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

Mobile robotics has profoundly transformed service robotics, expanding its possibilities and applications. By providing robots with autonomous mobility, it has become possible for them to move through complex environments without constant supervision.

Technologies such as computer vision, simultaneous localization and mapping (SLAM), and advanced sensors enable these robots to avoid obstacles and plan efficient routes. This has facilitated tasks such as cleaning, object transportation, and the delivery of food or medicine in hospitals and hotels. Their implementation has reduced operating times and optimized resource utilization across multiple industries.

In this context, the present work aims to investigate and analyze the feasibility of a mobile robotics project focused on the development of a set of algorithms that can be used by robotic systems to autonomously localize themselves in an unknown environment while generating a voxelized representation of it, commonly known as a map, using point clouds captured from the environment as the sole data source. To accomplish this task, this project relies on the ROS2 development framework [1] as well as libraries specialized in computer vision (Open3D [2]) and voxel map management (Bonxai [3]). In addition, a set of metrics and procedures will be proposed to validate the obtained results.

Keywords: Mobile robotics, Voxel maps, Point clouds, Exploration of unknown environments.

Resumen

La robótica móvil ha transformado profundamente la robótica de servicio, ampliando sus posibilidades y aplicaciones. Al dotar a los robots de movilidad autónoma, se ha logrado que puedan desplazarse en entornos complejos sin supervisión constante.

Tecnologías como la visión por computadora, el mapeo simultáneo (SLAM) y los sensores avanzados permiten que estos robots eviten obstáculos y planifiquen rutas eficientes. Esto ha facilitado tareas de limpieza, transporte de objetos y entrega de alimentos o medicinas en hospitales y hoteles. Su implementación ha reducido tiempos de operación y optimizado el uso de recursos en múltiples industrias.

En este contexto, el presente documento tiene como objetivo investigar y analizar la viabilidad de un proyecto de robótica móvil, centrado en el desarrollo de una serie de algoritmos que puedan ser usados por sistemas robóticos para localizarse de manera autónoma en un entorno desconocido a la vez que genera una representación voxelizada de este, comúnmente conocida como mapa, recibiendo como única fuente de datos nubes de puntos capturadas del entorno. Para llevar a cabo esta tarea este proyecto se apoya en la plataforma de desarrollo ROS2 [1] y librerias especializadas en la vision por computadora (Open3D [2]) y la gestion de mapas voxelizados (Bonxai [3]). Ademas se propondran una serie de métricas y procedimientos para validar los resultados obtenidos.

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

Contents

1 Introducción

1.1 Motivación

La robótica es una disciplina que ha ampliado sus fronteras en los últimos años, y su aplicación en la 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 el desarrollo de algoritmos que permitan a un sistema robótico de localizarse en un entorno desconocido a la vez que genera una represantación voxelizada de este, comúnmente conocido como mapa. Para llevar a cabo esta tarea, el robot recibirá como única fuente de datos nubes de puntos capturadas del entorno. Además, se calcularán una serie de métricas para validar la calidad de los resultados obtenidos en diferentes escenarios.

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.

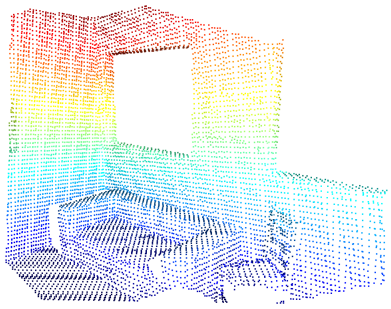

Figure 1: Imagen de una nube de puntos.

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 de Open3D [2]) para la alineación de nubes de puntos y, por consiguiente, la obtención de la posición y orientación relativa del sistema respecto de pasos anteriores. Además, se ha seleccionado la librería Bonxai [3] para la generación, manipulación y almacenamiento de mapas voxelizados que permite gestionados de manera eficiente. Tambien se hace uso de la amplia gama de herramientas y bibliotecas proporcionadas por ROS [1] 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++.

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar uan serie de algoritmos que permitan a un sistema robótico localizarse 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 posición y orientación:** Se debe implementar un algoritmo que permita calcular la posición y orientación del robot en el entorno a partir de las nubes de puntos recibidas. Esto puede incluir técnicas como ICP (Iterative Closest Point Open3D [2]) 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 [3], 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 sistema en el entorno, 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 sistema, considerando factores como la capacidad de representar adecuadamente el entorno.

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 [1] y Bonxai [3], 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 sistema 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 Conceptos clave en robótica móvil

En el contexto de la robótica móvil, los sensores que generan nubes de puntos —como LIDAR, cámaras RGB-D o sensores estereoscópicos— permiten obtener representaciones tridimensionales del entorno que son fundamentales para la localización y el mapeo. A partir de estas mediciones se definen diversas variables clave para describir el estado del robot y su relación con el entorno.

- **Pose**: Variable central y describe la posición y orientación del robot en el espacio. En el caso bidimensional se representa como:

  $$\mathbf{x} = [x, y, \theta]^T$$

  mientras que en 3D se extiende a:

  $$\mathbf{x} = [x, y, z, q_x, q_y, q_z, q_w]^T$$

  donde q_x, q_y, q_z, q_w corresponden a los cuaterniones que definen la orientación.

- **Traslación y rotación**: La trasalación se refiere al desplazamiento del robot entre dos instantes de tiempo consecutivos, estimado mediante el alineamiento de nubes de puntos (por ejemplo, usando algoritmos como ICP). Junto con la traslación, se obtiene la rotación, que describe el cambio de orientación.

- **Drift**: Corresponde al error acumulado en la estimación de la pose debido al ruido sensorial, imprecisiones en el alineamiento y propagación de errores de integración a lo largo del tiempo. La magnitud del drift determina la confiabilidad de la trayectoria estimada y, en ausencia de cierre de bucle, tiende a crecer indefinidamente.

