UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

Localización y construcción de mapas voxelizados en robótica móvil empleando cámaras RGB-D

Localization and building of voxelized maps in mobile robotics using RGB-D cameras

Realizado por
Pablo Vázquez Vera

Tutorizado por
José Raúl Ruiz Sarmiento
y
Jose Luis Matez Bandera

Departamento
INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2025

Fecha defensa: 9 de septiembre de 2025

Abstract

La robótica es una disciplina que ha crecido en los últimos años, y su aplicación en la industria ha demostrado ser muy beneficiosa en gran cantidad de materias. Desde la automatización de tareas repetitivas hasta la mejora de la precisión en procesos de manufactura, los robots han revolucionado la forma en que realizamos todo tipo de tareas, algunas inconcebibles tan solo unos años atrás.

En el campo de la robótica existen diversas ramas, cada una con su propio enfoque y aplicación. Una de estas ramas es la robótica móvil, que se centra en el diseño y desarrollo de robots capaces de moverse de manera autónoma en entornos dinámicos. La robótica móvil ha encontrado aplicaciones en una amplia variedad de campos, desde la exploración espacial hasta la logística y el transporte.

En este contexto, el presente documento tiene como objetivo investigar y analizar la viabilidad de un proyecto de robótica móvil, centrado en la creación de un robot capaz de localizarse de manera autónoma en un entorno desconocido a la vez que genera un mapa voxelizado de este, recibiendo como única fuente de datos nubes de puntos capturadas del entorno. Exploraremos los desafíos técnicos y las oportunidades que presenta este proyecto, así como su potencial impacto en la industria y la sociedad en general.

Palabras clave: Robótica móvil, Navegación autónoma, Mapas voxelizados, Nubes de puntos, Exploración de entornos desconocidos.

Contents

1 Introducción

1.1 Motivación

La robótica es una disciplina que ha ensanchado sus fronteras en los últimos años, y su aplicación en la industria ha demostrado ser increíblemente beneficiosa en gran cantidad de materias, atrayendo la atención tanto de investigadores y profesionales de todo el mundo como de empresas e inversores interesados en su desarrollo y aplicación.
Una de las ramas más prometedoras de la robótica es la robótica móvil, que se centra en el diseño y desarrollo de robots capaces de moverse de manera autónoma en entornos dinámicos. Esta rama es una de las más complejas y a la vez fascinantes, ya que implica la integración de diversas disciplinas como la inteligencia artificial, la visión por computadora y el control de sistemas que deben trabajar en perfecta sintonía para lograr que el robot sea capaz de navegar y operar de manera eficiente en entornos que pueden ser impredecibles y cambiantes.
La tarea que nos ocupa en este documento es la creación de un robot capaz de localizarse de manera autónoma en un entorno desconocido a la vez que genera un mapa voxelizado de este. Para llevar a cabo esta tarea, el robot recibirá como única fuente de datos nubes de puntos capturadas del entorno.
Este proyecto plantea una serie de desafíos ligados a la naturaleza de los datos que se utilizan, ya que las nubes de puntos son representaciones tridimensionales del entorno que pueden ser:

- **Computacionalmente costosas**: Las nubes de puntos pueden contener millones de puntos, lo que requiere un procesamiento intensivo para extraer información útil.
- **Ruidosas**: Las nubes de puntos pueden contener ruido debido a errores en la captura de datos intrínsecos a la naturaleza de los sensores.
- **Incompletas**: Las nubes de puntos pueden no representar completamente el entorno

debido a limitaciones en la cobertura del sensor o a obstrucciones físicas en el entorno.

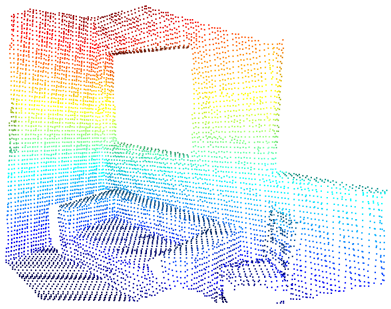

Figure 1: Imagen de una nube de puntos.

Esto nos llevará a explorar diversas técnicas de procesamiento de datos y algoritmos de localización y mapeo, así como a considerar las limitaciones y oportunidades que presenta el uso de nubes de puntos y los mapas voxelizados en la robótica móvil.
Entre las herramientas y técnicas de interés más destacadas relacionadas con la casuística que tratamos encontraremos el uso de ICP (Iterative Closest Point) para la alineación de nubes de puntos y, por consiguiente, la obtención de la pose relativa, Bonxai para la generación, manipulación y almacenamiento de mapas voxelizados gestionados de manera eficiente y aprovecharemos la amplia gama de herramientas y bibliotecas proporcionadas por ROS en su segunda versión, ROS2, para la implementación de los algoritmos de localización y mapeo, así como la potencia y versatilidad de lenguajes de programación como Python y C++.

Figure 2: Logo de la plataforma ROS2 [1].

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar un robot capaz de localizarse de manera autónoma en un entorno desconocido y generar un mapa voxelizado de este entorno utilizando

nubes de puntos como única fuente de datos. Para lograr este objetivo, se ha de llevar a cabo una serie de tareas específicas:

- **Diseño de un flujo de trabajo.** Para desarrollar este flujo de trabajo es necesario diseñar:
  - **Sincronización de mensajes:** Se debe establecer un mecanismo de sincronización que permita recibir y procesar las nubes de puntos de manera eficiente, asegurando que los datos estén disponibles en el momento adecuado para su procesamiento. Además, se debe considerar cómo se gestionará la comunicación entre los diferentes componentes del sistema, incluyendo la adquisición de datos, el procesamiento, localización y la generación del mapa.
  - **Preprocesamiento de los datos:** Se debe realizar un preprocesamiento de las nubes de puntos para eliminar ruido y mejorar la calidad de los datos. Esto puede incluir técnicas como filtrado, segmentación y reducción de ruido, así como la normalización de los datos para facilitar su procesamiento posterior.
  - **Cálculo de la pose:** Se debe implementar un algoritmo que permita calcular la pose del robot en el entorno a partir de las nubes de puntos recibidas. Esto puede incluir técnicas como ICP (Iterative Closest Point) para alinear las nubes de puntos y estimar la pose relativa del robot.
  - **Construcción del mapa voxelizado:** Se debe utilizar alguna herramienta para la generación y manipulación de mapas voxelizados, como Bonxai, para crear un mapa del entorno a partir de las nubes de puntos procesadas. Esto puede incluir la creación de una estructura de datos eficiente para almacenar el mapa y la implementación de algoritmos para actualizar el mapa a medida que se reciben nuevas nubes de puntos.
- **Procedimiento de validación:** Es necesario estudiar y definir un procedimiento que permita evaluar de manera objetiva la precisión y eficiencia del sistema de localización y mapeo desarrollado. Este procedimiento debe incluir:
  - **Definición de métricas sobre la calidad de la localización:** Se deben definir métricas que permitan evaluar la precisión de la localización del robot en el en-

torno, considerando factores como la desviación respecto a la posición real y la estabilidad de la localización a lo largo del tiempo.

  - **Definición de métricas sobre la calidad del mapa:** Se deben definir métricas que permitan evaluar la calidad del mapa generado por el robot, considerando factores como la capacidad de representar adecuadamente el entorno.
  - **Comparativa entre enfoques:** Se debe realizar una comparativa entre los diferentes enfoques desarrollados en el proyecto, evaluando su rendimiento y eficiencia en función de las métricas definidas anteriormente.

1.3 Estructura del documento

La estructura del documento se organiza en 6 capítulos principales, cada uno de los cuales aborda un aspecto clave del proyecto:

- **Capítulo 1: Introducción.** En este capítulo se presenta el contexto del proyecto, qué motiva el desarrollo y la investigación llevada a cabo, así como se definen una serie de objetivos que se pretenden alcanzar.
- **Capítulo 2: Bases.** En este capítulo se proporciona una visión general de las bases teóricas y técnicas que sustentan el proyecto, incluyendo conceptos clave como la localización, el mapeo y las nubes de puntos.
- **Capítulo 3: Tecnologías usadas.** En este capítulo se describen las tecnologías y herramientas utilizadas en el proyecto, incluyendo ROS2 y Bonxai, entre otros. Además, se presentan las ventajas y desventajas de cada una de ellas, así como su aplicabilidad en el contexto del proyecto.
- **Capítulo 4: Localización y mapeo.** Este capítulo se centra en los algoritmos y técnicas utilizados para la localización y el mapeo del robot en un entorno desconocido, incluyendo la alineación de nubes de puntos y la generación de mapas voxelizados.
- **Capítulo 5: Validación de resultados.** En este capítulo se presentan los resultados obtenidos en el proyecto, incluyendo la evaluación de la precisión y eficiencia del sistema de localización y mapeo, así como una comparativa entre los enfoques desarrollados.

- **Capítulo 6: Conclusiones.** En este capítulo se presentan las conclusiones del proyecto, incluyendo una reflexión sobre los logros alcanzados, las lecciones aprendidas y las posibles direcciones futuras de investigación y desarrollo en el campo de la robótica móvil.

2
Bases

2.1 Mapas voxelizados

2.1.1 Introducción a los mapas voxelizados

La idea de los mapas voxelizados surgió como una extensión natural de los conceptos utilizados en la representación de imágenes digitales en dos dimensiones, aplicados al espacio tridimensional. Mientras que una imagen está compuesta por píxeles (elementos de imagen), un volumen tridimensional puede representarse mediante vóxeles (volumetric pixels), es decir, pequeñas celdas cúbicas que subdividen el espacio en una cuadrícula regular.
Este enfoque se originó en el ámbito de la visualización médica y científica durante las décadas de 1970 y 1980, cuando surgió la necesidad de modelar órganos y estructuras internas del cuerpo humano a partir de escáneres como la tomografía computarizada (CT) y la resonancia magnética (MRI). Posteriormente, el concepto fue adoptado por otras disciplinas como la computación gráfica, la simulación física y, más recientemente, la robótica, donde los mapas voxelizados se utilizan para representar entornos 3D en tareas de navegación, percepción y planificación de movimiento.

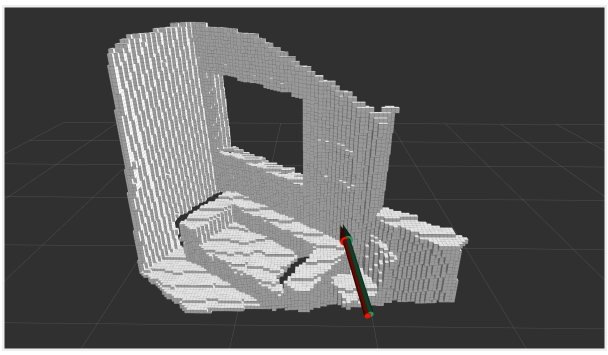

Figure 3: Imagen de un mapa voxelizado.

Los mapas voxelizados representan una evolución significativa en la forma en que se modelan

y procesan entornos tridimensionales y su desarrollo está vinculado al crecimiento de la capacidad computacional y la necesidad de representar espacios complejos de manera eficiente, especialmente en contextos donde se requiere análisis espacial preciso, como en navegación autónoma o reconstrucción 3D.

2.1.2 Características de los mapas voxelizados

Un mapa voxelizado divide un espacio tridimensional en una cuadrícula regular de celdas cúbicas, denominadas voxels (volumetric pixels). Cada voxel puede contener información binaria (ocupado/libre), probabilística (por ejemplo, con modelos bayesianos como los octomapas), o datos más complejos como intensidad, color, o etiquetas semánticas. Entre sus características principales se destacan:

- **Representación volumétrica discreta:** Los mapas voxelizados dividen el espacio tridimensional en una rejilla cúbica regular compuesta por pequeñas celdas llamadas voxeles. Cada voxel representa una porción del espacio con una resolución determinada, lo que permite una representación explícita del volumen y no solo de la superficie.

- **Resolución configurable:** La resolución del mapa puede ajustarse modificando el tamaño de los voxeles. Una resolución más alta (voxeles más pequeños) proporciona mayor detalle, pero aumenta el consumo de memoria y el coste computacional. Por el contrario, resoluciones más bajas reducen la precisión, pero permiten una operación más eficiente.

- **Estructura espacial regular:** La organización de los datos en una cuadrícula facilita la implementación de algoritmos paralelos y el acceso constante a la información espacial, lo que es útil en cálculos de visibilidad, planificación de trayectorias y simulaciones físicas.

- **Facilidad para representar ocupación:** Cada voxel puede almacenar un valor que indique si el espacio está ocupado, libre o desconocido. Esta propiedad es fundamental para la planificación y la detección de colisiones en robótica y simulación.

- **Extensibilidad de la información por voxel:** Los voxeles pueden contener no solo información binaria (ocupado/libre), sino también:

  - **Probabilidades:** Representando la probabilidad de ocupación de un voxel, lo que permite manejar incertidumbres en la percepción.
  - **Intensidad o color:** Almacenar información adicional como la intensidad de la señal o el color asociado a cada voxel, lo que es útil en aplicaciones de visión por computadora y reconstrucción 3D.
  - **Etiquetas semánticas:** Asignar etiquetas a los voxeles para identificar objetos o características del entorno, lo que es especialmente útil en aplicaciones de robótica móvil y percepción semántica.
  - **Normales de superficie:** Almacenar información sobre la orientación de las superficies dentro de los voxeles, lo que es útil para la reconstrucción de superficies y la simulación física.

- **Compatibilidad con estructuras jerárquicas:** Para mejorar la eficiencia en almacenamiento y consultas, los mapas voxelizados pueden implementarse con estructuras jerárquicas como los octrees, que dividen el espacio de forma adaptativa según la densidad de información.

- **Eficiencia en consultas espaciales:** Gracias a su estructura regular (o jerárquica en el caso de octrees), los mapas voxelizados permiten realizar operaciones como:

  - **Búsqueda de vecinos:** Encontrar voxeles adyacentes de manera eficiente.
  - **Intersección de rayos:** Determinar si un rayo intersecta con algún voxel, lo que es útil en simulaciones de iluminación y trazado de rayos.
  - **Colisiones:** Detectar colisiones entre objetos y el entorno representado por el mapa voxelizado, lo que es esencial en robótica y simulación física.

- **Idoneidad para entornos dinámicos:** La representación voxelizada puede actualizarse de forma local cuando se detectan cambios en el entorno, lo que permite mantener mapas actualizados en tiempo real, aspecto esencial para aplicaciones robóticas y vehículos autónomos.

2.1.3 Aplicaciones de los mapas voxelizados

El uso de los mapas voxelizados se ha ampliado a diversos campos con el paso del tiempo gracias a las mejoras en computación, tecnologías de captura de datos e inversiones en investigación y desarrollo del área. Algunas de las aplicaciones más destacadas incluyen:

- **Robótica móvil y navegación autónoma:** Los mapas voxelizados se utilizan para representar el entorno tridimensional de un robot o vehículo autónomo. Se emplean para la planificación de trayectorias, detección de obstáculos y navegación segura.
  - **Ventajas:** Permiten representar obstáculos a distintas alturas, útil en entornos no planos. Se integran bien con sensores como LIDAR o cámaras RGB-D. Facilitan el ray tracing para simulaciones de sensores y planificación.
  - **Inconvenientes:** Alta demanda de memoria y procesamiento, especialmente en espacios grandes. Requieren mecanismos eficientes de actualización en entornos dinámicos.
- **Reconstrucción 3D de escenas:** Los mapas voxelizados permiten reconstruir entornos tridimensionales desde datos de sensores (escáneres láser, cámaras estereoscópicas, etc.).
  - **Ventajas:** Buen manejo de ruido y fusión de múltiples vistas. Permiten interpolar datos faltantes o parciales de forma robusta.
  - **Inconvenientes:** La densidad del modelo puede hacer que sea difícil de almacenar o transmitir. Generalmente se requiere un posprocesamiento para renderizado o análisis.
- **Medicina e imagen biomédica:** Se usan para modelar órganos y tejidos en 3D, a partir de escaneos como tomografía (CT) o resonancia magnética (MRI), permitiendo diagnósticos y planificación quirúrgica.
  - **Ventajas:** Representan estructuras internas de forma precisa y continua. Permiten segmentación automática y simulaciones médicas.
  - **Inconvenientes:** Altos requisitos computacionales para procesar volúmenes detallados. Requieren software especializado para su interpretación.

- **Simulación física y entornos virtuales:** Se utilizan en motores de física y simulación para modelar el comportamiento de materiales, fluidos o entornos destructibles.
  - **Ventajas:** Representación volumétrica adecuada para materiales no rígidos o granularidad. Permiten simulaciones dinámicas y realistas de colisiones o fluidos.
  - **Inconvenientes:** Simulaciones físicas con vóxeles pueden ser computacionalmente más intensas que con mallas.
- **Videojuegos y gráficos por computadora:** Se usan para construir mundos destructibles o interactivos, como en juegos con estética voxel (por ejemplo, Minecraft).
  - **Ventajas:** Simplicidad para modelar entornos que cambian o se destruyen. Representación directa en memoria de entornos interactivos.
  - **Inconvenientes:** Apariencia visual menos realista comparada con modelos basados en polígonos. No adecuados para gráficos de alta fidelidad sin posprocesado.
- **Exploración subterránea y minería:** Los mapas voxelizados se utilizan para representar túneles, cavidades o cuerpos geológicos, permitiendo el análisis y la planificación de rutas de extracción.
  - **Ventajas:** Manejan entornos 3D complejos con estructuras irregulares. Permiten planificación segura en zonas de difícil acceso.
  - **Inconvenientes:** Requiere integrar datos de múltiples fuentes con resolución variable. Visualización y análisis pueden ser más complejos que con mapas 2D.

3

Tecnologías usadas

En esta sección nos centraremos en introducir las tecnologías usadas para el proceso completo de desarrollo del proyecto. Hablaremos de los lenguajes de programación utilizados, las plataformas de desarrollo, librerias externas y frameworks usados a lo largo del proyecto.

3.1 Lenguajes

En el desarrollo del proyecto se han utilizado varios lenguajes para abordar diferentes aspectos del sistema. Entre ellos encontramos:

- **Python:** Python es un lenguaje de programación interpretado, de alto nivel y con una sintaxis clara y legible. Fue diseñado para ser fácil de aprender y usar, lo que lo hace ideal tanto para principiantes como para desarrolladores avanzados. Es multiparadigma (soporta programación orientada a objetos, funcional e imperativa) y tiene una gran cantidad de bibliotecas disponibles. La integración de Python en ROS2 permite el desarrollo rápido de prototipos y la implementación de algoritmos complejos de procesamiento de datos, control y comunicación entre nodos. Python es el lenguaje principal de este proyecto, usado para la implementación de nodos de ROS2, controlando el flujo de datos, la lógica para el cálculo de la pose, métricas de validación y manipulación de nubes de puntos.

- **C++:** C++ es un lenguaje de programación compilado, de propósito general, conocido por su alto rendimiento, control sobre el hardware y capacidades de programación orientada a objetos. Es ampliamente utilizado en sistemas embebidos, videojuegos, y especialmente en aplicaciones donde el rendimiento y la eficiencia son críticos. La integración

de C++ en ROS2 permite el desarrollo de nodos de alto rendimiento, especialmente aquellos que requieren procesamiento intensivo de datos o interacción directa con hardware. En este proyecto, C++ se utiliza para nodos de almacenamiento y gestión del mapa voxelizado e interacción con la librería Bonxai, que proporciona una interfaz eficiente para la manipulación de mapas voxelizados.

- **XML:** XML (eXtensible Markup Language) es un lenguaje de marcado diseñado para almacenar y transportar datos de forma estructurada y legible tanto para humanos como para máquinas. En ROS2, XML se utiliza principalmente en la configuración y descripción de componentes del sistema robótico debido al sistema de descripciones de robots URDF y XACRO, legibilidad, interoperabilidad y amplia adopción en la industria. En este proyecto, XML se utiliza para definir la configuración de los nodos de ROS2, incluyendo parámetros, tópicos y dependencias entre servicios.

- **YAML:** YAML (Yet Another Markup Language, o más correctamente YAML Ain't Markup Language) es un formato de serialización de datos basado en texto, muy usado para representar información de forma estructurada y legible por humanos. Sus principales usos son como formato de configuración y para el intercambio de datos entre aplicaciones. En ROS2, YAML se utiliza para definir parámetros de nodos, configuraciones de lanzamiento y otros aspectos del sistema robótico. En este proyecto, YAML se utiliza para almacenar configuraciones de nodos, parámetros de algoritmos y otros datos estructurados necesarios para la ejecución del sistema.

- **CMAKE:** CMake es un lenguaje de configuración y scripting especializado para la construcción de software. Se usa como herramienta de automatización de compilación que utiliza archivos llamados CMakeLists.txt para describir cómo debe compilarse y enlazarse un proyecto de software. CMake es usado en ROS2 debido a que es la herramienta estándar para la construcción de paquetes y nodos, facilitando la gestión de dependencias, la configuración del entorno de compilación y la generación de archivos de configuración necesarios para la ejecución de los nodos desarrollados en C++.

- **LaTeX:** LaTeX es un sistema de composición de documentos basado en el lenguaje de tipografía TeX. Está diseñado para la creación de documentos de alta calidad tipográ-

fica, especialmente aquellos que incluyen fórmulas matemáticas complejas, referencias cruzadas, bibliografías y estructuras organizadas como capítulos, secciones y tablas.

3.2 Entorno, software y herramientas de desarrollo

En el desarrollo del proyecto se han utilizado varias herramientas y entornos para facilitar la implementación, el desarrollo y la prueba del código. Entre ellos encontramos:

- **ROS2 (Humble Hawksbill):** ROS 2 (Robot Operating System 2) es un marco de desarrollo y un conjunto de herramientas diseñadas para facilitar la creación de sistemas robóticos complejos, modulares y distribuidos. Aunque su nombre sugiere que es un sistema operativo, en realidad ROS 2 no es un sistema operativo tradicional, sino una capa de software que proporciona abstracciones y servicios esenciales para el desarrollo de robots, como:
  - **Comunicación entre componentes:** ROS 2 organiza el software robótico en nodos independientes que se comunican entre sí usando tópicos, servicios y acciones. Esto permite una arquitectura modular y escalable.
  - **Gestión de hardware:** ROS 2 interactúa con sensores y actuadores a través de drivers y interfaces de hardware.
  - **Control en tiempo real:** ROS 2 está diseñado para trabajar con sistemas en tiempo real, permitiendo ejecutar controladores que responden de manera precisa y rápida.
  - **Simulación, visualización y depuración:** ROS 2 se integra con herramientas como:
    - **RViz:** Una herramienta de visualización que permite ver datos de sensores, mapas y estados del robot en tiempo real.
    - **Ros2 bag:** Un sistema de registro que permite grabar y reproducir datos de sensores y mensajes de ROS 2, facilitando la depuración y el análisis de datos.
    - **Ros2 doctor, trace, topic, etc.:** Herramientas de diagnóstico y monitoreo que permiten analizar el estado del sistema, rastrear mensajes y verificar la comunicación entre nodos.

ROS 2 (Robot Operating System 2) es la evolución del sistema operativo para robots originalmente conocido como ROS 1. Fue diseñado desde cero para resolver las limitaciones arquitectónicas y técnicas de ROS 1, ofreciendo una plataforma más robusta, segura, flexible y adecuada para aplicaciones comerciales, industriales y en tiempo real. Estas mejoras vienen dadas por el uso de:

  - **DDS (Data Distribution Service):** Un middleware de comunicación en tiempo real que permite la interoperabilidad entre nodos, mejorando la escalabilidad, fiabilidad y seguridad de la comunicación.
  - **RTOS (Real Time Operating System):** Permite ejecutar nodos en sistemas operativos de tiempo real.
  - **Soporte para múltiples lenguajes y plataformas:** ROS 2 ofrece soporte nativo para varios lenguajes de programación como C++, Python y Rust, y es compatible con una amplia gama de sistemas operativos, incluyendo Linux, Windows y macOS.
  - **Mejoras en el sistema de construcción:** Se usa colcon en lugar de catkin, lo que permite una construcción más eficiente y flexible de los paquetes de ROS 2.

- **Ubuntu (22.04 LTS):** Ubuntu es una distribución del sistema operativo Linux, basada en Debian, desarrollada por Canonical. Es conocida por ser gratuita, de código abierto, estable y fácil de usar, tanto para usuarios nuevos como para desarrolladores. En concreto la versión 22.04 LTS (Long Term Support) es una versión de soporte a largo plazo, lo que significa que recibirá actualizaciones de seguridad y mantenimiento durante un período prolongado (5 años). Además de esto, se ha elegido esta versión en concreto por el soporte oficial de ROS2 en la distribución Humble Hawksbill, alta compatibilidad con las herramientas de desarrollo usadas, la fácil gestión de dependencias y el amplio uso por parte de la comunidad de robótica.

- **Visual Studio Code:** Visual Studio Code es un editor de código fuente ligero y multiplataforma, desarrollado por Microsoft. Es gratuito, de código abierto y compatible con una gran variedad de lenguajes de programación como C++, Python, XML, CMake, entre otros. Ofrece extensiones, depuración integrada, control de versiones (Git), y una

interfaz altamente personalizable. Se ha elegido usar este editor por su compatibilidad con los múltiples lenguajes usados en el proyecto, su ligereza y rapidez, su amplia gama de extensiones y su integración con herramientas de desarrollo como CMake y ROS2.

- **GitHub:** GitHub es una plataforma en línea para almacenar, compartir y colaborar en proyectos de software utilizando el sistema de control de versiones Git. Permitiendo a desarrolladores trabajar en proyectos, rastrear cambios en el código, revisar contribuciones y gestionar versiones del software, todo desde un entorno centralizado basado en la web.

- **Jupyter Notebook:** Jupyter Notebook es una herramienta interactiva que permite escribir y ejecutar código en fragmentos llamados celdas. Aunque originalmente fue diseñado para usarse en un navegador web, también se puede utilizar directamente desde entornos como Visual Studio Code (VS Code). Aunque esta orientado a python, tambien puede usarse con otros lenguajes. La principal ventaja de Jupyter Notebook es su capacidad para combinar código, texto, visualizaciones y otros elementos multimedia en un solo documento, lo que facilita la creación de informes interactivos y la documentación de proyectos. En este proyecto se ha utilizado para documentar el proceso de analisis de resultados y la validación de los algoritmos implementados.

- **Terminator:** Terminator es un emulador de terminal para sistemas operativos basados en Unix, que permite dividir la ventana de la terminal en múltiples paneles, facilitando la ejecución de varios comandos y la visualización de salidas simultáneamente. Es especialmente útil para desarrolladores y administradores de sistemas que necesitan trabajar con múltiples sesiones de terminal al mismo tiempo. En este proyecto se ha utilizado para ejecutar y monitorear múltiples nodos de ROS2 simultáneamente, facilitando la depuración y el control del flujo de datos entre los diferentes componentes del sistema.

4

Pipeline de Localización y Mapeo Simultaneo

En esta sección primero describiremos las funciones y componentes claves que entran en juego en el proceso. Posteriormente pasaremos a describir el flujo de datos y la interacción entre los diferentes componentes del sistema y por tanto describiremos el flujo de trabajo completo de la aplicacion.

4.1 Conceptos, componentes y funciones clave

En esta sección iremos detallando cada uno de los componentes clave que entran en juego en el flujo de trabajo del sistema, definiremos su base, las funcionos que desempeñan, los conceptos que lo sustentan y los detalles de su implementación.

4.1.1 Gestion y sincronización de mensajes

La gestión y sincronización de mensajes es un aspecto crucial en sistemas roboticos distribuidos, donde múltiples nodos trabajan de forma conjunta para lograr tareas complejas. En el contexto de ROS2, los nodos se comunican entre sí mediante el intercambio de mensajes a través de tópicos, servicios y acciones. La sincronización adecuada de estos mensajes es esencial para garantizar que los datos se procesen en el orden correcto y que las operaciones dependientes de múltiples fuentes se realicen de manera coherente.

Para lograr una gestión y sincronización correcta de los mensajes para este proyecto y dado que la fuente de datos esta almacenada en un archivo de tipo rosbag, se ha optado por el

desarrollo un nodo que se encargue de reproducir los mensajes adecuados en función de la dispobibilidad de los principales nodos de procesamiento.
Este nodo llamado ”input_data _node” se encarga de crear una estructura de datos que almacena los mensajes contenidos en el archivo rosbag (grabación con los datos) ya que estos vienen serializados. Una vez leido este archivo y creada esta estructura de datos con los datos serializados, se iran leyendo los mensajes de forma ordenada, deserializando, formateando segun el tipo de datos al que corresponda el mensaje y publicandolos en los tópicos correspondientes.

Conceptos clave: Los conceptos clave que sustentan este nodo son:

- **Archivos rosbag**: Un rosbag es un archivo usado en ROS (Robot Operating System) para registrar y reproducir datos de tópicos. Los archivos rosbag se almacenan en formato binario guardando la secuencia de mensajes ROS capturados. Esto mensajes incluyen cabeceras, marcas de tiempo y datos serializados. En nuestro caso, usaremos un archivo de este tipo para leer una grabación realizada previamente sobre un entorno simulado que intentaremos recrear usando nuestro proceso de localización y mapeo.

- **Serialización y deserialización de mensajes**: La serialización es el proceso de convertir un objeto o estructura de datos en una secuencia de bytes para su almacenamiento y la deserialización es el proceso inverso, es decir, convertir una secuencia de bytes en un objeto o estructura de datos. En ROS2, los mensajes se serializan para su transmisión a través de la red o su almacenamiento en archivos rosbag. En nuestro caso, los mensajes leidos del archivo rosbag vienen serializados, por lo que es necesario deserializarlos antes de poder usarlos. Esta acción sera clave una vez leido el archivo rosbag y antes de publicar los mensajes en los tópicos correspondientes.

- **Publicación y suscripción a tópicos**: En ROS2, los nodos pueden publicar mensajes en tópicos y suscribirse a ellos para recibir mensajes. La publicación y suscripción a tópicos es un mecanismo de comunicación fundamental en ROS2, que permite la comunicación asíncrona entre nodos. En nuestro caso, el nodo ”input_data _node” publicará los mensajes deserializados en el topico ”clean_pcl” para los datos que correspondan a las nubes de puntos que se usaran para el proceso de mapeo y localización y el tópico ”clean_pose” para los datos que correspondan a la pose original del robot que se usara

para aislar la pose real desde donde se capturaron las nubes de puntos y poder sacar métricas de precisión respecto a esta pose.

Funcionamiento del componente: Tal y como hemos ido describiendo, el nodo ”input_data _node” se encargará de leer el archivo rosbag, deserializar los mensajes y publicarlos en los tópicos correspondientes. El flujo de trabajo del nodo se puede describir de la siguiente manera:

- **Inicialización**: La inicialización de este nodo esta conformada por los siguientes pasos:
  - **Inicialización de publicadores y subscriptores**: Al instanciar el nodo, se inicia la escucha y se prepara para la recepción de mensajes de los siguientes tópicos:
    - **Publicación de ”clean_pose”**: Este servicio de publicación se encargara de enviar la pose original desde la que se capturo el mensaje.
    - **Publicación de ”clean_pcl”**: Este servicio de publicación se encargara de enviar los mensajes conteniendo la nube de puntos original.
    - **Subscripción a ”new_pose”**: Este subscriptor recibira mensajes de tipo ”PoseStamped” conteniendo la nueva pose calculada en función del calculo realizado por el servicio de publicación.
  - **Inicialización de lectura del archivo rosbag**: Tambien es necesario inicializar el sistema de lectura del archivo rosbag para que se pueda iterar facilmente a través de los mensajes que este contiene.
  - **Inicialización de variables auxiliares para la ejecucción**: Por ultimo, se inicializaran algunas variables auxiliares como indices entre otras, necesarias para la correcta ejecución del nodo.
- **Respuestas provocada por la recepción de mensajes a traves del tópico ”new_pose”**: Cada vez que se recive un mensaje a traves del tópico ”new_pose” se inicia un proceso de respuesta que involucra los siguientes pasos:
  - **Obtención de la última pose antes de la publicación de la nube de puntos**: Se leera el archivo rosbag almacenando siempre el último mensaje con etiqueta ”gt_odom” de forma que cuando se encuentre un mensaje conteniendo la nueva

nube de puntos, se pueda deserializar y rellenar un mensaje de tipo ”PoseStamped” que contendra la ultima pose asociada a este mensaje por el tópico ”clean_pose”.

- **Obtención de las nubes de puntos**: Se continuara la lectura de los mensajes del archivo rosbag hasta que se encuentre uno con la etiqueta ”cloud_in”, entonces se deserializaran los datos del mensaje y se rellenaran aquellos campos necesarios para publicar un mensaje de tipo ”PointCloud2”. Una vez tenemos este mensaje de tipo ”PointCloud2” relleno, este se publicara a traves del tópico ”clean_pcl”.

En resumen, este nodo se encarga de iniciar el flujo de datos de entrada publicando los mensajes asociados con la nubes de puntos y la pose asocidada. Una vez se ha iniciado este flujo, continuara plublicando estos mensajes cada vez que reciba un mensaje por parte del nodo de localización (determinando asi que este esta listo para recibir otro) y asi hasta que se complete la lectura.

4.1.2 Adición de ruido a los datos originales

Comprobar la resiliencia de nuestro flujo de trabajo frente al ruido es una parte esencial de nuestro estudio, es por esto que se ha desarrollado un nodo llamado ”noisy_cloud” que se encarga de recepcionar los mensajes de tipo ”PointCloud2” publicado a traves del tópico ”clean_pose” que contienen la nube de puntos libre de errores y En base a unos parametros determinados, añadira cierto error de forma aleatoria a los puntos contenido en la nube. Esta parte del proceso se usara solo cuando queramos probar la resiliencia frente a errores, es decir, el flujo de trabajo tambien esta preparado para trabajar directamente con la nube sin error como es de esperar ya que la presencia de error añade complejidad al proceso.

Conceptos clave: Un concepto clave a tener en cuenta para este proceso es la transformación que se aplica a la nube de puntos sin error para que se convierta en una nube de puntos que contenga ruido. En este caso hemos optado por aplicar la siguiente transformación:

- **Mascara de ruido**: El ruido se aplica a la nube de puntos a partir de una mascara que al operar con la nube original introducira errores usando una distribución de probabilidad normal con media N, desviación estandar S y que se aplicara a cualquier punto en cualquiera de sus ejes con una probabilidad X. El orden de operacion es el siguiente:

- **Creación de la mascara**: Se asigna una probabilidad aleatoria a cada uno de los puntos de la nube, y los puntos cuya probabilidad sea menor a la especificada, tendran valor 1 en la mascara y por tanto se les aplicara el error.
- **Distribución Normal**: Se genera un objeto que describe una distribución normal en función de los parametros elegidos en la configuracion.
- **Aplicación de ruido**: Se aplica el ruido a los puntos seleccionados por la mascara y se modifican en funcion de valor aleatorio generado por la normal para cada uno de sus ejes (X,Y,Z).

Funcionamiento del componente: Tal y como hemos ido describiendo, el nodo ”noisy_cloud” se encargara de añadir ruido a la nube de puntos original para probar la resiliencia del flujo de trabajo. Para ello, el funcionamiento basico de este nodo sera:

- **Inicialización**: Al iniciar el nodo, simplemente creara un subscriptor y un tópico de publicación:
  - **Subscripción al ”clean_pcl”**: Este subscriptor recibira mensajes de tipo ”PointCloud2” conteniendo la nueva nube de puntos a la que se añadira el ruido.
  - **Publicación de ”noisy_pcl”**: Este servicio de publicación se encarga de enviar mensajes de tipo ”PointCloud2” que contienen la nube de puntos con ruido.
- **Respuestas provocada por la recepción de mensajes a traves del tópico ”new_pcl”**: La recepción de este tipo de mensajes provoca la llamada de una función que se encarga de aplicar una mascara de ruido a la nube de puntos recibida y posteriormente publicarla a traves del tópico ”noisy_pcl”

En resumen, este nodo se encarga de transformar la nube de puntos original a un estado simulado mas cercano a la realidad de un sensor.

4.1.3 Calculo de la localización

La piedra angular de este proyecto se asienta sobre las bases de la localización, si sabemos la posición desde la que hemos capturado una nube de puntos podemos trasladar esta hasta esa posición para a posteriori ir calculando el desplazamiento entre nubes y componiendo este

desplazamiento sobre la pose existente, repitiendo asi este proceso hasta el infinito siempre y cuando seamos capaces de hacer buenas estimaciones de estos desplazamientos, filtrando ruido, calculos poco precisos y aislado estas estimaciones de pequeños errores consecutivos que pueden derivarnos lejos de la pose original si no se controlan.
Este es un proceso complejo que consta de varios componentes asociados que trabajan en sintonia para generar salidas de igual importacia a la pose como es el propio mapa. En este caso nos vamos a centrar en el proceso que lleva a cabo el nodo de localización llamado ”localization_node” y de la clase asociada a este proceso llamada ”KeyFrameSelector”. Estas dos clases trabajan en conjunto para calcular la pose desde la que se ha capturado una nube de puntos respecto de la anterior y comprobar si cumple los requisitos para formar parte del mapa.

Conceptos Clave: Este componente se construye alrededor de varios conceptos fundamentales para el correcto funcionamiento de este proyecto, entre ellos encontramos los siguientes:

- **Preprocesamiento de la nube de puntos**: Hay que tener en cuenta que las nubes de puntos tal cual son recibidas pueden no estar en el formato correcto para ser procesadas, pueden contener errores, estar incompletas o necesitar de transformaciones como traslación o reducción del muestreo (reducción del número de puntos agrupandolos en función de la distancia entre ellos). En nuestro caso, la mayoria de transformaciones de preprocesamiento que se aplican son:
  - **Transformación de formato**:
- **ICP (Iterative Closest Point)**:
- **Selección de fotogramas clave (Key frame selection)**: Como hemos comentado anteriormente, no todos las nube pueden formar parte del mapa ya que todos los emparejamientos realizados contienen un minimo de error ya que la naturaleza del metodo de emparejamiento es estadistico y no es perfecto. De forma que si componemos este error que aunque minimo o imperceptible a simple vista entre las capturas iniciales, se va componiendo en el tiempo dando lugar a que la pose termine por corromperse y dejar de representar de forma correcta la pose real.
  Esto nos lleva a desarrollar un proceso de curado y selección de frames que si cumplen una serie de requisitos, formaran parte del mapa y por tanto contribuiran a las siguientes

iteraciones, por lo que es de gran interes que esto sean emparejamientos lo mas precisos posibles y que se realizen con una frecuencia justa.

Funcionamiento del componente: Como se comenta, el nodo ”localization_node” se encargara de orquestar una serie de tareas cruciales que conciernen tanto a la localización global como a la construcción del mapa, por lo que detallar los pasos que este sigue, es clave para comprender el funcionamiento de la herramienta. Para ello vamos a describir los pasos que sigue:

- **Inicialización**: La inicialización de este nodo esta conformada por los siguientes pasos:
  - **Inicialización de publicadores y subscriptores**: Al instanciar el nodo, se inicia la escucha y se prepara para la recepción de mensajes de los siguientes tópicos:
    - **Subscripción de ”clean_pcl” o ”noisy_cloud”**: Este subscriptor se encargara de recibir los mensajes de tipo ”PointCloud2” conteniendo la nube de puntos original o con ruido en funcion del tipo de ejecución que hayamos decidido realizar.
    - **Publicacion de ”new_pose”**: Este servicio de publicación se encarga de publicar mensajes de tipo ”PoseStamped” conteniendo la pose actualizada calculada a traves de la ultima información recibida.
    - **Publicacion de ”new_pcl”**: Este servicio de publicación se encarga de publicar mensajes de tipo ”PointClou2” conteniendo la nube de puntos actualizada calculada a traves de la ultima información recibida.
  - **Lectura del archivo de configuración**: Es necesario leer un archivo de configuración de tipo YAML que contiene los parametros de evaluación de algunas condiciones, parametros de ICP entre otros.
  - **Inicialización de variables auxiliares para la ejecucción**: Por ultimo, se inicializaran algunas variables auxiliares como indices entre otras, necesarias para la correcta ejecución del nodo.
- **Respuestas provocada por la recepción de mensajes a traves del tópico ”clean_pcl” o ”noisy_pcl”**: Los mensajes recibidos por los tópicos de ”clean_pcl” o ”noisy_pcl” son

la unica entra que no se genera de forma recursiva por la ejecución del nodo, pero no es el unico input que usaremos. La ejecución de esta función se puede dividir en dos casos:

- **Primera llamada:** La primera llamada de este nodo se usa para inicializar una serie de elementos:
  - **Almacenamiento de la primera nube de puntos:** Como este algoritmo usara dos nubes de puntos (”t-1” y ”t”) para calcular el desplazamiento, en la primera llamada no tendremos ninguna nube previa por lo que unicamente haremos un preprocesamiento donde se transforma desde el tipo ”PointClou2” a ”open3d.geometry.PointCloud()”, se realizara un submuestreo para igualar su resolción a la que usa el mapa, se hace una limpieza de outliers, se trasladara al punto [x=0, y=0, x=0] y se calcularan sus normales, que son necesarias para realizar ICP dejandola asi lista para el siguiente paso.
  - **Almacenamiento del mapa e instaciación de la clase KeyFrameSelector:** Dado que es la primera nube de puntos, servira de punto de referencia, su pose sera el punto de origen por lo que se puede usar como inicialización para el mapa y la clase que seleccionara las sucesivas imagenes clave en función de esta y el mapa que describe.
  - **Reconstrucción del mensaje y publicación**: Dado que en esta fase no hay ningun emparejamiento que calcular, solo se transforma la nube de puntos de vuelta al tipo ”PointCloud2” y publicarla a traves del tópico ”new_pcl” para que sea recibida por el modulo de mapeo.
- **Llamadas sucesivas:** El resto de mensajes recibidos se trataran de igual manera y siguiendo este esquema:
  - **Preprocesamiento:** La nube de puntos se preprocesa de la misma manera que en el caso de la primera llamada, aunque se mantiene la copia original en caso de que se requiera usar para el proceso de selección de nubes claves. Es decir, la nube se transforma desde el tipo ”PointClou2” a ”open3d.geometry.PointCloud()”, se realiza el submuestreo a la resolción del mapa, se hace limpieza de outliers, se traslada a la pose calculada anteriormente y se calculan las normales para poder realizar el posterior emparejamiento por ICP.

* **Logica ICP:** Esta es la logica principal, encargada de suministrar la nueva pose y nube de puntos trasaladada a esa posición. Para implementar esta lógica, se ha creado una función llamada ”icp_logic” que sigue el siguiente esquema:
    - **PDTE**
* **Guardado de la nueva pose y publicación de la nueva información del mapa:** Se guarda la pose calculada por la logica ICP para puclicarla una vez que el mapa ha integrado la nueva nube de puntos en caso de que haya nueva información que añadir.

References

[1] Robotnik Automation S.L *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

Appendix A
Installation Manual

Requirements: