evaluación y comparación de algoritmos de planificación de trayectorias, se utilizan una variedad de métricas para medir su rendimiento y eficacia en la resolución de problemas de navegación autónoma. Estas métricas proporcionan una base objetiva para evaluar y seleccionar el algoritmo más adecuado para una aplicación robótica específica, ya que cada uno tiene unas ventajas y unas desventajas. Algunas de las métricas más comunes incluyen el tiempo de planificación, la calidad de la trayectoria generada, la completitud y corrección y la eficiencia energética.

El **tiempo de planificación** es una de las métricas esenciales a la hora de comparar algoritmos, ya que nos permite diferenciarlos en función del cual de ellos es el algoritmo que tiene el menor tiempo de planificación y que por lo tanto es el más eficiente. Sin embargo, es importante equilibrar el tiempo de planificación con la calidad de la trayectoria generada.

Otra de las métricas que también es primordial es la **calidad de la trayectoria o la suavidad** de la misma [51], ya que este tipo de métrica evalúa la optimización de un ruta. Una ruta suave es una trayectoria donde la distancia recorrida es la mínima pero teniendo en cuenta que no se producen movimientos innecesarios o redundantes.

La **completitud y corrección** son métricas que evalúan la capacidad del algoritmo para encontrar soluciones válidas [51] y evitar colisiones. Un algoritmo de planificación completo debería encontrar una solución si existe, mientras que la corrección se refiere a la capacidad de evitar colisiones y violaciones de restricciones. Estas métricas son cruciales para garantizar la seguridad y la confiabilidad del robot en su entorno operativo.

La **eficiencia energética** [52] es una métrica adicional que se vuelve cada vez más relevante en aplicaciones robóticas. Los algoritmos de planificación de trayectorias que minimizan el consumo de energía son preferibles, ya que permiten una mayor autonomía y eficiencia en el uso de los recursos energéticos disponibles.

En resumen es importante tener en cuenta que métrica se ha de usar en cada momento y dependiendo de los requisitos de un escenario. En función de eso, se debe realizar una evaluación detallista del entorno, las restricciones, las características del

robot y el objetivo que se quiere lograr para seleccionar un algoritmo de planificación de trayectorias adecuado en el contexto en el que se realiza.

4. Casos de estudio y experimentación

Tras la investigación realizada anteriormente en el estado del arte, se procede a definir los diferentes sistemas que se van a ser objeto de estudio. Para esta parte de la experimentación es importante resaltar los recursos hardware que se ha utilizado para la investigación. Durante la misma se han utilizado tanto recursos físicos como recursos preparados para simulaciones y que son los siguientes:

- Robot móvil Turtlebot 2, formado por una plataforma Kobuki y una cámara de profundidad Orbbec astra.
- Robot prototipo creado a partir de un hoverboard
- Robot móvil Turtlebot 3 (burger) simulado
- Roomba iCreate 3 con lidar A1

4.1. Conceptos básicos sobre ROS

Antes de sumergirse en la implementación y entrega de proyectos basados en ROS (Robot Operating System), es crucial tener una comprensión clara de los conceptos básicos de ROS, tanto en su versión 1 como en su versión 2. Estos conceptos fundamentales sientan las bases para el desarrollo exitoso de aplicaciones robóticas utilizando esta plataforma.

En primer lugar, es esencial comprender la arquitectura de ROS. Tanto en ROS 1 como en ROS 2, se utiliza una arquitectura distribuida y modular que se basa en la comunicación entre nodos. Los **nodos** son los componentes fundamentales de ROS y representan procesos individuales que realizan tareas específicas. Comunicarse entre nodos se realiza a través de tópicos, servicios y acciones, que permiten la transmisión de datos y comandos entre los diferentes componentes del sistema.

Los **tópicos** son canales de comunicación asincrónica utilizados para intercambiar mensajes entre nodos. Un nodo puede publicar mensajes en un tópico, mientras que otros nodos pueden suscribirse a ese tópico para recibir los mensajes. Esta arquitectura de publicación/suscripción es fundamental para el intercambio de datos en ROS.

Además de los tópicos, los **servicios** permiten la comunicación entre nodos de manera sincrónica. Un nodo puede ofrecer un servicio y otros nodos pueden llamar a ese servicio para solicitar una tarea o una respuesta específica. Esto es especialmente útil cuando se requiere una interacción más compleja entre los componentes del sistema.

En cuanto a las diferencias entre ROS 1 y ROS 2, es importante tener en cuenta las mejoras que ROS 2 ha introducido en comparación con su predecesor. ROS 2 se ha desarrollado para ser más escalable y adecuado para sistemas más grandes y

complejos, como robots multirobot y sistemas distribuidos a gran escala. También ha mejorado en términos de rendimiento, flexibilidad y robustez.

Una de las principales diferencias radica en el **middleware** utilizado para la comunicación. Mientras que ROS 1 utiliza su propio middleware basado en publicación/suscripción y servicios, ROS 2 se basa en el estándar Data Distribution Service (DDS). DDS proporciona una mayor flexibilidad y rendimiento, y también ofrece una mejor gestión de la calidad de servicio.

Además, ROS 2 tiene un enfoque multiplataforma más sólido, con soporte nativo para múltiples lenguajes de programación, como C++, Python y otros, a través de la especificación de interfaces (ROS 2 Interfaces). Esto permite una mayor flexibilidad en la elección del lenguaje de programación y facilita la interoperabilidad entre diferentes componentes del sistema.

Es por ello que antes de embarcarse en la implementación y entrega de proyectos basados en ROS, es esencial tener una sólida comprensión de los conceptos básicos de ROS en sus dos versiones. Comprender la arquitectura distribuida, los conceptos de nodos y la comunicación a través de tópicos, servicios y acciones es fundamental.

4.2. Principales diferencias de comunicación

Una de las diferencias clave entre ROS 1 y ROS 2 es cómo manejan la comunicación de nodos y si utilizan o no un nodo maestro (master node) para facilitar esta comunicación.

En ROS 1, se utiliza un **nodo maestro** (llamado roscore) para gestionar la comunicación entre los nodos. El nodo maestro actúa como intermediario centralizado, proporcionando servicios de registro y búsqueda de nodos, así como la gestión de la comunicación a través de tópicos y servicios. Los nodos deben registrarse con el nodo maestro para que otros nodos puedan descubrirlos y comunicarse con ellos.

En contraste, ROS 2 adopta un enfoque descentralizado para la comunicación de nodos y no requiere un nodo maestro. En su lugar, utiliza un sistema basado en el estándar **Data Distribution Service (DDS)**, que permite una comunicación punto a punto entre los nodos directamente, sin necesidad de intermediarios centrales. Cada nodo en ROS 2 es capaz de descubrir y comunicarse con otros nodos sin depender de un nodo maestro como se había mencionado anteriormente.

Este cambio en el enfoque de comunicación tiene varias implicaciones. Por un lado, eliminar el nodo maestro en ROS 2 reduce la dependencia de un único punto de falla y mejora la escalabilidad del sistema. Además, la comunicación punto a punto proporcionada por DDS puede ofrecer un mejor rendimiento y latencia reducida en comparación con la comunicación a través del nodo maestro en ROS 1.

Sin embargo, es importante tener en cuenta que, a pesar de la eliminación del nodo maestro en ROS 2, aún existen mecanismos para el descubrimiento de nodos y la gestión de la comunicación. ROS 2 utiliza un sistema llamado Domain Name System (DNS) para el descubrimiento automático de nodos en una red, lo que permite que

los nodos se encuentren entre sí sin la necesidad de un nodo maestro centralizado.

4.3. Análisis del hardware utilizado

Al usar ciertos componentes o recursos hardware en este proyecto, se ha considera importante realizar un análisis del hardware utilizado. Esta sección se centra en proporcionar una descripción concisa de los componentes de hardware utilizados en el proyecto, destacando sus especificaciones técnicas y características relevantes. El conocimiento detallado de estos componentes es fundamental para establecer las bases necesarias que permitirán una comprensión profunda de las etapas posteriores del estudio. A través de este análisis, se sentarán los cimientos para una evaluación rigurosa y una interpretación adecuada de los resultados obtenidos en el contexto del sistema de hardware implementado.

4.3.1. Plataforma kobuki

La plataforma robótica Kobuki es un sistema de hardware diseñado para facilitar el desarrollo de robots móviles de forma rápida y sencilla. Desarrollada por la compañía Yujin Robot, la plataforma Kobuki se ha vuelto popular en la comunidad de robótica debido a sus características versátiles y su fácil integración con diferentes aplicaciones.

Esta plataforma robótica esta equipada con ruedas omnidireccionales de tal forma que es considerada como robot holonómo, esto quiere que le permiten un movimiento en todas las direcciones sin girar previamente. También al igual que otras plataformas comerciales similares incluye múltiples sensores como infrarrojos y sensores de caída que permiten que este tipo de plataformas eviten los obstáculos y caídas.

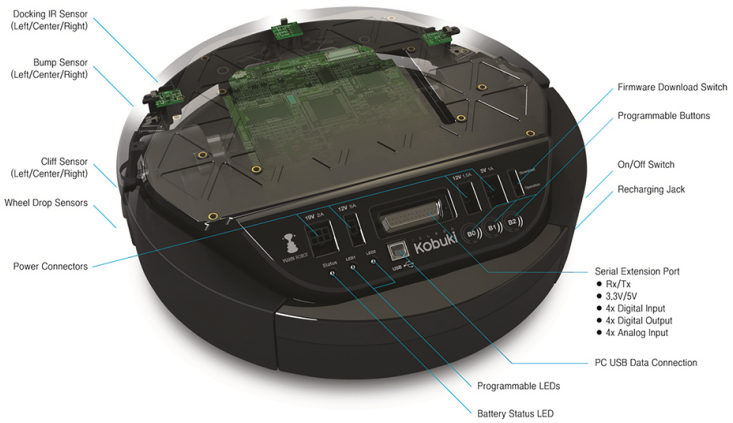

Figura 8: Composición de la plataforma Kobuki

Kobuki, a su vez, es compatible con ROS (Robot Operating System), lo que facilita la programación, el control y la integración del robot en sistemas más complejos

utilizando las herramientas y bibliotecas de ROS.

4.3.2. Hoverboard

Para este proyecto también se ha explorado el uso de un Hoverboard como una opción de bajo costo para crear un robot autonomo integrado en ROS. a elección de utilizar un Hoverboard como plataforma se basa en varias ventajas y características clave que ofrece. En las siguientes figuras 9 10 se puede ver la plataforma construida con la placa y los dos motores:

Figura 9: Prototipo de robot móvil con placa de Hoverboard

Figura 10: Prototipo de robot móvil con placa de Hoverboard (LIDAR A1)

En primer lugar, la utilización de la placa controladora y los motores del Hoverboard ha sido posible gracias a un proceso de ingeniería inversa que ha permitido desbloquear y acceder a su funcionalidad. Esto ha brindado la oportunidad de aprovechar la tecnología existente en el Hoverboard y utilizarla para la creación de un robot autónomo. La placa controladora del Hoverboard puede proporcionar la capacidad necesaria para realizar tareas como odometría, control de motores y comunicación con otros dispositivos, permitiendo la integración sin problemas en el entorno de ROS.

Otra ventaja significativa del uso de un Hoverboard es su bajo costo en comparación con otras plataformas robóticas disponibles en el mercado. Esto hace que sea accesible para proyectos con presupuestos limitados o académicos, sin comprometer la funcionalidad y el rendimiento requeridos para un robot autónomo.

Al aprovechar la funcionalidad de los motores del Hoverboard, es posible lograr un movimiento ágil y preciso del robot. Estos motores, una vez desbloqueados, pueden ser controlados y programados como cualquier otro tipo de motores utilizados en robótica, lo que proporciona una base sólida para la navegación del robot autónomo.

La integración del Hoverboard con ROS también brinda una serie de beneficios. ROS es una plataforma ampliamente utilizada y cuenta con una amplia gama de bibliotecas, herramientas y recursos que facilitan la programación, el control y la

integración de componentes robóticos. Al utilizar ROS, se pueden aprovechar estas capacidades existentes para agilizar el desarrollo y aumentar la eficiencia del proyecto.

4.3.3. Roomba iCreate 3

En esta sección de análisis de hardware, se explora la plataforma robótica Roomba iCreate 3 como una opción destacada para el desarrollo de robots autónomos. La Roomba iCreate 3 es una plataforma diseñada y fabricada por iRobot, reconocida por su línea de robots de limpieza doméstica.

La Roomba iCreate 3 se ha convertido en una elección popular debido a sus características y capacidades versátiles. Además, la Roomba iCreate 3 viene equipada con una serie de sensores integrados, como sensores de infrarrojos y de choque, que permiten la detección de obstáculos y la navegación inteligente en diferentes entornos.

Figura 11: Composición de la Roomba iCreate 3

La elección de utilizar la plataforma Roomba iCreate 3 se basa en que es el componente principal para el desarrollo de la plataforma Turtlebot 4 que esta integrada en una de las ultimas versiones de ROS 2 y que por lo tanto es importante para el desarrollo de una solución móvil en esta versión.

4.3.4. Sensores externos

En la construcción de un robot móvil no solo influye el medio de movimiento o desplazamiento, sino que es muy importante los sensores que son capaces de recoger la información de su entorno para que el propio robot sea capaz de procesar esa información y de navegar de manera autónoma. Es por ello que junto con las plataformas robóticas anteriores, que algunas de ellas ya se encuentran sensorizadas, se utilizan sensores externos para ampliar las capacidades de percepción y navegación de los robots.

Por lo general se utilizan sensores que captan o escanean la totalidad del entorno, en particular se destacará el uso de la cámara de profundidad Orbbec Astra y el lidar RP A1. La cámara de profundidad Orbbec Astra utiliza tecnología de sensores 3D para capturar información tridimensional del entorno. Esto permite al robot tener una percepción más precisa de los objetos y su posición en el espacio, lo que es especialmente útil en tareas de detección de obstáculos, navegación en interiores y reconocimiento de gestos.

Por otro lado, el lidar RP A1 utiliza láseres para generar un mapa tridimensional del entorno circundante. Este sensor mide la distancia y la ubicación de los objetos con alta precisión, permitiendo al robot mapear y comprender su entorno de manera más completa. El lidar RP A1 es especialmente útil en tareas de mapeo, localización y planificación de rutas, lo que contribuye a una navegación más segura y eficiente.

Figura 12: Cámara Orbbec Astra

Figura 13: RP Lidar A1

Durante la realización de este proyecto, se han llevado a cabo pruebas adicionales con sensores como los LIDAR S1 y A2, sensores TOF (Time of Flight) y cámaras Intel Real Sense. Sin embargo, es importante destacar que estas pruebas se realizaron principalmente por curiosidad y no se han obtenido resultados significativos o suficientes para su implementación en el proyecto. Aunque se exploraron estas opciones, se optó por centrarse en los sensores principales, como la cámara de profundidad Orbbec Astra y el lidar RP A1, que ofrecieron resultados más sólidos y adecuados para los objetivos específicos del proyecto.

4.4. Integración en ROS 1

En este caso para la integración de un robot móvil en ROS 1 se escogió la plataforma turtlebot 2 y un hoverboard. De tal manera que se buscaba integrar Turtlebot 2 con ROS, que con los paquetes que había era sencillo y de integrar un hoveaboard como prototipo de robot móvil con la versión Melodic de ROS 1.

4.4.1. Turtlebot 2

En esta sección, se describirán en detalle los pasos realizados para lograr la integración e instalación de ROS 1 Kinetic en la plataforma Kobuki. Se explorará específicamente el uso de los paquetes del Turtlebot 2, los cuales brindan una amplia gama de funcionalidades para la navegación y el mapeo. Mediante la configuración

y el ajuste de estos paquetes, se busca aprovechar al máximo las capacidades de la plataforma Kobuki.

Una vez completada la integración de ROS 1 Kinetic y los paquetes del Turtlebot 2 en Kobuki, se procederá a realizar el mapeo en 2D y 3D del entorno. Esto se logrará mediante el uso de sensores, como la cámara de profundidad Orbbec Astra, que permitirá capturar la información tridimensional del entorno y generar un mapa preciso y detallado.

Con el mapa generado, se implementará un sistema de navegación basado en el recorrido de puntos. Esto significa que se establecerán puntos de referencia, como la ubicación de una mesa, y se programará al robot para que pueda navegar de un punto a otro de manera autónoma. El sistema de navegación utilizará algoritmos y técnicas adecuadas para calcular las rutas óptimas y evitar obstáculos en el entorno.

4.4.1.1. Instalación

Para llevar a cabo la instalación de ROS 1 Kinetic, es necesario dirigirse a la página de referencia oficial de ROS, donde se proporcionan instrucciones detalladas para su instalación. Una vez completada la instalación de ROS 1 Kinetic, se procede a instalar los paquetes relacionados con el Turtlebot 2. Estos paquetes proporcionan las funcionalidades necesarias para utilizar la plataforma Kobuki y los sensores de forma conjunta. Una vez instalados, la plataforma y los sensores están listos para ser utilizados en el desarrollo de aplicaciones de mapeo y navegación autónoma.

4.4.1.2. Mapeo 2D y 3D

Para poder realizar un escaneo de una habitación en 2D o 3D, se ha utilizado una herramienta de SLAM como las mencionadas anteriormente, para el escaneo en 2D se ha utilizado la herramienta o método de mapping *Gmapping* y para el escaneo 3D la herramienta *RTAB-Map* para generar y visualizar la información tridimensional.

Gmapping es un algoritmo de SLAM para construir un mapa bidimensional, se ha utilizado este algoritmo / herramienta porque nos permite generar el mapa del robot móvil utilizando los datos de odometría de las ruedas y la información recogida por la cámara Orbbec Astra. Estos datos son utilizados para alimentar un filtro de partículas denominado *Rao-Blackwellized* para generar el mapa de cuadrículas de RVIZ. Es importante recordar que aunque la cámara escanea una nube de puntos solo utiliza los datos de profundidad al nivel de la cámara para generar el mapa bidimensional. En la siguiente figura 14 se puede ver como se realiza la generación del mapa:

Dado que nos encontramos en el entorno de desarrollo de ROS 1 y como se ha mencionado anteriormente en los conceptos básicos tenemos que inicializar el nodo maestro para poder realizar la comunicación entre los nodos, esto nos permitirá más tarde que se pueda comunicar los diferentes nodos de odometría y sensores como la cámara directamente con el nodo Gmapping que es el encargado de poner en marcha

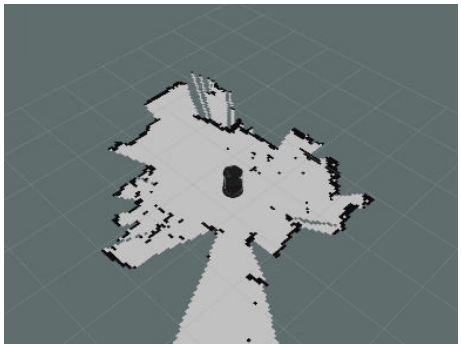

Figura 14: Escaneo 2D con cámara Orbbec Astra

el escaneo con esa información.

Por su puesto uno de los pasos importantes que se deben de dar en estos casos, en los que se integra un robot móvil con diferentes sensores es la configuración inicial que conllevan, algunos de ellos necesitan la instalación de unos drives iniciales o como en el caso de la cámara Orbbec Astra se necesitan configurar las reglas *udev* necesarias para que funcione el dispositivo.

En nuestro caso instalamos las bibliotecas necesarias para que la cámara Astra funcione correctamente, descargamos sus reglas udev del repositorio oficial y las copiamos en el directorio correspondiente.

```
sudo apt install ros-*-rgbd-launch ros-*-libuvc ros-*-libuvc-camera ros-*-libuvc-ros
wget https://raw.githubusercontent.com/orbbec/astra/master/install/orbbec-usb.rules
sudo cp orbbec-usb.rules /etc/udev/rules.d/.
```

Es importante recordar que se deberá realizar un reinicio del servicio que controlar estas reglas para que después de haberla añadido al directorio, la cámara funcione correctamente.

Además de ejecutar *roscore*, que permite la comunicación en ROS, es crucial inicializar los nodos necesarios para el funcionamiento de la plataforma y la cámara. Esto se logra mediante el comando *roslaunch turtlebot_ bringup minimal.launch*, el cual inicia los nodos esenciales del robot Turtlebot. Además, es importante inicializar la herramienta de gmapping para utilizar la odometría y la información de la cámara. Para esto, se utiliza el comando *roslaunch turtlebot_ navigation gmapping_ demo.launch*. Estos pasos son fundamentales para establecer la base de funcionamiento y permitir el mapeo y la navegación del robot.

Todo el proceso descrito anteriormente se puede visualizar y comprender mejor mediante un grafo que muestra la interconexión entre los diferentes nodos en ROS. En este grafo, se puede observar cómo los nodos de la plataforma, la cámara y las herramientas de mapeo y navegación se comunican entre sí a través de los tópicos y servicios correspondientes. Este grafo de la Figura 15 proporciona una representación visual clara de cómo se establecen las conexiones y fluye la información entre los componentes del sistema, permitiendo una comprensión más completa de la comunicación en ROS.

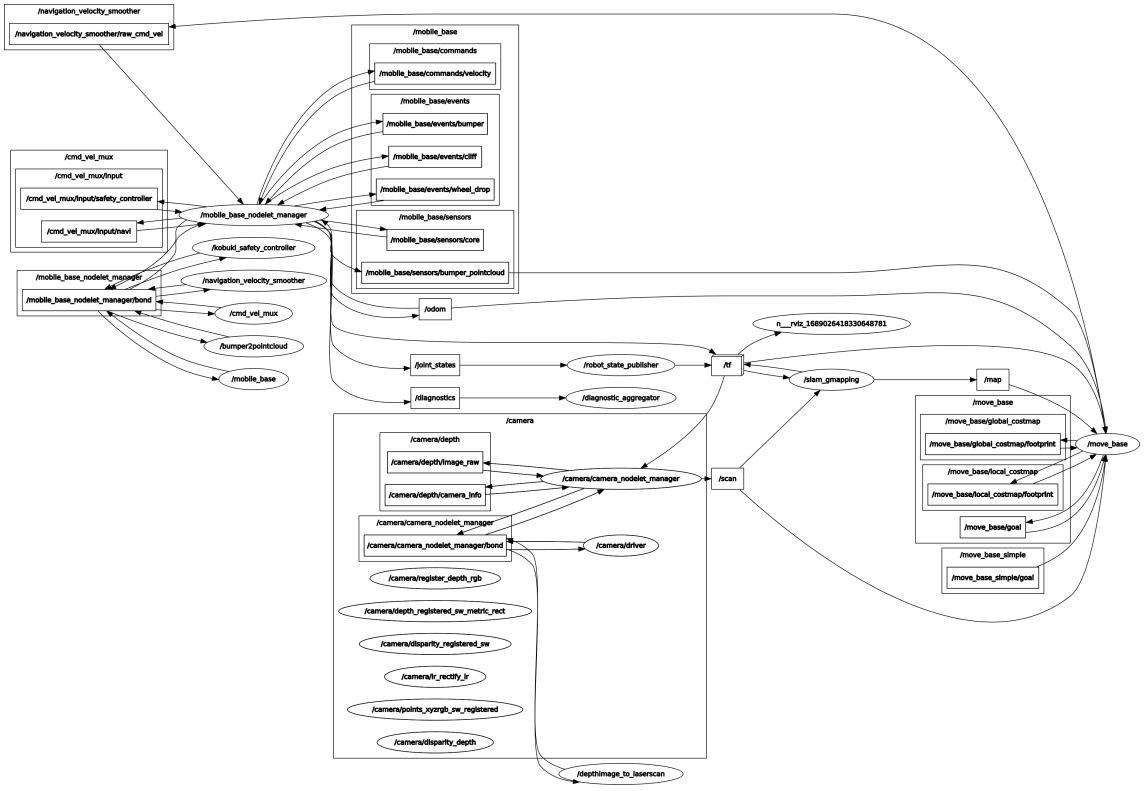

Figura 15: Grafo RQT_GRAPH de los nodos y tópicos ejecutados

La interconexión entre los nodos de los grafos es fundamental para el funcionamiento de varias funcionalidades en ROS. En este caso, podemos observar tres conexiones importantes:

Entre los nodos /**camera** y /**scan**: El nodo /**camera** proporciona información visual, mientras que el nodo /**scan** representa la información del sensor láser. Estos nodos se comunican para integrar datos visuales y de escaneo láser, lo que permite una mejor percepción del entorno y una generación de mapa más precisa en el nodo **Gmapping**.

Entre el nodo /**odom** y el nodo /**move_base**: El nodo /**odom** recopila información de la odometría del robot, como la posición y la orientación. Esta información es crucial para la planificación y navegación del robot. El nodo /**move_base** es responsable de la planificación de rutas y el control del movimiento del robot. La conexión entre estos nodos permite que la odometría se utilice en el proceso de navegación para estimar la posición y el movimiento del robot.

Entre el nodo /**tf** y el nodo **Gmapping**: El nodo /**tf** es responsable de gestionar las transformaciones entre los diferentes marcos de referencia (como el marco base del robot, el marco de la cámara, etc.). En este caso, se conecta con el nodo Gmapping para proporcionar las transformaciones necesarias que permiten al algoritmo de mapeo Gmapping entender y ajustar correctamente las posiciones relativas de los objetos en el mapa.

Estas conexiones entre los nodos permiten una comunicación efectiva y una co-

laboración entre los diferentes componentes del sistema en ROS, lo que facilita el mapeo, la navegación y la percepción del entorno para el robot. Cada nodo desempeña un papel específico y la interconexión entre ellos garantiza un funcionamiento fluido y coordinado del sistema robótico.

En cambio para la generación del mapa en tres dimensiones, se ha utilizado RTAB-Map como he mencionado antes. Es por ello que tras realizar el escaneo podemos obtener un mapa de tres dimensiones en color, ya que se fusiona la información del color y de la profundidad capturada por la cámara.

Para utilizar la herramienta RTABMAP en ROS 1, es necesario seguir unos pasos sencillos. Primero, se debe ejecutar el comando:

```
roslaunch rtabmap.launch database_path:=~/workspace/rtabmap.db
```

Este comando pone en marcha la herramienta RTABMAP y permite generar una base de datos para almacenar la información capturada durante el mapeo.

Es importante destacar que en el comando mencionado se debe indicar la ruta deseada donde se desea almacenar la base de datos. En el ejemplo anterior, se utiliza /workspace/rtabmap.db como ubicación de la base de datos, pero esto se puede modificar según las preferencias y la estructura del sistema.

Una vez ejecutado el comando, RTABMAP comenzará a realizar el mapeo utilizando los datos de los sensores y generará una base de datos en la ubicación especificada. Esta base de datos contendrá la información capturada durante el proceso de mapeo y será útil para futuros análisis y tareas relacionadas con la localización y el mapeo simultáneo (SLAM).

La ventaja clave de RTAB-Map es su capacidad para realizar mapeo con bucles bayesianos cerrados. Esto significa que el algoritmo tiene la capacidad de detectar y corregir automáticamente errores de estimación de posición que se producen al cerrar bucles en el mapa. Al utilizar esta técnica, RTAB-Map logra una mayor precisión y consistencia en los mapas generados, lo que mejora la calidad de la información espacial y permite una navegación más confiable para robots autónomos. Además de que repercute directamente en el rendimiento, produciendo que al cerrar esos bucles el uso de recursos del ordenador que se encarga de procesarlos no sea muy alto.

En el mapeado 2D, la información se guarda en dos archivos distintos: uno en formato .pgm, que contiene el mapa de ocupación en dos dimensiones, y otro en formato .yaml, que almacena los metadatos relacionados con el mapa, como su resolución y tamaño. Por otro lado, RTAB-Map utiliza una base de datos para almacenar los datos generados durante el proceso de mapeo, lo que permite un almacenamiento más eficiente y la posibilidad de realizar consultas y análisis más complejos de los datos capturados por los sensores del robot durante la exploración. Además RTAB-Map permite la visualización posterior en cualquier momento de los datos capturados, como se puede ver en la siguiente imagen:

Al finalizar un escaneo o proceso de mapeo en ROS, es fundamental conservar y guardar el mapa generado para futuras referencias o usos. Para lograr esto, se utiliza

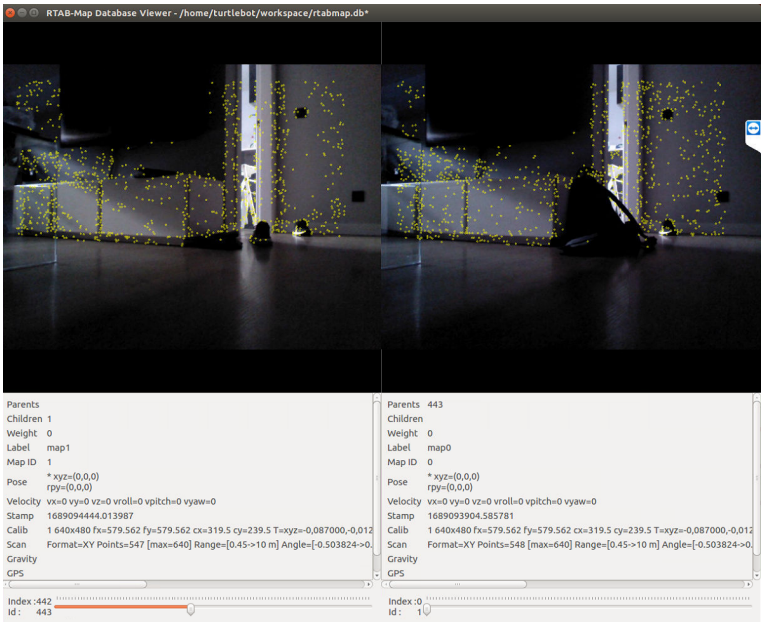

Figura 16: Base de datos de RTAB-Map

el siguiente comando:

```
rosrun map_server map_saver -f mymap
```

Al ejecutar este comando, el nodo map_server se activa y se encarga de guardar el mapa en un archivo. El parámetro -f seguido del nombre deseado (en este caso, "mymap") indica el nombre del archivo en el cual se guardará el mapa.

Una vez ejecutado el comando, se generará un archivo con extensión .pgm (Portable Graymap) y un archivo de metadatos con extensión .yaml. El archivo .pgm contiene el mapa en forma de imagen en escala de grises, donde diferentes valores de intensidad representan diferentes áreas del mapa. Por otro lado, el archivo .yaml contiene información adicional sobre el mapa, como su resolución, tamaño y posición.

Al conservar el mapa mediante el uso de este comando, se asegura la preservación y disponibilidad del mapa generado, lo que facilita su posterior uso en tareas de navegación, planificación de rutas o análisis del entorno. Es recomendable asignar un nombre significativo al archivo (en lugar de "mymap") para poder identificarlo fácilmente en el futuro.

4.4.1.3. Navegación autónoma

Para realizar la navegación del Turtlebot 2, se puede utilizar la herramienta RVIZ, que permite localizar manualmente al robot en un punto y establecer un objetivo al que debe dirigirse. Utilizando la información del mapa previamente escaneado, el sistema de navegación del Turtlebot, con el algoritmo de planificación

de rutas ROS Navigation Stack, será capaz de generar automáticamente una trayectoria para llegar al destino deseado, evitando obstáculos previamente escaneados. Incluso en el caso de que alguien se interponga en el camino del robot, el sistema será capaz de detectarlo y ajustar la ruta para evitar colisiones, asegurando así una navegación segura y autónoma.

Para poder realizar una navegación autónoma del robot debemos tener en cuenta dos componentes principales, uno es el mapa que se ha guardado, ya que es su entorno de navegación y el otro es la localización del robot. Al no tener un sistema que nos permita localizar de forma automática al robot, este a partir de los puntos intentará localizarse pero seguramente necesitaremos realizar una estimación 2D de su posición en la interfaz visual RVIZ para poder situarlo de forma correcta. En la figura 17 podemos ver como damos la localización al robot, tan solo debemos seleccionar 2D Pose Estimate y situarlo en el mapa.

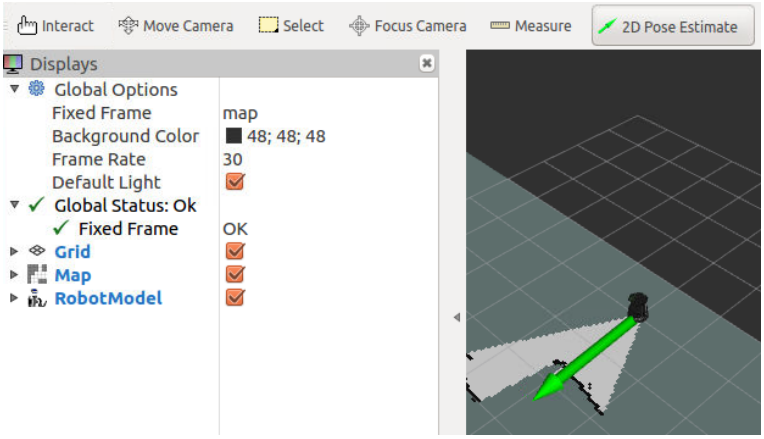

Figura 17: Estimación de la posición del robot móvil

4.4.1.4. Sistema de navegación basados en puntos de ruta

Por ultimo en está integración de la plataforma kobuki en ROS 1 se ha creado un sistema simple de navegación basado en puntos o marcas de referencia (coordenadas de un mapa). El sistema de navegación por puntos es una solución eficiente y simple para guiar al robot hacia diferentes ubicaciones específicas. El funcionamiento de este sistema se basa en un programa de **Python** que lee las posiciones desde un tópico en el que se publican estas coordenadas. Una vez que el programa recibe la posición de destino, *utiliza algoritmos de planificación de rutas para calcular la trayectoria más óptima para alcanzar ese punto.*

Al utilizar el tópico que devuelve las posiciones, el sistema de navegación por puntos permite una flexibilidad en la selección de los destinos del robot. Simplemente publicando nuevas posiciones en el tópico, se pueden actualizar y cambiar las ubicaciones a las que se desea que el robot se dirija.

Una vez que se ha calculado la trayectoria, el robot sigue las instrucciones del sistema de navegación para moverse hacia el punto de destino. El sistema ajusta

continuamente la ruta en función de la información del entorno y los obstáculos detectados para evitar colisiones.

En resumen, el sistema de navegación por puntos es un enfoque sencillo y efectivo para guiar al robot hacia ubicaciones específicas. Por ejemplo, al leer las posiciones de unas mesas, en un mapa, desde un tópico y utilizar algoritmos de planificación de rutas, el sistema calcula trayectorias óptimas y permite una navegación precisa y flexible. A través del siguiente código, podemos ver un poco como funciona el sistema:

```
class GoToPose():
    def __init__(self):
        self.goal_sent = False
        # Qué hacer si se cierra (por ejemplo, Ctrl-C o fallo)
        rospy.on_shutdown(self.shutdown)
        # Indicar al cliente de acción que queremos un hilo en ejecución de forma predeterminada
        self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
        rospy.loginfo("Esperando que el servidor de acción esté activo")
        # Esperar hasta 5 segundos a que el servidor de acción esté activo
        self.move_base.wait_for_server(rospy.Duration(5))
    def goto(self, pos, quat):
        # Enviar un objetivo
        self.goal_sent = True
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose = Pose(Point(pos['x'], pos['y'], 0.000),
                                     Quaternion(quat['r1'], quat['r2'], quat['r3'], quat['r4']))
        # Comenzar el movimiento
        self.move_base.send_goal(goal)
        # Permitir que TurtleBot tenga hasta 60 segundos para completar la tarea
        success = self.move_base.wait_for_result(rospy.Duration(60))
        state = self.move_base.get_state()
        result = False
        if success and state == GoalStatus.SUCCEEDED:
            result = True
        else:
            self.move_base.cancel_goal()
        self.goal_sent = False
        return result
    def shutdown(self):
        if self.goal_sent:
            self.move_base.cancel_goal()
        rospy.loginfo("Detenido")
        rospy.sleep(1)
```

4.4.2. Hoverboard

Para utilizar el hoverboard como plataforma, se ha desarrollado un prototipo de robot con una configuración específica. El prototipo se ha diseñado con dos ruedas locas y dos ruedas provenientes del hoverboard, permitiendo así la movilidad del robot. Además, se ha integrado un sensor Lidar A1 en la parte superior del robot para obtener datos de escaneo láser y realizar un mapeo detallado del entorno circundante. Esta configuración del prototipo permite al robot moverse de manera ágil y precisa, mientras que el Lidar A1 proporciona información crucial para la percepción y la navegación autónoma del robot en su entorno.

Al igual que el anterior robot era integrado en ROS 1, pero en la versión *kinetic*, este ha sido desarrollado e integrado en la versión *Melodic*. La integración exitosa de este robot ha sido posible gracias al desbloqueo y al desarrollo de un firmware personalizado para las placas de hoverboard. Además, se ha utilizado un repositorio de *robaka-ros* de GitHub que ha permitido a la placa controladora de los motores

publicar información precisa sobre las revoluciones de las ruedas, lo que a su vez proporciona datos de odometría para el robot.

4.4.2.1. Instalación

Para llevar a cabo la integración de este robot, se ha seguido un proceso específico. En primer lugar, se realizó la instalación de ROS 1 en su versión Melodic, proporcionando el entorno de desarrollo necesario. A continuación, se realizó el clonado del repositorio de Robaka-ROS en la carpeta de trabajo (workspace), permitiendo así acceder a los recursos y paquetes necesarios para la implementación del robot. Además, se llevó a cabo la instalación de Google Cartographer, una herramienta importante para el mapeo y la generación de mapas en 2D y 3D. Estos pasos iniciales sientan las bases para la integración del robot y la utilización de las herramientas necesarias para el desarrollo y control del mismo en el entorno de ROS.

4.4.2.2. Mapeado 2D

Para realizar el mapeado 2D de una habitación o espacio interior, se ha utilizado la plataforma construida en combinación con el sensor Lidar A1, el cual está configurado para realizar escaneos a una altura determinada. Esta configuración permite obtener datos precisos de distancia y ubicación en un plano horizontal, lo que resulta fundamental para generar un mapa detallado y preciso del entorno. Al utilizar la plataforma junto con el Lidar A1, se logra un mapeado 2D efectivo y confiable, proporcionando información espacial crucial para la navegación y la toma de decisiones autónomas del robot en el entorno interior.

En esta integración, se utilizó Google Cartographer en lugar de Gmapping para el proceso de mapeado. La principal diferencia entre Cartographer y Gmapping radica en los algoritmos y enfoques utilizados. Mientras que Gmapping se basa en un filtro de partículas para generar mapas 2D, Cartographer utiliza un enfoque más avanzado llamado SLAM (Simultaneous Localization and Mapping) basado en optimización de grafo. Esta técnica permite una mejor estimación de la posición del robot y una mayor precisión en el mapeo del entorno. Además, Cartographer es capaz de generar mapas en 3D, lo que brinda una representación más detallada del entorno tridimensional, aunque en este caso no hayamos usado el escaneo 3D.

Para entender mejor como se funciona Google Cartographer a diferencia de Gmapping como en el ejemplo anterior, tenemos que verlo como una opción mucho más precisa. Gmapping solo utiliza un filtro de partículas como se ha mencionado en otros momentos de esta memoria, sin embargo Google Cartographer es una solución basada en grafos y que utiliza mucha más información de forma optimizada.

Google Cartographer utiliza la información inercial por parte de la IMU si en ese caso el robot móvil dispone de ella (no es nuestro caso) y de los sensores de forma optimizada, por lo que puede manejar entornos más complicados que Gmapping, aunque es más complicada de configurar.

Para que se entienda el concepto de como funciona Google Cartographer tenemos que entender dos conceptos:

1. Auto-generación de relaciones de "verdad fundamental"
2. Evaluación de los datos de prueba contra la verdad del terreno generada

En Cartographer, la generación de estas relaciones de verdad se realizan a partir de las trayectorias de cierre de bucle, como la técnica de cierre de bucles bayesianos que comentábamos anteriormente. Este tipo de cierres se realizan en base a unos criterios, como pueden ser la mínima distancia recorrida, valores atípicos líneales o en radianes.

Por ejemplo si suponemos que las relaciones entre los nodos de una trayectoria cumplen los requisitos son localmente correctas. En este caso no tenemos una optimización de cierre de ciclo pero nos sirve para explicarlo.

La siguiente figura 18 ilustra esto que estoy contando. En la figura de la izquierda se ven las conexiones verdes entre los nodos de una trayectoria optimizada. Y en la figura de la derecha tenemos una trayectoria no optimizada. La métrica real es la diferencia entre esas dos métricas.

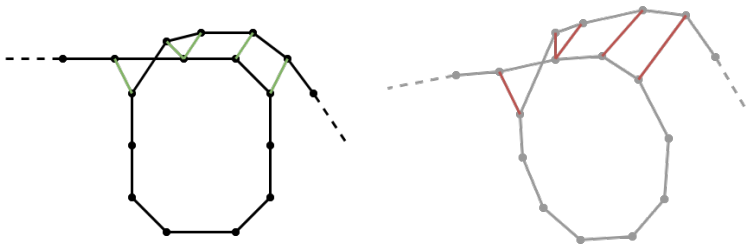

Figura 18: Evaluación SLAM en Google Cartographer

Otra de las cosas importantes es configurar adecuadamente el archivo Lua de configuración de Google Cartographer es de vital importancia para lograr un mapeo preciso y eficiente. Este archivo contiene una serie de parámetros que permiten ajustar el comportamiento del sistema de mapeo, desde la selección de sensores hasta la configuración de algoritmos y opciones de optimización. Al personalizar esta configuración, los usuarios pueden adaptar el rendimiento y la precisión del sistema a las necesidades específicas de su entorno de mapeo. Mediante la cuidadosa manipulación de estos parámetros, es posible maximizar la calidad de los datos recopilados, minimizar errores y garantizar una experiencia de mapeo robusta y confiable. Por lo tanto, invertir tiempo y esfuerzo en comprender y ajustar el archivo Lua de configuración de Google Cartographer es un paso fundamental para lograr resultados superiores en proyectos de cartografía y percepción del entorno.

```
options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "imu",
  published_frame = "base_footprint",
```

```
  odom_frame = "odom",
  provide_odom_frame = true,
  publish_frame_projected_to_2d = true,
  use_odometry = false,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 10,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}
```

Estas opciones de configuración en el archivo Lua de Google Cartographer determinan varios aspectos clave del sistema de mapeo. Aquí hay un análisis de algunas de las opciones más relevantes:

- **map_frame:** Especifica el nombre del marco de referencia del mapa en relación con el cual se alinearán los datos del sensor.
- **tracking_frame:** Define el marco de referencia utilizado para el seguimiento del sistema, típicamente asociado con los datos provenientes de los sensores inerciales.
- **published_frame:** Establece el marco de referencia desde el cual se publicarán los mensajes del mapa y los datos de odometría.
- **odometry_frame:** Indica el marco de referencia para los datos de odometría.
- **use_odometry:** Controla si se utiliza la odometría como fuente de información para el sistema de mapeo.
- **use_nav_sat:** Determina si se utilizan los datos de navegación por satélite (GPS) para el mapeo.
- **num_laser_scans y num_multi_echo_laser_scans:** Estos parámetros definen el número de escaneos láser que se utilizarán en el mapeo.
- **num_subdivisions_per_laser_scan:** Define el número de subdivisiones de cada escaneo láser, lo que afecta la densidad y precisión del mapa generado.
- **lookup_transform_timeout_sec:** Establece el tiempo máximo permitido para buscar una transformación entre marcos de referencia.
- **submap_publish_period_sec, pose_publish_period_sec y trajectory_publish_period_sec:** Estos parámetros controlan la frecuencia de publicación de submapas, poses y trayectorias, respectivamente.
- **topic_sampling_ratio:** Estos parámetros definen la frecuencia de muestreo para diferentes tipos de datos de sensores, permitiendo controlar la cantidad de información utilizada en el proceso de mapeo.

En conlusión, comprender a fondo Google Cartographer y sus opciones de configuración ha sido fundamental para lograr una exitosa integración en nuestro proyecto. Además, comprender adecuadamente cómo se realiza el paso de información entre los nodos del sistema ha sido de vital importancia. Gracias a esta comprensión, hemos podido optimizar y ajustar las configuraciones para adaptarlas a nuestras necesidades específicas y garantizar un rendimiento óptimo en la generación de mapas y la percepción del entorno. La capacidad de configurar adecuadamente las opciones y comprender el flujo de información nos ha permitido realizar una integración efectiva y aprovechar al máximo las capacidades de Google Cartographer en nuestro proyecto.

4.5. Integración en ROS 2

En este otro apartado, se aborda la transición de ROS 1 a ROS 2 y se exploran las mejoras que esta nueva versión ofrece en comparación con su predecesora. Se propone integrar la plataforma Kobuki en ROS 2 y evaluar tanto una solución simulada como una solución real para analizar y comparar las ventajas y beneficios que ROS 2 puede ofrecer en relación con la versión anterior.

ROS 2 ha sido desarrollado con el objetivo de abordar las limitaciones y desafíos identificados en ROS 1, buscando mejorar la modularidad, la escalabilidad y la interoperabilidad de la plataforma. Entre las mejoras más destacadas se encuentran la comunicación más eficiente entre nodos, el soporte nativo para sistemas operativos en tiempo real, la compatibilidad con diferentes lenguajes de programación y la mayor flexibilidad en la arquitectura del sistema.

La integración de la plataforma Kobuki en ROS 2 permitirá evaluar y comparar las capacidades y rendimiento de ROS 2 en comparación con ROS 1. Se llevará a cabo una simulación para validar el comportamiento y las funcionalidades del sistema, así como una implementación real en un entorno físico para evaluar la viabilidad y la eficiencia del uso de ROS 2 en aplicaciones prácticas.

A través de este análisis comparativo, se busca obtener una visión más completa de las mejoras y beneficios que ROS 2 puede aportar en términos de rendimiento, robustez y facilidad de desarrollo. El objetivo final es determinar si la adopción de ROS 2 en la plataforma Kobuki es una opción prometedora para futuros desarrollos y aplicaciones en el ámbito de la robótica.

4.5.1. Turtlebot 2

En este caso el objetivo perseguido de nuevo con Turtlebot 2 o la plataforma Kobuki era conseguir integrar en ROS 2 la plataforma y la cámara Orbbec Astra. Para ello los propios desarrolladores de los paquetes de Turtlebot habían desarrollado una serie de paquetes para lograr la conexión entre la plataforma y la cámara para poder hacer algo tan sencillo como un escaneo.

El proceso de integración de ROS 2, se inició utilizando paquetes que facilitaron

la integración y permitieron lanzar los nodos necesarios para controlar los motores de la plataforma y gestionar la publicación de la cámara en sus respectivos nodos. Estos paquetes proporcionaron una base sólida para la configuración inicial del sistema.

Posteriormente, el objetivo principal fue aprovechar los beneficios de Google Cartographer, una herramienta que previamente había demostrado su utilidad y precisión en el desarrollo del prototipo utilizando motores de hoverboard. Se optó por utilizar Google Cartographer en lugar de Gmapping, ya que se evidenció que era una opción más efectiva y precisa para construir mapas. La integración de Google Cartographer en el entorno de ROS 2 permitió generar mapas más detallados y de mayor calidad, lo que a su vez mejoró la navegación y la toma de decisiones autónomas del robot.

Esta combinación de paquetes de integración y el uso de Google Cartographer en ROS 2 fue un enfoque estratégico para obtener resultados superiores en términos de generación de mapas y rendimiento general. El objetivo final era aprovechar las capacidades y mejoras de ROS 2, junto con la precisión y eficiencia de Google Cartographer, para lograr un sistema de mapeo y navegación más avanzado y confiable en comparación con la versión anterior de ROS.

4.5.1.1. Instalación

Para llevar a cabo esta integración, los componentes esenciales que se han requerido instalar son el repositorio de Turtlebot para ROS 2, la herramienta **Google Cartographer** y la herramienta **SLAM ToolBox**. Estas tres piezas fundamentales proporcionan las funcionalidades necesarias para la integración exitosa de la plataforma con ROS 2, así como para el mapeo y la generación de mapas de alta precisión. No he hablado antes de SLAM ToolBox pero es otra herramienta que permite el uso de SLAM y en términos de uso es mucho más rápida que Google Cartographer de entender.

4.5.1.2. Resultados iniciales

Durante el proceso de integración, se utilizaron los paquetes iniciales proporcionados para Turtlebot 2, lo que permitió lograr la teleoperación de la plataforma Kobuki. Gracias a la publicación de la odometría de las ruedas, se pudo controlar el robot de forma remota utilizando el teclado como interfaz.

Posteriormente, se logró manualmente que la cámara Orbbec Astra comenzara a publicar información. Mediante la creación de un pequeño programa, se implementó un sistema en el que la cámara seguía constantemente un objeto fijo dentro de su campo de visión. Esta funcionalidad añadida permitió un mayor nivel de interacción y seguimiento visual por parte del robot.

El tercer paso en la integración implicaba utilizar la plataforma y la cámara junto con Google Cartographer para realizar un escaneo y mapeo del entorno. Sin embargo, se encontró un obstáculo en la publicación de las **transformaciones (TF)**

Figura 19: Seguimiento de un código QR

necesarias para llevar a cabo el mapeo. Esto se debió a que el repositorio utilizado, así como la plataforma misma, eran versiones antiguas y había un fallo en las transformaciones al que no se le podía dar una solución y no se logró poder realizar el escaneo.

A pesar de esta dificultad, los primeros pasos en la integración fueron exitosos, permitiendo la teleoperación de la plataforma y la implementación de seguimiento visual con la cámara. Aunque el objetivo final de lograr la creación de un mapa y la navegación autónoma del robot en él no se alcanzó debido a las limitaciones tecnológicas, estos avances iniciales sientan las bases para el desarrollo posterior que hice para intentar lograr un mapeo con esta plataforma.

4.5.1.3. Resultados posteriores

A pesar de los esfuerzos realizados, se intentó utilizar el Lidar A1 en conjunto con la plataforma Kobuki para generar un mapa, pero desafortunadamente no se logró el resultado esperado. Durante el proceso de integración, se configuró el Lidar A1 y se llevó a cabo la configuración correspondiente en RVIZ. Sin embargo, al desplegar todos los componentes y visualizar en RVIZ, no se mostraba ningún mapa. Esta vez ya no se utilizo Google Cartographer como herramienta para realizar el SLAM sino que se utilizo la herramienta SLAM ToolBox que parecía estar mucho más completa para ROS 2 Foxy y que permitía un control más exhaustivo de los errores que se producían.

Este problema puede haber surgido debido a varias razones, como un problema de comunicación o compatibilidad entre el Lidar y la plataforma, o problemas asociados a las transformaciones necesarias para que la **odometría** y la información del **láser** se publiquen de forma correcta.

Aunque no se pudo generar un mapa exitosamente en esta instancia, este resultado nos brinda valiosos aprendizajes y oportunidades para futuras mejoras. Se requiere una investigación adicional para identificar las posibles causas y explorar

Figura 20: Plataforma Kobuki con lidar RP A1

soluciones alternativas que permitan el correcto funcionamiento del Lidar A1 con la plataforma Kobuki y la generación de mapas precisos. A continuación se muestra el grafo de conexión de los nodos, que demuestra la interconexión de todos los nodos sin explicar porque no llega a publicar el mapa:

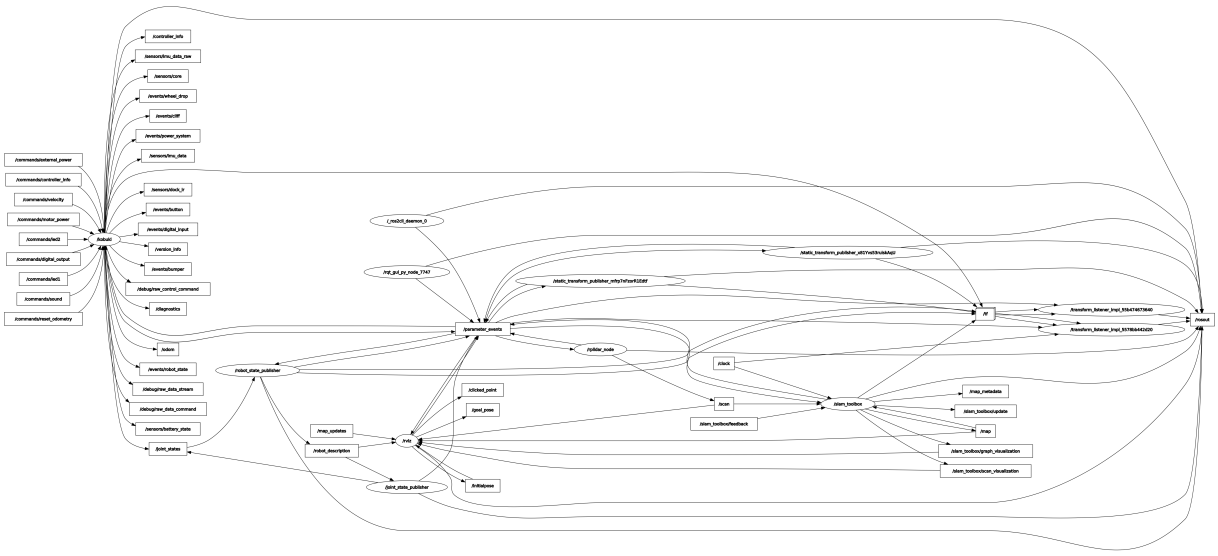

Figura 21: Grafo rqt de ROS 2 sobre los nodos activos

En este grafo se puede ver toda la publicación de nodos y tópicos activo en el sistema, hay que destacar los dos nodos en los que radica básicamente el funcionamiento de un escaneo que son /odom y /scan. El nodo **/odom** es el encargado de publicar la odomotería que por lo que se ha revisado parece ser que el problema es que el nodo que publica la información no llega a el nodo de SLAM ToolBox y por lo tanto no se puede ver de forma visual en RVIZ. A diferencia del nodo **/scan** de rplidar que si que se comunica con SLAM ToolBox como podemos ver enel grafo y esto hace que se puede mostrar el escaneo generado por el lidar como se ve en la figura 21.

Es importante recordar que los desafíos forman parte del proceso de desarrollo y la integración de tecnologías, y cada obstáculo permite el descubrimiento de técnicas que no son buenas prácticas y cambiar el entendimiento de ciertos conceptos teóricos en robótica.

A pesar de los desafíos encontrados, se mantuvo una persistencia continua en el proceso de integración y se dedicó tiempo a solucionar los problemas que surgieron. Uno de los desafíos identificados fue la publicación de las transformaciones en el árbol de transformaciones (TF tree), lo cual afectaba la correcta visualización y funcionamiento del sistema. Mediante un análisis detallado y un enfoque meticuloso, se logró identificar y corregir los problemas relacionados con las transformaciones, lo que *permitió establecer un* ***TF tree adecuado***. En la figura 22 se puede ver como es un árbol de transformaciones correcto:

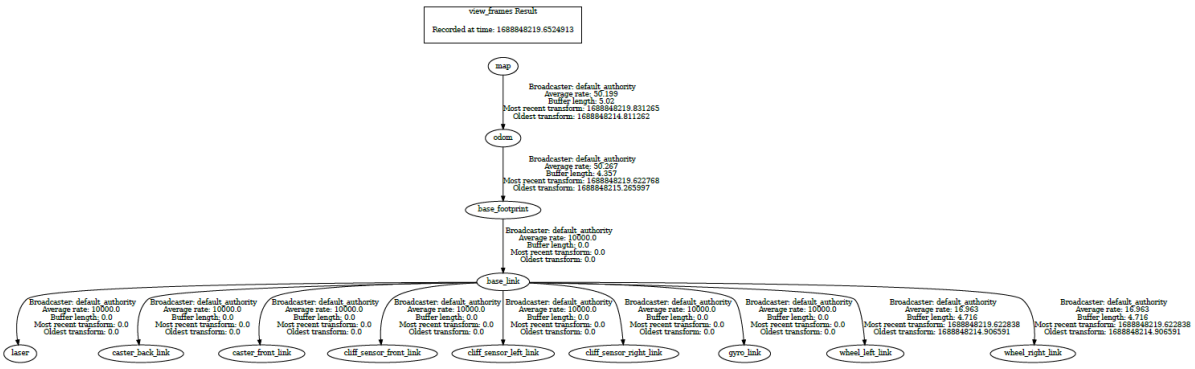

Figura 22: Grafo rqt de ROS 2 del árbol de transformaciones

En está figura se puede ver como las transformaciones manuales, para el paso de información, que hice para poder interpretarlo en RVIZ se ven de forma correcta. Realmente lo que tenemos que observar es que las transformaciones estén correctamente ordenadas, siendo el padre siempre un frame como map o world, seguido de los datos de odometría y a su vez seguido de el /los frames encargados de los sensores externos.

Los círculos o elipses que se pueden ver en la figura 22 son los nombres de los frames, que son el sistema de coordenadas correspondientes de los componentes del robot y las transformaciones son cada una de la flechas que relacionan cada frame. Para entenderlo hay que entender antes bien, como funciona un frame:

Los **marcos de referencia**, también conocidos como marcos de coordenadas, son sistemas de coordenadas tridimensionales que se utilizan para representar la posición y orientación de los objetos en un entorno de robótica. En ROS, se utilizan para describir la ubicación y orientación de los diferentes componentes de un robot, como sensores, actuadores y partes del cuerpo. Cada marco de referencia tiene un nombre único y se puede relacionar con otros marcos de referencia mediante transformaciones. La funcionalidad de los marcos de referencia en ROS incluye:

1. Proporcionar una forma estandarizada de representar y relacionar las ubica-

ciones y orientaciones de los componentes del robot.

2. Permitir el cálculo de transformaciones entre diferentes marcos de referencia para realizar la coordinación entre los componentes del robot.

3. Facilitar la fusión de datos de múltiples sensores y la integración de los sistemas de percepción y control del robot.

Los **nodos** son los componentes de software individuales en ROS que realizan tareas específicas como la publicación de la información láser del LIDAR.

Es por ello que tan solo tenemos que comprender que cada uno es como un sistema de coordenadas que se comunica con el padre que tienen inmediatamente superior, hasta llegar al último que es el mapa.

Además, para esta integración se resolvieron las dificultades relacionadas con la publicación de la odometría (no estaba siendo publicada), lo cual fue esencial para obtener mediciones precisas de la posición y la trayectoria del robot. Con estos avances, finalmente se pudo realizar un mapeo semi exitoso utilizando el Lidar A1 y la plataforma Kobuki, no existoso del todo, porque sigue habiendo un fallo con la odometría, que es que aunque está sea publicada, para que el nodo controlador de SLAM ToolBox no es capaz de recibir esa información. La generación del mapa parcial (porque en algunos momentos se congelaba) proporcionó una representación visual del entorno circundante.

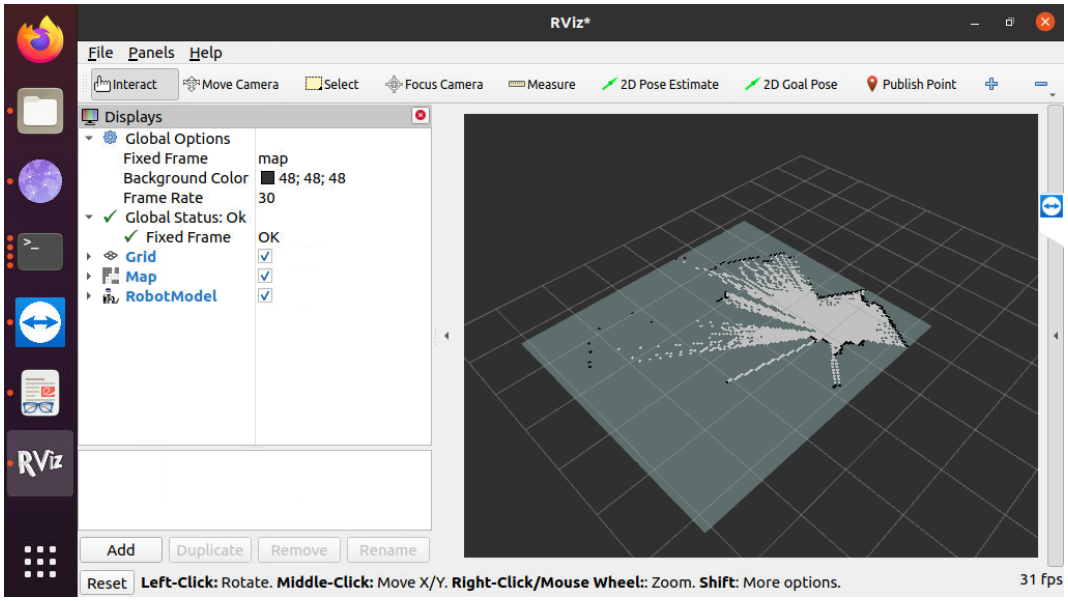

Figura 23: Interfaz de RVIZ durante el escaneo o mapping

Este logro representa un hito significativo en el proceso de integración, ya que se superaron obstáculos y se encontraron soluciones a problemas técnicos. La persistencia y la dedicación demostradas en este proyecto reafirman la importancia de la resolución de problemas y la búsqueda de soluciones efectivas. Los resultados obtenidos en la generación exitosa del mapa refuerzan el valor del esfuerzo continuo y el enfoque riguroso en la implementación de tecnologías de robótica y mapeo.

4.5.2. Turtlebot 3 (burger)

Después de explorar diversas integraciones en un entorno físico con un robot real, uno de los objetivos adicionales fue lograr la integración de un robot simulado. Para este propósito, se seleccionó el robot Turtlebot 3 Burger como plataforma de simulación. En esta sección, se describirá el proceso seguido para la instalación del Turtlebot 3 Burger en el entorno de simulación y se detallará cómo se llevó a cabo el mapeado y la navegación utilizando herramientas específicas.

Para comenzar, se procedió a la instalación del Turtlebot 3 Burger en el entorno de simulación, utilizando **Gazebo** como plataforma de simulación en la **versión Foxy de ROS 2**. Esto implicó la configuración de los paquetes y controladores necesarios para permitir la simulación del robot y su interacción con el entorno virtual.

Una vez completada la instalación, se procedió a realizar el mapeado y la navegación en el entorno simulado. Se utilizaron herramientas como el paquete de mapeo **Google Cartographer** y el algoritmo de navegación proporcionado por el Turtlebot 3. Estas herramientas permitieron realizar un mapeo 2D del entorno virtual, así como planificar rutas y llevar a cabo la navegación autónoma del robot simulado.

A través de este enfoque de simulación, se pudo evaluar y validar el funcionamiento del Turtlebot 3 Burger en diferentes escenarios virtuales, lo que brindó una plataforma segura y flexible para probar y optimizar algoritmos de mapeado y navegación antes de implementarlos en un entorno real.

4.5.2.1. Instalación

Para habilitar la simulación del Turtlebot 3 en ROS2, se requiere la instalación de varios componentes clave. En primer lugar, es necesario instalar ROS2 en su versión Foxy, que proporciona el entorno de desarrollo necesario para la integración. A continuación, se deben instalar los paquetes específicos de Turtlebot 3, que contienen los controladores y configuraciones necesarios para simular el robot.

Además, se necesita instalar Google Cartographer, una herramienta esencial para el mapeo y la generación de mapas en 2D, ya que está simulación solo utiliza un lidar RP A1. Esta herramienta permitirá obtener mediciones precisas del entorno durante la simulación.

Por último, se debe instalar Gazebo, una plataforma de simulación ampliamente utilizada y compatible con ROS2. Gazebo proporciona un entorno virtual realista donde se puede simular y probar el Turtlebot 3.

Una vez que todos estos componentes están instalados correctamente, se podrá iniciar la simulación del Turtlebot 3 en ROS2.

4.5.2.2. Simulación

Para llevar a cabo la simulación del Turtlebot 3 Burger, se utilizó el entorno proporcionado por Gazebo. Este entorno de simulación, representado como un hexágono virtual, permitió teleoperar y controlar el Turtlebot 3 Burger de forma remota. Gracias al uso del sensor LIDAR A1, se pudo realizar un mapeo completo de este entorno simulado. A continuación en la siguiente figura 24 se puede ver el entorno de simulación de Turtlebot 3:

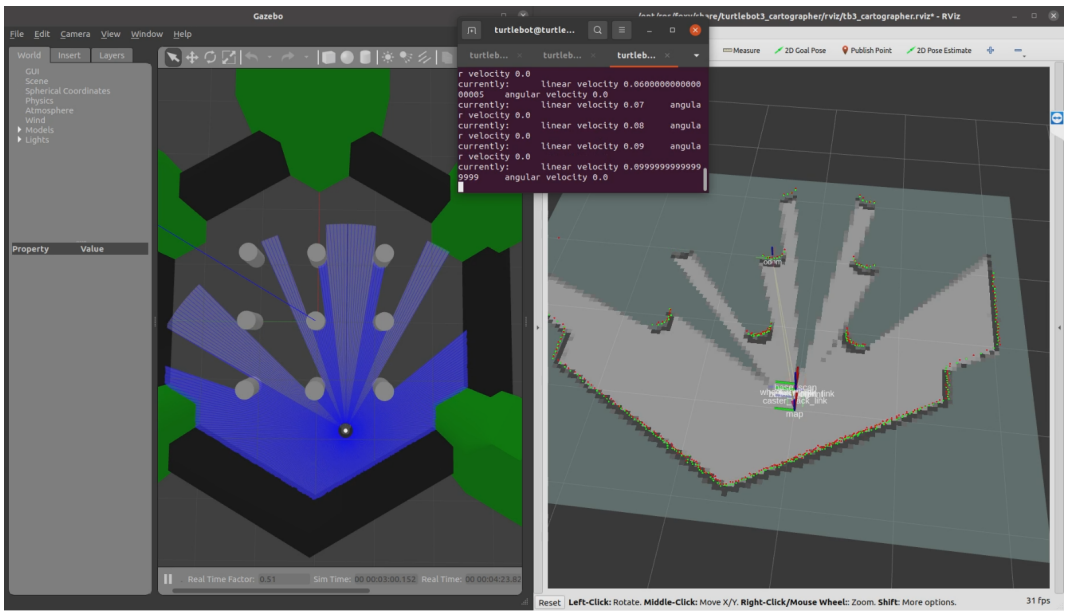

Figura 24: Mapeado del entorno de simulación usando Google Cartographer

Durante la simulación, el LIDAR A1 escaneó el entorno y recopiló datos precisos de distancia y ubicación. Estos datos fueron procesados y utilizados para generar un mapa detallado del entorno en el que se encontraba el Turtlebot 3 Burger. Este mapa proporcionó una representación visual clara de todas las características y obstáculos presentes en el entorno simulado. Al igual que cuando se trata de un robot real, en un simulación obtenemos los dos tipos de ficheros, tanto el fichero con extensión pgm que es el propio como si fichero de configuración con extensión yaml.

Una vez generado el mapa, se pudo utilizar para realizar la navegación autónoma del robot. Con el mapa como referencia, el Turtlebot 3 Burger fue capaz de planificar rutas y evitar obstáculos de manera inteligente mientras se desplazaba por el entorno simulado. Se puede ver en la figura 27 tomada de las simulaciones como el robot traza una ruta hasta el objetivo seleccionado y trata de recorrerla:

En resumen, mediante el uso del entorno de simulación proporcionado por Gazebo, se pudo teleoperar el Turtlebot 3 Burger y utilizar el LIDAR A1 para realizar el mapeo del entorno. El resultado fue la generación de un mapa detallado que permitió la navegación autónoma del robot en el entorno simulado. Esta simulación proporcionó una valiosa plataforma de pruebas y desarrollo si se desarrollará el robot real.

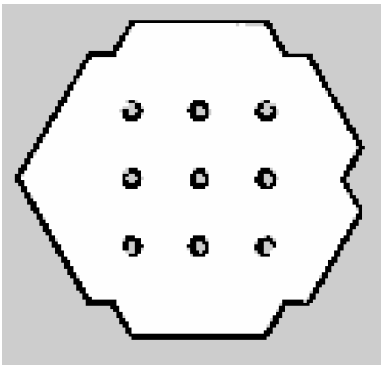

Figura 25: Fichero de extensión PGM

```
image: turtlebot3_sim_map.pgm
mode: trinary
resolution: 0.05
origin: [-1.23, -2.55, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

Figura 26: Detalles del propio mapa en el fichero YAML

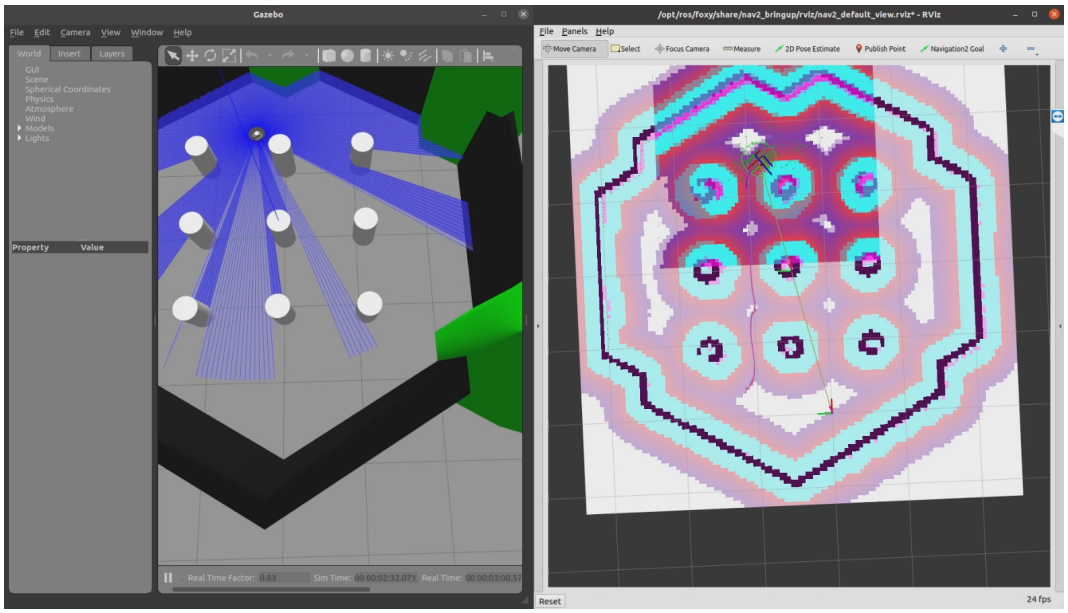

Figura 27: Navegación autónoma del Turtlebot 3 Burger

4.5.2.3. Comparación entre simulación y mundo real: evaluando el rendimiento del sistema en ROS 2

La comparación entre la simulación y el mundo real es un aspecto crucial para evaluar el rendimiento del sistema en ROS 2. La simulación proporciona un entorno controlado y predecible donde se pueden probar y depurar los algoritmos y las funcionalidades del sistema de manera segura y eficiente. Permite una iteración rápida y la identificación temprana de posibles problemas antes de la implementación en el mundo real.

Sin embargo, la simulación tiene limitaciones inherentes, ya que no puede capturar completamente las complejidades y variabilidades del mundo real. En contraste, el entorno del mundo real presenta desafíos reales, como la incertidumbre del sensor, la variabilidad del entorno y las limitaciones físicas del robot. Estos factores pueden afectar el rendimiento del sistema y requerir ajustes y adaptaciones adicionales.

En el mundo real, **las condiciones no siempre son perfectas** y se presentan desafíos adicionales en comparación con la simulación. En un entorno físico, existen huecos entre objetos y obstáculos que pueden dificultar la percepción y la detección mediante el sensor láser. Además, el material de los objetos puede afectar la forma en que la información del láser rebota y se procesa, lo que a su vez puede influir en la calidad de los datos obtenidos.

Además, es común encontrarse con objetos que son demasiado pequeños para ser detectados por el sensor, como las patas de sillas y mesas. Estos objetos pueden representar desafíos significativos en el proceso de mapeo y navegación, ya que su presencia puede no ser correctamente identificada o representada en el mapa.

Estos factores del mundo real introducen **incertidumbre y complejidad** en el proceso de mapeo y navegación, y requieren un enfoque cuidadoso para abordarlos. Es importante considerar estas limitaciones y ajustar los algoritmos y estrategias en consecuencia, con el fin de lograr una navegación segura y eficiente en entornos reales.

En la simulación, es posible obtener resultados más consistentes y repetibles, lo que facilita la comparación de diferentes configuraciones y estrategias. Además, se pueden realizar pruebas exhaustivas y variar los parámetros de manera controlada para evaluar el rendimiento en diferentes escenarios.

Por otro lado, *el mundo real permite evaluar el comportamiento del sistema en situaciones más desafiantes y realistas*. Las interacciones con el entorno, la presencia de ruido en los sensores y las limitaciones físicas del robot brindan una comprensión más precisa del desempeño y la viabilidad del sistema en condiciones reales.

En conclusión, tanto la simulación como el mundo real son componentes importantes para evaluar el rendimiento del sistema en ROS 2. La simulación proporciona una plataforma para el desarrollo inicial y las pruebas, mientras que el mundo real brinda información valiosa sobre el comportamiento del sistema en situaciones más complejas y reales. Tras haber probado una navegación simulada con ROS 2 no hay evidencias por ejemplo tomando como métrica el tiempo que ROS 2 sea una plataforma mucho más eficiente que ROS 2.

4.5.3. Roomba iCreate 3

En este último desarrollo, se ha llevado a cabo la integración de los paquetes del Turtlebot 4 adaptados a una instalación personalizada, en la cual no se utiliza el producto comercial como base, sino que se aprovechan los componentes y recursos disponibles para construir una solución a medida.

Para lograr esta adaptación, se ha realizado un análisis exhaustivo de los componentes y recursos disponibles, evaluando su compatibilidad y funcionalidad con los paquetes del Turtlebot 4. A partir de esta evaluación, se ha llevado a cabo una integración personalizada, donde se han implementado las configuraciones y conexiones necesarias para garantizar el correcto funcionamiento del sistema.

Este enfoque de adaptación permite aprovechar los componentes y recursos disponibles de manera eficiente, maximizando su utilidad y reduciendo la dependencia de productos comerciales específicos. Además, brinda la flexibilidad necesaria para ajustar y personalizar el sistema según las necesidades y restricciones de la instalación en particular.

Para este ultimo, *caso de estudio en vez de utilizar la plataforma comercial*, se ha utilizado el mismo Roomba iCreate 3 pero con distinto hardware, en vez de la raspberry pi 4 que se utiliza en la solución comercial, se ha utilizado un mini ordenador intel Nuc y en vez de utilizar el conjunto de cámara Oak con el lidar RP A1 solo se ha utilizado este ultimo. El objetivo propuesto era conseguir realizar una navegación autónoma con el Roomba iCreate 3 sin necesidad de tener todos los componentes disponibles del turtlebot 4. En la siguiente imagen se puede ver todos los componentes conectados en la realización de las pruebas:

Figura 28: Prototipo de Turtlebot 4 con iCreate 3

4.5.3.1. Instalación

Para la instalación los recursos necesarios para utilziar el Roomba iCreate 3 ha sido necesario utilizar la **versión de ROS 2 Humble**, junto a los paquetes que proporcionan de Turtlebot 4. Para la instalación de los paquetes se ha seguido el manual de Turtlebot 4.

4.5.3.2. Resultados obtenidos

El resultado obtenido con el Roomba iCreate 3 ha sido exitoso en términos de teleoperación y la integración de un Lidar A1 para realizar un escaneo en dos dimensiones del entorno. Estas funcionalidades permitieron obtener datos de distancia y ubicación precisos, lo cual fue un logro importante en el desarrollo del proyecto.

Sin embargo, lamentablemente, se encontraron dificultades al intentar implementar técnicas de SLAM (Simultaneous Localization and Mapping) y navegación en esta plataforma. Uno de los principales desafíos fue la falta de una documentación clara sobre el estado actual de los paquetes utilizados, los cuales eran bastante

recientes. Además, se encontraron algunos bugs en los paquetes, lo que dificultó su implementación exitosa. Este tipo de información se puede encontrar en el repositorio de github de Turtlebot 4, ya que hay se muestran la cantidad de Issues relacionados con el mapeo y navegación al utilizar estos paquetes. En la Figura 29 se muestra el problema que hay en relación a que no se procesa la información del LIDAR y no se realiza el escaneo, y que se encuentra en el issue publicado por mi en el repositorio:

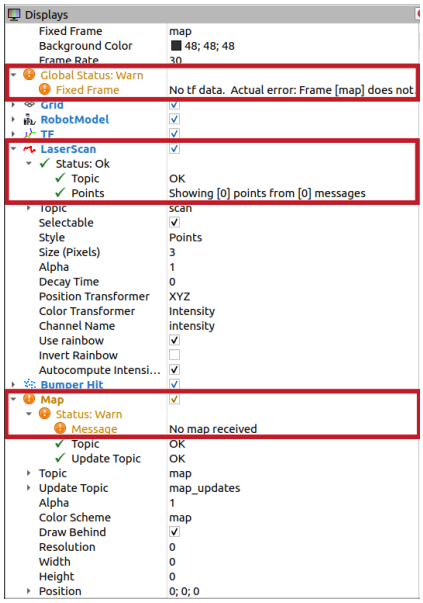

Figura 29: Problema de procesamiento del mapa

Estos obstáculos relacionados con la falta de documentación y los bugs identificados demuestran la importancia de contar con una base sólida de recursos y documentación actualizados al desarrollar proyectos de robótica. La disponibilidad de información precisa y actualizada es crucial para el éxito de la implementación de técnicas avanzadas como el SLAM y la navegación autónoma.

A pesar de no haber logrado implementar completamente estas técnicas en la plataforma Roomba iCreate 3 debido a las limitaciones mencionadas, se obtuvieron valiosos aprendizajes y se sentaron las bases para futuras mejoras y desarrollos en la integración de estas funcionalidades. Esta experiencia resalta la importancia de una investigación exhaustiva, la colaboración con la comunidad de desarrollo y la actualización constante de los recursos para lograr resultados exitosos en proyectos de robótica.

5. Limitaciones y desafíos

Durante el desarrollo del proyecto, se encontraron diversas limitaciones y desafíos que afectaron la integración y el uso de determinadas plataformas y paquetes en ROS 2.

Uno de los desafíos principales fue utilizar la **plataforma Kobuki en ROS 2**, ya que resultó ser una plataforma antigua con poca disponibilidad de software compatible. Esto dificultó la integración y limitó las funcionalidades que se podían utilizar con esta plataforma.

Además, los paquetes de Turtlebot 2 presentaron numerosos errores y dependencias obsoletas en el entorno de ROS 2. Esto generó dificultades para su instalación y funcionamiento adecuado, lo que afectó la implementación de esta plataforma junto con la cámara Orbbec Astra en ROS 2. Fueron múltiples módulos que se encontraban obsoletos como el de **python flowdas_meta_1.0.1** que impedía el uso correcto de los launch en la versión *Bouncy* de ROS 2.

No solo eso sino que para usar la plataforma Kobuki en ROS 2, los paquetes necesarios para controlar los motores, publicarla odometría y utilizar los sensores que están integrados dentro de la propia plataforma no son muy útiles, ni claros a la hora de usarlos. Esto se puede ver en los diferentes issue's que se presentan en su repositorio indicando múltiples problemas con el movimiento de las ruedas y con el uso de la plataforma mediante teleoperación.

Otro desafío importante fue la **falta de documentación en las implementaciones específicas para ROS 2**. Esta carencia dificultó el proceso de desarrollo y la resolución de problemas, ya que no se contaba con guías claras y detalladas para el uso de ciertos paquetes y herramientas.

Asimismo, la *comunidad había experimentado en menor medida con ROS 2 en comparación con ROS 1*. Esto hizo que había menos recursos y ejemplos disponibles para aprovechar y aprender de experiencias previas, lo que aumentó la complejidad del desarrollo en el entorno de ROS 2.

Por último, las **versiones de ROS 2** resultaron ser **menos estables** en comparación con sus otras versiones de ROS 1. Esto significó que, a medida que se trabajaba con una versión de ROS 2, surgían rápidamente nuevas versiones que requerían ajustes y actualizaciones constantes, lo que añadió complejidad al proceso de desarrollo.

A pesar de estas limitaciones y desafíos encontrados, se pudo superar muchos obstáculos y obtener resultados significativos en el proyecto. Estas dificultades resaltan la importancia de una investigación exhaustiva, la colaboración con la comunidad y la adaptación constante a los cambios en el ecosistema de ROS 2.

6. Conclusiones

A lo largo de este proyecto, se ha llevado a cabo la investigación en cuanto a la integración de un robot móvil en el entorno de ROS (Robot Operating System) lo que ha permitido que se desarrollen pequeñas soluciones que permiten un rápido uso de varias plataformas para el posterior desarrollo de otros sistemas o arquitecturas. Esta investigación ha permitido aumentar de gran manera el conocimiento sobre este marco de trabajo y allanar el terreno sobre la investigación relacionada sobre todo con las plataformas comerciales como Turtlebot.

Durante este proyecto se han llevado a cabo diversos estudios y se han realizado integraciones de ROS en diferentes plataformas robóticas. Se ha investigado y evaluado técnicas de escaneo o mapeo, así como técnicas de navegación, lo que ha permitido adquirir un mayor conocimiento sobre estos temas.

Se ha logrado integrar tanto ROS 1 como ROS 2 en la plataforma Kobuki (aunque no en su totalidad), así como en un hoverboard se ha conseguido integrar ROS 1, lo que ha ampliado el abanico de posibilidades para realizar experimentos y pruebas. Sin embargo, se ha encontrado que la integración de ROS 2 en el Roomba iCreate 3 ha sido más desafiante debido a la falta de documentación y a la relativa novedad de la versión Humble de ROS 2. Aunque en la versión de ROS 2 Foxy, se ha conseguido grandes avances en ROS 2 todavía queda mucho trabajo en estas nuevas versiones, ya que su alto grado de cambios y de pocas pruebas por parte de la comunidad hacen que sea mucho más difícil conseguir integrar los diferentes componentes hardware.

En términos de estabilidad y documentación, se ha observado que ROS 1 cuenta con una amplia documentación y versiones más estables, mientras que ROS 2 aparenta mejoras no muy notables en los tiempos de navegación, por otra parte si que está maás destina a algoritmos de navegación más precisos. Sin embargo, todavía existe una brecha significativa en cuanto a la documentación y el apoyo de la comunidad, lo que hace que la mayoría de los usuarios continúen trabajando con ROS 1.

Con este trabajo se ha logrado un avance significativo en la comprensión del entorno y el funcionamiento de ROS (Robot Operating System), así como en el dominio de diversas herramientas de SLAM (Simultaneous Localization and Mapping), como gmapping, Google Cartographer y SLAM Toolbox. Esta comprensión más profunda ha permitido que los desarrollos futuros en el campo de la robótica sean mucho más rápidos y sencillos. Ahora, los investigadores y desarrolladores tienen a su disposición una amplia gama de herramientas y bibliotecas que les permiten crear sistemas de navegación y mapeo más eficientes y precisos. Gracias a esto, se ha reducido significativamente el tiempo y los recursos necesarios para implementar soluciones robóticas avanzadas, abriendo las puertas a un nuevo mundo de posibilidades en la automatización y la interacción hombre-máquina.

En conclusión, este proyecto ha sido enriquecedor y ha permitido adquirir conocimientos valiosos sobre ROS, explorar diferentes plataformas robóticas y evaluar las capacidades y limitaciones de las versiones de ROS. Se ha evidenciado el poten-

cial de ROS 2, aunque también se ha destacado la importancia de contar con una comunidad sólida y una documentación exhaustiva para aprovechar al máximo las capacidades de esta versión.

7. Líneas de trabajo futuras

En cuanto a las líneas futuras de desarrollo de este proyecto, se plantean diversas áreas de enfoque que permitirán seguir avanzando en la mejora de las capacidades y funcionalidades de los sistemas robóticos integrados con ROS.

Una de las líneas de investigación es la aplicación de algoritmos para **determinar qué algoritmo de navegación y mapeo es más adecuado** para cada entorno específico. Esto implica evaluar y comparar diferentes algoritmos, teniendo en cuenta factores como el tamaño del entorno, la presencia de obstáculos y las restricciones de movimiento. Se busca seleccionar y adaptar los algoritmos más óptimos para garantizar una navegación y mapeo eficientes en una amplia gama de escenarios.

Otro aspecto importante es la exploración de la **integración de ROS 2 en la placa controladora del hoverboard**. Esto permitiría aprovechar las capacidades de ROS 2 en esta plataforma, mejorando el rendimiento y la estabilidad de los sistemas implementados en ella.

Además, se propone realizar **benchmarks y pruebas de rendimiento sobre los algoritmos DWA Planners** y otros algoritmos de planificación integrados en ROS. Estas pruebas permitirán evaluar y comparar el rendimiento y la eficacia de los algoritmos en diferentes escenarios y condiciones, proporcionando información valiosa para la selección y optimización de algoritmos de navegación.

Una línea importante de investigación es lograr la implementación e **integración exitosa de la Roomba iCreate 3 en ROS 2**. Esto implica superar los desafíos identificados anteriormente, como la falta de documentación y el soporte limitado de la comunidad para esta plataforma en ROS 2. Una vez lograda la integración, se podrá aprovechar las mejoras y funcionalidades de ROS 2 en la Roomba iCreate 3, ampliando sus capacidades y posibilidades de uso.

Por último, se propone explorar el uso de un planificador de trayectorias como **OMPL (Open Motion Planning Library)** para comparar y evaluar diferentes algoritmos de planificación en entornos complejos. Esto permitirá mejorar la planificación de trayectorias en tiempo real, garantizando movimientos eficientes y seguros para los robots en diferentes contextos y escenarios.

Estos enfoques permitirán continuar mejorando los sistemas robóticos integrados con ROS y explorar nuevas capacidades y aplicaciones en el campo de la robótica.

Referencias

[1] Kang Li, Shiying Sun, Xiaoguang Zhao, Jinting Wu, and Min Tan. Inferring user intent to interact with a public service robot using bimodal information analysis. *Advanced Robotics*, 33(7-8):369 – 387, 2019. Cited by: 6. [*Citado en pág. 10.*]

[2] Yanyu Zhang, Xiu Wang, Xuan Wu, Wenjing Zhang, Meiqian Jiang, and Mahmood Al-Khassaweneh. Intelligent hotel ros-based service robot. In *2019 IEEE International Conference on Electro Information Technology (EIT)*, pages 399–403, 2019. [*Citado en pág. 10.*]

[3] Ruchik Mishra and Arshad Javed. Ros based service robot platform. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 55–59, 2018. [*Citado en pág. 11.*]

[4] Sergio García González et al. Desarrollo de una plataforma robótica para la gestión de pedidos en la restauración. 2022. [*Citado en pág. 11.*]

[5] Clara Palacios Marín. Estudio de la aplicabilidad del transporte automatizado para el transporte de material en hospitales. 2021. [*Citado en pág. 11.*]

[6] Jonatan E. Salinas-Avila, Hugo G. Gonzalez-Hernandez, Nashely Martinez-Chan, Carlos M. Bielma-Avendano, Ximena A. Salinas-Molar, and Martin O. Garcia-Garcia. Assistant delivery robot for nursing home using ros: Robotic prototype for medicine delivery and vital signs registration. page 141 – 148, 2022. Cited by: 0; All Open Access, Bronze Open Access. [*Citado en pág. 11.*]

[7] Antica Eam Opha, Sittikorn Titaviriya, and Wibool Piyawattanametha. Fully autonomous drug and food delivery robot for hospital and patient care logistic management based on artificial intelligence. 2020. Cited by: 0. [*Citado en pág. 11.*]

[8] Anis Koubâa, Mohamed-Foued Sriti, Yasir Javed, Maram Alajlan, Basit Qureshi, Fatma Ellouze, and Abdelrahman Mahmoud. Turtlebot at office: A service-oriented software architecture for personal assistant robots using ros. page 270 – 276, 2016. Cited by: 25. [*Citado en pág. 11.*]

[9] Pasqual Joan Ribot Lacosta. Robot móvil manipulador tiago en un entorno hospitarlario. 2019. [*Citado en pág. 11.*]

[10] Harish Ram Nambiappan, Krishna Chaitanya Kodur, Maria Kyrarini, Fillia Makedon, and Nicholas Gans. Mina: A multitasking intelligent nurse aid robot. page 266 – 267. Association for Computing Machinery, 2021. Cited by: 5; All Open Access, Bronze Open Access. [*Citado en pág. 11.*]

[11] Javier Ramírez de la Pinta. Integration of service robots in the smart home. 2017. [*Citado en pág. 11.*]

[12] L.A.I. Zhi-Jie, C.A.O. Ming-Yu, Zhang Zhi-Ming, and Y.U. You-Ling. Design and implementation of home service robot. page 3541 – 3546, 2020. Cited by: 0. [*Citado en pág. 11.*]

[13] E.D.G. Sanjeewa, K.K.L. Herath, B.G.D.A. Madhusanka, H.D.N.S. Priyankara, and H.M.K.K.M.B. Herath. *Understanding the hand gesture command to visual attention model for mobile robot navigation: Service robots in domestic environment.* 2021. Cited by: 1. [*Citado en pág. 11.*]

[14] Jiansheng Peng, Hemin Ye, Qiwen He, Yong Qin, Zhenwu Wan, and Junxu Lu. Design of smart home service robot based on ros. *Mobile Information Systems*, 2021, 2021. Cited by: 8; All Open Access, Gold Open Access. [*Citado en pág. 11.*]

[15] Chengguang Xu, Wenyu Li, Jeffrey Too Chuan Tan, Zengqiang Chen, Han Zhang, and Feng Duan. Developing an identity recognition low-cost home service robot based on turtlebot and ros. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 4043–4048, 2017. [*Citado en pág. 12.*]

[16] Kailong Li and Haiyan Tu. Design and implementation of autonomous mobility algorithm for home service robot based on turtlebot. In *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, volume 5, pages 1095–1099, 2021. [*Citado en pág. 12.*]

[17] David St-Onge and Damith Herath. *The Robot Operating System (ROS1 &2): Programming Paradigms and Deployment*, pages 105–126. Springer Nature Singapore, Singapore, 2022. [*Citado en pág. 16.*]

[18] ¿quién controla tu robot? una evaluación de los mecanismos de seguridad de ros. In *2021 7.ª Conferencia Internacional sobre Automatización, Robótica y Aplicaciones (ICARA)*. [*Citado en pág. 16.*]

[19] Ritwik Gupta, Zachary T. Kurtz, Sebastian A. Scherer, and Jonathon M. Smereka. Open problems in robotic anomaly detection. *CoRR*, abs/1809.03565, 2018. [*Citado en pág. 16.*]

[20] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. [*Citado en pág. 16.*]

[21] David S Michal and Letha Etzkorn. A comparison of player/stage/gazebo and microsoft robotics developer studio. In *Proceedings of the 49th Annual Southeast Regional Conference*, pages 60–66, 2011. [*Citado en pág. 16.*]

[22] Walter Fetter Lages, Darlan Ioris, and Diego Caberlon Santini. An architecture for controlling the barrett wam robot using ros and orocos. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–8. VDE, 2014. [*Citado en pág. 17.*]

[23] Sinan Barut, Marco Boneberger, Pouya Mohammadi, and Jochen J Steil. Benchmarking real-time capabilities of ros 2 and orocos for robotics applications. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 708–714. IEEE, 2021. [*Citado en pág. 17.*]

[24] Paul Fitzpatrick, Elena Ceseracciu, Daniele E Domenichelli, Ali Paikan, Giorgio Metta, and Lorenzo Natale. A middle way for robotics middleware. *Journal of Software Engineering for Robotics*, 5(2):42–49, 2014. [*Citado en pág. 17.*]

[25] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013. [*Citado en pág. 17.*]

[26] Rui Wang, Xiaoge Li, and Shuo Wang. A laser scanning data acquisition and display system based on ros. In *Proceedings of the 33rd Chinese Control Conference*, pages 8433–8437, 2014. [*Citado en pág. 17.*]

[27] Karl Pauwels, Vladimir Ivan, Eduardo Ros, and Sethu Vijayakumar. Real-time object pose recognition and tracking with an imprecisely calibrated moving rgb-d camera. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2733–2740, 2014. [*Citado en pág. 17.*]

[28] M Basavanna, M Shivakumar, and KR Prakash. Navigation of mobile robot through mapping using orbbec astra camera and ros in an indoor environment. In *Recent Advances in Manufacturing, Automation, Design and Energy Technologies: Proceedings from ICoFT 2020*, pages 465–474. Springer, 2022. [*Citado en pág. 18.*]

[29] Francisco Javier Bernal Cerpa. Proyecto fin de grado grado en ingeniería de las tecnologías de telecomunicación intensificación: Sistemas de telecomunicación. [*Citado en pág. 18.*]

[30] José Gomes da Silva Neto, Pedro Jorge da Lima Silva, Filipe Figueredo, João Marcelo Xavier Natário Teixeira, and Veronica Teichrieb. Comparison of rgb-d sensors for 3d reconstruction. In *2020 22nd Symposium on Virtual and Augmented Reality (SVR)*, pages 252–261. IEEE, 2020. [*Citado en pág. 19.*]

[31] Análisis de algoritmo slam de robot móvil basado en lidar. In *Conferencia de control chino (CCC) de 2019.* [*Citado en págs. 19 y 24.*]

[32] Yuqing Yang and Takuya Azumi. Exploring real-time executor on ros 2. In *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–8. IEEE, 2020. [*Citado en pág. 19.*]

[33] George Stavrinos. Ros2 for ros1 users. *Robot Operating System (ROS) The Complete Reference (Volume 5)*, pages 31–42, 2021. [*Citado en pág. 20.*]

[34] Motor de comportamiento flexible basado en ros 2 para navegación flexible. [*Citado en pág. 20.*]

[35] Vincenzo DiLuoffo, William R Michalson, and Berk Sunar. The transition from ros 1 to ros 2. 1917. [*Citado en pág. 20.*]

[36] Gustavo Acosta, José Gallardo, and Ricardo Pérez. Arquitectura de control reactiva para la navegación autónoma de robots móviles. *Ingeniare. Revista chilena de ingeniería*, 24(1):173–181, 2016. [*Citado en pág. 20.*]

[37] Oscar Enrique Goñi, José A Fernández León, and Nelson Acosta. Arquitectura de un controlador híbrido para navegación robótica en ambientes parcialmente conocidos. In *VIII Workshop de Investigadores en Ciencias de la Computación*, 2006. [*Citado en pág. 21.*]

[38] Antonio Javier Pérez Rodríguez et al. Desarrollo de sistema de navegación para vehículos autónomos terrestres utilizando ros. 2017. [*Citado en pág. 21.*]

[39] Adriana María Padilla Ericksen. Implementación y comparativa de algoritmos de control y planificación local para robots móviles utilizando ros. 2017. [*Citado en pág. 21.*]

[40] David Leonardo Martínez, Oscar Hernan Rojas Barreto, and William Fernando Bernal. Reconocimiento de espacios con optimización de trayectorias utilizando el robot "turtlebot3 burger". *Ingenio Magno*, 11(1):112–122, 2020. [*Citado en pág. 22.*]

[41] Iván Palacios Serrano, Christyan Cruz Ulloa, and Manuel Barraza Rodríguez. Análisis de los algoritmos de planificación de trayectorias rrt, prm y voronoi en la solución de un laberinto modular controlado por una plataforma de dos gdl. *Ingeniare. Revista chilena de ingeniería*, 30(1):157–170, 2022. [*Citado en pág. 22.*]

[42] ANGELLO JAHIR HOYOS IBARRA. Planeación de movimientos con índices métricos. 2016. [*Citado en pág. 22.*]

[43] Douglas Viana, Israel Amaral, Adriano Rezende, Héctor Azpúrua, Gustavo Pessin, and Gustavo Freitas. Análise e comparação dos algoritmos rrt* e dijkstra para planejamento de caminhos em terrenos acidentados. In *Simpósio Brasileiro de Automação Inteligente-SBAI*, volume 1, 2021. [*Citado en pág. 22.*]

[44] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International journal of robotics research*, 32(9-10):1164–1193, 2013. [*Citado en pág. 23.*]

[45] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Berlin, Germany, 2013. [*Citado en pág. 23.*]

[46] Sachin Chitta. Moveit!: an introduction. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 3–27, 2016. [*Citado en pág. 23.*]

[47] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. [*Citado en pág. 23.*]

[48] Zachary Kingston and Lydia E Kavraki. Robowflex: Robot motion planning with moveit made easy. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3108–3114. IEEE, 2022. [*Citado en pág. 23.*]

[49] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783, 2021. [*Citado en pág. 24.*]

[50] Kirill Krinkin, Artyom Filatov, Art yom Filatov, Artur Huletski, and Dmitriy Kartashov. Evaluation of modern laser based indoor slam algorithms. In *2018 22nd Conference of Open Innovations Association (FRUCT)*, pages 101–106. IEEE, 2018. [*Citado en pág. 25.*]

[51] Kuisong Zheng, Guangda Chen, Guowei Cui, Yingfeng Chen, Feng Wu, and Xiaoping Chen. Performance metrics for coverage of cleaning robots with mocap system. In *Intelligent Robotics and Applications: 10th International Conference, ICIRA 2017, Wuhan, China, August 16–18, 2017, Proceedings, Part III 10*, pages 267–274. Springer, 2017. [*Citado en pág. 25.*]

[52] Amir Ibrahim, Mani Golparvar-Fard, and Khaled El-Rayes. Metrics and methods for evaluating model-driven reality capture plans. *Computer-Aided Civil and Infrastructure Engineering*, 37(1):55–72, 2022. [*Citado en pág. 25.*]