Otras variables relevantes derivadas de las nubes de puntos incluyen:

- **Correspondencias**: asociaciones entre puntos de la nube actual y del mapa previo.
- **Transformación estimada**: matriz homogénea $\mathbf{T} \in SE(3)$ que combina traslación y rotación para llevar la nube actual al marco de referencia global.
- **Error de registro**: medida de la calidad del alineamiento de nubes, utilizada como criterio de convergencia en algoritmos iterativos.

El análisis de estas variables es crucial para el diseño de algoritmos de SLAM [4], navegación autónoma y reconstrucción de entornos tridimensionales.

2.2 Mapas voxelizados

2.2.1 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.

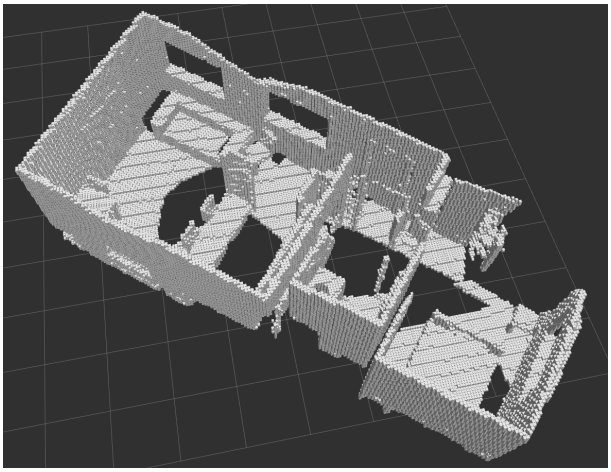

Figure 2: Imagen de un mapa voxelizado.

El uso de voxels para modelar estructuras internas a partir de tomografías y resonancias comenzó en los años 70 y 80, cuando las capacidades computacionales posibilitaron el procesamiento volumétrico de datos médicos [5]. El proyecto pionero VOXEL-MAN en 1986 representó las primeras visualizaciones 3D voxelizadas, incluyendo la combinación de imágenes de CT y MRI [6]. Un voxel, equivalente tridimensional del píxel, ha sido definido como una unidad de valor en una rejilla 3D utilizada especialmente en visualización médica y científica [7]. Además, el algoritmo Marching Cubes de 1987 fue un hito en la extracción eficiente de superficies isosuperficiales desde datos voxelizados [8]. Estos desarrollos precedentes sentaron las bases para la adopción posterior de mapas voxelizados en robótica, navegación y planificación de movimiento.

2.2.2 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.

2.2.3 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: [9]** 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 [1] 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.

Figure 3: Logo de Python.

- **C++: [10]** 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 [1] 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.

Figure 4: Logo de C++.

- **XML: [11]** 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.

Figure 5: Logo de XML.

- **YAML: [12]** 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.

Figure 6: Logo de YAML.

- **CMAKE: [13]** 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++.

Figure 7: Logo de CMake.

- **LaTeX: [14]** 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.

Figure 8: Logo de LaTeX.

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 [1] (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.

Figure 9: Logo de Ros2.

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 [15] (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.

Figure 10: Logo de Ubuntu.

- **Visual Studio Code [16]:** 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.

Figure 11: Logo de Visual Studio Code.

- **GitHub [17]:** 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.

Figure 12: Logo de Github.

- **Jupyter Notebook [18]:** 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.

Figure 13: Logo de Jupyter Notebook.

- **Terminator [19]:** 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.

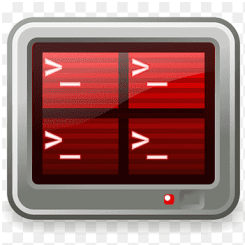

Figure 14: Logo de Terminator.

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 aplicación.

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 funciones que desempeñan, los conceptos que lo sustentan y los detalles de su implementación.

4.1.1 Gestión y sincronización de mensajes

La gestión y sincronización de mensajes es un aspecto crucial en sistemas robóticos 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 está almacenada en un archivo de tipo rosbag, se ha optado por el de-

sarrollo de un nodo que se encargue de reproducir los mensajes adecuados en función de la disponibilidad 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 leído este archivo y creada esta estructura de datos con los datos serializados, se irán leyendo los mensajes de forma ordenada, deserializando, formateando según el tipo de datos al que corresponda el mensaje y publicándolos 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. Estos 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 leídos del archivo rosbag vienen serializados, por lo que es necesario deserializarlos antes de poder usarlos. Esta acción será clave una vez leído 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 tópico ”clean_pcl” para los datos que correspondan a las nubes de puntos que se usarán 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 usará

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 se ha descrito, 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 está conformada por los siguientes pasos:
  - **Inicialización de publicadores y suscriptores**: 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 encargará de enviar de enviar mensajes de tipo ”PointCloud2” que contienen la pose original desde la que se capturó el mensaje.
    - **Publicación de ”clean_pcl”**: Este servicio de publicación se encargará de enviar de enviar mensajes de tipo ”PointCloud2” que contienen la nube de puntos original.
    - **Suscripción a ”new_pose”**: Este suscriptor recibirá mensajes de tipo ”PoseStamped” conteniendo la nueva pose calculada en función del cálculo realizado por el servicio de publicación.
  - **Inicialización de lectura del archivo rosbag**: Es necesario inicializar el sistema de lectura del archivo rosbag para que se pueda iterar fácilmente a través de los mensajes que este contiene.
  - **Inicialización de variables auxiliares para la ejecución**: Es necesario inicializar algunas variables auxiliares como índices, entre otras, para la correcta ejecución del nodo.
- **Respuestas provocadas por la recepción de mensajes a través del tópico ”new_pose”**: Cada vez que se recibe un mensaje a través 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 leerá 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 contendrá la última pose asociada a este mensaje por el tópico "clean_pose".

- **Obtención de las nubes de puntos**: Se continuará la lectura de los mensajes del archivo rosbag hasta que se encuentre uno con la etiqueta "cloud_in", entonces se deserializarán los datos del mensaje y se rellenarán aquellos campos necesarios para publicar un mensaje de tipo "PointCloud2". Una vez tenemos este mensaje de tipo "PointCloud2" relleno, este se publicará a través del tópico "clean_pcl" junto con la pose asociada por el tópico correspondiente.

En resumen, este nodo se encarga de iniciar el flujo de datos de entrada publicando los mensajes asociados con las nubes de puntos y la pose asociada. Una vez se ha iniciado este flujo, continuará publicando estos mensajes cada vez que reciba un mensaje por parte del nodo de localización (determinando así que este está listo para recibir otro), y así 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 este estudio, ya que acerca la simulación de la que disponemos a lo que podría ser una ejecución en el entorno real. Es por esto que se ha desarrollado un nodo llamado "noisy_cloud", que se encarga de recepcionar los mensajes de tipo "PointCloud2" publicados a través del tópico "clean_pose", que contienen la nube de puntos libre de errores. En base a unos parámetros determinados, añadirá cierto error de forma aleatoria a los puntos contenidos en la nube. Esta parte del proceso se usará solo cuando queramos probar la resiliencia frente a errores, es decir, el flujo de trabajo también está preparado para trabajar directamente con la nube sin error, como es de esperar, ya que la presencia de error añade complejidad al proceso y reducir la complejidad de este no debería impactar negativamente.

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 se ha optado por aplicar la siguiente transformación:

- **Máscara de ruido**: El ruido se aplica a la nube de puntos a partir de una máscara que, al operar con la nube original, introduce errores usando una distribución de probabilidad normal con media μ, desviación estándar σ y que se aplica a cualquier punto en cualquiera de sus ejes con una probabilidad p. El orden de operación es el siguiente:
  - **Creación de la máscara**: Se asigna una probabilidad aleatoria $u_i \sim \mathcal{U}(0,1)$ a cada punto i de la nube. Aquellos puntos para los que $u_i < p$ tendrán valor 1 en la máscara y, por tanto, se les aplicará el error.
  - **Distribución normal**: Se genera una variable aleatoria $n \sim \mathcal{N}(\mu, \sigma^2)$ que describe la distribución normal en función de los parámetros elegidos en la configuración.
  - **Aplicación de ruido**: Para cada punto seleccionado por la máscara, se modifica cada coordenada (x, y, z) como:

    $$x' = x + n_x, \quad y' = y + n_y, \quad z' = z + n_z$$

    donde $n_x, n_y, n_z \sim \mathcal{N}(\mu, \sigma^2)$ son muestras independientes de la distribución normal.

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

- **Inicialización**: Al iniciar el nodo, simplemente creará un suscriptor y un tópico de publicación:
  - **Suscripción al "clean_pcl"**: Este suscriptor recibirá mensajes de tipo "PointCloud2" conteniendo la nueva nube de puntos a la que se añadirá 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 provocadas por la recepción de mensajes a través del tópico "new_pcl"**: La recepción de este tipo de mensajes provoca la llamada de una función que se encarga de aplicar una máscara de ruido a la nube de puntos recibida y, posteriormente, publicarla a través del tópico "noisy_pcl".

En resumen, este nodo se encarga de transformar la nube de puntos original a un estado simulado más 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**: Dado que las nubes de puntos son recibidas en formato ”PointCloud2”, es necesario transformarlas a un formato ”open3d.geometry.PointCloud()” para poder trabajar con ellas usando las librerías Open3D que se usarán a lo largo del proceso.

- **Submuestreo (Downsampling)**: Dado que el mapa con el que trabajamos tiene una resolución determinada, es necesario reducir la resolución de las nubes de puntos que recibimos para que esta coincida con la del mapa. Esto se hace para reducir la cantidad de puntos a procesar, lo que mejora la eficiencia computacional y reduce el ruido. En nuestro caso, se usa un método de submuestreo basado en un voxel grid, donde se agrupan los puntos dentro de celdas de un tamaño determinado (tamaño del voxel) y se reemplazan por un solo punto representativo (generalmente el centroide de los puntos en esa celda).
- **Eliminación de outliers**: Las nubes de puntos pueden contener puntos erróneos o aislados que no representan la realidad del entorno. Estos outliers pueden ser causados por ruido en la captura, errores de medición o reflejos. La eliminación de outliers es crucial para mejorar la calidad de la nube de puntos y la precisión de los cálculos posteriores. En nuestro caso, se usa un método basado en la distancia media a los vecinos más cercanos, donde se eliminan los puntos que están demasiado lejos de sus vecinos. Este proceso se realiza después del submuestreo para evitar eliminar puntos que podrían ser relevantes en la nube original.
- **Traslación al origen**: Las nubes de puntos capturadas suelen estar referenciadas a un sistema de coordenadas distinto al origen (0,0,0). Para facilitar el cálculo del desplazamiento entre nubes y su emparejamiento con la nube anterior, es necesario trasladarlas al sistema de coordenadas del origen. Esto permite que las transformaciones posteriores sean más consistentes y precisas.
- **Cálculo de normales**: El cálculo de las normales de los puntos en la nube es esencial para muchos algoritmos de procesamiento de nubes de puntos, incluyendo el emparejamiento ICP. Las normales proporcionan información sobre la orientación de las superficies representadas por la nube de puntos, lo que ayuda a mejorar la precisión del emparejamiento. En nuestro caso, se calcula la normal de cada punto en función de sus vecinos más cercanos, usando un método basado en la covarianza local.

- **ICP (Iterative Closest Point)**: El algoritmo *Iterative Closest Point* (ICP) es un método ampliamente utilizado para la alineación de nubes de puntos en aplicaciones de visión

por computadora, robótica y reconstrucción 3D. Su objetivo es encontrar la transformación rígida óptima (rotación y traslación) que minimiza la distancia entre dos conjuntos de puntos: una nube de puntos *fuente* $\mathcal{P} = \{\mathbf{p}_i\}_{i=1}^{N}$ y una nube de puntos *objetivo* $\mathcal{Q} = \{\mathbf{q}_j\}_{j=1}^{M}$. Ahora describiremos su formulación matemática y el algoritmo paso a paso.

- **Formulación Matemática**: El problema se modela como la búsqueda de una transformación rígida $(\mathbf{R}, \mathbf{t})$, donde $\mathbf{R} \in SO(3)$ es una matriz de rotación y $\mathbf{t} \in \mathbb{R}^3$ es un vector de traslación, que minimice la siguiente función de costo:

$$\min_{\mathbf{R},\mathbf{t}} \; E(\mathbf{R}, \mathbf{t}) = \frac{1}{|\mathcal{C}|} \sum_{(i,j)\in\mathcal{C}} \left\| \mathbf{q}_j - \left(\mathbf{R}\mathbf{p}_i + \mathbf{t}\right) \right\|^2$$

  donde $\mathcal{C}$ es el conjunto de correspondencias entre puntos de $\mathcal{P}$ y $\mathcal{Q}$ determinado en cada iteración.

- **Algoritmo**: El procedimiento estándar de ICP consiste en los siguientes pasos iterativos:

  1. **Inicialización:** Se establece una estimación inicial de la transformación $(\mathbf{R}_0, \mathbf{t}_0)$, que puede ser la identidad si no se dispone de información previa.
  2. **Correspondencia:** Para cada punto $\mathbf{p}_i$ en la nube fuente, se busca el punto más cercano $\mathbf{q}_j$ en la nube objetivo según la métrica euclidiana:

$$j = \arg\min_{k} \left\| \mathbf{q}_k - \left(\mathbf{R}\mathbf{p}_i + \mathbf{t}\right) \right\|$$

     formando así el conjunto de correspondencias $\mathcal{C}$.
  3. **Optimización:** Se resuelve el problema de mínimos cuadrados para encontrar la nueva transformación $(\mathbf{R}, \mathbf{t})$ que minimiza el error cuadrático medio sobre las correspondencias. La solución cerrada se obtiene usando el método de Umeyama o descomposición en valores singulares (SVD):
     (a) Calcular los centroides de los puntos emparejados:

$$\bar{\mathbf{p}} = \frac{1}{|\mathcal{C}|} \sum_{(i,j)\in\mathcal{C}} \mathbf{p}_i, \qquad \bar{\mathbf{q}} = \frac{1}{|\mathcal{C}|} \sum_{(i,j)\in\mathcal{C}} \mathbf{q}_j$$

     (b) Construir la matriz de correlación:

$$\mathbf{H} = \sum_{(i,j)\in\mathcal{C}} (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_j - \bar{\mathbf{q}})^T$$

(c) Obtener $\mathbf{R}$ y $\mathbf{t}$ mediante SVD:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad \Rightarrow \quad \mathbf{R} = \mathbf{V}\mathbf{U}^T, \qquad \mathbf{t} = \bar{\mathbf{q}} - \mathbf{R}\bar{\mathbf{p}}$$

4. **Transformación:** Actualizar la nube fuente aplicando la transformación obtenida.
5. **Convergencia:** Repetir los pasos anteriores hasta que el cambio en el error $E(\mathbf{R}, \mathbf{t})$ entre iteraciones esté por debajo de un umbral ε o se alcance el número máximo de iteraciones.

- **Variantes de ICP**: Existen variantes que mejoran la robustez y velocidad del algoritmo, entre ellas:
  - **ICP punto a punto (Point-to-Point):** Minimiza la distancia euclidiana entre puntos correspondientes (como en la formulación anterior).
  - **ICP punto a plano (Point-to-Plane):** Minimiza la distancia entre el punto transformado y el plano tangente en $\mathbf{q}_j$, lo que conduce a una convergencia más rápida:

    $$E(\mathbf{R}, \mathbf{t}) = \frac{1}{|\mathcal{C}|} \sum_{(i,j)\in\mathcal{C}} \left(\mathbf{n}_j^T \left(\mathbf{q}_j - (\mathbf{R}\mathbf{p}_i + \mathbf{t})\right)\right)^2$$

    donde $\mathbf{n}_j$ es el vector normal asociado a $\mathbf{q}_j$.
- **Implementación en Open3D**: La librería `Open3D` proporciona una implementación optimizada de ICP en C++ y Python. Su función principal `registration_icp()` permite elegir entre varias métricas de alineación (`TransformationEstimationPointT` `TransformationEstimationPointToPlane`) y utiliza aceleradores de búsqueda de vecinos como KD-Trees para obtener correspondencias eficientes. Además, Open3D admite:
  - **Jerarquía multiescala:** aplicación de ICP en niveles de resolución decreciente para mejorar la convergencia global.
  - **Criterios de parada:** tolerancia de convergencia en función de la reducción relativa del error y límite de iteraciones.
  - **Transformación inicial:** opción de especificar $(\mathbf{R}_0, \mathbf{t}_0)$ para mejorar los resultados cuando se dispone de una estimación previa.

- **Selección de fotogramas clave (Key frame selection)**: Anteriormente se comenta cómo no todas las nubes pueden formar parte del mapa, ya que todos los emparejamientos realizados contienen un mínimo de error debido a que la naturaleza del método de emparejamiento es estadística y no es perfecta. De forma que, si componemos este error, aunque mínimo o imperceptible a simple vista entre las capturas iniciales, se va acumulando con el tiempo, dando lugar a que la pose termine por corromperse y deje de representar de forma correcta la pose real.
  Esto nos lleva a desarrollar un proceso de curado y selección de fotogramas que, si cumplen una serie de requisitos, formarán parte del mapa y, por tanto, contribuirán a las siguientes iteraciones. Por ello, es de gran interés que estos emparejamientos sean lo más precisos posibles y que se realicen con una frecuencia adecuada. En concreto, se ha desarrollado una clase llamada ”KeyFrameSelector” que se encarga de gestionar este proceso. Esta clase se discutira mas en profundidad mas adelante.

Funcionamiento del componente: Tal y como se ha ido introduciendo, 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. A contuación se describen 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 pub-

licar 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:
    - **Pendiente: Resumen actual**
    - **El numero de frames es que han pasado desde la ultima comprobación es menor que un umbral** Se calcula ICP entre la nube de puntos actual y la anterior, se obtiene la transformación resultante y se compone con la pose. **Si FITNESS, RMSE, TRANSLATION y ROTATION cumplen los requisitos** Se actualiza la pose y se almacena la nube de puntos actual como la anterior
    - **El numero de frames es que han pasado desde la ultima comprobación es mayor o igual que el umbral o el emparejamiento no cumple los requisitos anteiores** Se comprueba si el frame actual puede ser una imagen clave.
  - **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.

4.1.4 Seleccion de fotogramas clave

El principio fundamental de la tecninca de emparejamiento que se usa en este proyecto (ICP), al ser un metodo estadistico, es que siempre va a contener un minimo de error. Aunque este error sea minimo, si se va acumulando con el tiempo, puede llegar a corromper la pose y que esta deje de representar la realidad. Por ello, es necesario un proceso de curado y seleccion de fotogramas que construyan el mapa. Este proceso debe ser riguroso ya que si se aceptan muchas nubes de puntos, el error se acumulara mas rapidamente y si se aceptan pocas, el mapa sera pobre, no representara la realidad y hara mas complicado que se encuentren emparejamientos entre la nube actual y el mapa.

Conceptos Clave: El unico concepto clave que sustenta este proceso es el siguiente:

- **Criterios de selección de fotogramas clave**: La selección de fotogramas clave se basa en la evaluación de varios criterios que determinan si una nube de puntos debe ser añadida al mapa. Estos criterios incluyen:
  - **Distancia mínima desde la última clave**: Se define una distancia mínima que la nueva nube de puntos debe tener respecto a la última nube clave añadida al mapa. Esto asegura que las nubes de puntos añadidas al mapa estén suficientemente separadas espacialmente, lo que ayuda a cubrir más área del entorno y reduce la redundancia. Matemáticamente, si $\mathbf{p}_{\text{new}}$ es el centroide de la nueva nube de puntos y $\mathbf{p}_{\text{last}}$ es el centroide de la última nube clave, se requiere que:

    $$\|\mathbf{p}_{\text{new}} - \mathbf{p}_{\text{last}}\| > d_{\text{min}}$$

    donde d_{min} es la distancia mínima requerida.
  - **Cambio angular mínimo**: Se establece un umbral de cambio angular que la nueva nube de puntos debe superar respecto a la última nube clave. Esto garantiza que las nubes de puntos se capturen desde diferentes orientaciones, lo que mejora la diversidad de perspectivas en el mapa. Matemáticamente, si $\mathbf{R}_{\text{new}}$ y $\mathbf{R}_{\text{last}}$ son las matrices de rotación asociadas a las nubes, el cambio angular θ se calcula como:

    $$\theta = \arccos\left(\frac{\text{trace}(\mathbf{R}_{\text{new}}\mathbf{R}_{\text{last}}^{\top}) - 1}{2}\right)$$

y se requiere que $\theta > \theta_{\text{min}}$, donde θ_{min} es el umbral mínimo de cambio angular.

- **Ratio de superposición**: Se calcula el ratio de superposición entre la nueva nube de puntos y el mapa existente. Si este ratio es demasiado alto, indica que la nueva nube no aporta información significativa al mapa, por lo que no se añade. Matemáticamente, si N_{overlap} es el número de puntos de la nueva nube que coinciden con el mapa y N_{total} es el número total de puntos en la nueva nube, el ratio de superposición r se define como:

$$r = \frac{N_{\text{overlap}}}{N_{\text{total}}}$$

y se requiere que $r < r_{\text{max}}$, donde r_{max} es el umbral máximo permitido.

Funcionamiento del componente: La clase "KeyFrameSelector" se encargara de gestionar este proceso de selección, almacenamiento y gestion de nubes clave. El funcionamiento de esta clase se puede describir de la siguiente manera:

- **Inicialización**: La inicialización de esta clase esta involucra al primer frame del proceso ya que este sera el frame de referencia. Ademas se compone por los siguientes pasos:
  - **Inicialización de variables esenciales**: Con la inicialización de la clase, se inicializan una serie de variables esenciales para el correcto funcionamiento de la clase, como la última nube clave, su pose asociada y el mapa actual, que en este punto unicamente contiene la primera nube de puntos.
  - **Lectura del archivo de configuración**: Es necesario leer un archivo de configuración de tipo YAML que contiene los parametros de evaluación de las condiciones para aceptar una nueva nube como clave.
- **Evaluación de nuevas nubes de puntos**: Este proceso recibe la nube de puntos actual sin ninguna transformacion, es decir en el tipo original del mensaje ("PointClou2), por lo que es necesario aplicar un preprocesamiento que en este caso incluye la transformar la nube al tipo "open3d.geometry.PointCloud()", realizar un submuestreo para igualar su resolción a la del mapa, trasladarla a la ultima pose en la que se añadio una nube clave, es decir correcta desde nuestra ultima estimación, eliminacion de outliers y calculo de normales. Una vez tenemos la nube de puntos preprocesada, se sigue el siguiente esquema:

- **Pendiente: Resumen actual:**: Al ser nubes de puntos que pueden estar mas alejadas de lo normal o complicar el emparejamiento por ICP, primero se realiza un alineamiento inicial usando un metodo de alineamiento global basado en características (RANSAC) que nos da una estimación inicial de la transformación entre la nube actual y el mapa. Esta estimación inicial ayuda a mejorar la convergencia del ICP posterior. Posteriormente, se realiza un emparejamiento ICP (Iterative Closest Point) entre la nube actual y la ultima nube clave del mapa. Con esta ultima transformacion se calculan las metricas de evaluación. En caso de que las cumpla, se empareja de nuevo esta nube pero esta vez con el mapa completo, y se devuelve la transformación resultante al nodo de localización. En caso de que no se cumpla igualmente se devulve la transformación resultante del emparejamiento con la ultima nube clave, ya que ayuda a reanclar la pose al mapa y reducir el drift, pero sin actualizar el mapa.

4.1.5 Construcción del mapa

El mapeo es un componente esencial en aplicaciones de robótica y visión por computadora, donde la representación eficiente y precisa del entorno 3D es crucial. En este contexto, las nubes de puntos seleccionadas como fotogramas clave se integran en una estructura de datos que permite almacenar y consultar información espacial de manera eficiente. Para este propósito se ha diseñado un nodo llamado ”mapping_node” que utiliza la biblioteca `Bonxai` para gestionar el mapa 3D. `Bonxai` es una biblioteca de código abierto diseñada para la representación, manipulación y consulta eficiente de datos volumétricos en 3D mediante una estructura jerárquica y dispersa de tipo *Voxel Grid*. Su objetivo es proporcionar una alternativa moderna y de alto rendimiento a bibliotecas como `Octomap`, manteniendo soporte para entornos de tamaño ilimitado (*unbounded*), almacenamiento eficiente de memoria y tiempos de acceso rápidos.
La estructura de datos de Bonxai permite representar el espacio 3D como una colección de celdas cúbicas (voxels) de tamaño uniforme $\Delta \in \mathbb{R}^+$, organizadas jerárquicamente para garantizar eficiencia en memoria y operaciones. Esta característica lo hace especialmente útil en aplicaciones de mapeo y localización en robótica, donde la representación del entorno debe ser consistente, escalable y fácilmente actualizable.

Conceptos clave: Los conceptos fundamentales que sustentan la construcción del mapa son conceptos intrinsecos del funcionamiento de Bonxai, siendo estos los siguientes:

- **Voxel Grid Jerárquico**: El espacio continuo $\mathbb{R}^3$ se discretiza en una cuadrícula regular de voxels con resolución Δ. Cada punto del espacio $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ se asigna a un voxel mediante:

$$\text{coord}(\mathbf{x}) = \left(\left\lfloor \frac{x}{\Delta} \right\rfloor, \left\lfloor \frac{y}{\Delta} \right\rfloor, \left\lfloor \frac{z}{\Delta} \right\rfloor \right) \in \mathbb{Z}^3$$

  donde cada coordenada entera representa un índice de voxel. Esta discretización define una partición del espacio:

$$\mathbb{R}^3 = \bigcup_{\mathbf{k} \in \mathbb{Z}^3} V_{\mathbf{k}}, \quad V_{\mathbf{k}} = \{\mathbf{x} \in \mathbb{R}^3 : \text{coord}(\mathbf{x}) = \mathbf{k}\}$$

  donde $V_{\mathbf{k}}$ es la celda cúbica asociada al índice $\mathbf{k}$.

- **Estructura jerárquica y esparcida**: Los voxels se almacenan de forma dispersa usando un árbol jerárquico, donde únicamente las celdas activas (ocupadas o modificadas) se mantienen en memoria. Esto permite que el costo de almacenamiento sea proporcional al número de celdas activas N:

$$\mathcal{O}(\text{memoria}) = \mathcal{O}(N)$$

  en lugar de depender del volumen total del espacio explorado.

- **Acceso eficiente mediante Accessor**: Bonxai proporciona un objeto `Accessor` que permite acceso de lectura/escritura en tiempo cercano a $\mathcal{O}(1)$ para operaciones de inserción, actualización y consulta:

$$\text{setValue} : \mathbf{k} \mapsto v, \qquad \text{value} : \mathbf{k} \mapsto v \text{ o } \varnothing$$

  donde $v \in T$ es el valor almacenado (por ejemplo, probabilidad de ocupación, intensidad de un sensor, etc.).

- **Almacenamiento de datos personalizados**: Cada voxel puede almacenar un tipo de dato arbitrario T, lo que permite utilizar Bonxai no sólo para mapas de ocupación binaria, sino también para mapas de distancia firmada (ESDF), probabilidades bayesianas de ocupación, valores de intensidad o información semántica.

- **Escalabilidad y precisión**: La resolución Δ se elige según la aplicación. Con coordenadas enteras de 32 bits, el rango máximo por eje es:

$$R_{\max} = 2^{32} \cdot \Delta$$

lo que, para $\Delta = 0.01$ m, cubre hasta ≈ 42 km por eje, suficiente para la mayoría de aplicaciones robóticas en entornos extensos.

- **Operaciones principales**: Además de inserción y consulta, Bonxai soporta iteración eficiente sobre los voxels activos:

$$\text{IterateAllCells} : \{\mathbf{k}_1, \ldots, \mathbf{k}_N\} \mapsto \{(\mathbf{k}_i, v_i)\}_{i=1}^{N}$$

lo que permite exportar, serializar o realizar operaciones globales sobre el mapa.

Funcionamiento del componente: Bonxai se integra en el nodo ”mapping_node” a traves de un componente facilitado por la propia libreria llamado ”BonxaiServer” que se encarga de gestionar la estructura de datos y proporcionar una interfaz sencilla para interactuar con el mapa. El funcionamiento de este nodo se puede describir de la siguiente manera:

- **Inicialización**: La inicialización de este nodo esta conformada por los siguientes pasos:
  - **Subscripción de ”new_pcl”**: Este subscriptor se encargara de recibir los mensajes de tipo ”PointCloud2” conteniendo la nube de puntos trasladada a la posicion calculada por el algoritmo.
  - **Inicialización del servidor Bonxai**: Se crea una instancia del servidor Bonxai como un componente al que podremos acceder para gestionar el mapa.
- **Respuestas provocada por la recepción de mensajes a traves del tópico ”new_pcl”**: Los mensajes recibidos por el tópico de ”new_pcl” iniciaran el proceso de integración de la nueva nube de puntos en el mapa. La ejecución de esta función comprueba que el mensaje recibido no es nulo, y en caso de que no lo sea, se llama a la funcion proporcionada por el servidor Bonxai para integrar la nube de puntos en el mapa y ademas publicar el mapa actualizado a traves de un tópico llamado ”bonxai_point_cloud_centers”.

5

Pruebas y validación de resultados

La validación de los resultados obtenidos por el sistema de localización y mapeo es crucial para asegurar su precisión y fiabilidad en aplicaciones del mundo real. Para ello, se han diseñado y ejecutado una serie de pruebas que evalúan el rendimiento del sistema tanto a nivel de localización como de construcción del mapa. A continuación, se describen las metricas de interes en cada caso, metodologías empleadas, los escenarios de prueba y los resultados obtenidos.

5.1 Métricas de interés

Antes de comenzar a describir las pruebas realizadas, es importante definir las métricas clave que se utilizarán para evaluar el rendimiento del sistema. Ademas de esto, como debemos evaluar las dos salidas principales del sistema (pose y mapa), las métricas se dividen en dos categorías: métricas de localización y métricas de mapeo.

5.1.1 Métricas de Evaluación de Localización

En el contexto de *SLAM* sin cierre de bucle, es fundamental cuantificar la precisión de la estimación de la trayectoria del robot a lo largo del tiempo. En esta sección se describen tres métricas ampliamente utilizadas: **Absolute Trajectory Error (ATE)**, **Relative Pose Error (RPE)** y **KITTI-style Drift**, incluyendo su formulación matemática, motivación y forma de interpretación.

Absolute Trajectory Error (ATE)

Definición matemática : Sea $\mathbf{T}_i^{gt} \in SE(3)$ la pose de referencia (ground-truth) en el instante i y $\mathbf{T}_i^{est} \in SE(3)$ la pose estimada por el sistema SLAM. El ATE mide el error de traslación absoluto después de alinear la trayectoria estimada con la de referencia mediante una transformación de similitud rígida $S \in Sim(3)$ (obtenida mediante un ajuste de mínimos cuadrados de tipo Horn o Umeyama):

$$\text{ATE}_{rmse} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| \text{trans}\big((\mathbf{T}_i^{gt})^{-1} \cdot (S\mathbf{T}_i^{est})\big) \right\|^2}$$

donde $\text{trans}(\cdot)$ extrae el vector de traslación de la transformación homogénea.

Motivación: El ATE evalúa el error acumulado de la trayectoria en coordenadas globales. Es especialmente relevante en escenarios sin cierre de bucle porque revela de manera directa la **deriva acumulada** en posición a lo largo de la secuencia.

Área de interés:

- Precisión global de localización
- Consistencia con el sistema de referencia absoluto
- Comparación directa de trayectorias completas

Interpretación:

- Valores bajos de ATE indican que la trayectoria estimada está bien alineada con la de referencia.
- Un incremento progresivo de ATE a lo largo del tiempo es indicativo de deriva acumulativa.
- El ATE es sensible a errores de escala (en monocular SLAM) y a falta de cierre de bucle.

Resultados:

Relative Pose Error (RPE)

Definición matemática: El RPE mide la consistencia local de la estimación de movimiento entre poses separadas por un intervalo temporal Δt. Definimos el error relativo entre la transformación de referencia y la estimada como:

$$\mathbf{E}_i = \left[(\mathbf{T}_i^{gt})^{-1}\mathbf{T}_{i+\Delta}^{gt}\right]^{-1} \cdot \left[(\mathbf{T}_i^{est})^{-1}\mathbf{T}_{i+\Delta}^{est}\right]$$

La métrica RPE de traslación (en RMSE) se calcula como:

$$\text{RPE}_{trans} = \sqrt{\frac{1}{M}\sum_{i=1}^{M} \|\text{trans}(\mathbf{E}_i)\|^2}$$

donde M es el número de pares válidos de poses. De forma análoga puede calcularse un error angular usando $\text{rot}(\mathbf{E}_i)$.

Motivación: El RPE mide la **precisión local del movimiento relativo**, independientemente de la deriva global. En ausencia de cierre de bucle, el RPE es muy útil porque puede permanecer bajo incluso si la trayectoria global se ha desviado, siempre que las estimaciones locales sean coherentes.

Área de interés:

- Consistencia local del odometría
- Estabilidad de la integración incremental
- Detección de saltos o discontinuidades en la estimación de pose

Interpretación:

- Valores bajos de RPE indican que la odometría es localmente coherente.
- Un RPE alto suele señalar errores en el cálculo de la transformación incremental (por ejemplo, fallos en la estimación de ICP, ruido excesivo de sensores o reseteos de pose).

Resultados:

KITTI-style Drift:

Definición matemática: En el benchmark KITTI, la deriva se evalúa sobre tramos de longitud L (p.ej. $L \in \{100, 200, \ldots, 800\}$ m). Para cada segmento se calcula el error relativo de traslación y de rotación:

$$\text{drift}(L) = \frac{1}{|S_L|} \sum_{s \in S_L} \frac{\|\text{trans}(\mathbf{E}_s)\|}{L}$$

donde S_L es el conjunto de todos los subtramos de longitud L y $\mathbf{E}_s$ es el error relativo de pose en el subtramo.

Motivación: Esta métrica ofrece una visión **normalizada por distancia recorrida** de la deriva, lo que permite comparar secuencias de distinta longitud.

Área de interés:

- Tasa de acumulación de error por metro recorrido
- Comparación con otros sistemas de odometría visual/SLAM en datasets estándar
- Identificación de segmentos de trayectoria con peor rendimiento

Interpretación:

- Un drift de traslación bajo (p.ej. $< 1\%$) indica que el sistema es robusto y que la trayectoria estimada sigue de cerca la referencia en cada tramo.
- Un incremento del drift con L es normal, pero un crecimiento excesivo puede indicar que la integración de movimiento está acumulando error rápidamente.

Resultados:

5.1.2 Métricas de Evaluación de Mapeo

En escenarios de *SLAM* sin cierre de bucle, además de la trayectoria, es fundamental evaluar la calidad del mapa reconstruido. Las métricas de mapeo cuantifican tanto la exactitud de los puntos estimados como la cobertura de la escena.

Chamfer Distance (Bidireccional)

Definición matemática: Sea P_{gt} el conjunto de puntos de referencia (ground-truth) y P_{slam} el conjunto de puntos reconstruidos por SLAM. La Chamfer distance bidireccional se define como:

$$d_{CD}(P_{gt}, P_{slam}) = \frac{1}{|P_{gt}|} \sum_{p \in P_{gt}} \min_{q \in P_{slam}} \|p - q\|^2 + \frac{1}{|P_{slam}|} \sum_{q \in P_{slam}} \min_{p \in P_{gt}} \|q - p\|^2$$

Descripción: Esta métrica calcula, para cada punto de un conjunto, la distancia al punto más cercano del otro conjunto y hace la media. La primera suma evalúa la cobertura de la reconstrucción (si faltan regiones), mientras que la segunda suma evalúa la exactitud (si se agregaron puntos espurios).

Relevancia en SLAM sin cierre de bucle En ausencia de cierre de bucle, los mapas pueden sufrir deformaciones locales y acumulación de errores. La Chamfer distance permite cuantificar estas desviaciones sin requerir correspondencias explícitas de topología.

Área de interés:

- Precisión global del mapa
- Cobertura y completitud de la escena
- Detección de artefactos o puntos espurios

Interpretación:

- Valores bajos: la reconstrucción es precisa y completa.
- Valores altos en $P_{gt} \to P_{slam}$: la reconstrucción es incompleta.
- Valores altos en $P_{slam} \to P_{gt}$: hay puntos espurios o ruido.

Resultados:

F-score τ

Definición matemática: Para un umbral de distancia τ, se define:

$$\text{Precision} = \frac{|\{q \in P_{slam} : \exists p \in P_{gt}, \|q - p\| < \tau\}|}{|P_{slam}|}$$

$$\text{Recall} = \frac{|\{p \in P_{gt} : \exists q \in P_{slam}, \|p - q\| < \tau\}|}{|P_{gt}|}$$

$$F\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Descripción: El F-score combina precisión y recall para dar un valor global de la calidad del mapa bajo un umbral de tolerancia. Se centra en si los puntos estimados están suficientemente cerca de la referencia y si la reconstrucción cubre la escena. Esta métrica es importante en el SLAM sin cierra de bucle ya que permite evaluar la calidad de la reconstrucción sin depender de la alineación perfecta global, lo cual es útil cuando la trayectoria ha acumulado deriva.

Área de interés:

- Completitud vs. exactitud bajo un umbral de tolerancia
- Comparación rápida entre distintas reconstrucciones
- Evaluación de regiones críticas de la escena

Interpretación:

- F-score cercano a 1: buena precisión y cobertura.
- Baja precisión: muchos puntos espurios.
- Bajo recall: la reconstrucción no cubre todo el mapa.

Resultados:

Distribución de Errores (Percentiles)

Definición matemática: Sea $d_i = \min_{q \in P_{slam}} \|p_i - q\|$ la distancia del punto $p_i \in P_{gt}$ a su vecino más cercano en P_{slam}. Se calculan percentiles $p_{50}, p_{90}, p_{95}, p_{99}$ del conjunto $\{d_i\}$:

$$p_x = \text{percentil}_x(\{d_i\})$$

Descripción: La distribución de errores permite conocer no solo el error típico (mediana), sino también cómo se comportan los casos más extremos.

Relevancia en SLAM sin cierre de bucle: Es especialmente útil para detectar zonas donde la reconstrucción falla debido a deriva local, ya que puede haber outliers importantes incluso si la mayoría de la nube es correcta.

Área de interés:

- Errores típicos y extremos
- Detección de outliers
- Evaluación de robustez local de la reconstrucción

Interpretación:

- p_{50}: error típico de la mayoría de puntos.
- p_{90}, p_{95}: errores significativos de regiones concretas.
- p_{99}: outliers extremos.
- Histogramas de error ayudan a identificar patrones de error espacial.

Resultados:

References

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[2] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[3] D. Faconti, "Bonxai: Fast, hierarchical, sparse voxel grid." `https://github.com/facontidavide/Bonxai`, 2024. Release v0.6.0 (2024-11-10). MPL-2.0 license.

[4] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1987.

[5] Wired Staff, "Voxels allow doctors to get beneath the surface." Wired (online, 1996), 1996. Describes the usage of voxels for 3D visualization from CT and MRI.

[6] VOXEL-MAN Project, "History of the voxel-man project (first 3d medical voxel visualizations)." University Medical Center Hamburg-Eppendorf (VOXEL-MAN documentation), 1986. Early 3D voxel-based visualization combining CT and MRI.

[7] Wikipedia contributors, "Voxel." Wikipedia, latest revision, 2025. Definition and application of voxels in visualization and GIS.

[8] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987. Algorithm extracting polygonal meshes from voxel data.

[9] Python Software Foundation, "Python programming language," 2025. Version 3.x, accessed 2025-09-08.

[10] ISO/IEC JTC1/SC22/WG21, "Iso/iec 14882:2020 – programming language c++," 2020. C++20 Standard, accessed 2025-09-08.

[11] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml) 1.0 (fifth edition)," 2008. W3C Recommendation, accessed 2025-09-08.

[12] O. Ben-Kiki, C. Evans, and B. Ingerson, "Yaml ain't markup language (yaml) version 1.2," 2021. Specification for human-friendly data serialization standard.

[13] Kitware, Inc., "Cmake build system," 2025. Cross-platform build-system generator, accessed 2025-09-08.

[14] L. Lamport, *LaTeX: A Document Preparation System*. Reading, Massachusetts: Addison-Wesley, 2 ed., 1994. 2nd edition.

[15] Canonical Ltd., "Ubuntu operating system," 2025. Debian-based GNU/Linux distribution, accessed 2025-09-08.

[16] Microsoft Corporation, "Visual studio code," 2025. Source-code editor developed by Microsoft, accessed 2025-09-08.

[17] GitHub, Inc., "Github," 2025. Web-based platform for version control and collaboration, accessed 2025-09-08.

[18] Project Jupyter Contributors, "Project jupyter," 2025. Open-source platform for interactive computing, accessed 2025-09-08.

[19] Terminator Project, "Terminator terminal emulator," 2025. Terminal emulator supporting multiple split panes, accessed 2025-09-08.

Appendix A
Installation Manual

Requirements: