Estudio para la mejora de la precisión mediante la simultánea localización y mapeado para la robótica

David Valencia

1 de julio de 2024

Resumen–
ESTE documento aborda la mejora de la precisión en la localización de robots mediante la técnica de SLAM. Se establecen objetivos clave para implementar técnicas de mapeo y localización simultánea, vital en la robótica para espacios reducidos y robots de tamaño reducido. Se desarrolla un prototipo de robot con capacidades de movimiento y integración de datos sensoriales para una precisa localización en un plano 2D, utilizando el framework ROS2 y código en Arduino.

Palabras clave– Arduino, hardware, localización, precisión, robot, ROS2, SLAM (Simultaneous Localization And Mapping), técnica.

Abstract–
THIS document focuses on enhancing robot localization accuracy through SLAM technique. Key objectives are outlined for implementing simultaneous mapping and localization techniques, crucial in robotics for confined spaces and small-scale robots. A robot prototype is developed with movement capabilities and integration of sensor data for precise 2D localization, utilizing ROS2 framework and Arduino code.

Keywords– Arduino, hardware, localization, precision, robot, ROS2, SLAM (Simultaneous Localization And Mapping), technique

1 INTRODUCCIÓN

ESTE documento se centra en la mejora de la precisión de localización de un robot mediante la técnica de SLAM (técnicas de mapeo y localización simultánea). La precisión es crucial en ámbitos como la robótica para espacios cerrados y para robots pequeños. Toda la extensión del trabajo se realiza en el contexto de una empresa llamada Movvo, especialmente en el departamento de robótica.

El objetivo principal de este trabajo es crear un prototipo de robot que pueda moverse e integrar la información recibida por sensores para determinar con precisión su localización en un plano 2D. Para ello, el robot utilizará el framework ROS2 y código en Arduino.

El documento incluye la definición de los objetivos del trabajo, la metodología empleada, los resultados obtenidos tras la implementación de ciertas soluciones y las conclusiones alcanzadas.

• E-mail de contacto: davidvalenciac01@gmail.com
• Mención realitzada: Enginyeria de Computadors
• Trabajo tutorizado por: Bernat Gastón Braso (Department of Information and Communications Engineering)
• Curso 2023/24

2 ESTADO DEL ARTE

2.1. Tipos de robots

LOS robots móviles se pueden clasificar según la tecnología que implementan en su sistema de navegación, dividiéndose en AGV (Automated Guided Vehicles) y AMR (Autonomous Mobile Robots) [9]. Aunque nuestro proyecto se centra en la localización y no en la navegación, ambas funcionalidades están estrechamente relacionadas, ya que no puede haber navegación sin una localización precisa.

Tanto los AGV como los AMR son tipos de robots móviles utilizados en la automatización industrial, pero presen-

tan diferencias significativas en cuanto a tecnología, flexibilidad y aplicaciones.

Los AGV se llaman guiados porque su tecnología se basa en recibir una trayectoria predefinida. Siguen rutas fijas marcadas en el suelo, que pueden ser cintas magnéticas, cables enterrados, líneas pintadas o bandas reflectantes. En su más reciente implementación los AGV se basan en mapas virtuales. Su movimiento está restringido a estas rutas y cualquier cambio requiere modificaciones físicas en el entorno (salvo en los basados en mapas virtuales), aunque mediante la sensórica que dispongan podrán ser capaces de detectar obstáculos y detenerse de ser necesario. Esta implementación puede ser costosa y llevar tiempo debido a la necesidad de instalar y mantener las guías físicas. Los AGV son más adecuados para aplicaciones con rutas repetitivas y predecibles, como el transporte de materiales en líneas de montaje o entre áreas de almacenamiento y producción.

Por otro lado, los AMR utilizan tecnologías avanzadas de navegación, como LIDAR, cámaras, sensores y algoritmos de mapeo, para moverse de manera autónoma y flexible por el entorno. Los AMR tienen un planificador que les permite planificar rutas mediante algoritmos sin intervención humana. Pueden crear y ajustar rutas en tiempo real, evitando obstáculos y adaptándose a cambios en el entorno. Son altamente flexibles y adaptables, ideales para entornos donde se requiere mayor flexibilidad y adaptabilidad, como almacenes, centros de distribución y plantas de fabricación con diseños cambiantes. La implementación de un sistema AMR es generalmente más rápida y menos costosa, ya que no requiere modificaciones físicas extensas [10].

Actualmente, ambas tecnologías coexisten y cada una tiene sus aplicaciones. La facilidad de los AMR para localizarse sin infraestructura adicional es especialmente atractiva, ya que el mantenimiento de balizas y la poca resiliencia frente a cambios en el entorno no son sostenibles a largo plazo para un proyecto de mejora de localización. Por esta razón, optaremos por un modelo AMR.

2.2. Modelo cinemático del robot

RESPECTO al tipo del robot físico, necesitamos uno que cumpla con ciertos parámetros. El robot debe ser robusto, poco costoso y fácil de implementar. A partir de documentos como [7], encontraremos un modelo que encaje con nuestras necesidades y que ofrezca la facilidad de implementación que nos permitirá centrarnos en los objetivos de este trabajo.

2.3. Componentes necesarios para el robot

PARA construir un robot que pueda localizarse en un mapa que él mismo haya creado, necesitaremos una combinación de componentes de hardware y software que permitan la percepción, la navegación y el control del robot. En cuanto al hardware, uno de los componentes fundamentales será la unidad de procesamiento, que puede ser un microcontrolador o una computadora a bordo con suficiente capacidad de procesamiento para manejar tareas de SLAM (Simultaneous Localization and Mapping) y navegación.

También necesitaremos sensores de percepción, como LIDAR para obtener datos de distancia precisos y detallados del entorno, cámaras RGB o RGB-D para la visión computarizada, reconocimiento de objetos y navegación visual, sensores ultrasónicos para la detección de obstáculos a corta distancia, y una IMU (Inertial Measurement Unit) que incluye acelerómetros y giroscopios para medir la orientación y la aceleración del robot.

Además, los sensores de movimiento, como los encoders de rueda para medir la rotación de las ruedas y calcular la distancia recorrida, serán esenciales. Los actuadores, como los motores y controladores de motor para la propulsión y dirección del robot, y los servos para mover componentes específicos del robot, también serán necesarios, junto con una fuente de energía, como baterías recargables con suficiente capacidad para alimentar todos los componentes durante el tiempo necesario de operación.

2.4. Framework de robótica ROS2

SEGUIMOS con otro de los núcleos del proyecto, ROS2. ROS 2 (Robot Operating System 2) es una plataforma de software de código abierto diseñada específicamente para el desarrollo de sistemas robóticos. Surgió como una evolución de la primera versión de ROS para abordar sus limitaciones y mejorar su adaptabilidad a una variedad de aplicaciones y entornos [5].

Algunas de las características clave de ROS 2 incluyen:

- Arquitectura Distribuida: ROS 2 está diseñado desde cero para soportar sistemas distribuidos. Utiliza el middleware de comunicación DDS (Data Distribution Service) para permitir la comunicación entre nodos de forma eficiente y confiable, incluso en entornos heterogéneos.

- Multiplataforma: ROS 2 es compatible con una amplia gama de plataformas, incluyendo sistemas operativos como Linux y macOS, así como diferentes arquitecturas de hardware. Cabe destacar que ROS2 es principalmente usado en Linux por su facilidad de uso de terminal y compatibilidad con los paquetes.

- Soporte para Diferentes Lenguajes de Programación: ROS 2 está diseñado para admitir múltiples lenguajes de programación. Actualmente, cuenta con soporte para C++, Python, en los que se desarrolla principalemnte. Estos lenguajes tienen documentación, tutoriales, bibliotectas y herramientas actualizadas para el desarrollo en ROS2.

- Mayor Modularidad: ROS 2 está diseñado con una arquitectura más modular, lo que permite una mayor flexibilidad y reutilización de componentes de software. Esto facilita el desarrollo, la integración y el mantenimiento de sistemas robóticos complejos.

- Mejoras en la Seguridad y la Robustez: ROS 2 incluye características diseñadas para mejorar la seguridad y la robustez de los sistemas robóticos. Esto incluye mecanismos para la gestión de errores, la recuperación ante fallos y la autenticación de nodos.

- Herramientas de Desarrollo Mejoradas: ROS 2 proporciona una variedad de herramientas de desarrollo y depuración que facilitan el proceso de desarrollo de software para robots. Esto incluye herramientas para la vi-

sualización de datos, la monitorización del sistema y la depuración de nodos.

2.5. Algoritmos SLAM

OTRO tema importante del proyecto es la localización y mapeo simultáneos (SLAM) que son tareas fundamentales para la navegación autónoma de robots móviles. SLAM permite a un robot construir un mapa de su entorno mientras estima su posición dentro de ese mapa. En este trabajo, se presenta una revisión del estado del arte en técnicas SLAM 2D para robots móviles en el marco del Robot Operating System (ROS)[3]. Las técnicas SLAM 2D se basan en sensores de distancia como LiDAR para estimar la posición del robot y construir un mapa del entorno. También existen las técnicas VSLAM que utilizan la visión por computador para realizar el proceso de localización y mapeado simultáneo a partir de imágenes recibidas por el computador. Aún habiendo técnicas de precisión comprobada se demuestra que las técnicas basadas en localización a partir de grafos (GBL) son superiores a las basadas en filtro de partículas 3D (PF) [6]. Es por esto que nos centraremos en las alternativas basadas en grafos.

Algunas de las técnicas SLAM 2D más populares son:

- Gmapping: utiliza un filtro de partículas Rao-Blackwell para estimar la pose del robot y construir un mapa de celdas.
- Hector SLAM: utiliza un método de Gauss-Newton para el registro de escaneo y un filtro de Kalman para la estimación de la pose del robot.
- Cartographer: desarrollado por Google, utiliza un filtro de partículas asíncrono y un esquema de optimización basado en grafos para generar mapas de alta calidad.
- Slam_toolbox: un método reciente que se centra en la facilidad de uso y el mapeo en línea, utilizando un filtro de Kalman no lineal y un esquema de fusión de mapas.
- Iris_lama: otro método reciente que se centra en el mapeo en línea, utilizando un filtro de partículas Rao-Blackwell y un esquema de fusión de mapas.

Las técnicas SLAM 2D se pueden comparar en base a varios criterios, como la precisión de la estimación de la pose, la calidad del mapa generado, la complejidad computacional y la facilidad de uso.

- Precisión: Cartographer generalmente ofrece la mayor precisión en la estimación de la pose y la generación de mapas.
- Calidad del mapa: Cartographer y Slam_toolbox generan mapas de alta calidad, mientras que Gmapping y Hector SLAM pueden producir mapas menos detallados.
- Complejidad computacional: Cartographer es la más compleja computacionalmente, mientras que Gmapping y Hector SLAM son las menos exigentes.

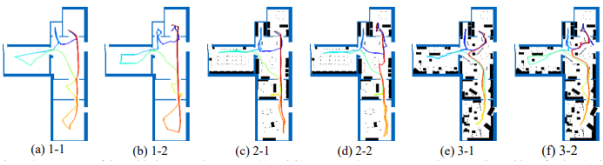

Fig. 1: Escenarios para la comparación de figura 2

Method	1-1		1-2		2-1		2-2		3-1		3-2	
AMCL	8,49	0,44	8,47	0,50	33,68	2,71	37,44	3,26	63,04	3,29	65,12	3,37
GMCL	8,27	0,24	7,86	0,24	24,27	2,57	52,38	4,37	66,60	3,70	126,91	4,46
SLAM Toolbox	**3,69**	**0,17**	**3,95**	**0,17**	28,69	1,50	23,57	1,50	**37,84**	**1,34**	**37,96**	**1,70**
Cartographer	11,89	0,22	4,04	0,21	**7,19**	**0,15**	**4,11**	**0,21**	-	-	-	-

Fig. 2: Comparación de alternativas SLAM, PF y GBL, con precisón traslacional en cm y grados angulares RMSE

- Facilidad de uso: Slam_toolbox e Iris_lama están diseñadas para ser fáciles de usar y ofrecen interfaces intuitivas.

Las anteriormente mencionadas alternativas de SLAM para ROS están disponibles para la primera versión de ROS, nosotros al usar ROS2 centraremos nuestra atención en Cartographer y Slam-toolbox. Los resultados que estos dos algoritmos pueden presentar son dispares como lo podemos ver en artículos como [3], donde se muestra que Cartographer es superior en mapeado a Slam, y [6] donde se muestra que Slam es superior a Cartographer en mapeado. El primer estudio nos muestra una comparativa en la que Cartographer tiene mayor precisión y es más robusto como algoritmo, aunque también menciona su principal problemática que es la necesidad de altas prestaciones computacionales para funcionar en tiempos aceptables para la localización en tiempo real. El segundo estudio muestra mediante métricas, en una tabla comparativa (como se puede ver en las figuras 1 y 2 pertenecientes a [6]) y con diferentes entornos , como Slam-toolbox tiene mejor rendimiento que Cartographer y es más eficiente en cuanto al cálculo de la localización. Por esta razón escogeremos Slam-toolbox para nuestro proyecto.

3 OBJETIVOS

EL objetivo principal del trabajo es estudiar las diferentes alternativas que se tiene para mejorar la precisión de la localización de un robot mediante la implementación de técnicas tipo SLAM (Simultaneous Localization And Mapping) en entornos cerrados e implementando el framework de ROS2 para la programación del robot.

3.1. Metodología y Planificación de subobjetivos

PARA llevar a cabo el objetivo principal del proyecto necesitamos cumplir con ciertos subobjetivos:

- Idear modelo del robot: En esta etapa, estableceremos el diseño y las especificaciones del robot diferencial. Esto implica definir su estructura mecánica, como la disposición de las ruedas y los motores, así como determinar sus dimensiones, peso y capacidades funcionales. Este proceso tomará aproximadamente 10 horas.

Plantearemos el modelo del robot después de haber adquirido el conocimiento necesario para decidir qué tipo de tecnología (AGV o AMR) usaremos y qué tipo de modelo cinemático empleará el robot.

- Obtención del HW: Una vez que tengamos el diseño del robot, procederemos a adquirir los componentes de hardware necesarios para construirlo. Esto incluye motores, ruedas, chasis, sensores (como encoders, IMUs, sensores de proximidad), controladores de motor, placas de circuito (como Arduino o Raspberry Pi), baterías, entre otros. Aproximadamente 10 horas permitirán realizar la búsqueda y compra de componentes que se ajusten al modelo de robot seleccionado.

- Montado del robot: En esta fase, ensamblaremos físicamente todos los componentes del robot de acuerdo con el diseño establecido. Esto implica conectar los motores a las ruedas, fijar los sensores en la ubicación deseada, instalar la placa controladora y la fuente de alimentación, y asegurarse de que todo esté correctamente ensamblado y funcione adecuadamente. Estimamos que este proceso tomará 30 horas. Seguiremos el procedimiento de montar todos los componentes sobre la base del robot, haciendo huecos con un taladro donde sea necesario y atornillando sensores y actuadores. También haremos huecos para pasar los cables y usaremos una pistola de silicona para adherir elementos no perforables, como las baterías.

- Creación de drivers: Los drivers son programas o códigos que permiten la comunicación entre el hardware del robot y el software que lo controla. En esta etapa, desarrollaremos o configuraremos los drivers necesarios para interactuar con los componentes específicos del robot, como los motores, sensores y otros dispositivos electrónicos. Debido a nuestra limitada experiencia con Arduino y los posibles problemas que puedan surgir al programarla, asignaremos 60 horas a esta tarea. Los drivers de nuestra Arduino deberán ser capaces de mover los motores hacia adelante o hacia atrás, corregir la velocidad si es necesario y enviar y recibir datos de velocidad con el PC.

- Comunicación Arduino-PC: Para permitir la interacción entre el robot y un PC, estableceremos un canal de comunicación entre la placa controladora del robot (por ejemplo, Arduino) y el PC. Esto se puede lograr utilizando diversos protocolos de comunicación, como USB, Bluetooth, Wi-Fi o UART. Desarrollaremos los programas tanto en el PC como en el Arduino para gestionar esta comunicación de manera eficiente. Debido a la necesidad de investigar cómo comunicarnos con el PC desde la Arduino y resolver posibles problemas de buffers o sincronización, asignaremos 40 horas a esta tarea. Estudiaremos las tecnologías de comunicación disponibles para transmitir las velocidades de las ruedas de manera óptima y posteriormente implementaremos la solución en el código del driver de Arduino a través del IDE.

- Creación de funcionalidades en ROS2: ROS (Robot Operating System) es un conjunto de herramientas y bibliotecas ampliamente utilizadas en la robótica para facilitar el desarrollo de software. En esta etapa, crearemos los nodos y paquetes de ROS2 que permitirán controlar y coordinar las diferentes funciones del robot, como la navegación, la percepción del entorno, el control de los actuadores y la comunicación con otros sistemas. Asignaremos 70 horas a esta tarea, ya que será la parte central del trabajo. Primero nos documentaremos sobre cómo funciona el framework de ROS2, realizaremos cursos y revisaremos los diferentes repositorios de la empresa Movvo para entender cómo avanzar en caso de encontrar problemas. Empezaremos creando funcionalidades de envío de datos por joystick para interactuar con el robot, transmitiendo los datos recibidos al Arduino y recibiendo información de este (otra funcionalidad). Describiremos las características de nuestro robot para que sea localizable en el plano 2D y desarrollaremos funcionalidades que nos permitan realizar la localización con SLAM.

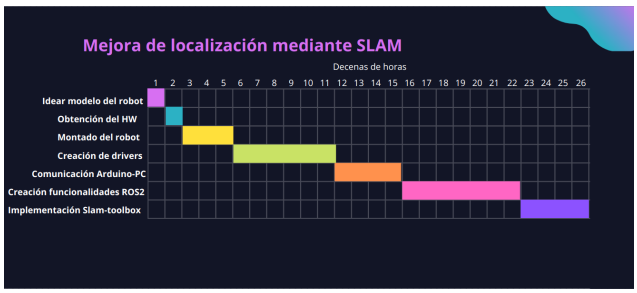

Fig. 3: Planificación en decenas de horas

- Implementación de Slam-toolbox: SLAM (Simultaneous Localization and Mapping) es una técnica utilizada para que un robot pueda construir un mapa de su entorno y al mismo tiempo localizarse dentro de este mapa. La implementación de SLAM en el robot implicará utilizar algoritmos y bibliotecas especializadas para procesar los datos de los sensores (como cámaras, LIDAR o ultrasonidos) y estimar la posición y orientación del robot en relación con su entorno. La Slam-toolbox es una herramienta específica que facilita la implementación de SLAM en ROS2, proporcionando nodos y algoritmos predefinidos para realizar esta tarea de manera eficiente. Asignaremos 40 horas para implementar el algoritmo SLAM. Para lograr esto, nos informaremos a través de cursos y repositorios de Github que contengan ejemplos o diagramas que nos guíen en el proceso [11].

Todas las funcionalidades que tienen que ver con código serán implementadas desde Visual Studio Code, mientras que las relacionadas con Arduino, como la creación de drivers de motor, serán desarrolladas desde Arduino IDE. Estas dos partes del proyecto estarán guardadas y separadas por funcionalidad en un repositorio de Github, donde cada funcionalidad estará en una rama diferente. Una vez que una funcionalidad haya sido testeada, se hará un merge a la rama de desarrollo, desde la cual se añadirá la nueva funcionalidad al fichero que ejecuta todo el programa del robot.

4 METODOLOGÍA

La metodología que hemos utilizado para la creación del proyecto ha sido SCRUM [1] desde el primer momento. Esta metodología ha contribuido significativamente mediante la reunión diaria, Daily Scrum, en la que se exponían de forma rápida los avances realizados y los problemas encontrados. De esta manera, era sencillo registrar un progreso adecuado y hacer consciente al equipo de las dificultades que necesitaban ayuda para ser solventadas. Tras la identificación de los problemas, organizábamos reuniones para revisarlos, definiendo de la manera más rápida y clara posible cómo ubicar a nuestros compañeros eficientemente en nuestro escenario. Se probaban diferentes posibles soluciones y, en caso de no encontrar solución, se marcaba un camino a seguir para averiguar de dónde provenía el comportamiento inesperado. Además, cada dos semanas teníamos reuniones más completas, las reuniones de Sprints.

Para desarrollar el trabajo exitosamente, es necesario cumplir con ciertos aspectos. Es indispensable contar con los fondos de la empresa para poder obtener los componentes del robot. En este aspecto, el departamento de robótica ya cuenta con la mayoría de las piezas que se necesitarán para el proyecto. Por otra parte, es necesario el conocimiento en el framework de ROS2, programación para Arduino y comunicación por puerto serie, así como conocimiento sobre la información recopilada de sensores (odometría, IMU, LiDAR). Esta ha sido la manera de conseguir los objetivos del proyecto.

5 DESARROLLO DEL TRABAJO

5.1. Idear modelo del robot

El robot que hemos creado es un robot diferencial. Esto significa que consta de dos ruedas motrices delanteras y una rueda caster trasera. Esta configuración permite que el robot gire sobre su propio eje sin necesidad de realizar maniobras complicadas. El robot tiene una placa metálica sobre la cual se montará todo el hardware.

5.2. Obtención de HW

Los componentes del robot son proporcionados por la empresa con la que se desarrolla el trabajo final de grado. El departamento de robótica de la empresa ofrece lo necesario para crear un robot capaz de localizarse en un plano 2D. Estos componentes incluyen una base metálica para montar todos los elementos, dos servomotores de rotación continua, dos controladores de motor (**L298N**), cuatro ruedas para asegurar el agarre en diferentes superficies, una **Arduino Mega 2560** para controlar el hardware, el cableado necesario, una batería capaz de alimentar los motores y la Arduino, encoders para los motores, un pulsador para encender el robot, un lidar para el mapeo del entorno (**A2M7**), un mini PC para controlar las funcionalidades del robot en **ROS2** (**NUC** de **Intel**) y una fuente de voltaje para alimentar la batería del robot.

Con el avance del proyecto, hemos visto la necesidad de cambiar ciertos elementos. Las cuatro ruedas iniciales se han cambiado por dos de las iniciales para los motores y una rueda caster trasera para implementar un robot diferencial.

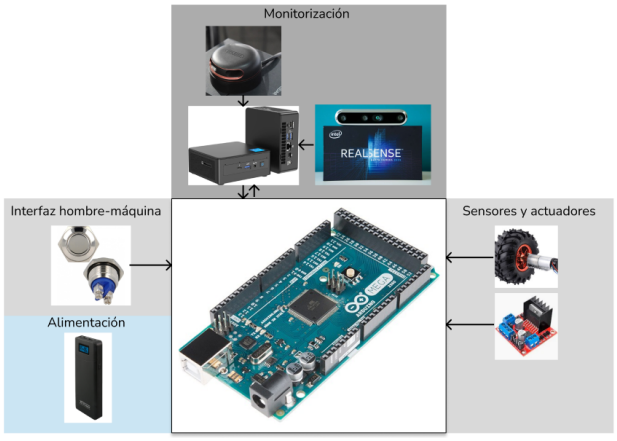

Fig. 4: Diagrama de componentes

Inicialmente, no se había considerado la falta de precisión en la odometría, como los errores en los giros o el deslizamiento de las ruedas en ciertas superficies. Por ello, es indispensable acompañar el cambio a la rueda caster con la adición de un sensor **IMU** (Inertial Measurement Unit). Este sensor, junto con la odometría, proporcionará una mejor localización. Inicialmente, la IMU era una con magnetómetro (**WT901C-485**), pero resultó ser defectuosa tras integrarla con el resto del software. Realizamos comprobaciones visuales de cómo interpretaba la orientación al girarlo manualmente y analizamos los datos que proporcionaba el sensor. Fue una tarea complicada, ya que tuvimos que probar múltiples alternativas para solucionar el problema. Finalmente, usamos otro sensor IMU integrado en la cámara **D455 Realsense** de **Intel**.

Inicialmente, teníamos la intención de aprovechar la batería que anteriormente hacía funcionar una versión sencilla del mismo robot, pero debido al paso del tiempo, la autonomía que esta batería ofrecía era de 5 minutos como máximo. En las primeras fases del trabajo, utilizamos una fuente de voltaje constantemente conectada al robot a 19V y 1A. Con el tiempo, vimos la necesidad de comprar una nueva batería o utilizar un power bank para alimentar la NUC. El robot necesita ser alimentado a 19V para la NUC y a 12V para los servomotores (**75:1 Metal Gearmotor 25Dx69L mm HP 12V con 48 CPR Encoder, item 4846 Pololu**). Como los servos reciben el voltaje mediante los controladores de motor, requeriría mayor esfuerzo cambiar la batería y el cableado, por lo que optamos por integrar un power bank con un conector personalizado.

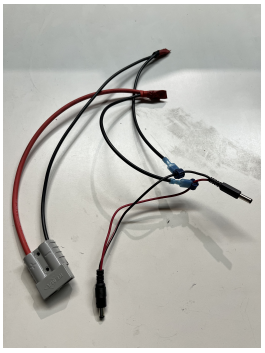

Fig. 5: Conector soldado para alimentar robot y NUC

5.3. Montado del robot

La placa metálica que hace de base del robot tiene la Arduino Mega 2560, los encoders, los motores, la batería del robot y la rueda caster, además de todo el cableado necesario, en la parte inferior de esta placa. En la parte superior está la cámara, el lidar, la NUC, el pulsador y el powerbank.

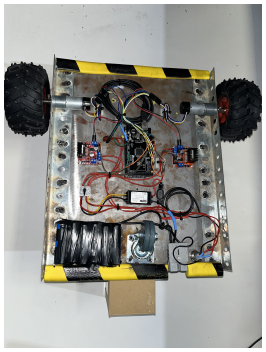

Fig. 6: Modelo de Robot y cableado

5.4. Creación de drivers

Incluye la definición del **PINOUT** utilizado en la **Arduino**, la creación de variables de control para el driver, la configuración de los pines e interrupciones, las funciones controladoras de encoders con filtros pasabajos y promedios para limpiar la señal leída por los encoders de ambas ruedas, funciones para el control de timeouts en la comunicación serie con la **NUC**, funciones de escritura por el puerto serie, controladores **PI** para rectificar la velocidad de los motores y obtener la velocidad deseada aunque haya resistencia en la rueda, algoritmos de regresión para la obtención de **PWM** en función de la velocidad angular deseada y la implementación de todas las funciones dentro de la función **loop**. Las velocidades se tratan en base a 100 debido a su sencillez en comparación con el paso de decimales por puerto serie.

5.5. Comunicación Arduino-PC

El protocolo de comunicación que hemos utilizado para comunicar la parte de **ROS2** con el driver del motor del robot es el puerto Serie. La conexión se establece desde el **PC** con **ROS2** definiendo el dispositivo que se conectará, en nuestro caso la **Arduino** por el puerto **/dev/ttyACM0**, utilizando el protocolo serie, con un baudrate de 230400 y un timeout de 0.1. La razón de esta comunicación es enviar la velocidad deseada al robot y recibir el feedback de la velocidad a la que van las ruedas de nuestro robot así como enviar esta velocidad a otros paquetes del robot para calcular la posición en la que podríamos estar (Odometría). Posteriormente, utilizaremos la información recibida de los encoders para fusionarla con diferentes sensores (**IMU** y **Lidar**). Este puerto ha presentado múltiples problemas, como la necesidad de que el **baudrate** sea el mismo en ambas partes de la comunicación, el **overflow** del buffer del puerto serie que llevaba a comportamientos anómalos, el manejo de decimales y los timeouts.

5.6. Creación de funcionalidades en ROS2

ROS2 es un framework muy útil para la robótica, ya que permite separar las funcionalidades del robot en paquetes y facilita la comunicación entre ellas de manera visual, utilizando aplicaciones que grafican el comportamiento del entorno que estamos creando [5]. Las funcionalidades, o paquetes, en las que dividiremos el comportamiento del robot son: **agl_arduino**, donde especificamos el driver del Arduino; **agl_bringup**, desde donde lanzamos todos los paquetes del robot; **agl_description**, donde definimos cada componente del robot para usar las transformadas de sensores y partes del robot en un plano; **agl_joystick**, para controlar el robot desde un gamepad; **agl_lidar**, para realizar llamadas a submódulos que controlan el lidar; **agl_odometry**, para recibir el feedback de las velocidades del robot desde **agl_serial_interface** y calcular la posición y orientación basándose en las velocidades angulares de las ruedas del robot; **ast_camera**, un paquete heredado de un repositorio de la empresa para controlar la IMU (permitiendo detectar la orientación con precisión); **ast_imu_filter_madgwick**, para transformar la información de una IMU sin magnetómetro a una que lo tenga, aplicándole un filtro para obtener mejores datos; **ast_robot_localization_ekf**, que integra el paquete open-source **robot_localization**, permitiendo fusionar datos como odometría e IMU. A partir de un fichero de configuración de este paquete, se define qué sensores se utilizan y desde qué topic (canal de datos) se deben escuchar los datos de estos sensores; y, por último, **rplidar_ros**, un paquete open-source que permite controlar el lidar RPLIDAR A2M7 instalado en nuestro robot.

5.7. Implementación de Slam-toolbox

La implementación de la técnica de SLAM requiere preparar el entorno de ROS2 para que sea posible. Para poder usar el mapeado con slam-toolbox [14], debemos asegurarnos de poder recibir el topic de **scan**, proveniente del paquete **agl_lidar**, así como el topic de **/EKF/Odometry**, que es el topic que sale del paquete **ast_robot_localization_ekf**.

Tuvimos problemas con el lidar y después de debuggear el problema optamos por probar con un lidar idéntico para descartar por completo un fallo en el software y vemos que con el nuevo RPLidar A2M7 funciona a la primera, por lo que trabajamos con este. Los nuevos componentes pueden obstaculizar el escaneado del lidar, por lo que tuvimos que cambiar el trípode de la cámara con la IMU por un tornillo que la ancla al cuerpo del robot. De esta manera, eliminamos el problema de que la inercia ocasionalmente moviera la cámara y el trípode, lo que arruinaba la información de la IMU desde ese momento.

También tuvimos que conseguir un conector tipo L para el puerto USB tipo C de la cámara Realsense D455 y fijamos el cable de la cámara para que no obstaculizara el escaneado. Para integrar completamente el paquete en el proyecto, tuvimos que solucionar un problema con los ejes de la cámara que podrían ocasionar fallos en la odometría al depender de la IMU presente en dicha cámara, así como rotar los ejes del lidar porque mapeaba al revés debido a una definición incorrecta de sus ejes en el paquete **agl_description**.

Al utilizar slam-toolbox, observamos un mapeado co-

rrecto siempre que el robot esté estático, pero cuando se mueve, crea mapas erróneos. El comportamiento observado es una superposición de nuevos mapas en lugar de completar el mapa del primer escaneado. Revisando foros del paquete slam-toolbox y consultando con compañeros de la empresa, llegamos a la conclusión de que lo más probable es que nuestra odometría sea errática, lo que produce un mal mapeado.

5.8. Validación de odometría

La validación de nuestra odometría es necesaria por diferentes motivos. Por un lado, necesitamos cuantificar qué tan bien nos localizamos únicamente mediante sensórica, sin contar con mapeado, ya que el objetivo del trabajo es mejorar esta localización al implementar técnicas de SLAM. Inicialmente, se realizaron pruebas utilizando un visualizador de ROS2 llamado **Rviz2** para observar el comportamiento de la odometría. Estas pruebas incluían la localización después de avanzar ciertos metros, la velocidad a la que iba el robot y la precisión en los giros. Esta primera aproximación para validar la odometría no permite extraer métricas relevantes por lo que se busca una nueva manera de adquirir estos datos.

La segunda aproximación para la validación la haremos investigando el paquete **evo** [13] para ROS2. Este paquete es de gran utilidad, ya que nos permite graficar las trayectorias de nuestro robot, así como las trayectorias ideales, y realizar comparaciones cuantitativas mediante comandos como **evo_ape** o **evo_rpe** para obtener el *absolute pose error* o *relative pose error*, donde *pose* es una combinación de posición (x,y,z) y orientación en cuaterniones. **Evo** trabaja recibiendo ficheros de trayectorias como **bag/bag2**, **euroc**, **tum** y **kitti**.

Hemos logrado usar esta herramienta para graficar trayectorias en diferentes formatos, pero no hemos podido obtener las métricas a partir de evo debido a los diferentes timestamps que tienen las trayectorias cuando son recorridas en real y en simulación, esto será explicado más adelante. La obtención de las trayectorias también ha sido un desafío. Inicialmente, se definía una trayectoria ideal desde un PC distinto, donde se grababa la información de la trayectoria (topic **/odom** de una simulación en el software Gazebo a partir del repositorio de turtlebot3 para ROS2 distribución humble) y también grabábamos las velocidades que enviábamos (topic **/cmd_vel** también al turtlebot3). Como lo ideal sería que tanto la simulación de la trayectoria como la trayectoria real fueran lo más simultáneas posibles, se cambió el enfoque para hacerlo todo desde el mismo PC donde se ejecutaría la simulación y el proyecto **AGL**, mientras se grababan los topics **/odom** y **/EKF/Odometry** que definen estas trayectorias respectivamente.

Como ya hemos mencionado, el problema de no poder obtener las métricas desde el paquete **evo** se debe a un desalineamiento entre los timestamps de cada trayectoria al ser una simulada y otra real, presuntamente. Esta dificultad no se ha podido solventar, por lo que optamos por graficar la trayectoria ideal desde un script de **Python** [12] con planos dimensionados de manera similar a como se grafican con **evo**. Las trayectorias, también llamadas *ground truth*, son una línea recta de 5 metros de largo, un cuadrado de 2 metros de lado y un triángulo equilátero de 2 metros de lado.

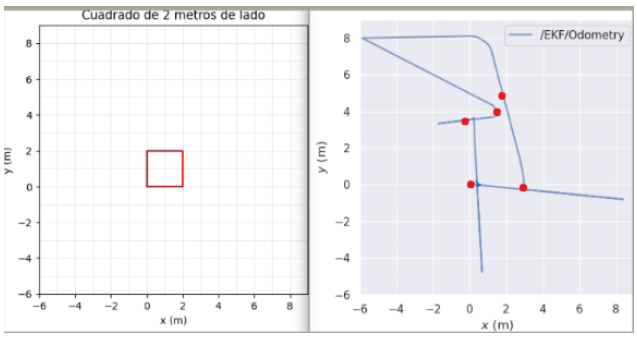

Fig. 7: Comparación de trayectoria deseada con real, se observa corrección de posición lenta

Luego, se han definido puntos de obtención de datos para la posterior comparación.

5.9. Uso de Slam-toolbox

Usar Slam-toolbox ha implicado estudiar cómo funciona el paquete de ROS2. Después de implementarlo en el proyecto y asegurarnos de que no haya errores, hemos configurado un fichero de parámetros que especifica cómo debería comportarse Slam-toolbox si lo ejecutamos desde ese fichero con el comando **ros2 launch slam_toolbox online_async_launch.py slam_params_file:=src/mapper_params_online_async.yaml use_sim_time:=false**. El proceso de mapeado ha sido detallado en el **README** del repositorio **Github** del proyecto, pero requiere iniciar el robot (instanciar la distribución de ROS2 y lanzar el fichero **ros2 launch agl_bringup start_launch.py**), iniciar **rviz2** de manera que podamos visualizar todos los topics que nos interesen analizar, como **grid** (genera una cuadrícula con cuadrados de 1 metro por 1 metro y ayuda a comprobar el comportamiento del robot), **TF** (para ver que tengamos todas las transformadas necesarias publicadas), **odometry** (necesaria para la ejecución de Slam-toolbox como input topic y para ver la orientación del robot), **LaserScan** (también necesaria para la ejecución de Slam-toolbox como input topic y ver qué detecta el lidar del robot y si va en consonancia con lo que mapeamos) y, por último, **Map** (para visualizar cómo generamos un mapa a partir del lidar).

Hemos tenido numerosos problemas con este apartado porque nos dimos cuenta de varios aspectos que necesitaban ser corregidos para poder usar Slam-toolbox. Como mencionamos en el apartado anterior, la odometría no es ideal, al menos la odometría generada al fusionar los datos IMU-Odometría mediante el paquete **ast_filter_madgwick** (convierte la información de la IMU de acelerómetro y giroscopio en un mensaje **imu** de ROS2 fusionable mediante **robot_localization**) y **robot_localization**. Al usar esta odometría y el mapeado mediante Slam-toolbox, nos encontramos con comportamientos erráticos donde el robot no navega como debería por el mapa y el mapeado no construye un mapa, sino que genera diferentes mapas para cada nueva posición/orientación del robot. Esto ha requerido un largo tiempo de depuración, que ha conllevado una mejor validación de la odometría para identificar de dónde proviene el problema. Desde la validación, corregimos los fallos posibles, pero la odometría fusionada seguía sin funcionar

correctamente. El paquete **madgwick** fue descartado como fuente del problema por su correcto funcionamiento en otro proyecto. El problema reside en **robot_localization** y, después de varias sesiones de prueba y configuración de este fichero de parámetros sin conseguir mejoras, se deja de lado la odometría fusionada para usar únicamente la odometría de las ruedas. De esta forma, comprobaremos la posición en **x** e **y** pero no la orientación, ya que no usaremos la IMU y la comparación carecería de sentido.

Para usar solo esta odometría, tuvimos que modificar el paquete de odometría para que publique una transformada en el topic **/odom** y que Slam-toolbox pueda identificar como fuente de odometría al topic **/Odometry/odom** (odometría de ruedas). Otro inconveniente que tuvimos fue la falta de documentación actualizada sobre cómo funciona Slam-toolbox. Para ejecutar el mapeado, debemos especificar el fichero de configuración de slam, y eso se hace con el argumento **params_file**, pero al hacerlo de esta manera se usaba un fichero por defecto diferente al que habíamos definido. Finalmente, encontramos en un foro de ROS2 que desde la distribución **Humble** de ROS2 el argumento es **slam_params_file**, algo que no está documentado en el repositorio de Github.

5.10. Integración de Nav2 (AMCL)

PARA comparar algoritmos que puedan mejorar la localización, podríamos utilizar algoritmos básicos como la odometría, pero esto no ofrecería una comparación útil en la actualidad. Necesitamos comparar los dos algoritmos más utilizados para la localización en ROS2. Por esta razón, vamos a integrar el paquete de **Nav2** para ROS2, que se basará en el mismo mapa que hayamos creado con Slam-toolbox (lanzamos AGL con Slam cargando el mapa y utilizamos el paquete de **map_saver** de Nav2 para obtener el **map.yaml**), pero la parte de localización la realizaremos usando **AMCL** (Adaptive Monte Carlo Localization) [15], también presente en Nav2.

Los algoritmos que estamos utilizando son comparables, pero tenemos en cuenta que aunque Slam-toolbox se localiza en el mapa, también lo está creando, y esto no será igual que un mapa que no se actualice a medida que pasa el tiempo. Se han investigado diferentes alternativas para encontrar un modo en que Slam-toolbox únicamente se localice en un mapa ya creado. Esto se podría lograr mediante el modo de localización, pero al ser un algoritmo que hace ambas cosas a la vez, tiene sentido que siga mapeando a partir de un mapa cargado. Investigamos los parámetros que se pueden modificar en el fichero de configuración de Slam-toolbox y encontramos un parámetro llamado **paused_new_measurements**. Sin embargo, modificarlo a **true** no detiene los nuevos escaneos.

No logramos detener el mapeo utilizando el paquete instalado a través del gestor de paquetes de Linux **apt**, ni con un plugin para **rviz2**, ni mediante una llamada al servicio **Pause.srv** de ROS2 proporcionado por Slam-toolbox, creamos un submódulo de **git** en nuestro proyecto para modificar una variable que controla si seguimos mapeando o no, sin detener por completo el mapeo mientras estamos en modo localización. No hemos encontrado información sobre cómo detener el mapeo o si existe tal posibilidad.

Debido al tiempo que potencialmente invertiríamos en investigar este problema, consideramos un escenario en el cual, aunque no logremos detener el mapeo, los algoritmos aún puedan ser comparables. Este escenario ocurre cuando tenemos un mapa completamente creado que no es dinámico.

6 RESULTADOS

6.1. Resultados de la validación de odometría

IDENTIFICAMOS que la odometría del topic **/EKF/Odometry**, que combina el sensor de orientación **IMU** y el sensor de vueltas de servomotores, realiza los giros con precisión suficiente, desviándose de la orientación en pocos casos, como se puede observar en la Fig. 8 y Fig. 9. La precisión en posición es generalmente incorrecta, y se ha aplicado un factor de corrección para la coordenada **x**, que era la más problemática. Este factor es de 1.22, lo que ha mejorado significativamente la precisión.

Identificamos un comportamiento inesperado en el que el robot sigue avanzando después de que hemos dejado de moverlo y corrige su posición más tarde, lo que genera trayectorias erróneas y se traduce en errores de odometría en determinados casos. Investigando este problema, llegamos a la conclusión de que está relacionado con la desconexión esporádica del sensor **IMU**, lo que genera errores sin seguir un patrón específico. Intentamos depurar este error revisando la frecuencia de envío de datos y modificándola, revisando el uso de CPU y memoria de la **NUC** y revisando las transformadas definidas en nuestro **urdf** que define nuestro robot, las cuales podrían causar comportamientos inesperados. Sin embargo, no pudimos solucionar este problema. Además, observamos la acumulación de errores en la odometría y el *drift* de las ruedas, que también puede considerarse normal.

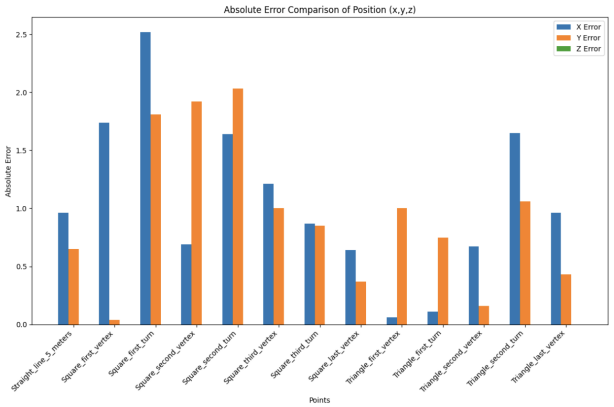

Fig. 8: Comparación del Error Absoluto (EA) de la posición con IMU

6.2. Mapeado mediante slam-toolbox

GRACIAS a la correcta configuración e implementación de los diferentes subobjetivos que nos hemos planteado conseguimos mapear correctamente el entorno. En Fig. 10 podemos ver el plano 2D de lo que será nuestro entorno de pruebas para sacar las métricas en las que se basa la validación de la odometría. Así podremos calcular la mejora y hacer comparaciones con los algoritmos.

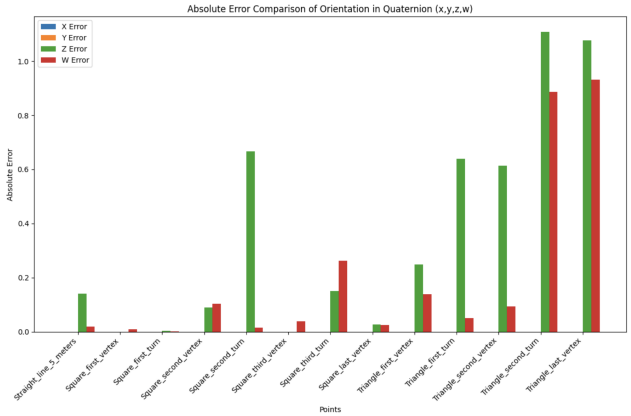

Fig. 9: Comparación del EA de la orientación con IMU

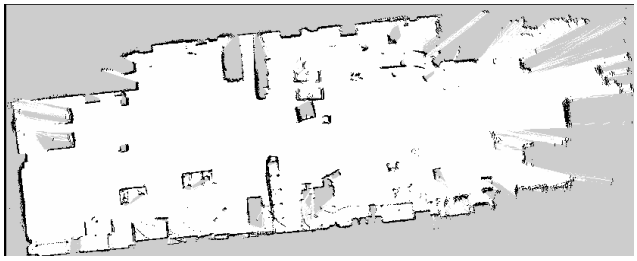

Fig. 10: Mapa de comparación para los algoritmos

6.3. Comparación Slam-toolbox, AMCL y /Odometry/odom

PAPERS como [6] indican que algoritmos como slam-toolbox deberían ser superiores en cuanto a localización, teniendo en cuenta que ambos algoritmos son punteros por lo que la diferencia será en todo caso pequeña. Al realizar la comparación esperamos ver esta mejora y una mejora en cualquier caso a la odometría del topic /Odometry/odom, partiendo en todo momento desde el mismo mapa. Las comparaciones han requerido un entorno de pruebas que no fuera dinámico, como ya hemos comentado en secciones anteriores del trabajo. Este nuevo entorno apenas cuenta con tránsito de personas o de robots de la empresa. El mapa que se crea es el siguiente:

Las figuras 11, 12 y 13 son diagramas comparativos de la precisión de localización mediante estas tres alternativas:

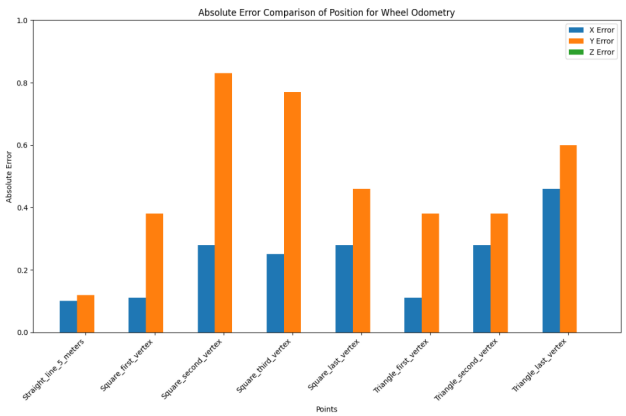

Fig. 11: Comparación del EA de la posición con Odometría de ruedas

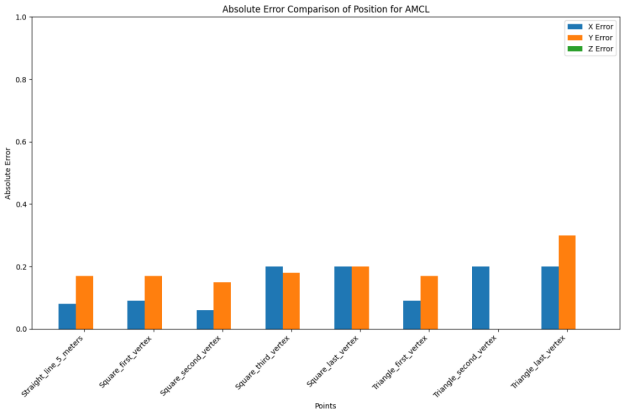

Fig. 12: Comparación del EA de la posición con AMCL

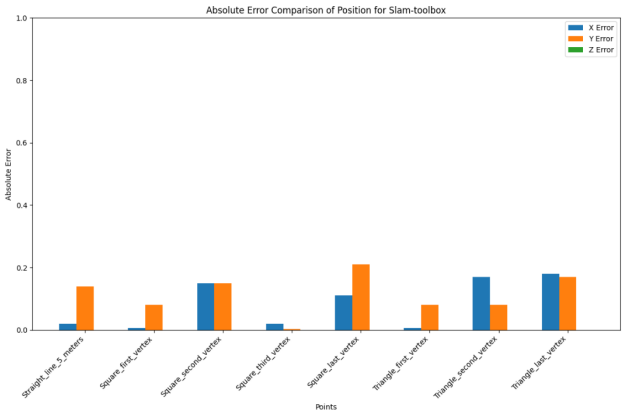

Fig. 13: Comparación del EA de la posición con slam-toolbox

7 CONCLUSIONES

ESTE trabajo ha descrito de forma secuencial como hemos diseñado y ensamblado un robot diferencial que consta de la sensórica necesaria para llevar a cabo tareas de localización mediante Arduino y el framework de ROS2. Hemos explicado la comunicación Arduino-PC y como hemos llevado a cabo la implementación de slam-toolbox así como del paquete Nav2 que incluye AMCL para su posterior comparación.

Aunque hemos podido llevar a cabo todos los objetivos que habíamos puesto en la planificación del proyecto es cierto que no todo lo que hemos desarrollado lo hemos implementado para la versión final. Ejemplos de esto son paquetes como el robot_localization que a partir del input de dos sensores puede fusionar los datos para tener una odometría filtrada mejorada. La IMU de la cámara d455 tampoco se usa debido a problemas constantes de desconexión que no se han podido solventar. Debido a esto hemos realizado las pruebas sin poder depender de la precisión de giro, nada catastrófico debido a que el objetivo final del trabajo era mejorar la localización centrándonos más en la posición que en la orientación .

Como conclusión de la comparación entre algoritmos podemos decir que al tener que sacar el error absoluto para medir el porcentaje de la precisión de los algoritmos, y este depender de una fórmula que pondría como divisor el error de referencia, no podremos usar el vértice de inicio de la trayectoria cuadrada ni de la triangular. Dicho esto vemos

que la precisión de la odometría únicamente con ruedas es del 77.5 %, que el del algoritmo AMCL (principal alternativa a algoritmos SLAM) es del 91.2 % y que la precisión de localización para Slam-toolbox en nuestro proyecto es del 94.9 % dejando claro que slam-toolbox podría mejorar en gran medida la localización de robots sin GPS. Está exactitud permitiría mantener una mejor precisión a trayectorias más largas ya que el drift natural, que sucede en localización, tendría menos penalidades en un algoritmo tan fiable.

8 Posibles mejoras

Futuras extensiones a este trabajo serían conseguir integrar el paquete de robot_localization exitosamente, de manera que no diera problemas de fusión de datos. Arreglar el problema de desconexión de transferencia de datos de la IMU de la cámara realsense d455 podría mejorar en gran medida la odometría filtrada y también sería un cambio prioritario para la mejora del proyecto.

Una mejora al trabajo sería un análisis exhaustivo de diferentes trayecorias, a poder ser más extensas y con mayor cantidad de giros de manera que podamos analizar si la precisión de localización se mantiene con el tiempo como debería ser posible gracias a algoritmos de tipo SLAM.

Aunque no afecte a la localización del robot, también sería interesante cambiar los servomotores de manera que tengamos más precisión en el control de su potencia y así conseguir realizar giros precisos. Esto nos permitiría integrar el paquete de Nav2 por completo y conseguiríamos un robot que pudiera navegar evitando obstáculos, haciéndolo perfecto para cualquier aplicación que le quisiéramos dar.

Referencias

[1] K. S. ., "SCRUM: AN AGILE PROCESS", Int. J. Res. Eng. Technol., vol. 02, n.º 03, pp. 337–340, marzo de 2013. Accedido el 21 de marzo de 2024. [En línea]. Disponible: https://doi.org/10.15623/ijret.2013.0203019

[2] W. Burgard et al., .ª comparison of SLAM algorithms based on a graph of relations,"2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, USA, 2009, pp. 2089-2095, doi: 10.1109/IROS.2009.5354691. keywords: Simultaneous localization and mapping;Mobile robots;Contracts;Particle filters;Intelligent robots;USA Councils;Error correction;Robot sensing systems;Laser modes;Transportation,

[3] Y. S. Castro, A. R. Martínez, and M. R. Torres, "(PDF) comparison of different techniques of 2D simultaneous ...," Comparison of different techniques of 2D simultaneous localization and mapping for a mobile robot by using ROS, https://www.researchgate.net/publication/344522360 (accessed Mar. 21, 2024).

[4] H. Temeltas y D. Kayak, "SLAM for robot navigation", IEEE Aerosp. Electron. Syst. Mag., vol. 23, n.º 12, pp. 16–19, diciembre de 2008. Accedido el 21 de marzo de 2024. [En línea]. Disponible: https://doi.org/10.1109/maes.2008.4694832

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette y W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild", Sci. Robot., vol. 7, n.º 66, mayo de 2022. Accedido el 21 de marzo de 2024. [En línea]. Disponible: https://doi.org/10.1126/scirobotics.abm6074

[6] [1] M. A. V. Torres, A. Braun, and A. Borrmann, "Occupancy grid map to pose graph-based map: Robust Bim-based 2D-lidar localization for lifelong indoor navigation in changing and dynamic environments," arXiv.org, https://arxiv.org/abs/2308.05443 (accessed Mar. 21, 2024).

[7] R. Siegwart, I. R. Nourbakhsh, R. C. Arkin y D. Scaramuzza, Introduction to Autonomous Mobile Robots. MIT Press, 2011.

[8] D. S. Dawoud y P. Dawoud, Serial Communication Protocols and Standards. River Publ., 2022.

[9] Y. Bekishev, Z. Pisarenko y V. Arkadiev, "FMEA Model in Risk Analysis for the Implementation of AGV/AMR Robotic Technologies into the Internal Supply System of Enterprises", Risks, vol. 11, n.º 10, p. 172, septiembre de 2023. Accedido el 22 de marzo de 2024. [En línea]. Disponible: https://doi.org/10.3390/risks11100172

[10] D. Cubillo Llanes. "NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE MEDIANTE UNA CÁMARA LIDAR". https://repositorio.comillas.edu. Accedido el 22 de marzo de 2024. [En línea]. Disponible: https://repositorio.comillas.edu/xmlui/bitstream/handle/11531/62114/TFM-CubilloLlanes,Diego.pdf?sequence=1

[11] S. Macenski, "On Use of SLAM Toolbox", en ROSCon2019, Macau, China, 31 de octubre–1 de noviembre de 2019. Mountain View, CA: Open Robot., 2019. Accedido el 22 de marzo de 2024. [En línea]. Disponible: https://doi.org/10.36288/roscon2019-900903

[12] Tosi, S. (2009) Matplotlib for Python Developers, Google Libros. Disponible: https://books.google.co.jp/books?hl=es (Accessed: 21 April 2024).

[13] Grupp, M. (2024) MichaelGrupp/Evo: Python package for the evaluation of odometry and slam, GitHub. Available at: https://github.com/MichaelGrupp/evo?tab=readme-ov-file (Accessed: 21 April 2024).

[14] Macenski, S. (2024) Stevemacenski/slam_toolbox: Slam Toolbox for lifelong mapping and localization in potentially massive maps with Ros, GitHub. Available at: https://github.com/SteveMacenski/slam_toolbox (Accessed: 21 April 2024).

[15] T. Lauttia, Ädaptive Monte Carlo localization in ROS", Trepo, 2022. [Online]. Available: https://trepo.tuni.fi/handle/10024/134867. [Accessed: 29-Jun-2024].