Universidad Miguel Hernández de Elche

MÁSTER UNIVERSITARIO EN ROBÓTICA

INTEGRACIÓN DE TECNOLOGÍAS LASER Y GPS/RTK PARA NAVEGACIÓN AUTÓNOMA EN EXTERIORES

Trabajo de Fin de Máster

2022

Autor: Enrique Heredia Aguado
Tutores:
· Arturo Gil Aparicio
· David Valiente García

Agradecimientos

Agradezco en primer lugar a mis tutores Arturo y David por todo el apoyo y optimismo que me han aportado a lo largo de los meses que ha durado el proyecto. Han sabido guiar y acotar de manera muy eficiente y correcta el desarrollo de este trabajo hasta alcanzar el resultado que aquí se presenta. Sin toda esta ayuda estos resultados no hubieran sido posibles.

Agradezco también a Alejandro Rujano por su ayuda revisando el documento.

Por último pero no menos importante a mis padres y mi hermana por su apoyo continuo, en esta y otras aventuras.

Resumen

En este proyecto se revisan diferentes técnicas de SLAM. Se ha tomado el algoritmo Cartographer como el más adecuado para el caso de uso planteado (entornos semi-estructurados de interior y exterior). Se valida el algoritmo mediante diferentes experimentos que incluyen tanto una plataforma robótica real como en un entorno simulado. El trabajo incluye la preparación y puesta a punto de la plataforma robótica así como de los sensores elegidos, su calibración, integración y validación independiente e integrada.

Palabras clave: SLAM, robótica móvil, navegación, LIDAR, GPS-RTK, ROS

Índice general

Índice de figuras

Índice de tablas

Glosario

BDS BeiDou Navigation Satellite System.

DEM Digital Elevation Map.

DGPS Differential GNSS.

frame Dentro del ecosistema ROS, un *frame* es un sistema de coordenadas. ROS incluye una librería para gestionar estos sistemas de coordenadas así como las transformadas entre los mismos. Es parte del núcleo de ROS y permite gestionar diferentes tipos de robots compuestos tanto por partes fijas como móviles.

GALILEO Sistema GNSS Europeo.

Gazebo Es un simulador de robótica 3D de código abierto. Integra el motor físico ODE y emplea OpenGL para la visualización. Aunque no es exclusivo, se integra con el ecosistema ROS mediante diferentes drivers para comunicar las diferentes funcionalidades del simulador con la información de ROS.

GLONASS Globalnaya Navigazionnaya Sputnikovaya Sistema.

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

IMU Unidad de medición inercial.

LIDAR Light Detection and Ranging.

NMEA National Marine Electronics Association.

nodo ROS Dentro del ecosistema ROS constituyen la unidad básica de ejecución. Es un módulo compuesto por un solo ejecutable que, utilizando las diferentes librerías provistas por ROS, podrá comunicarse con otros nodos o acceder a los servicios provistos por el ROS Master.

OBC On Board Computer. Ordenador de a bordo del robot.

PointCloud También llamado nube de puntos, es un conjunto de puntos en un sistema de coordenadas en tres dimensiones. Típicamente para cada vértice se almacena la posición X,Y,Z. Diferentes sensores que permiten medir distancias exportarán la información a este tipo de formato.

ROS Acrónimo de Robot Operating System.
Se compone de un conjunto de librerías que proporcionan un ecosistema diseñado específicamente para trabajar con robótica. Provee de funcionalidades para encapsular los diferentes módulos desarrollados ayudando en la comunicación de los mismos mediante diferentes métodos que permitirán ejecutar tanto en modo centralizado, en un solo ordenador, como de manera distribuida a lo largo de diferentes máquinas conectadas en red.
Permite el uso e integración de módulos desarrollados en diferentes lenguajes de programación incluyendo herramientas para asegurar una correcta compilación y ejecución de las mismas, generando el código necesario para la correcta comunicación y sincronización de los diferentes módulos.

ROS Master Dentro del ecosistema ROS el ROS Master es un proceso que funciona permitiendo a los diferentes Nodos ejecutados localizarse e iniciar la comunicación entre los mismos. Provee además de un sistema de gestión de parámetros disponible para todos los Nodos conectados al mismo.

rosbag Es un paquete de ROS que permite trabajar con ficheros *bag*, que permiten almacenar mensajes de ROS tal y como se han ejecutado. Posteriormente pueden reproducirse como si de un vídeo se tratara.

RRT Rapidly-exploring random tree.

RTK Real-Time Kinematic.

rviz Es una herramienta de visualización para robótica integrada en ROS. Incluye visualización 3D y 2D para diferentes tipos de datos así como posibilidad para interactuar y comandar directamente la plataforma. Es extensible mediante plugins que permiten añadir funcionalidades específicas adaptadas al caso de uso, plataforma robótica o entorno en el que se trabaje.

SLAM Simultaneous Localization And Mapping.

SPS Single Point Positioning or Standard GNSS.

Tk Librería para desarrollo de interfaces de gráficas usuario. Proporciona un conjunto de herramientas para administrar ventanas así como diferentes *widgets* que permiten desarrollar de manera rápida interfaces gráficas. Está integrado en diferentes lenguajes, como Python.

TOF Time Of Flight.

topic Dentro del ecosistema ROS componen la unidad básica de comunicación entre Nodos. Son buses de datos que siguen un patrón de publicación/suscripción anónima. Esto significa que al iniciar, cada Nodo notificará al ROS Master su rol como proveedor/suscriptor de información sobre un topic específico, identificándolo mediante el nombre del mismo.

UDP User Datagram Protocol.

UHF Ultra High Frequency.

URDF Unified Robot Description Format.

UTM Universal Transverse Mercator.

Capítulo 1

Introducción

La robótica en general y la robótica móvil en particular ha ido experimentando en los últimos años un incremento en su popularidad empezando a ocupar cada vez más espacios tanto en la investigación como en aplicaciones reales. Sin ir más lejos, en un análisis de las búsquedas realizadas en IEEE Xplore, referencia en el mundo de investigación en ámbitos técnicos y tecnológicos, desarrollado en [15] se pudo comparar el crecimiento de búsquedas relacionadas, no solo con la robótica, si no también con la robótica móvil. Puede apreciarse el notable aumento experimentado en los últimos años en la figura 1.1.

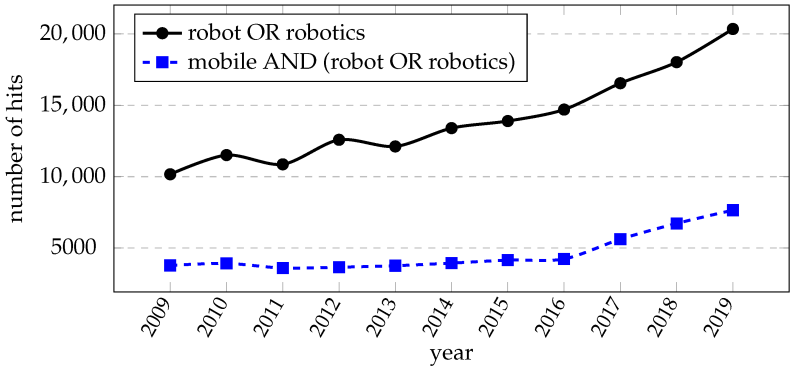


Figura 1.1: búsquedas que incluyen las palabras robot o robótica y concretamente robótica móvil en el buscador de IEEE Xplore

Fuente Imagen: Imagen del extraída de [15]

Este creciente interés no solo se ve reflejado en la investigación, si no que tiene un impacto directo a nivel comercial, tal y como presenta [23], viéndose un incremento en las ventas en diferentes aplicaciones como la robótica para logística, ambientes públicos, limpieza profesional o robótica médica.

Pero ¿qué es lo que entendemos por robótica móvil? Antes de llegar ahí conviene revisar algunas definiciones básicas. Centrándonos en las definiciones propuestas por la norma ISO8373 en [29] cabe destacar las siguientes, que han sido traducidas para incluirlas en este documento. Nótese que se ha mantenido la numeración en cuanto a las secciones extraídas del documento original:

3.1 robot: mecanismo actuado y programado con un cierto grado de autonomía (3.2) para llevar a cabo locomoción, manipulación o posicionamiento.
3.2 autonomía: habilidad para llevar a cabo una tarea dada basándose en el estado actual y percibido, sin intervención humana.
4.15 robot móvil: robot (3.1) capaz de desplazarse bajo su propio control. Además de las operaciones autónomas, un robot móvil puede incluir medios para su control remoto. Un robot móvil puede ser una plataforma móvil incluyendo o no manipuladores.
4.16 plataforma móvil: conjunto de componentes ensamblados que permiten la locomoción.

Otros autores como [30] coinciden en la definición de robot móvil remarcando relación en cuanto a autonomía entre percepción y acción por parte de la máquina o robot. Algunas características destacadas incluyen la capacidad de desplazarse, enfatizando la seguridad tanto para sí mismos como para el entorno, maximizando la flexibilidad en sus funcionalidades. Además de la definición vista, [16] incide en la relevancia de la necesidad de cualquier robot móvil de estar equipado con:

- actuadores que permiten el movimiento del robot.
- sensores que permiten adquirir conocimiento sobre el medio en que se encuentra el robot así como de sí mismo.
- inteligencia (algoritmo, controlador, etc) que le permitirá relacionar los componentes anteriores para llevar a cabo las tareas determinadas.

Como puede verse, tanto las definiciones como requerimientos son agnósticas en cuanto al ámbito de aplicación del robot. En este proyecto nos centraremos en la relación entre estos tres aspectos enfocados a robots móviles que combinan ámbitos de interior de edificios como de exterior.

Dentro de las posibilidades que pueden desarrollarse en la parte nombrada como *inteligencia* pueden establecerse diferentes grados de autonomía. Desde autonomía en cuanto a localización, a generación de mapas, a reconocimiento de objetivos o incluso autonomía en cuanto a ejecución de misiones completas. En general estos niveles de autonomía se superponen unos sobre otros por lo que cabría empezar por el nivel inferior: autonomía en la locomoción. Para que un robot pueda desplazarse por un entorno de manera segura habrá que cubrir diferentes aspectos como tener (o construir) un mapa del entorno, ser capaz de localizarse en el y poder tanto planificar como ejecutar trayectorias para desplazarse de un punto a otro. Volviendo sobre las definiciones propuestas por [29] se entiende el proceso de navegación como aquel que incluye la planificación de trayectorias, la localización y el mapeado.

El término navegación también ha sufrido un aumento notable del interés que suscita esta área. [15] presenta un análisis sobre las búsquedas en IEEE Xplore, en este caso de los términos *Navegación*, *Autonomía* y *Optimización/Aprendizaje* (en inglés). puede verse en la figura 1.2 como la navegación en robótica móvil acumula el mayor interés y crecimiento de los últimos años.

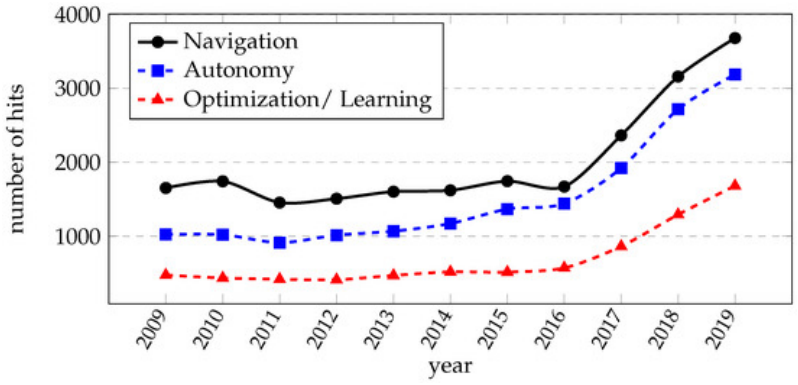


Figura 1.2: búsquedas de los términos navegación, autonomía y optimización/aprendizaje en el buscador de IEEE Xplore

Fuente Imagen: Imagen del extraída de [15]

Todas las partes involucradas en la navegación se verán fuertemente influenciadas por la representación que se haga del entorno, es decir, del mapa que se cree. El tipo de mapa limitará las operaciones que se pueden realizar así como influirá en el modo de localización y planificación de trayectorias sobre el mismo. En el siguiente capítulo se hará un repaso del estado del arte en el ámbito principalmente del SLAM, que incluye la formación de mapas y la localización sobre los mismos.

En el resto del capítulo se presenta de manera resumida la motivación del proyecto, los objetivos marcados así como la estructura del documento completo.

1.1. Motivación del proyecto

Previo a la aplicación en cualquier entorno de una solución de robótica autónoma, es necesario delimitar tanto la plataforma robótica (incluyendo actuadores) como los sensores que se van a emplear. Este proyecto busca integrar diferentes sensores en una plataforma de manera que puedan explorarse diferentes aspectos de la navegación autónoma, concretamente la formación de mapas y localización del robot o SLAM. De las diversas soluciones existentes se pondrá especial atención a trabajar con implementaciones optimizadas que permitan una ejecución en tiempo real sin abusar de los recursos computacionales disponibles. Hay que destacar el interés por mezclar entornos de interior como de exterior.

Esta base permitirá a futuro continuar investigando en otras áreas teniendo una plataforma autónoma como base.

1.2. Objetivos

Visto el enfoque que se plantea en el proyecto, es necesario acotar unos objetivos concretos. Se plantean los siguientes, que serán revisados nuevamente al final del documento:

- Puesta en marcha de la plataforma robótica: La plataforma robótica debe quedar operativa configurándose accesos y configuración básica e implementando diferentes procedimientos para gestionar tanto la configuración como funcionamiento de la propia plataforma como del hardware integrado.

- Integración de un sensor láser en la plataforma robótica elegida: el sensor LIDAR quedará montado sobre la plataforma, comunicando la información al OBC. Los datos deben ser integrados dentro del ecosistema ROS, publicándose en un topic con el tipo de mensaje adecuado.

- Integración, calibración y puesta en marcha de un sensor IMU: el sensor quedará conectado a la plataforma robótica comunicándose a su vez con el OBC. Se calibrará y ajustará de manera que quede operativa, publicándose la información en un topic dentro del ecosistema ROS.

- Integración de una antena GNSS-RTK con su par homólogo: se harán las pruebas pertinentes para validar el funcionamiento en diferentes condiciones para finalmente integrar la antena en la plataforma robótica. La información deberá publicarse en un topic haciendo uso del tipo de mensaje adecuado.

- Exploración de diferentes algoritmos de SLAM existentes e integración y pruebas tanto en interiores como exteriores. Se tomará uno integrado en el ecosistema ROS para hacer diferentes pruebas con el mismo y validar su uso tanto en interiores como en exteriores.

- Navegación autónoma en exteriores utilizando el *navigation stack* de ROS: partiendo de la localización y mapas generados mediante el sistema de SLAM se adaptarán las herramientas existentes para su correcto funcionamiento en la plataforma elegida y en exteriores navegando por el entorno esquivando los posibles obstáculos.

1.3. Herramientas utilizadas

Durante el proyecto se ha trabajado con diferentes herramientas, las más relevantes se presentan a continuación ya que serán referenciadas a lo largo del documento:

- bash: Es un interprete de lenguaje de scripting integrado en Linux. Permite desarrollar herramientas para simplificar y automatizar tareas.

- Python: Es un lenguaje de programación interpretado. Se empleará para diferentes herramientas de análisis, interfaces gráficas y en algunos módulos integrados en ROS.

- C++: Es un lenguaje de programación fuertemente tipado y compilado. Se utilizará en algunos módulos integrados en ROS.
- ROS: Tanto el robot como las pruebas simuladas utilizan ROS como base.
- Gazebo: Las pruebas simuladas se efectuarán sobre un escenario virtual con un robot y sensores análogos.
- rviz: La monitorización de cómo se generan los mapas se podrá visualizar en la herramienta rviz. Además permite visualizar otros topics de interés como son la posición del robot o los PointClouds generados.
- Cartographer: Herramienta de SLAM integrada en ROS que se evaluará en diferentes escenarios.

1.4. Estructura del documento

A continuación y para facilitar la lectura del documento, se detalla el contenido de cada capítulo:

- En el capítulo 1 se realiza una introducción.
- En el capítulo 2 se hace un repaso del estado del arte en torno a la generación de mapas en robótica móvil.
- En el capítulo 3 se describe la base teórica de los diferentes conceptos que se aplicarán en este proyecto.
- En el capítulo 4 se describe la plataforma que se ha utilizado así como el hardware incluido y la estructura de software y configuración desarrollada.
- En el capítulo 5 se detalla el desarrollo experimental llevado a cabo.
- En el capítulo 6 pueden encontrarse algunas conclusiones generales del proyecto así como el planteamiento de desarrollos futuros.
- Al final del documento puede encontrarse una lista con la bibliografía incluida a lo largo del documento.

Capítulo 2

Estado del arte

En este capítulo se hace un repaso sobre las diferentes opciones en cuanto a generación de mapas y localización en los mismos que existen, revisando los diferentes enfoques y datos empleados.

Como se ha introducido en el primer capítulo el tipo de mapa que se utilice marcará notablemente los algoritmos e información que pueda manejar el robot. Tal y como se destaca en [22], un mapa deberá cumplir con:

- La precisión del mapa deberá ir acorde con la precisión esperada por el robot para poder cumplir con sus objetivos.
- Las características representadas en el mapa deben provenir de las características del entorno extraídas a través de los sensores implicados.
- La coste computacional tanto en el proceso de mapeado, como de localización se verá afectada por la complejidad en la representación del mapa.

De forma general, tal y como explica en [22], en el ámbito de la robótica móvil los mapas suelen dividirse en dos tipos:

- Mapas métricos: muestra información geométrica del entorno. Buscan ser muy precisos en la representación de objetos en el entorno y son muy intuitivos de interpretar desde el punto de vista humano. En un mapa métrico una localización o un objeto se presenta como una coordenada en el espacio representado. En general pueden modelarse mapas representando la posición de diferentes características relevantes (*Feature-based Map Representation*) o puede optarse por representar todo el espacio al que puede acceder el robot (*Free-space Map Representation*). Esta segunda es la más común. Puede verse como sería una representación del espacio en un mapa métrico en la figura 2.1.
- Mapas topológicos: son mapas que no se centran en la geometría exacta del entorno, si no en características relevantes del entorno con las que localizarse y navegar de manera adecuada. Este tipo de mapas emplean nodos, representando diferentes localizaciones en el entorno a las que el robot puede acceder, unidas mediante líneas que conectarán nodos adyacentes, es decir, nodos que están físicamente comunicados. Puede verse una representación topológica del espacio en la figura 2.1.

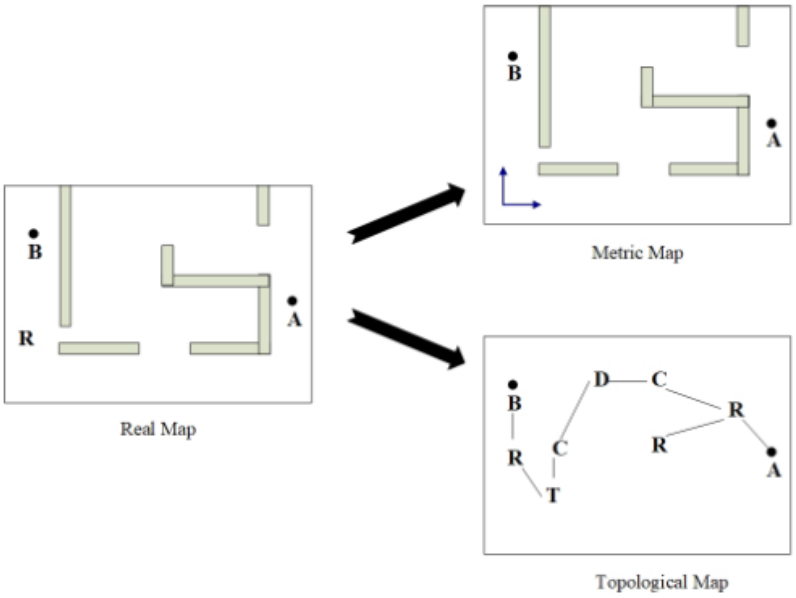


Figura 2.1: comparación de mapas topológicos y métricos
Fuente Imagen: Extraída de [9]

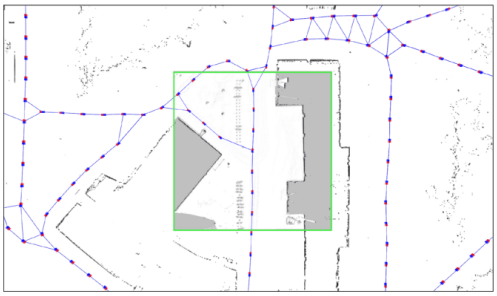
Nótese que el uso de uno u otro no es exclusivo, ambas representaciones pueden emplearse al mismo tiempo o a diferentes niveles de procesamiento. Existen mapas que combinan ambas representaciones.

Aparte de esta división en base a como se representa el espacio, se pueden diferenciar diferentes tipos de mapa en base a que representan del espacio. Se pueden encontrar:

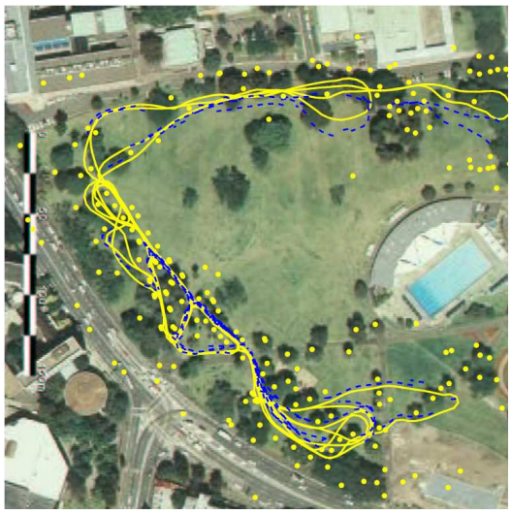
- Mapa de grafo: Representa el entorno en base a nodos y uniones entre ellos. En la figura 2.2a, extraída de [17] puede verse un ejemplo de este tipo de mapas. En este caso concreto se combina un mapa topológico, como el presentado en la figura, con mapas de ocupación locales que entrarán en juego cuando el robot se encuentre en cada uno de los nodos.

- Mapa de *landmarks*: destacando por la gran compacidad en cuanto a la representación del terreno, un mapa de *landmkars* representan la posición de un conjunto de puntos concretos del entorno en un sistema de coordenadas global. Estos puntos pueden ser reconocidos por el robot para así localizarse con respecto a ese sistema de coordenadas global. Puede verse un ejemplo de este tipo de mapas, con los diferentes *landmarks* dibujados, en la figura 2.2b extraído de [10]. Se trata de un tipo de mapa métrico que se basa en una representación de características del entorno o *Feature-based Map Representation*.

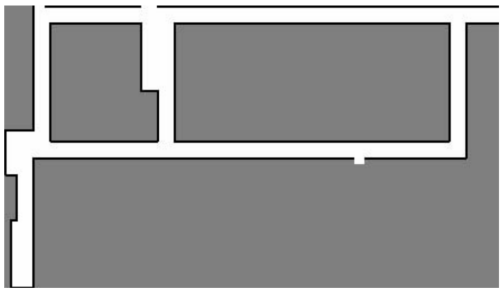
- Ocupación del entorno. El uso de mapas de ocupación u *Occupancy Grid* predomina en el modelado del entorno dentro de la robótica móvil. Consisten en representaciones del espacio discretizado mediante rejillas, de manera que para cada celda se asocia un valor representando la ocupación o no del espacio. La obtención de estos mapas no es sencilla y hay diferentes métodos para obtener



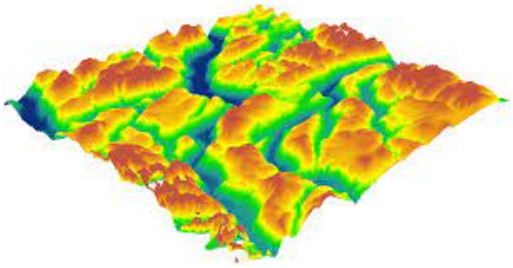
(a) Mapa topológico representado sobre un mapa de opcupaciónl



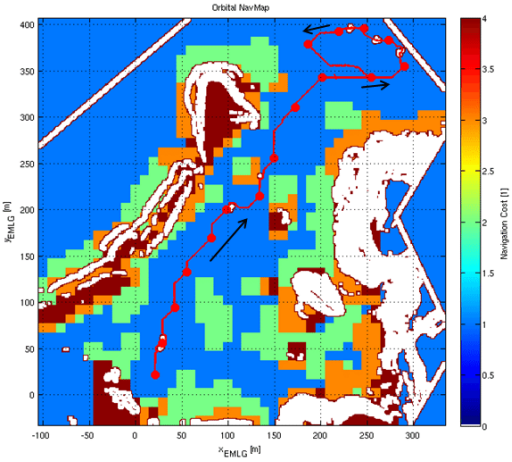
(b) mapa de landmarks representado sobre una imagen tomada por satélite



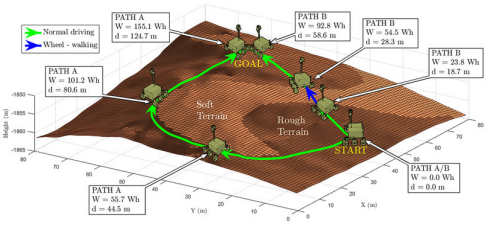
(c) Mapa de ocupación ideal



(d) mapa de elevación del terreno obtenido por satélite



(e) mapa centrado en la peligrosidad del terreno



(f) mapa centrado en la traversabilidad del terreno

Figura 2.2: diferentes tipos de mapas

la probabilidad de ocupación de cada celda, un análisis comparativo de diferentes métodos se muestra en [7]. En la figura 2.2c puede verse un ejemplo de un mapa de ocupación ideal donde, representado en blanco queda el espacio transitable, en negro los obstáculos y en gris oscuro el espacio sin explorar. Se trata de un mapa métrico basado en una representación *Free-space*.

- Elevación del entorno. Suelen representarse como rejillas codificando en cada celda la altitud de la misma, ya sea absoluta o relativa a un punto de referencia. Puede verse un ejemplo de un DEM generado por satélite en la figura 2.2d donde los diferentes colores representan la altitud. Nuevamente se trata de un mapa métrico que representa el entorno del robot, o una representación *free-space*.

- Otros como información sobre la peligrosidad del entorno añaden (o sustituyen) la información de cada una de las celdas por información de la peligrosidad del terreno, tal y como trabajan y muestran en [20]. En la figura 2.2e puede verse un mapa de este tipo, en el que cada color representa una dificultad del terreno en base a la variación de altitud como a la presencia de obstáculos, desde el blanco (terreno no transitable) bajando de dificultad desde el rojo pasando por el naranja y el verde hasta el azul, de menor dificultad. Otros mapas pueden almacenar en las celdas la facilidad o dificultad de atravesar ciertos terrenos, muy práctico en robots con diferentes modos de locomoción, tal y como plantean en [26]. Estos autores plantean representar en un mapa en el que se representen diferentes tipos de terreno, desde terrenos más duros que permitirán una locomoción normal (con menor coste asociado) hasta terrenos más blandos o arenosos que forzarán a un avance más lento (con mayor coste asociado). UN ejemplo de este enfoque puede verse en la figura 2.2f. También se tratan de mapas métricos, representando el entorno del robot, una representación *free-space*.

No necesariamente deben utilizarse estas representaciones por separado, por ejemplo en [28] se plantean mapas donde para cada celda se asocian valores de ocupación, elevación y variación de dicha elevación.

Finalmente se pueden clasificar a su vez los algoritmos de mapeado según la información a la que recurren para la formación del mapa. Podrán ser:

- Basados en información visual: utilizan cámaras para mediante técnicas de procesamiento de imagen extraer las diferentes características y generar el mapa. Ejemplos de SLAM basados en *landmarks* visuales pueden verse en [10] o en [8]. Se emplean diferentes tipos de descriptores visuales para almacenar y reconocer los diferentes *landmarks* detectados, almacenando dichos descriptores con la posición asociada a los mismos.

- Basado en información de distancia: generalmente utilizando sensores de tipo LIDAR, aunque en algunos casos se emplean medidas de distancia extraídas de cámaras.

Recapitulando sobre la información vista podemos enfocar hacia soluciones que se adapten al entorno y caso de uso en el que se centra este proyecto. Como se ha mencionado en el primer capítulo, resaltado por parte de [30], una de las características que influyen en el proceso de navegación pasa por asegurar la integridad tanto de la plataforma como del entorno. Se optará por representaciones del espacio que permitan detectar obstáculos y así esquivarlos. Una buena forma de hacer esto es empleando mapas métricos que, como se ha visto, representan de manera fiel y precisa la geometría del entorno. Una de las restricciones en las que se ha puesto énfasis al inicio del capítulo es el coste computacional que puede conllevar representaciones del entorno más complejas. Tratándose de un entorno semi-estructurado, tanto en exteriores como interiores se optará por soluciones basadas en mapas de ocupación pues se considera la representación más sencilla dentro de las disponibles en el grupo de mapas métricos. Finalmente se dará preferencia a soluciones ya integradas en el ecosistema ROS, que será el utilizado como referencia a lo largo del proyecto.

Dentro del ecosistema ROS uno de los algoritmos más empleados es el presentado en [12] y [11], Gmapping. Emplea filtros de partículas para generar los mapas de ocupación. Cada partícula almacena la pose y un mapa del entorno. Aunque un filtro de partículas de este tipo puede resultar computacionalmente costoso, Gmapping aprovecha las ventajas del SLAM para reducir la incertidumbre en cuanto a la pose del robot, reduciendo así el número de partículas necesario.

Otra de las opciones integradas en ROS es KartoSLAM. Desarrollado originalmente por SRI International's Karto Robotics, e integrado posteriormente en ROS es un algoritmo de SLAM basado en grafos. Cada nodo del grafo representa una pose del robot a lo largo de la trayectoria y un conjunto de medidas de los sensores disponibles. Cada nodo se une con otros nodos mediante arcos que representan el movimiento entre sucesivas poses. Para cada nuevo nodo se calcula el mapa buscando la configuración espacial de nodos consistente con las restricciones incluidas por cada uno de los arcos. Incluye tanto *scan-matching* como *loop-closing* para asegurar la consistencia del mapa generado.

HectorSLAM es otra de las opciones típicas integradas dentro de ROS combina SLAM 2D con un proceso de *scan-matching* y navegación 3D. Basa sus estimaciones de movimiento haciendo uso de datos en alta frecuencia de LIDAR sin depender de datos de odometría. Al no depender de la odometría puede ser utilizado en robots aéreos como drones, aunque será especialmente sensible a la disponibilidad de información de PointClouds a alta frecuencia. Una explicación de este algoritmo y una comparación detallada con otros puede verse en [21].

Por último, otra de las opciones, desarrollada por Google es el algoritmo Cartographer, presentado en [14]. Es una solución que provee de SLAM 2D y 3D en tiempo real, pensado para adaptarse a diferentes plataformas o sensores utilizados. Se trata de un algoritmo que basa su funcionamiento en la construcción de un grafo con las restricciones correspondientes entre sus nodos para, finalmente, reconstruir un mapa de ocupación con la información disponible. Se trata de la solución escogida por

estar diseñada para funcionar en tiempo real manteniendo un coste computacional realmente bajo, se explica en detalle en el siguiente capítulo.

Nota: puede ampliarse información sobre diferentes algoritmos de SLAM en la página web de OpenSLAM: `https://openslam-org.github.io/`.

Capítulo 3

Navegación, conceptos teóricos

Como se ha venido adelantando desde la introducción, la navegación consta de tres partes principales: localización, mapeado y planificación y seguimiento de trayectorias. Tal y como se desarrolla en [10], cada uno de ellos implica:

- Mapeado: Se refiere al proceso de generación de un mapa interpretando la información proveniente de distintos sensores para representar el entorno. La salida de este proceso será alguno de los tipos de mapas mencionados anteriormente, aunque de ahora en adelante nos centraremos en los mapas de ocupación. Por lo general se asume que el proceso de mapeado parte de una localización y movimiento de la plataforma conocidas, de manera que se integrará la información proveniente de los diferentes sensores para ir *descubriendo* el mapa según se avance.

- Localización: Se refiere al proceso de estimación de la posición del robot. En robots móviles típicamente la pose del robot está definida por una traslación y una rotación con respeto al eje de giro del robot: (x,y,θ). El proceso de localización típicamente incluye un mapa o modelo del entorno y en base a los sensores que equipa podrá descubrir en que punto del mapa se encuentra.

- Planificación de trayectorias: La planificación de trayectorias busca encontrar el camino óptimo para desplazarse de un punto A a un punto B del entorno. Nótese que se pueden optimizar diferentes parámetros como tiempo empleado, distancia recorrida, energía consumida o seguridad del robot entre otros.

En la figura 3.1 pueden verse como los conceptos individuales de la navegación pueden combinarse de distinta forma dando lugar a operaciones de:

- Exploración: Se trata de un proceso de mapeado de un área desconocida. Normalmente se utilizarán algoritmos de planificación que vayan decidiendo a qué punto desplazarse para poder mapear el entorno al completo. Un ejemplo, integrado y probado en el entorno de ROS, es el mostrado en [31]. Haciendo uso de los mapas generados por el paquete Gmapping (también integrado en ROS) y de un algoritmo RRT (exploración rápida de árbol aleatorio) es capaz de decidir a qué puntos deberá desplazarse para explorar el entorno. Aunque las pruebas están realizadas sobre un mapa 2D, presentan la opción

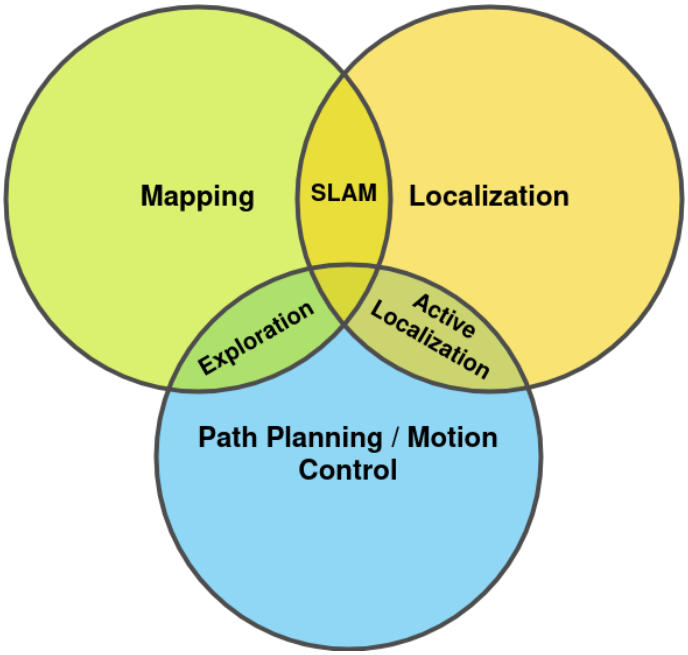


Figura 3.1: navegación

Fuente Imagen:

de expandir a más dimensiones gracias a la versatilidad del algoritmo RRT. La exploración supone un ejemplo claro donde la planificación de trayectorias no pretende optimizar la distancia de una trayectoria en si misma, sino optimizar el proceso de exploración.

- Localización Activa: En contraposición con la localización pasiva (basada solo en la información proveniente de sensores), descrita anteriormente, la localización activa asume que en paralelo al proceso de localización se tiene el control del movimiento del robot, lo que permite aumentar la eficiencia y robustez de la localización estimada. En el trabajo presentado en [3] sobre localización activa, se parte de un mapa del entorno en el que el robot deberá localizarse haciendo uso de la información de los sensores planificando moverse hacia puntos que permitan disminuir la incertidumbre en la posición estimada. La localización activa presenta otro ejemplo en el que la planificación no busca optimizar la distancia recorrida per se, sino localizarse de la manera más fiable posible.

- SLAM: Este proceso combina las operaciones de mapeado y localización. Busca construir un mapa y localizarse en el mismo a la vez. Cualquier error inducido en la localización afectará al proceso de mapeado distorsionando el resultado, de igual manera cualquier error en el mapeado (interpretación o fusión de los datos de los sensores) afectará a la localización. Este trabajo se centrará en este proceso concretamente, profundizando en el ya mencionado algoritmo Cartographer, que se describe en la sección siguiente.

Nótese que trata de procesos nuevos combinando las funciones previas para desarrollar una nueva funcionalidad específica, tener mapeado y planificación de trayectorias

no implica la capacidad de exploración por si mismo, aunque la capacidad de exploración si requiere de las dos previas. Es muy común combinarlas entre distinto grado, por ejemplo teniendo SLAM y planificación de trayectorias proporciona un grado de autonomía muy elevado para un robot móvil.

3.1. Cartographer

En esta sección se resume el funcionamiento del algoritmo propuesto en [14], así como algunos aspectos más relevantes configuración y parametrización del mismo. Además del citado artículo existe documentación online donde se profundiza en los diferentes parámetros así como ejemplos de los mismos.

Cartographer es un algoritmo de SLAM pensado para operar en tiempo real tanto en 2D como en 3D permitiendo una parametrización a diferentes entornos, plataformas y sensores utilizados. Cómo otros algoritmos de SLAM, produce un mapa de ocupación y la localización de la plataforma en el mismo, con la particularidad de que el algoritmo está diseñado para reducir el coste computacional de métodos existentes permitiendo hacer SLAM en tiempo real. Aunque el algoritmo está contenido en una librería C++, se ha publicado todo un módulo que permite la integración del algoritmo en el entorno de ROS.

Desde un punto de vista de la arquitectura global, el Cartographer está compuesto principalmente por dos subsistemas: un SLAM global y un SLAM local. El subsistema de SLAM local está encargado de generar submapas a partir de la información suministrada mientras que el módulo de SLAM global irá uniendo dichos submapas para formar el mapa general. Las posibles entradas aceptadas por el algoritmo incluyen:

- Información de distancia a objetos: podrá tener diferentes formatos desde un escaneado en 2D hasta un PointCloud completo.
- Movimiento de la plataforma a través de información de odometría.
- Alineamiento de los datos de distancia con respecto al plano del suelo a través de un sensor IMU (aceleración lineal y velocidad angular).
- Posición global a través de una antena GNSS o similar.

Las tres primeras entradas intervienen en el proceso de SLAM local, mientras que las tres últimas lo hacen a la hora de unir los submapas en el SLAM global.

La figura 3.2 representa la arquitectura de alto nivel del algoritmo, donde pueden verse las entradas descritas previamente así como los dos subsistemas mencionados al inicio. Además de los algoritmos propiamente dichos, la información de entrada es tratada para su posterior uso de la siguiente manera:

- Los datos de distancia son discretizados a una cuadrícula del tamaño configurado. El proceso de medida mediante cualquier sensor de distancia en varios

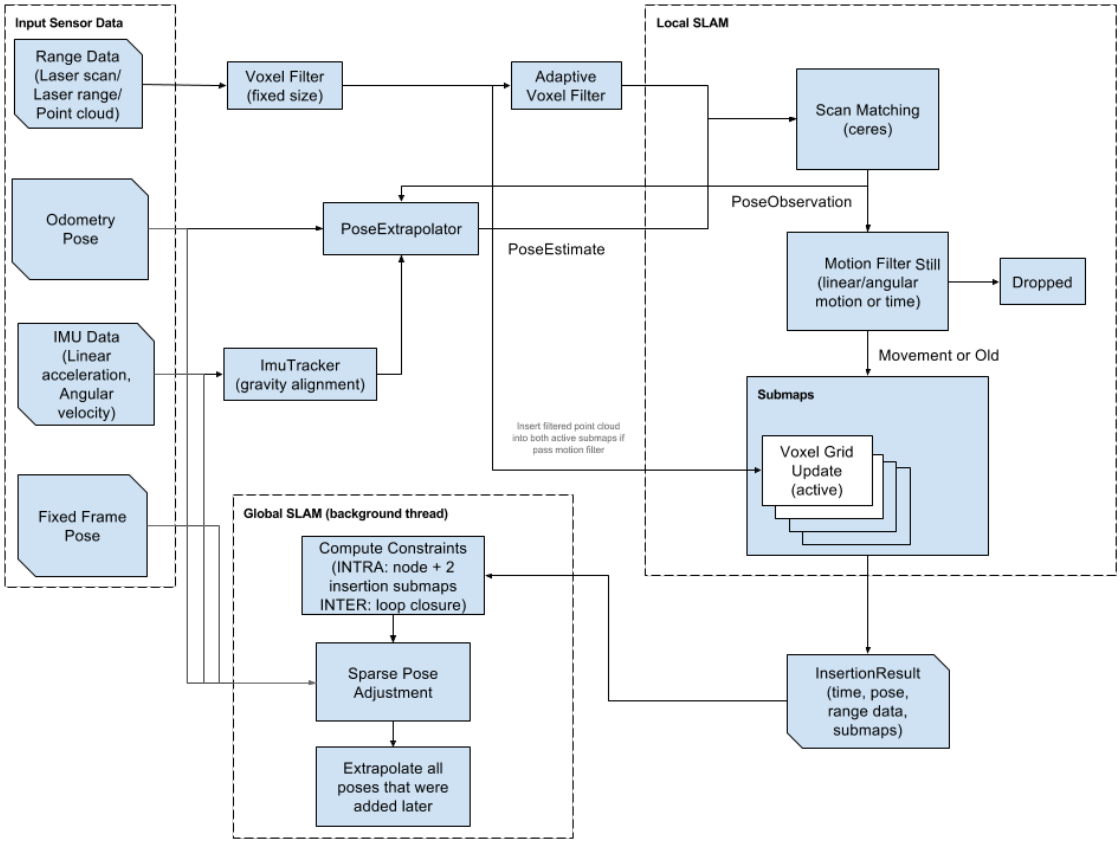


Figura 3.2: arquitectura del algoritmo Cartographer

Fuente Imagen: Extraído de la documentación oficial del Cartographer

ángulos implica que objetos cercanos reciban potencialmente muchos más impactos y se almacenen más puntos que objetos lejanos. Dependiendo del sensor esto puede resultar en una elevada densidad de información siendo muy dispar entre diferentes zonas. Para solucionar este problema la información es preprocesada para submuestrear la información discretizándola. Para ello se divide el espacio en cubos manteniéndose el centroide de aquellos cubos en los que hubiera puntos originalmente. Este procesado se lleva a cabo en el módulo llamado *Voxel Filter (fixed size)* visto en el diagrama.

- Se vuelve a procesar la información previa en este caso mediante un algoritmo adaptativo que intentará determinar el tamaño del cubo para aunar un número configurado de puntos sin pasarse del tamaño establecido.

- El módulo *PoseExtrapolator* proporciona una estimación inicial de donde deberá insertarse el nuevo *scan* procesado integrando información de los sensores que proporcionan datos sobre el movimiento o posición de la plataforma. Puede verse como el resultado del *scan-matching* es parte de las entradas de este módulo.

- El módulo nombrado como *ImuTracker (gravity alignment)* se encarga de alinear la información de distancia (*scan*, PointCloud) con el plano del suelo, haciendo uso de la información de la gravedad proveniente de la IMU. Es especialmente relevante en entornos donde haya cambios de inclinación así como con plataformas que no mantengan constante su orientación con respecto al suelo. Ajustar bien este filtro permitirá evitar el ruido de las lecturas de la IMU.

Centrado en reducir el consumo de recursos computacionales Cartographer lleva a cabo un proceso de *scan-to-map matching* considerado más eficiente y robusto. Los nuevos *scan* son insertados en submapas locales, en los que se presupone un error mínimo. Para evitar que el error pueda divergir se ejecuta una optimización de la *pose* (posición X, Y del robot y orientación Yaw). Cuando un submapa se da por finalizado entra a formar parte del *scan-matchin* del SLAM global, donde todos los submapas son empleados en un algoritmo de *loop-closing* de manera iterativa para añadir restricciones al problema de optimización global. Este proceso permite ser ejecutado en tiempo real permitiendo que el proceso de *close-loop* ocurra en el momento en el que una posición es revisitada por la plataforma. Este proceso se detalla a continuación, tanto para el SLAM local como para el SLAM global. Finalmente se hace una revisión de los parámetros configurables más relevantes que permite ajustar la integración del algoritmo en ROS.

SLAM Local

El algoritmo de SLAM local busca optimizar la pose (compuesta de información de traslación (x,y) y de rotación θ) de la información proveniente del LIDAR, preprocesada y simplificada, que llamaremos *scan*. Cada uno de los *scan* va uniéndose

en pequeños submapas utilizando una optimización no lineal que alineará cada nuevo *scan* al submapa al que pertenezca.

Cada submapa tiene la forma de un mapa de probabilidad, que almacena, discretizado en una rejilla, la probabilidad de que un punto esté obstruido. Cada vez que un nuevo *scan* se inserta se actualiza la probabilidad de cada celda de estar obstaculizada. Se crean dos conjuntos disjuntos de celdas, uno con aquellas en las que hay un obstáculo detectado, y otro con el espacio entre el origen de la medida (ubicación del sensor) y el punto detectado. Puede verse este proceso y ambos conjuntos en la figura 3.3.

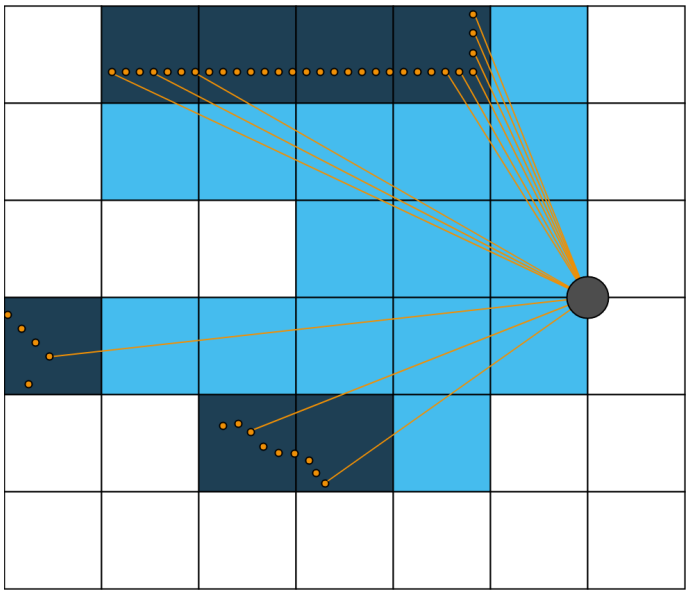


Figura 3.3: procesado de la información del *scan*

Fuente Imagen: Autor, basado en [14]

Usar un mapa probabilístico permite asociar diferente confianza a las medidas tanto de espacio ocupado como espacio libre a la hora de parametrizar el algoritmo. Este modo de funcionamiento permitirá además ir filtrando información de obstáculos móviles según la probabilidad de estar obstaculizado disminuya al incluir nuevos *scans* de esa misma zona. También permite amortiguar posibles errores al posicionar los nuevos *scans* incluidos.

Antes de incluir un *scan* en el submapa se lleva a cabo un proceso de optimización de la pose en la que se va a insertar dicho *scan*. Para ello se utiliza un optimizador no lineal desarrollado por Google, el Ceres Solver (puede verse más información en [1]). El optimizador está a cargo del proceso de *scan-matching* para buscar una pose que maximice la correspondencia entre los puntos del *scan* y el submapa considerado. Se trata de un proceso de optimización local, por lo que requiere tener buenas estimaciones iniciales para asegurar su convergencia al mínimo esperado. Es muy importante contar con información del desplazamiento y rotación de la plataforma para mejorar tanto la convergencia como el tiempo empleado para ello.

SLAM Global

Como se ha visto previamente la creación de los submapas implica acumular error en la posición estimada. Este error, cuando el número de *scans* es reducido se mantiene muy pequeño, pero irá creciendo conforme se acumulen más número de *scans* o en su defecto submapas. Para compensar dicho error el algoritmo implementa un problema de optimización mediante mínimos cuadrados no lineales (haciendo uso nuevamente del Ceres Solver mencionado previamente). Este algoritmo tiene como entrada una serie de restricciones para buscar la solución que mejor las satisfaga. Estas restricciones incluyen:

- Información de la posición de la plataforma que ahora incluye datos de odometría, IMU y GNSS.
- Las posiciones en las que se han ido uniendo los diferentes *scans* para formar los submapas. Todas las posiciones previamente optimizadas en su entorno local se almacenan para incluirse en el problema de optimización global.
- Al igual que al incluir cada nuevo scan se optimiza y busca su posición en el submapa, para cada nuevo submapa se buscarán correspondencias en los submapas previamente almacenados. Este proceso, de *loop-closing*. Resultados del algoritmo de *loop-closing*. Para mantener la restricción de tiempo real el algoritmo utiliza un mecanismo de Ramificación y Poda (*Branch and Bound)* para ir eliminando las opciones subóptimas. Puede ampliarse la información sobre este enfoque en [4]. Cuando se encuentra una correspondencia se recurre nuevamente al Ceres Solver para optimizar la transformada entre *scan* y submapa.

Una vez se han obtenido todas las restricciones se pasa al problema de optimización para buscar la solución que une los submapas satisfaciendo las restricciones calculadas en base a la confianza que se tenga en cada una de las restricciones (llevan un peso configurable asociado).

El resultado es un mapa completo, que irá mejorándose conforme se añada más información.

Parametrización

La complejidad del algoritmo de SLAM descrito implica una gran cantidad de parámetros que podrán configurarse para adaptarse a las condiciones específicas en que se utilice. Esta sección pretende cubrir los más básicos y relevantes que se han considerado, pudiendo encontrarse una descripción completa en la documentación presentada en el apartado siguiente. Nótese que lo descrito en este apartado hace uso de la integración desarrollada para el Cartographer en ROS.

La configuración del Cartographer se encuentra separada en diferentes ficheros de configuración para cada uno de los submódulos, todos ellos ficheros con extensión y sintaxis `.lua`. Generalmente lo recomendable es hacer una copia del fichero

principal desde el que se podrán sobrescribir los parámetros del resto de submódulos que se desee. De esta manera los parámetros que no se sobrescriban quedarán con un valor por defecto. De la configuración principal del Cartographer cabe destacar los siguientes parámetros:

- **tracking_frame**: Será el nombre del frame a monitorizar. Es obligatorio que si se incluyen datos de IMU este frame sea el propio de la IMU.
- **published_frame**: Es el nombre del frame utilizado para publicar la posición.
- **use_odometry**: cuando se activa el algoritmo utilizará datos en formato ROS de tipo *nav_msgs/Odometry* publicados sobre un topic llamado *"odom"*.
- **use_nav_sat**: cuando se activa el algoritmo utilizará datos en formato ROS de tipo *sensor_msgs/NavSatFix* publicados sobre un topic llamado *"fix"*.
- **num_point_clouds**: número de PointClouds al que suscribirse. Vendrán en formato ROS de tipo *sensor_msgs/PointCloud2* en el topic *points2*.

Referido a los otros submódulos del Cartographer algunos parámetros que se han considerado relevantes para una primera aproximación son:

- **TRAJECTORY_BUILDER_nD.use_imu_data**: Si se activa se tendrá en cuenta la información de la IMU.
- **TRAJECTORY_BUILDER_nD.imu_gravity_time_constant**: Los datos de la IMU pueden fluctuar bastante, es por ello que se pueden filtrar integrándolos durante una cantidad de tiempo. Se configura en segundos.
- **TRAJECTORY_BUILDER_nD.min_z**: altura en Z mínima a considerar. Es importante filtrar las alturas que no supongan un obstáculo para la plataforma para evitar que puntos del suelo o que no supongan obstáculo, captados en el PointCloud, puedan acabar en el mapa incluidos.
- **TRAJECTORY_BUILDER_nD.max_z**: altura Z máxima a considerar.
- **TRAJECTORY_BUILDER_nD.min_range**: Establece una distancia mínima para considerar información del PointCloud al rededor del origen.
- **TRAJECTORY_BUILDER_nD.max_range**: Permite restringir la cantidad de información empleada. Demasiada información puede ralentizar el algoritmo. Puede ser interesante restringir el alcance máximo que se tendrá en cuenta.
- **TRAJECTORY_BUILDER_nD.ceres_scan_matcher.translation_weight**: Como se ha visto el problema de optimización admite asignar diferente peso en base a la confianza que se tenga a cada tipo de información. Cuanto mayor sea mayor importancia dará el algoritmo a la información de traslación.
- **TRAJECTORY_BUILDER_nD.ceres_scan_matcher.rotation_weight**: Equivalente al parámetro anterior, en este caso modulando la importancia de las rotaciones.

- **TRAJECTORY_BUILDER_nD.submaps.num_range_data**: Configura el número de *scans* que se incluirán en cada submapa. Cuanto mayor sea, mayor será el error acumulado. Nótese que deberá ser lo suficientemente grande como para que el *loop-closing* pueda funcionar correctamente.

Información Extra

Además de las referencias citadas a lo largo del texto, algunas páginas web y blogs pueden resultar de gran ayuda además de ampliar la información aquí descrita. Se consideran interesantes las siguientes:

- Documentación online del Cartographer:

  `https://google-cartographer.readthedocs.io/en/latest/index.html`

- Repositorio GitHub con el código del Cartographer:

  `https://github.com/cartographer-project/cartographer`

- Documentación online de la adaptación del Cartographer a ROS:

  `https://google-cartographer-ros.readthedocs.io/en/latest/index.html`

- Repositorio GitHub con el código de la adaptación del Cartographer a ROS:

  `https://github.com/cartographer-project/cartographer_ros`

Capítulo 4

Plataforma Experimental

En este capítulo se describen tanto la plataforma robótica utilizada como los sensores y diferentes aspectos de la configuración empleada para las pruebas posteriores.

A lo largo de este capítulo se presentan diferentes diagramas con la arquitectura de cada uno de los paquetes, el *hardware* implicado y los diferentes nodos ROS que ejecutan cada uno. Todos estos diagramas utilizan la misma notación, para mayor claridad se ha extraído una leyenda aparte que se presenta en la figura 4.1.

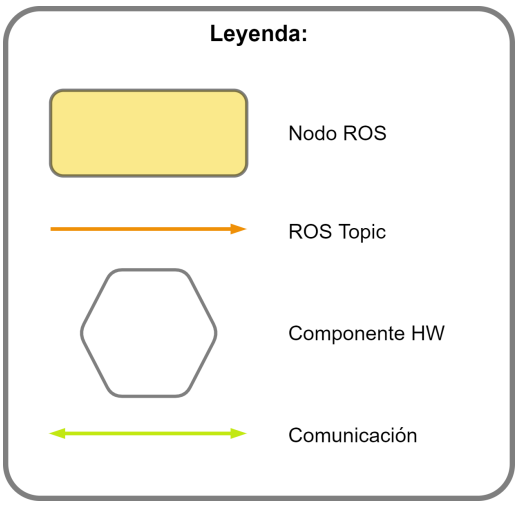


Figura 4.1: simbología utilizada en los diagramas

Fuente Imagen: Autor

Cabe remarcar en el caso de los topics que están representados con una flecha con un sentido específico. El patrón proveedor-suscriptor utilizado por ROS para los topics implica una dirección en la comunicación. El mensaje proviene de un único nodo ROS y puede llegar a uno o más nodos ROS, siendo la flecha representada consistente con este flujo de información. En el caso de las comunicaciones más genéricas, en verde, se especificará en cada caso, tanto diagrama como explicación, si la comunicación es o no bidireccional.

4.1. Plataforma robótica Husky

La plataforma empleada como base es la plataforma Husky de la empresa Clearpaht Robotics. Puede verse la versión original en la figura 4.2. Sobre la base superior se montarán posteriormente los diferentes sensores.

Figura 4.2: plataforma robótica Husky

Fuente Imagen: Fabricante

Se trata de una plataforma de pequeño tamaño con una configuración cinemática diferencial clásica. El robot cuenta con cuatro ruedas fijas que, variando su velocidad de giro permitirá controlar diferentes modos de locomoción. La cinemática queda definida por la ecuación general para robots diferenciales, que podrá concretarse haciendo uso de las medidas presentadas en la figura 4.3.

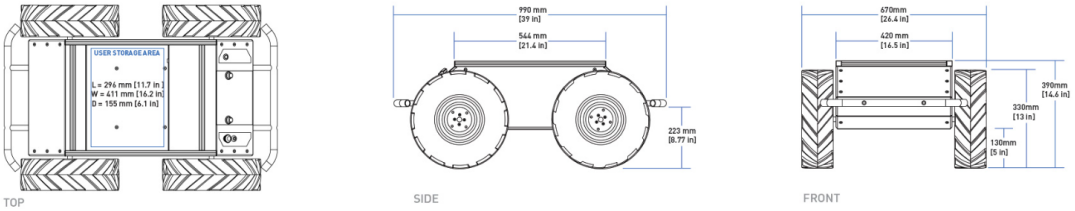


Figura 4.3: plataforma robótica Husky, dimensiones principales

Fuente Imagen: Fabricante

Las ecuaciones 4.1 y 4.2 permiten obtener la velocidad angular de ambas ruedas, ω_L para la rueda izquierda y ω_R para la derecha, en base a los parámetros del movimiento:

- V: Velocidad lineal.
- ω: velocidad angular de rotación del robot.
- b: ancho del robot.
- r: radio de las ruedas.

$$\omega_L = \frac{V - \omega * b/2}{r} \tag{4.1}$$

$$\omega_R = \frac{V + \omega * b/2}{r} \tag{4.2}$$

El modelo cinemático ha sido obtenido directamente del controlador integrado en ROS, que se presentará más adelante. Con esta disposición cinemática el robot puede conseguir desplazarse mediante dos modos de locomoción: *Point Turn* o *Ackerman*, representados en la figura 4.4. Aunque la plataforma consigue realizar dichos modos de desplazamiento, la configuración diferencial es especialmente sensible al deslizamiento de las ruedas cuando giran a distintas velocidades o velocidades con signo contrario, siendo más marcado en los giros sobre su propio eje o *point turn*. Esto tendrá un impacto relevante en la estimación de odometría y la posterior reconstrucción del mapa, tal y como podrá verse en el capítulo con el desarrollo experimental.

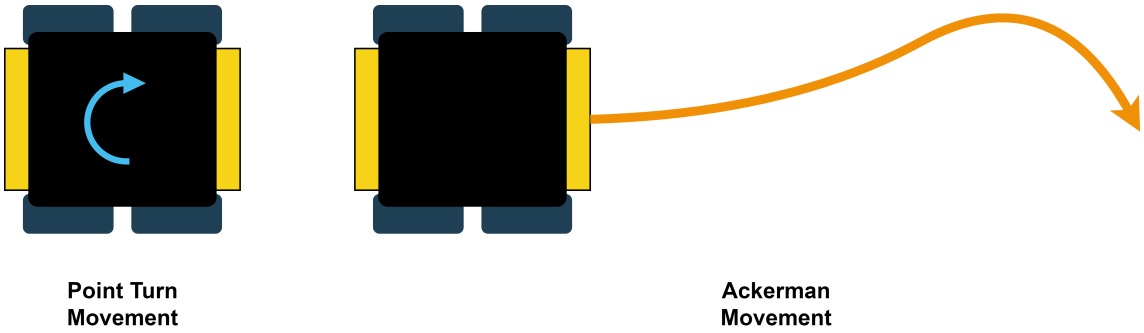


Figura 4.4: modos de locomoción disponibles para la plataforma Husky

Fuente Imagen: Autor

Algunas de las especificaciones más relevantes de la plataforma robótica Husky se han extraído de [6], y pueden verse en la tabla 4.1. Cabe remarcar la altura inferior de la plataforma, que delimitará los obstáculos que podrán obviarse en el proceso de Navegación así como tener en cuenta la temperatura de operación. Nótese que en caso de superarse pueden provocarse fallos en el OBC que interfieran con el control del robot.

Cuenta con un ordenador a bordo, de ahora en adelante referido como OBC. Se trata de un ordenador Mini-ITX preparado con el software básico del robot ya preconfigurado. Será utilizado para conectar los sensores y configurar el funcionamiento base de los mismos así como de red.

Tabla 4.1: especificaciones técnicas de la plataforma Husky

Diámetro ruedas	330 mm
Altura inferior	130 mm
Máxima capacidad de carga	75 kg
Velocidad máxima	1 m/s
Máximo ángulo de marcha	30°
Duración batería - Uso nominal	3 horas
Tiempo de carga	4 horas
Temperatura ambiente para operación	-10° a 40° (sin sol directo)

4.2. Sensores

Sobre la plataforma Husky se han montado una serie de sensores que se describen en esta sección.

4.2.1. Antenas GNSS-RTK

Para la localización GNSS se ha elegido las antenas Harxon Smart Antenna TS100 Series, tanto Base como Rover. En funcionamiento la antena base se ubica y configura en una posición conocida, esta posición conocida será utilizada para corregir el error y aumentar la precisión de la posición de la antena rover, que será la antena móvil. Puede verse el modelo elegido en la figura 4.5 así como sus principales características en la tabla 4.2.

Figura 4.5: antena GNSS-RTK TS100 de Harxon

Fuente Imagen: Fabricante

Tabla 4.2: especificaciones técnicas de las antenas *GNSS* TS100

Señal recibida	GPS; GLONASS; BDS; GALILEO
Frecuencia máxima de posición	10 Hz
Horizontal Position Accuracy (RMS)	Single point 1.5 m; DGPS 0.4 m; RTK 1 cm
Communication Ports	Serial; Port Bluetooth
Communication Ports (correction)	3G/4G; built in radio

Para comunicar ambas antenas y poder llevar a cabo las correcciones en la posición se ofrecen diferentes métodos. Se ha optado por la radio integrada en ambas antenas. A lo largo del documento se hará referencia a este tipo de comunicación llamándola UHF. Según datos del fabricante el rango de distancia, en línea recta vista, en que las antenas podrían comunicarse será de aproximadamente 1.5km.

La configuración así como lectura de los datos provistos por la antena se realizará en todo momento a través del puerto serie. Se verá más en detalle el proceso de configuración y uso de las antenas en el capítulo 5.1.1.

Las antenas ofrecen tres tipos de datos en función de la comunicación entre ambas. En la tabla 4.2 pueden verse las precisiones estimadas para cada uno de los modos de funcionamiento explicados a continuación:

- SPS: Es el modo de funcionamiento básico para cualquier antena GNSS que trilatera la posición en base a los satélites con los que se comunique.
- DGPS: Utilizando una base en una posición conocida permite llevar a cabo correcciones sobre la posición GNSS para eliminar error. En base a la posición conocida y las lecturas realizadas por la base puede estimarse el error para cada satélite empleado, esa estimación de error se utilizará en la antena móvil para corregir la posición estimada.
- RTK: Es un tipo de DGPS más moderno, actualiza tanto la tecnología como el protocolo y algoritmo utilizado permitiendo mayor precisión en la corrección efectuada.

A lo largo de la sección 5.1.1 se comentarán y pondrán a prueba diferentes aspectos del posicionamiento GNSS mediante estas antenas en el entorno real.

4.2.2. Sensor IMU

El sensor inercial elegido es el modelo UM7 de Redshiftlabs. Este sensor combina acelerómetros de tres ejes, giróscopo y magnetómetro que combinados mediante un Filtro de Kalman extendido, proporcionan información de la orientación así como la velocidad lineal y aceleración angular.

Las características principales pueden verse en la tabla 4.3, han sido extraídas de [18], donde puede ampliarse la información.

Figura 4.6: IMU UM7 de Redshift Labs

Fuente Imagen: Fabricante

Tabla 4.3: especificaciones técnicas del sensor UM7

Frecuencia de la estimación del EKF	500 Hz
Precisión (pitch and roll) estática	±1°
Precisión (pitch and roll) en movimiento	±3°
Precisión (yaw) estática	±3°
Precisión (yaw) en movimiento	±5°

Para el correcto funcionamiento del sensor es necesario calibrarlo. En este caso el sensor se ha adquirido junto con un adaptador a USB. Se conectará el sensor, mediante el cable USB a un ordenador con sistema operativo Windows y se seguirá el proceso de calibración descrito en [19] haciendo uso de la interfaz proporcionada por el fabricante.

El sensor IMU codifica los mensajes en el formato NMEA, los mensajes para configuración, específicos para el sensor pueden verse más detalladamente en la sección 'SPI Communication' de [18].

4.2.3. Sensor LIDAR

Los sensores LIDAR se emplean para medir distancias entre el sensor y distintos obstáculos. Se trata de un sensor activo que emite una luz y mide el tiempo que tarda dicha fuente de luz en llegar al obstáculo y volver al receptor del propio sensor (tiempo de vuelo o TOF). Típicamente utilizan luz focalizada LÁSER y en su versión más sencilla permiten medir la distancia a un punto ya sea un objeto o un

reflector específico. El fabricante SICK ofrece una descripción muy detallada de la tecnología así como de las diferentes versiones que comercializan en [27], se resume a continuación algunos aspectos relevantes.

El uso de LÁSER permite su uso para medir distancias independientemente de las fuentes de luz externas y emplearse tanto de noche, bajo tierra, etc. manteniendo la misma eficacia que con luz artificial o exteriores. Nótese que al tratarse de un sensor basado en LÁSER muchos cristales y materiales transparentes dejarán pasar la luz impidiendo ser detectados como obstáculos, situación que podría darse comúnmente en interiores.

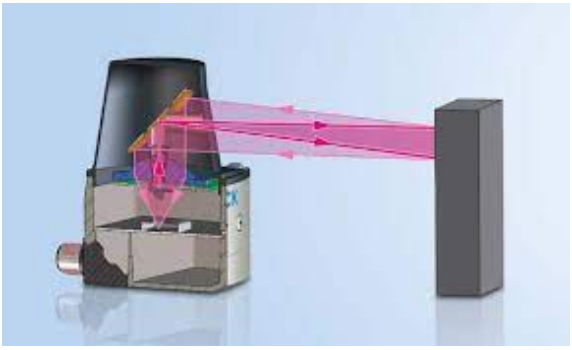


Figura 4.7: funcionamiento de un LIDAR 2D
Fuente Imagen: Imagen del fabricante SICK, extraída de [27]

El sensor básico en una dimensión puede ampliarse mediante el uso de un espejo giratorio que permitirá hacer medidas a lo largo del ángulo que pueda girar. Las características como el largo alcance o la capacidad de medida en diferentes condiciones de iluminación se mantendrán, ahora midiendo a lo largo de una línea de puntos. Puede verse en la figura 4.7 una representación de como un espejo redirige el foco de luz y como esta luz choca contra el objeto. El ángulo de medida, la velocidad que es capaz de alcanzar o la resolución a la que es capaz de captar los puntos dependerá del modelo elegido.

Extrapolando este concepto a tres dimensiones, mediante varios emisores y receptores se pueden crear sensores que exploren simultáneamente líneas a varios ángulos. Los más avanzados permiten medir al rededor del sensor, 360° así como varias líneas en el plano horizontal. Cabe destacar que la resolución, tanto en el número de líneas (vertical) como en el número de puntos (horizontal) variará tanto en función del modelo como de la configuración con que se ejecute, y afectará de diferente manera según para que se quiera aplicar. En la figura 4.8 puede verse una visualización de la nube de puntos obtenida de un LIDAR 3D.

Para el desarrollo de este proyecto y proyectos futuros se ha adquirido e integrado el LIDAR de la marca Ouster, concretamente el modelo OS1-128. Se trata de un sensor láser 3D que es capaz de proporcionar medidas a lo largo de 128 líneas horizontales. El sensor integra a su vez un sensor IMU. Puede verse en la tabla 4.4 las

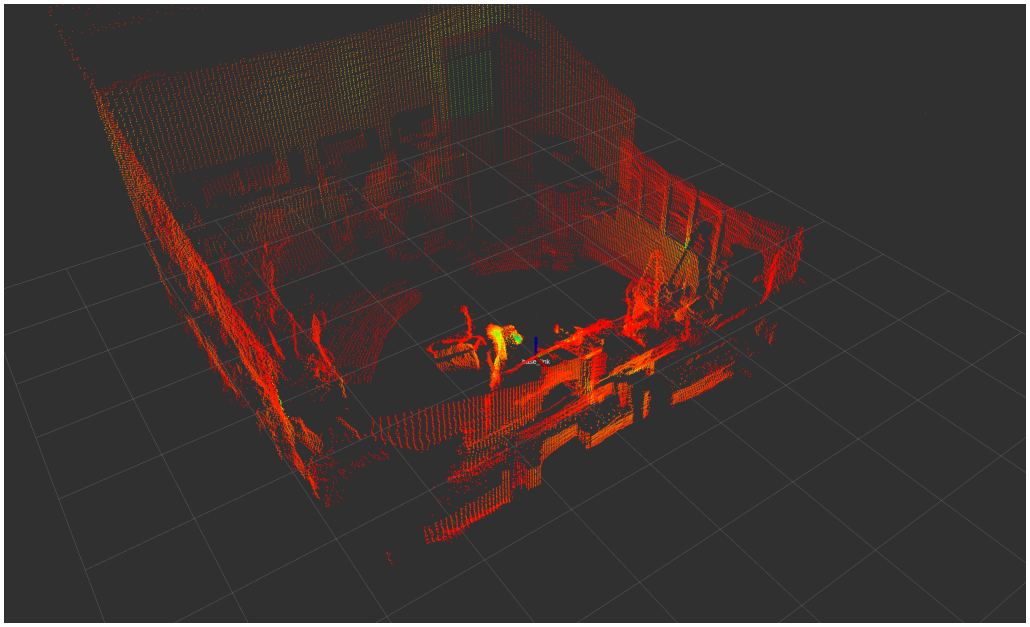


Figura 4.8: vista en rviz de un PointCloud obtenido con un LIDAR 3D

Fuente Imagen: Autor

Figura 4.9: LIDAR OS1 de Ouster

Fuente Imagen: Fabricante

características principales del *LIDAR* utilizado. Esta información puede ampliarse en [25].

Tabla 4.4: especificaciones técnicas del *LIDAR* OS1-128

Resolución vertical	128 canales
Resolución horizontal	512, 1024 o 2048
Rango	120 m
Campo de visión vertical	45°
Frecuencia de rotación	10 o 20 Hz

El sensor incluye un reloj interno mediante el cual se establece el *timetag* para cada uno de los mensajes enviados por UDP al puerto definido. Este reloj se inicia cuando se alimenta el sensor y será necesario sincronizarlo con el reloj del OBC para que los tiempos de todos los mensajes del sistema sean coherentes.

Para sincronizar ambos relojes se ha recurrido al servicio *PTP* de Linux, que configurará el reloj del ordenador como maestro del que se tomará el *timetag* para actualizar el reloj interno del sensor LIDAR. Al lanzar el driver del sensor se deberá especificar el modo `TIME_FROM_PTP_1588`. De esta manera tanto los datos del propio sensor LIDAR como de la IMU integrada llevarán un código de tiempo asociado coherente con el tiempo del resto del sistema.

4.3. Configuración de red

La plataforma robótica viene preparada para conectarse a una red Wifi a través de las antenas que tiene integradas. Por defecto viene con un sistema operativo sin escritorio, un Ubuntu Server. Durante las pruebas se ha instalado un escritorio para poder trabajar con pantalla y teclado directamente conectadas a la plataforma. Por defecto la plataforma utiliza *Netplan* para la configuración de red, aunque se ha cambiado a *Network Manager*, el gestor tradicional con Ubuntu Desktop.

Es importante contar con una red fiable para poder comunicarse entre ordenadores y robot, normalmente mediante *ssh*. En algunas pruebas se ha podido comprobar como la red puede caerse al intentar pasar datos de gran tamaño, como los utilizados por el algoritmo de SLAM. Además de para pasar datos de gran tamaño, es interesante contar con una red consolidada para poder ejecutar y monitorizar los diferentes módulos del robot desde cualquier ordenador o teléfono vía *SSH* sin depender de la pantalla y teclados, que en ejecución no estarían disponibles.

La versión actual hace uso de un router Cisco Linksys E4200 como el de la figura 4.10, que será el encargado de gestionar la red del robot. El router va ubicado

Figura 4.10: modelo de router utilizado

Fuente Imagen: Fotografía del fabricante

en la superficie de la plataforma robótica estando el OBC de la misma conectado mediante un cable Ethernet al mismo. Cualquier dispositivo que necesite conectarse a la red interna del robot podrá hacerlo ya sea a través de la red Wifi o a través de un cable Ethernet al mismo router. Algunas de las características más relevantes del router son:

- LAN y WAN Gigabit con doble banda de 2.4 GHz y 5GHz.
- Velocidades de hasta 300 Mbps en la banda de 2.4 GHz y 450 Mbps en la banda de 5 GHz.
- Disponibilidad de puerto USB para conectar disco duro al router.
- Alimentación a un voltaje de 12 V y corriente de 2 A, proporcionada por la plataforma robótica.

4.4. Estructura y configuración del robot

Tanto los sensores como el router descritos van montados sobre la plataforma robótica. Puede verse el aspecto actual de los mismos en la figura 4.11.

Todos los componentes descritos previamente van conectados al OBC de diferente forma. Puede verse en la figura 4.12 un diagrama esquemático con los diferentes componentes:

- IMU: se comunica con el OBC a través de un cable USB. Desde el OBC se abrirá el puerto serie para recibir la información del sensor, así para configurarla y calibrarla cuando sea necesario. El driver en el OBC recibirá estos mensajes traduciéndolos a mensajes de ROS.
- LIDAR: se comunica a través de un cable Ethernet. A través del mismo pasan los mensajes tanto del propio LIDAR como de la IMU integrada. Hace uso de un protocolo propio que mediante UDP provee de la información a un puerto e IP configurados. Desde el OBC se interpretarán estos mensajes para traducirlos a mensajes de ROS.

Figura 4.11: plataforma robótica Husky con los sensores instalados

Fuente Imagen: Fotografía del autor

- GPS: incluye una salida por puerto serie que será la empleada para comunicar con el OBC. Utiliza un protocolo propio para la configuración de la antena así como mensajes estándar NMEA para la información de posición. El driver abrirá el puerto serie y traducirá dichos mensajes a mensajes de ROS.
- Router: El OBC se comunicará con el router, proveedor de la red de la plataforma robótica, a través de un cable Ethernet. A través de cable Ethernet o Wifi también podrán conectarse otros dispositivos a la red Wifi así como al OBC.

Dentro del ecosistema de ROS los mensajes deben ir referenciados a un *frame* específico que actuará como referencia para relacionar los mensajes obtenidos en diferentes sensores. La estructura completa del robot se configura mediante un fichero de descripción mediante el formato URDF. En este fichero se van añadiendo las partes del robot así como las diferentes articulaciones y las relaciones de transformación entre las mismas. En ROS existe una herramienta que simplifica la generación de ficheros URDF, esta herramienta se llama Xacro. Los ficheros Xacro (y la herramienta en sí) suponen una extensión a la sintaxis propia de un URDF que permite añadir cierta lógica (bucles, condicionales, etc).

Para añadir los diferentes sensores se ha modificado el fichero base del robot para añadir los nuevos *frames*. Puede verse en la figura 4.13 las referencias añadidas correspondientes a la posición de la antena GNSS, la IMU y el LIDAR. Se han representado además los frames *top_plate_link* y *top_plate_rear_link* que representan el centro y el extremo delantero de la superficie del robot. Además aparece representado el frame *base_link*, que ubicado en el centro del robot es la base del árbol de transformadas y se utilizará como referencia principal.

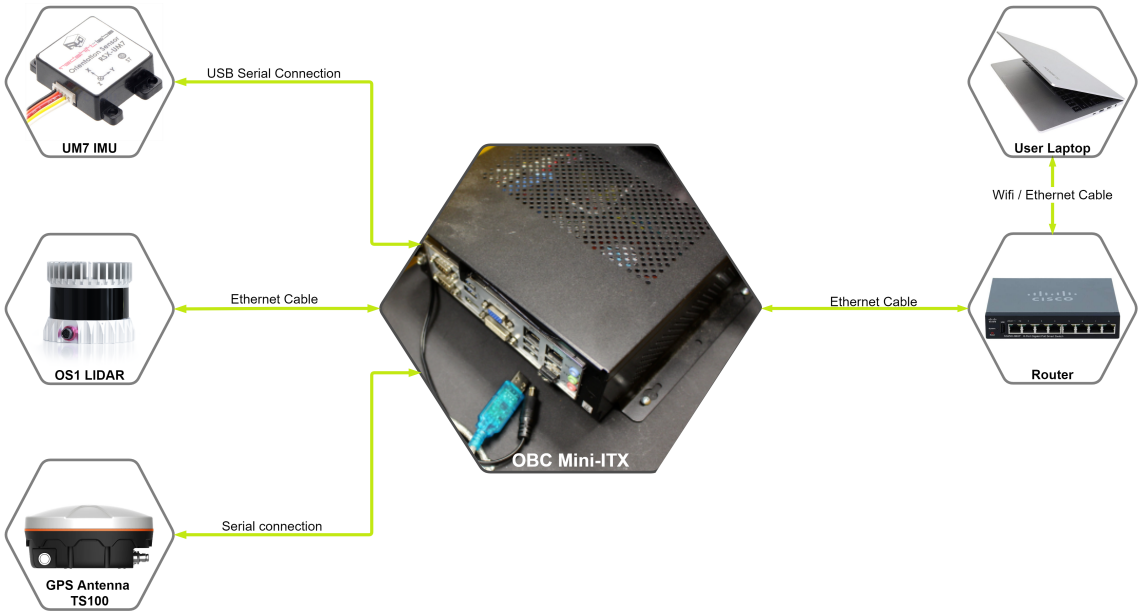


Figura 4.12: esquema de componentes hardware conectados al OBC del robot

Fuente Imagen: Autor

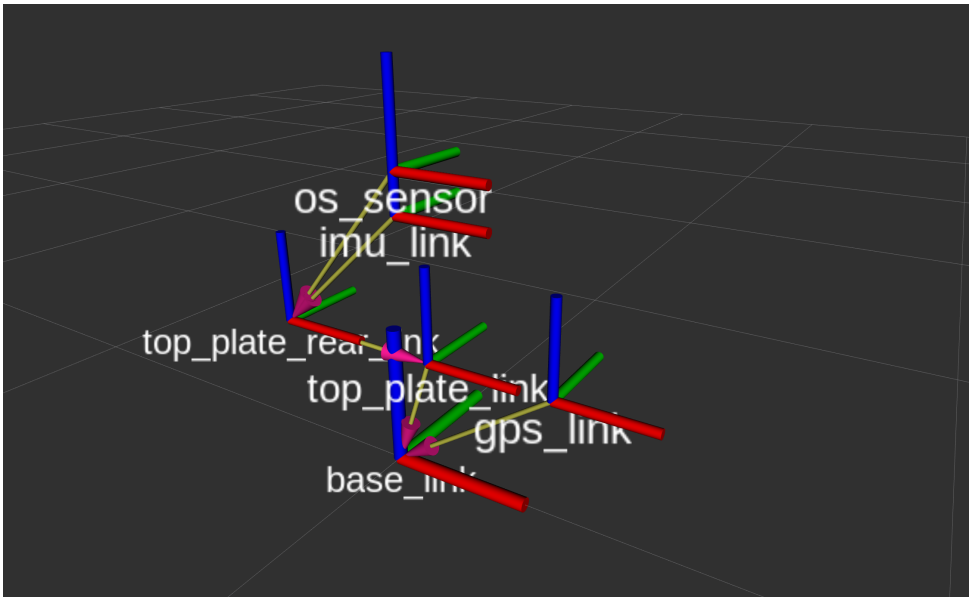


Figura 4.13: transformadas representativas en el robot. Visualización en rviz

Fuente Imagen: Captura de pantalla en rviz

4.5. Estructura de paquetes SW

Dentro del ecosistema ROS se pueden agrupar los diferentes paquetes y nodos ROS en paquetes que se ejecutarán al mismo tiempo a través de ficheros *launch*. De esta manera se han definido una serie de paquetes base, cada uno lanzado a través de un fichero con extensión *launch*, que ejecutarán de manera coordinada los diferentes módulos necesarios.

4.5.1. Husky Base

Es el paquete base de la plataforma Husky, se lanza por defecto cuando el OBC se enciende. Incluye los paquetes base del robot, control del mismo, odometría e interfaces para comandarlo ya sea a través de diferentes topics así como del mando remoto incluido. Parte de esta información, e información de cómo utilizar y configurar la plataforma robótica puede encontrarse en [5].

Por lo general es este paquete el encargado de lanzar el ROS Master que será utilizado durante la ejecución.

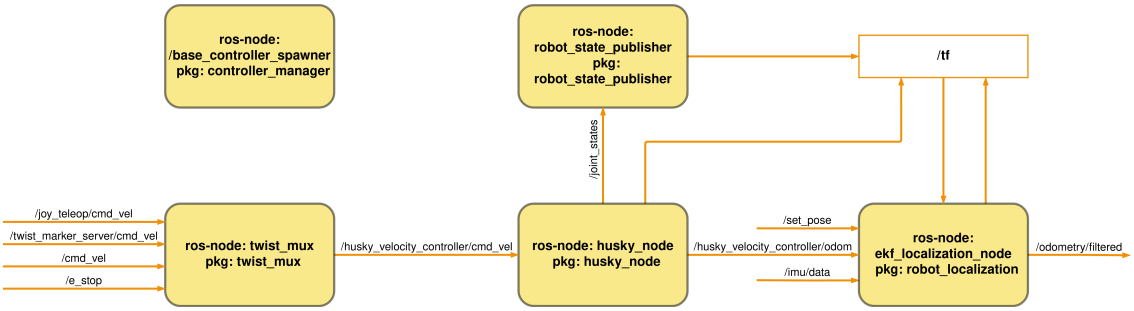


Figura 4.14: esquema de componentes involucrados en el paquete base lanzado por la plataforma robótica

Fuente Imagen: Autor

Los componentes más relevantes, representados en la figura 4.14 se encargan de:

- El nodo **base_controller_spawner** prepara el control del robot así como los límites de velocidad configurados. Incluye las interfaces hardware que mandarán los comandos a los motores. En este caso se lanza un controlador para robot diferencial, que recibe comandos a través del topic */husky_velocity_controller/cmd_vel*. Aunque los controladores son lanzados por este nodo, la interfaz entre la plataforma y ROS la gestiona el nodo **husky_node**.
- El nodo **twist_mux** es el encargado de gestionar los comandos de velocidad de diferentes fuentes. En base a las prioridades configuradas elegirá que comando enviar a través del topic */husky_velocity_controller/cmd_vel*. La información se codifica mediante el tipo de mensaje *geometry_msgs/Twist*. De mayor a menor prioridad estos comandos pueden provenir de:

  - Mando teleoperado: un nodo se encargará de hacer de interfaz con el mando remoto para publicar los comandos recibidos en el topic */joy_teleop/cmd_vel*.
  - rviz: un nodo se encarga de procesar los comandos provenientes de la interfaz gráfica para publicarlos sobre */twist_marker_server/cmd_vel*
  - Cualquier fuente podrá además publicar sobre el topic */cmd_vel*. Esta interfaz está pensada para que el usuario pueda utilizarla publicando desde los nodos ROS desarrollados.

  El nodo incluye además un topic para parar el robot, */e_stop*, que tiene mayor prioridad que cualquiera de los anteriores.

- El nodo **husky_node** actúa como interfaz tanto de control como de comunicación entre la plataforma y ROS. Publica la odometría en base al movimiento de las ruedas en el topic */husky_velocity_controller/odom* así como recibe comandos de velocidad a través del topic */husky_velocity_controller/cmd_vel*.

- El nodo **robot_state_publisher** se encarga de publicar todas las transformadas entre los sistemas de coordenadas definidos en el robot, además de actuar como interfaz para actualizar las mismas.

- Una librería permite publicar diferentes sistemas de coordenadas así como las transformadas entre las mismas. El patrón de suscripción/publicación funciona similar al resto de topics, aunque se hace a través de una librería específica. Se ha representado en la imagen de manera excepcional, con el nombre de **/tf**. Almacena todos los sistemas de coordenadas existentes en el sistema permitiendo a los diferentes nodos acceder tanto para leerlos como para actualizarlos.

- El nodo **ekf_localization_node** combina datos de odometría e IMU (proveniente de los topics */husky_velocity_controller/odom* e */imu/data* respectivamente) mediante un filtro de Kalman Extendido para proveer una odometría en conjunto */odometry/filtered*. Incluye una interfaz para inicializar la posición del robot, mediante el topic */set_pose*.

4.5.2. Husky Sensors

Es el paquete que conecta los diferentes sensores que se han instalado en el robot, publicando la información de los mismos en diferentes topics que estarán disponibles para usarse.

Lanza las interfaces con la *IMU*, la antena GNSS y el *LIDAR*. Puede verse en la figura 4.15 un diagrama de los componentes principales que intervienen cuando se ejecuta este paquete.

Como puede verse cada sensor comunica directamente con un nodo que actuará como driver, recibiendo la información del sensor y traduciéndola a mensajes de ROS. Concretamente:

- IMU: El driver **um7_driver** es el encargado de abrir el puerto serie configurado para recibir la información de la IMU. Esta información se codifica en

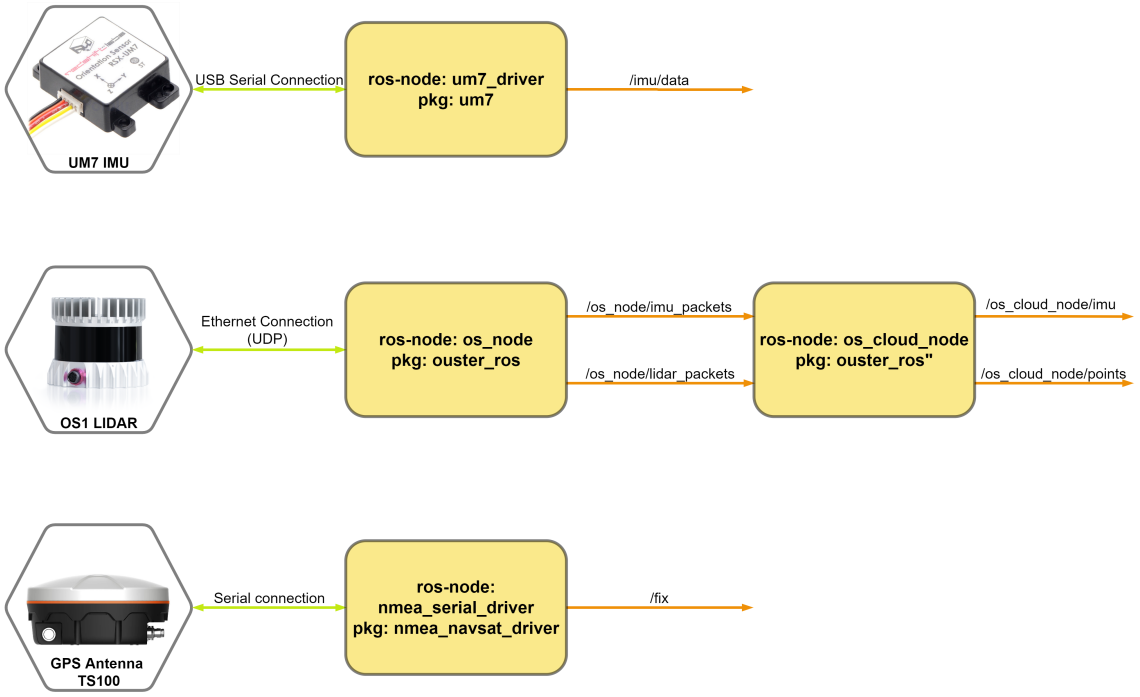


Figura 4.15: esquema de componentes involucrados en el paquete de sensores

Fuente Imagen: Autor

diferentes tipos de mensajes para ser publicada dentro del ecosistema ROS. El topic más relevante para el uso que se dará es el de */imu/data* que incluye información en formato *sensor_msgs/Imu*.

- LIDAR: El sensor LIDAR utiliza dos nodos: **os_node** y **os_cloud_node**. El primero será el encargado de abrir el puerto UDP para recibir los paquetes de información del sensor LIDAR. Estos mensajes se traducirán al ecosistema ROS y serán procesados por el segundo nodo. El sistema del driver del LIDAR está pensado para poder conectar y sincronizar diferentes sensores al mismo tiempo, en este caso solo se dispone de uno. El segundo nodo será el que publique la información que podrá utilizarse posteriormente, concretamente sobre los topics: **/os_cloud_node/imu** (mismo tipo de mensaje que la IMU) y el topic **/os_cloud_node/points**, donde se codifican los mensajes en formato *sensor_msgs/PointCloud*.

- GNSS: La antena GNSS a través del driver **nmea_serial_driver** publicará los datos en el topic **/fix**. El mensaje publicado es de tipo *sensor_msgs/NavSatFix*.

4.5.3. Husky SLAM

El módulo de SLAM, del Cartographer, incluye dos nodos que gestionan todo el proceso de SLAM. Puede verse el diagrama en la figura 4.16. Cada nodo estará encargado de:

- **cartographer_node**: es el nodo que lleva a cabo todo el proceso de SLAM. Si se ha configurado para ello, tomará los datos de GNSS, IMU, PointCloud y

odometría para ello. Además está suscrito a las transformadas del sistema, /**tf**, donde también publicará la transformada entre el robot y el mapa generado. Genera una lista de sub-mapas con la referencia entre cada uno de ellos. Es un mensaje de tipo *cartographer_ros_msgs/SubmapList)* publicado sobre el topic /**submap_list**.

- **cartographer_occupancy_grid_node**: Traduce la lista de sub-mapas a un mapa de ocupación en formato típico de ROS, utilizando el tipo de mensaje *nav_msgs/OccupancyGrid* publicado sobre el topic /**map**. Nótese que el proceso de generar el mapa es costoso y se publicarán con una frecuencia de alrededor de 1Hz (dependiendo también de la capacidad del ordenador donde se ejecute).

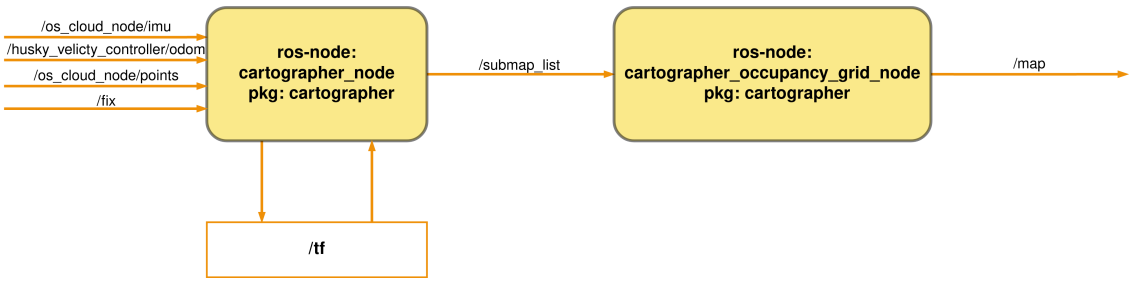


Figura 4.16: diagrama con componentes principales de ROS del módulo de SLAM
Fuente Imagen: Autor

Nótese que en el diagrama los nombres de los topics se han mantenido tal cual están en el sistema. Dentro del módulo de SLAM se debe hacer un *remap* para algunos de los topics de la siguiente manera:

- El topic /**os_cloud_node/imu** se renombrará a /**imu**.
- El topic /**os_cloud_node/points** se renombrará a /**points2**.
- El topic /**husky_velocity_controller/odom** se renombrará a /**odom**.

4.5.4. Configuración y ejecución de los diferentes paquetes

Se han desarrollado diferentes *scripts* que facilitan la configuración del entorno para trabajar con los sensores y plataforma descritos en este capítulo así como funciones preparadas para simplificar el manejo de los mismos. Pueden encontrarse almacenados en el repositorio en GitHub que puede consultarse en el siguiente enlace `https://github.com/enheragu/test_utils/tree/master/scripts`. Dichas funcionalidades están separadas en los siguientes ficheros:

- log_utils.sh: Funciones específicas para escribir mensajes por terminal aprovechando diferentes formatos.
- husky_private_functions.sh: Funciones de configuración para detectar automáticamente las IPs a configurar, puertos y cargar variables de entorno; función para sincronizar el reloj interno del sensor *LIDAR* utilizando el reloj del OBC.

- husky_setup.sh: Funciones principales para cargar el entorno y ejecutar los diferentes módulos explicados previamente. Se describen más en detalle las diferentes funciones disponibles a continuación.

El fichero `husky_setup.sh` será invocado para disponer de las variables de entorno y funcionalidades de manera que los siguientes comandos estarán habilitados.

Se configurará el entorno con el comando siguiente. Este comando se encargará de cargar todas las variables de entorno y funcionalidades de ROS y del Worskpace en la sesión actual. Además detectará las IPs y puertos de los sensores correspondientes preparando las variables de entorno específicas para el correcto funcionamiento de las mismas.

```
husky_ros_setup
```

El siguiente comando permite lanzar de nuevo toda la capa funcional del robot. Por norma general este módulo se lanza cuando arranca el OBC de la plataforma y no debería ser necesario ejecutarlo a mano.

```
husky_launch_base
```

Para lanzar el módulo que permitirá disponer de la información de los sensores es utilizará el comando presentado a continuación. Este comando se encarga de comprobar si la sincronización del reloj del LIDAR ha sido efectuada, llevarla a cabo si no, y de lanzar los drivers de los diferentes sensores.

```
husky_launch_sensors
```

Para las pruebas el módulo de SLAM se ha ejecutado *offline* y no está instalado en la propia plataforma. Cuando se instale se procederá a poner un comando equivalente a los anteriores para encapsular su funcionamiento.

Capítulo 5

Desarrollo experimental

Tanto la validación del hardware instalado como del software requiere realizar una serie de pruebas para comprobar que el funcionamiento es el esperado. En este capítulo se describen los diferentes experimentos realizados tanto con los sensores, como con el software en entornos simulados y reales. La primera sección describe los diferentes experimentos llevados a cabo para validar la antena GNSS-RTK en diferentes condiciones. La siguiente sección recoge la información referente a los experimentos realizados con el software de SLAM elegido.

5.1. Pruebas con GNSS-RTK

Como se ha descrito en la sección 4.2.1 las antenas GNSS utilizadas se componen de una pareja de antenas. Una de ellas, nombrada como *base* a partir de ahora, se situará en una ubicación fija y conocida. Esta posición se configura en la propia antena para ser utilizada como referencia en las correcciones que realizará con la otra antena. La segunda antena, nombrada como *rover*, va montada sobre la plataforma robótica. Nótese que la visibilidad entre ambas antenas debe ser óptima para que puedan establecer la comunicación por radio para funcionar en modo RTK.

Una vez configuradas ambas antenas se han llevado a cabo diferentes pruebas, se mencionan a continuación las más relevantes junto a las conclusiones a las que se ha podido llegar.

Para la realización de los experimentos se ha utilizado el driver de GNSS descrito previamente y se ha volcado la información del topic correspondiente a un fichero para su posterior análisis. Se ha desarrollado una herramienta en *Python* para extraer la información, proyectar sobre los mapas y extraer las gráficas presentadas a continuación. El código que lleva a cabo estas operaciones puede encontrarse en el siguiente enlace: `https://github.com/enheragu/test_utils/tree/master/python`. Se presentan en esta sección diferentes resultados y conclusiones extraídas del análisis de dichos datos.

A lo largo de esta sección se presentan diferentes representaciones gráficas de la información GNSS obtenida, se ha representado en todos los casos siguiendo el

esquema de colores presentado en la figura 5.1. Se podrán ver diferentes experimentos en donde las antenas se ven forzadas a trabajar en los diferentes modos de funcionamiento: RTK, DGPS, SPS. En ciertas circunstancias la antena es incapaz de proveer datos correctos, habiendo partes de las trayectorias que se han perido. Estas trayectorias se ha dibujado de manera aproximada para facilitar la lectura e interpretación de los resultados.

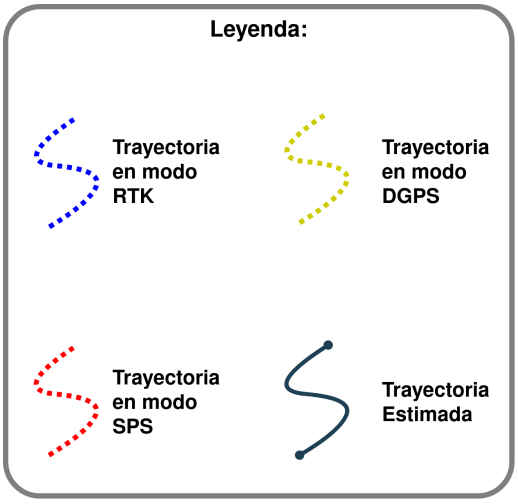


Figura 5.1: nomenclatura utilizada en la representación de mapas y diagramas

Fuente Imagen: Autor

Se han representado las trayectorias proyectando las coordenadas GNSS (latitud,longitud) sobre mapas 2D extraídos de Google Maps. Para ayudar en la interpretación además se han representado gráficamente la variación de error en el plano del suelo. En este caso se ha optado por una representación diferente, en base al llamado UTM. Se trata de una proyección cartográfica que permite dividir la tierra en cuadrantes tal y como puede verse en la figura 5.2. Cualquier coordenada GNSS se representará en coordenadas UTM con la identificación del cuadrante en el que esté así como la transformada en metros (este, norte) con respecto a la esquina inferior izquierda de dicho cuadrante.

El número de cuadrante se obtendrá en base a la longitud y la letra en base a la latitud. Las longitudes, que van desde -180° a 180°, se dividirán en 60 *husos* de 6° de longitud cada una; las latitudes, que van desde los -80° hasta 84°, se dividirán en 20 *bandas* de 8° de longitud cada una. Los *husos* se identifican mediante un identificador numérico que va desde el número 1 al 60; las *bandas* se identifican mediante una letra que va desde la 'C' hasta la 'X' excluyendo las letras 'I' y 'O' De manera genérica (nótese que los cuadrantes 31, 32, 33, 35 y 37 se engrosan en algunas zonas) se podrá obtener el número de cuadrante con la siguiente transformación:

```
int((long + 180) / 6) + 1
```

La letra correspondiente se podrá obtener de la siguiente manera:

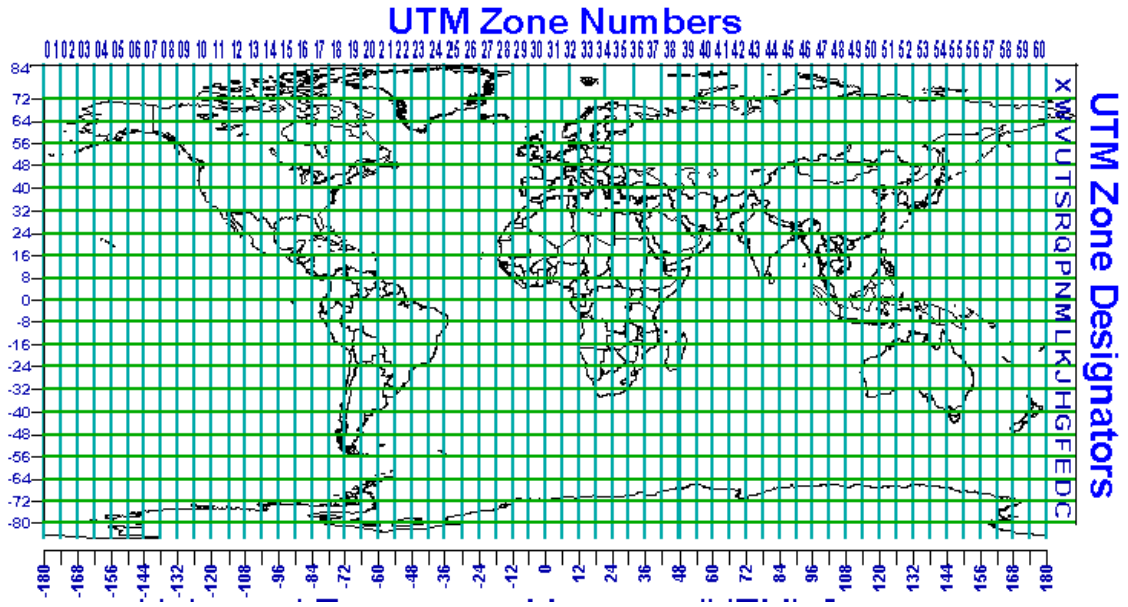


Figura 5.2: mapa de cuadrantes UTM

Fuente Imagen: Extraída de [2]

```
LETTERS = "CDEFGHJKLMNPQRSTUVWXX"
letter = LETTERS[int(float(latitude + 80) / 8.0)]
```

La transformada final, entre el (0,0) del cuadrante obtenido se podrá calcular como la distancia entre las coordenadas de dicho centro y las coordenadas GNSS obtenidas.Para la realización de estos cálculos se ha empleado la librería *utm* integrada en *Python*.

5.1.1. Interfaz y Configuración de la antena GNSS

La configuración de las antenas se hace a través del puerto serie. Es necesario configurar el tipo de comunicación así como que rol tendrá cada una de las antenas, una como base y otra como antena móvil sobre la plataforma. Para este fin, el fabricante incluye para la configuración de las antenas una interfaz gráfica que está disponible solo para Windows. La configuración inicial se ha llevado a cabo con dicha interfaz. Para mayor flexibilidad a la hora de trabajar, y puesto que el sistema operativo tanto del robot como de los ordenadores de trabajo está basado en Linux, se ha desarrollado una interfaz gráfica con las funcionalidades mínimas que permiten configurar las antenas, tanto la antena *base* como la del *rover*, a través del puerto serie.

Para el desarrollo de esta interfaz se ha utilizado el motor de interfaces Tk integrado en *Python*. Se ha extraído de [13] la información sobre el protocolo de comunicación empleado por las antenas para mandar y procesar los comandos intercambiados.

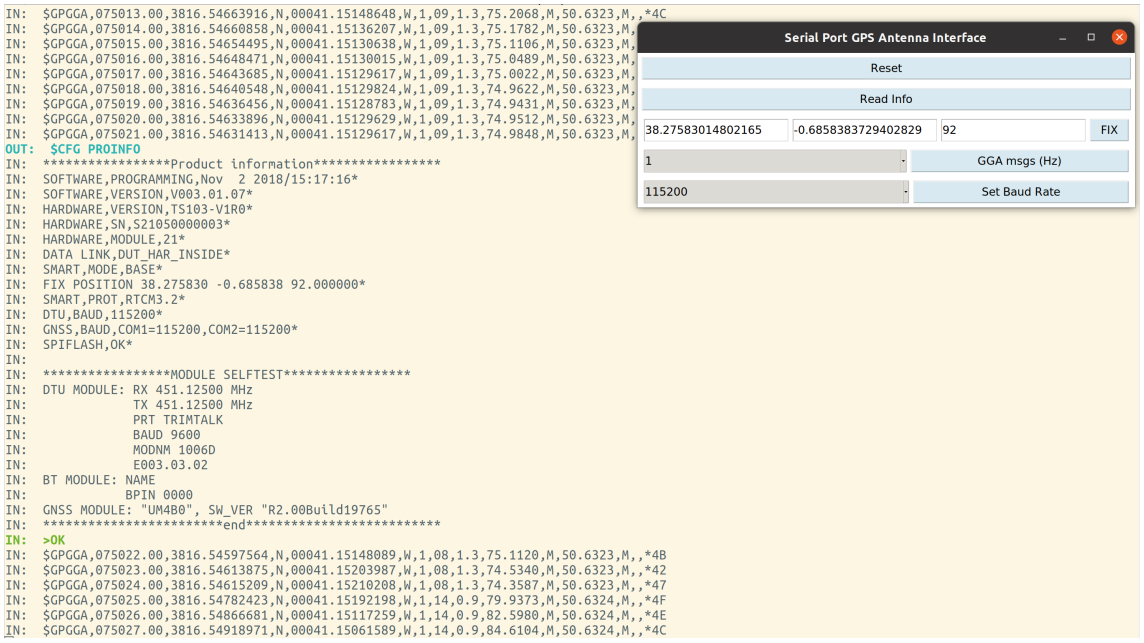


Figura 5.3: comando para recibir la información de la configuración actual de la antena

Fuente Imagen: Autor

Se ha aprovechado para modificar el *log* por terminal para hacerlo más claro para el usuario. Pueden ver ejemplos de diferentes comandos en las figuras a continuación, donde se remarcan los mensajes clave, tanto salientes como entrantes.

En la figura 5.3 puede verse tanto la interfaz como el *log* en la terminal viendo el contenido de la información que hay configurada en la antena en el momento de ejecución. Antes y después de dicho mensaje pueden verse los mensajes correspondientes a la posición de la antena.

En la figura 5.4 puede verse una de las operaciones realizadas a la antena funcionando como *base*. Se configuran las coordenadas GNSS donde está ubicada la misma. Puede verse en la terminal tanto el comando enviado como la confirmación de su correcta interpretación y ejecución.

Por defecto las antenas utilizan el protocolo estándar *NMEA* para retransmitir la información. Este protocolo define un formato específico en las tramas de información enviadas, tanto entre antenas, como en este caso entre antena y ordenador. Incluye además diferentes prefijos que permiten configurar y procesar el tipo de información que se codifica en cada mensaje. En este caso se ha utilizado mensajes de tipo *GGA*, que proporciona datos de localización 3D y precisión. Puede verse tanto en la figura 5.4 como en la figura 5.3 los mensajes recibidos de la antena, que inician como *GPGGA*. *GP* es el prefijo utilizado para dispositivos *GNSS* y GGA es el tipo de mensaje.

Se puede consultar tanto al documentación como el código fuente de la interfaz

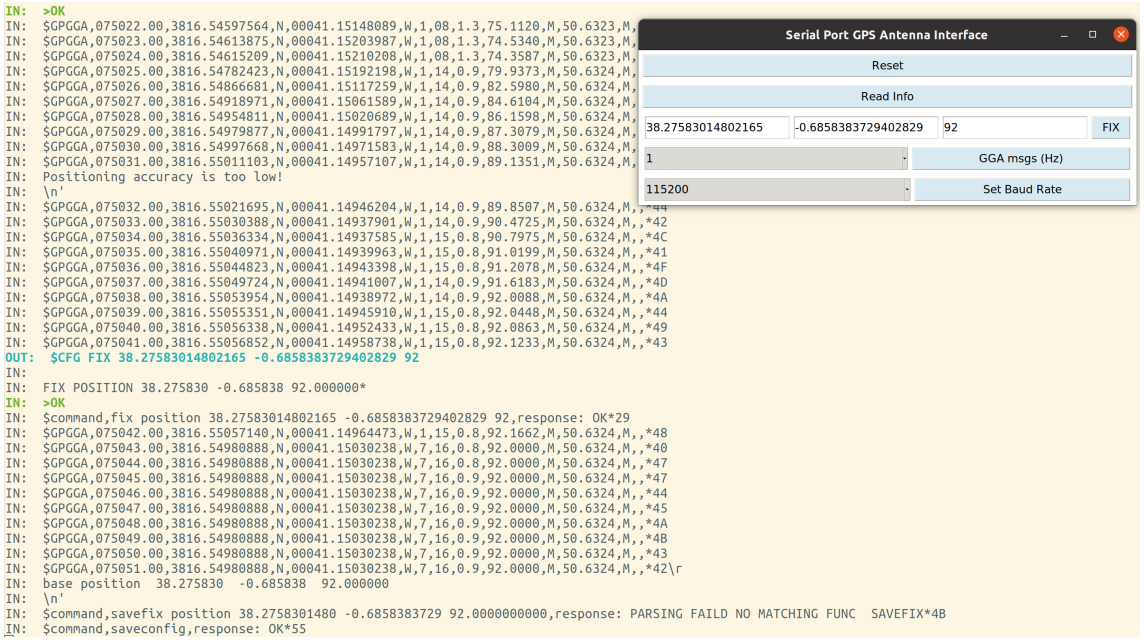


Figura 5.4: comando para fijar la posición actual de la *base*

Fuente Imagen: Autor

en un repositorio GitHub en el siguiente enlace: `https://github.com/enheragu/test_utils/tree/master/python#gui_antena_configpy`.

5.1.2. Correspondencia RTK con Mapas existentes

El objetivo de este experimento es comprobar la correspondencia dentro del campus de Elche entre las coordenadas GNSS provistas por Google Maps y las obtenidas de las antenas GNSS.

Se comprobará en diferentes puntos diferentes tanto de la *base* como de la antena *rover* la comunicación entre ambas antenas, el modo de funcionamiento, error y la correspondencia de dichas coordenadas con las observables en Google Maps. Se puede comprobar que la resolución máxima que puede obtenerse en Google Maps es de 10 cm, que se usará como referencia al no disponerse de un *ground truth* con mayor precisión. Esta falta de un *ground truth* preciso impude obtener gráficas de la variación de error a lo largo de los experimentos.

Para comprobar se han realizado experimentos alrededor del edificio Innova, seleccionando seis puntos sobre los que se realizarán los experimentos, a continuación se muestran las coordenadas GNSS de cada uno de ellos. Puede verse en las figuras 5.5, 5.6 y 5.7 la correspondencia sobre el mapa de dichos puntos. En estas imágenes se han proyectado sobre un mapa extraído de Google Earth las coordenadas GNSS proporcionadas por las antenas. Las coordenadas extraídas de Google Maps para configurar la antena *base* así como para utilizar de referencia son las siguientes:

- p1: [38.27537888125739, -0.6856797963437212]

- p2: [38.27526464974311, -0.6855993300726063]
- p3: [38.27520411223183, -0.6856107294610143]
- p4: [38.27518538528904, -0.6856249366190161]
- p5: [38.27478893101137, -0.686135354928984]
- p6: [38.27464507180332, -0.6869747851455184]

En cada caso se han guardado un mínimo de diez lecturas de la posición GNSS (funcionando a 10Hz), que podrán verse representadas sobre imágenes aéreas a continuación. Sobre las mismas puede verse en color azul los puntos en que las antenas funcionaban en modo RTK así como en rojo cuando las antenas funcionaban en modo SPS.

Figura 5.5: puntos de *base* y *rover* representados sobre imagen por satélite
Fuente Imagen: Puntos GNSS interpolados sobre imagen

Por simplicidad se han resumido varios experimentos sobre la figura 5.5. Se han ido variando las posiciones de las antenas *rover* y *base*. Se han alternado entre las siguientes configuraciones:

- test0: *base* ubicada sobre p2 (sin configurar la posición como fija).

- test1: *base* en p2 (configurada como posición fija) y antena *rover* en p1.
- test2: *base* como en el test1 y la antena *rover* sobre p3.
- test3: *base* como en el test1 y la antena *rover* sobre p4.
- test6: *base* posicionada en p1.
- test7: *base* posicionada en p4.

En esta serie de experimentos se pudo comprobar la correcta correspondencia entre coordenadas del mapa y la proporcionada con la antena, validando así la posibilidad de comandar el robot mediante coordenadas directamente extraídas de Google Maps. La configuración de la *base* se puede hacer de manera manual y permanecerá configurada aun apagando la antena. Ambas antenas son capaces de comunicarse para funcionar en modo RTK y actualizar la posición de la antena *rover* con las correcciones de la antena *base*. Puede comprobarse como la precisión con las antenas estáticas es muy elevada, así como la variación entre lecturas muy pequeñas. En algunos casos se pueden ver algunas lecturas alejadas del punto correcto.

Figura 5.6: coordenadas *rover* modo RTK en la segunda zona representados sobre imagen por satélite

Fuente Imagen: Puntos GNSS interpolados sobre imagen

En el test4 la *base* permaneció en p2 posicionando la antena *rover* en el punto p5, tal y como puede verse en la figura 5.6. Puede verse representado en azul la posición provista por la antena, puede comprobarse que la precisión sigue siendo elevada (en el *log* puede verse como la covarianza es superior al caso previo).

En el test5, manteniendo la antena *base* en p2, se alejó más la antena *rover* hasta que perdió la comunicación con la antena *base*, en el punto p6. En este caso la antena pasa a modo SPS recibiendo posiciones GNSS pero sin llevar a cabo corrección alguna. Pueden verse en la figura 5.7 las lecturas almacenadas así como la elipse de error estimada por la antena en estas circunstancias. Aunque no es capaz de llevar a cabo correcciones con la antena *base* la precisión sigue siendo elevada. Nótese que en los ejemplos previos la elipse de error no es lo suficientemente grande como para verse representada en la imagen.

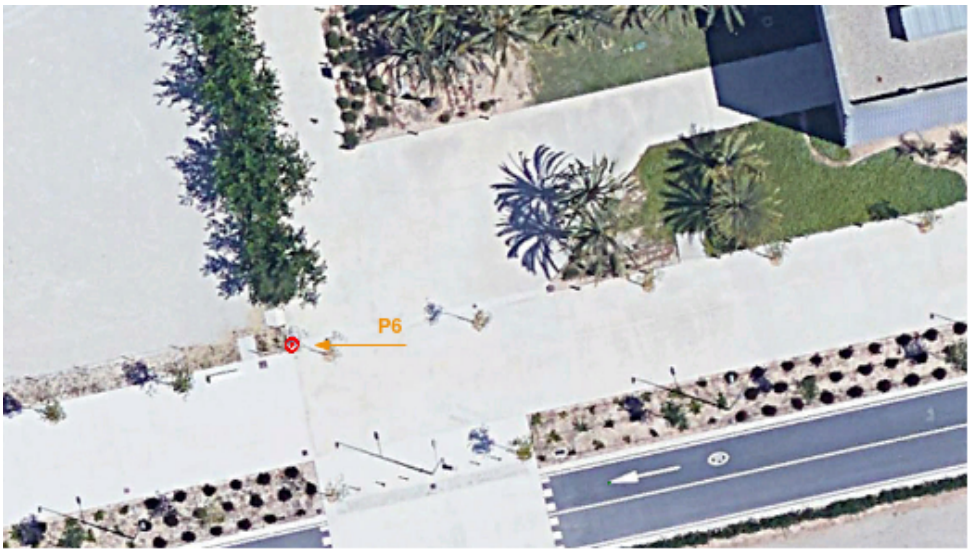


Figura 5.7: coordenadas *rover* modo SPS en la tercera zona representados sobre imagen por satélite

Fuente Imagen: Puntos GNSS interpolados sobre imagen

Puede verse como las coordenadas obtenidas cuadran dentro de los márgenes posibles con las representadas tanto sobre las imágenes como en Google Maps. Cabe destacar, que además de la resolución máxima de 10 cm medible sobre los mapas, la proyección de los mismos no es completamente vertical, puede verse muy claramente en la figura 5.7 como la esquina del edificio aparece a lo largo de varios puntos. En este caso, puntos que deberían tener las mismas coordenadas sobre el mapa difieren dando una distancia de desplazamiento de varios metros, detalle que puede verse más claro en la figura 5.8. Cuando se usen coordenadas obtenidas de Google Maps habrá que tener en cuenta estas limitaciones para no inducir mayores errores.

Figura 5.8: error de proyección de los mapas
Fuente Imagen: Captura de Google Maps

5.1.3. RTK en exteriores

El objetivo de este experimento es comprobar el funcionamiento nominal en exteriores de ambas antenas funcionando en modo RTK. Se eligió una trayectoria dentro del campus (puede verse parte del edificio Innova en la figura 5.9 así como el aparcamiento cercano al edificio del Rectorado).

En la figura 5.9 pueden verse los resultados de este experimento. En azul pueden verse las coordenadas GNSS obtenidas de la antena *rover* y proyectados sobre la imagen. También se ha representado la posición de la antena *base*. Además se ha representado la elipse de error para cada una de las lecturas, aunque la resolución de la imagen no permite distinguirla del propio punto. Puede verse cómo durante toda la trayectoria el modo de funcionamiento se mantiene en RTK salvo en unos puntos que cambia a DGPS.

En la figura 5.10 se ha representado la misma información en forma de gráfica. Puede verse tanto el desplazamiento en X e Y en metros (este y norte en coordenadas UTM) con respecto al punto inicial. Además, asociada a cada una de las medidas y para cada uno de los ejes se ha representado el error σ en metros. En esta gráfica se puede comprobar como los errores asociados a cada modo de funcionamiento varían dentro de los órdenes de magnitud esperados.

Nótese que las imágenes por satélite no son exactamente de este año y por tanto la vegetación que se observa no es la actual. Puede verse en la gráfica como a partir de aproximadamente la mitad del tiempo empieza a fluctuar más el error. Se ha asociado a una zona con mayor vegetación, provocando esta mayores interferencias en la señal GNSS. Aun cambiando momentáneamente a modo DGPS puede verse como la comunicación entre ambas antenas no se ha perdido y en ningún momento se ha pasado a modo SPS. Es relevante comprobar el gran área que se podrá mapear contando con información precisa de GNSS.

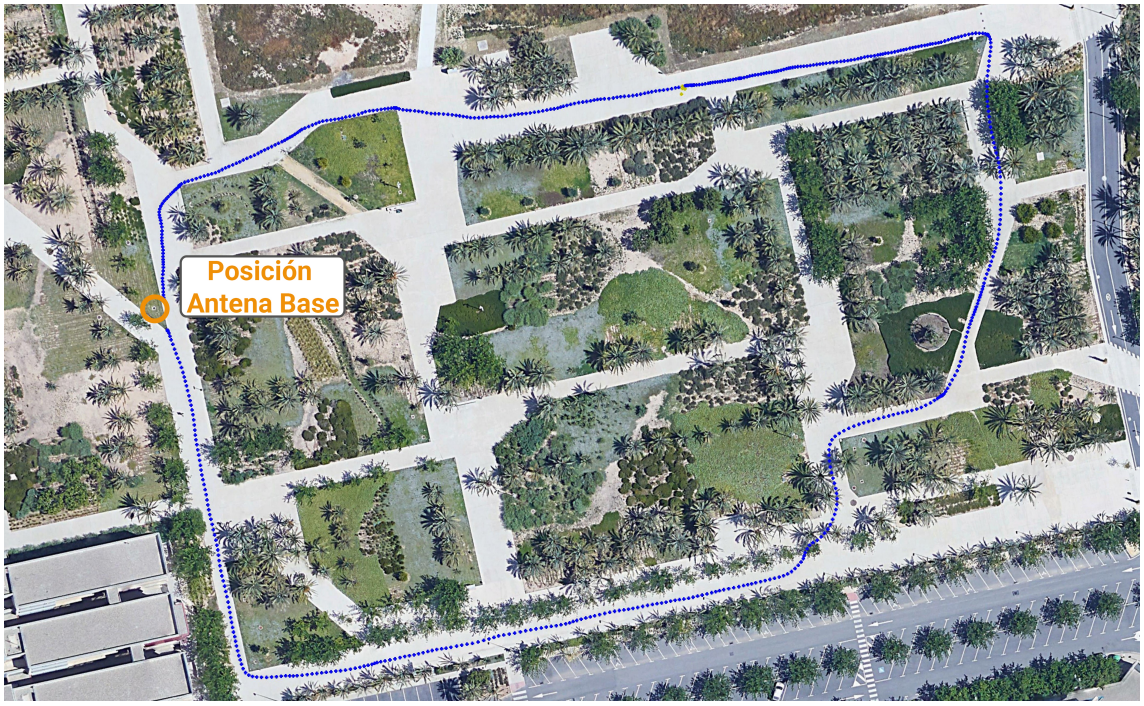


Figura 5.9: trayectoria nominal en RTK de la antena *rover* representados sobre imagen por satélite

Fuente Imagen: Puntos GNSS interpolados sobre imagen

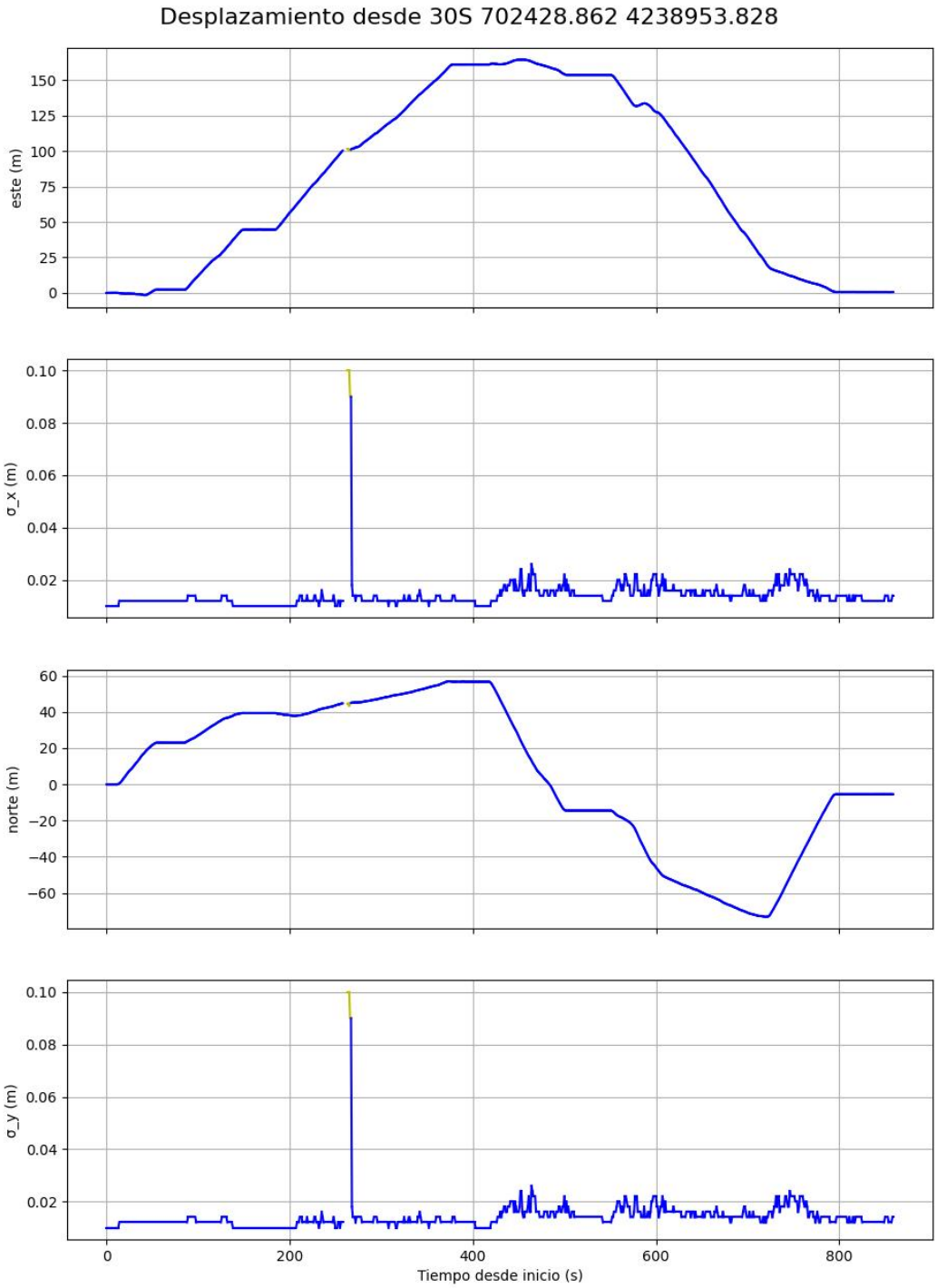


Figura 5.10: representación gráfica de los datos de desplazamiento X e Y en coordenadas UTM así como del error asociado a cada una en la trayectoria RTK nominal

Fuente Imagen: Autor

5.1.4. Pérdida de conexión antena rover-base cerca de edificios

El objetivo de este experimento es comprobar cómo se comporta la antena tanto cuándo se pasa cerca de un edificio como cuando se pierde la visibilidad con la antena *base*, así como cuando se recupera la misma.

Como en los casos anteriores, se ha proyectado las coordenadas GNSS sobre imágenes de Google Earth. Se han representado en color azul los puntos cuya medida se ha realizado en modo RTK, en amarillo se representan los puntos registrados en modo DGPS y por último en rojo están representados los puntos en modo SPS, cuando no hay corrección entre antena *rover* y *base*. Se han representado además las elipses de error, aunque en muchos de los puntos no son lo suficientemente grandes para verse en la imagen. La antena *base* se ha posicionado en el mismo punto que en el caso de la figura 5.9.

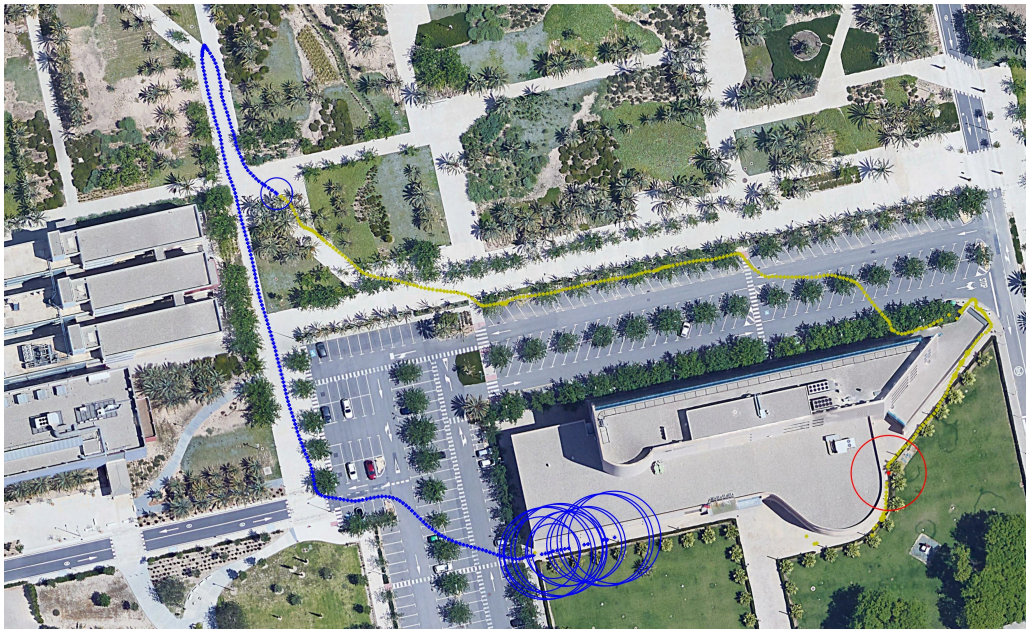


Figura 5.11: trayectoria exterior representada sobre imagen por satélite. Pérdida de conexión por edificio

Fuente Imagen: Puntos GNSS interpolados sobre imagen

El experimento se ha hecho saliendo desde el lateral del edificio Innova y rodeando el edificio del Rectorado. Puede verse en la figura 5.11 los resultados de dicho experimento. Puede verse que la trayectoria empieza y acaba al lado de la antena *base*, en la parte superior de la imagen.

En la figura 5.12 se ha ampliado la zona más interesante del experimento. Puede verse como según se llega al edificio (puntos en azul de la izquierda de la imagen), el error empieza a aumentar debido a la influencia de la pared y de las reflexiones de la señal con la misma. Puede verse como a continuación se ha perdido tanto la

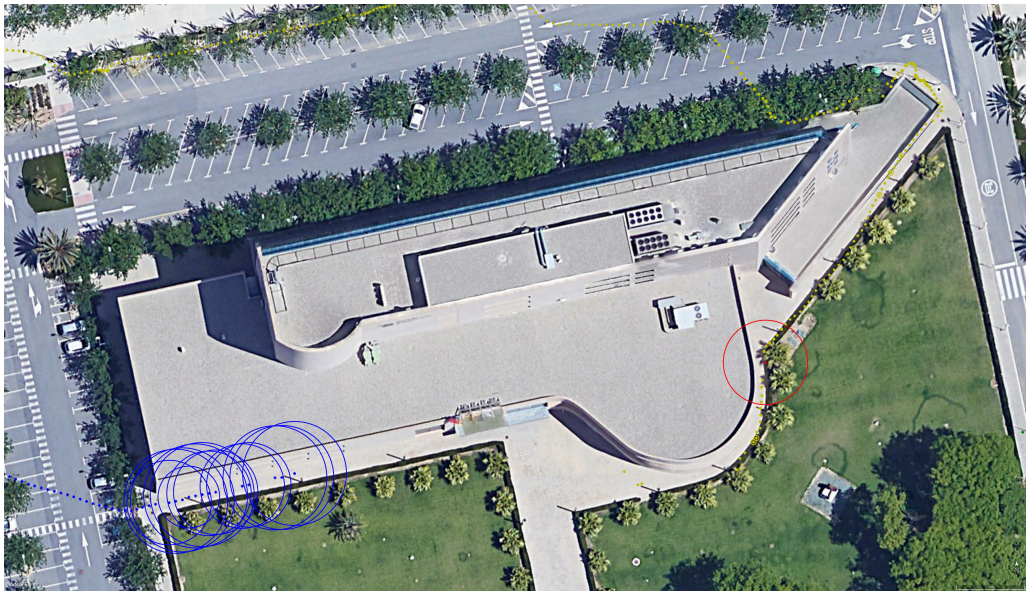


Figura 5.12: detalle pérdida de conexión y reconexión entre las antenas *rover-base* con un edificio provocando interferencia

Fuente Imagen: Puntos GNSS interpolados sobre imagen

señal UHF con la antena *base* como la señal GNSS de la antena, impidiendo que trabaje en modo SPS. Un poco más adelante la antena *rover* recupera la señal GNSS pudiendo verse un punto en modo SPS con bastante error. A partir de ahí la comunicación con la antena *base* se recupera y continúa en modo DGPS durante buena parte de la trayectoria. Se deduce que tanto el edificio como la vegetación posterior, pasando cerca de arbustos y por debajo de las palmeras es la causa de que la antena se mantenga trabajando en modo DGPS hasta casi el final, cuando sale de la zona de palmeras y tiene visibilidad con la antena *base*.

Es importante remarcar que tanto las interferencias producidas por el edificio, reflexiones en la pared y vegetación afectarán de dos formas: tanto a la señal GNSS recibida por la antena *rover*, como a la comunicación entre esta antena y la antena *base*.

5.1.5. Pérdida de conexión antena rover-base interior-exterior

Este experimento continúa el anterior forzando una pérdida de comunicación entre ambas antenas, pero en este caso entrando en un edificio (edificio Innova). Se pretende comprobar como es la reconexión con la antena *base* así como las interferencias causadas tanto a la entrada como a la salida del edificio.

La figura 5.13 muestra la proyección de los puntos obtenidos de la antena *rover* sobre el mapa. Los colores empleados son los mismos que en experimentos previos así como la ubicación de la antena *base*. Nuevamente la trayectoria comienza y finaliza junto a la antena *base*, en la parte superior de la imagen.

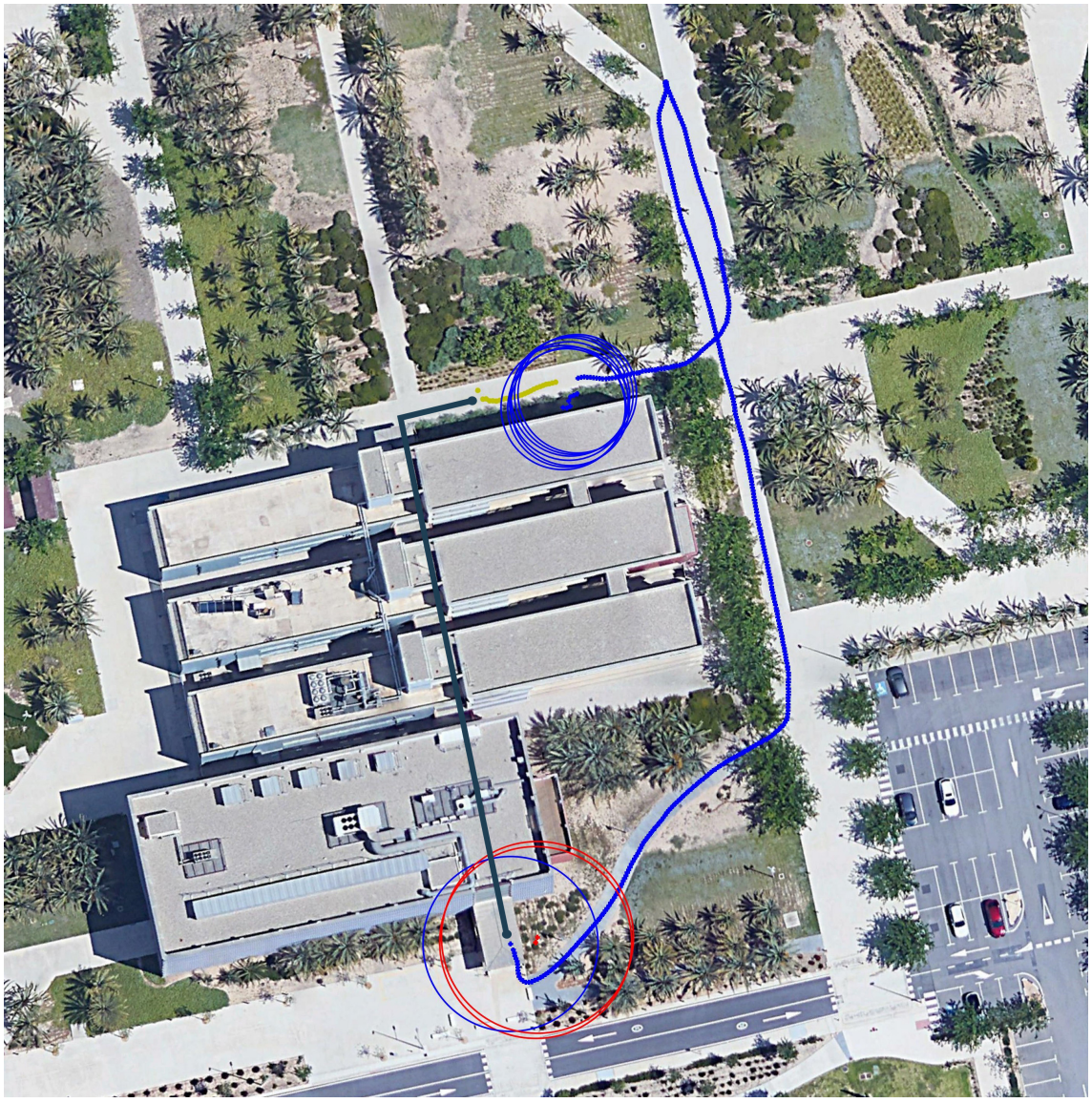


Figura 5.13: trayectoria interior-exterior de puntos GNSS representada sobre imagen
Fuente Imagen: Puntos GNSS interpolados sobre imagen

A lo largo de la trayectoria exterior puede verse un comportamiento equivalente al del experimento anterior, manteniendo la producción de datos en modo RTK con bajo error de medida. Las situaciones más interesantes se pueden observar a continuación en detalle, tanto a la entrada como salida del edificio.

Figura 5.14: detalle pérdida de conexión entre las antenas *rover-base* a la entrada del edificio

Fuente Imagen: Puntos GNSS interpolados sobre imagen

La figura 5.14 focaliza en detalle la entrada al edificio Innova. Toda la trayectoria de coordenadas se van generando en modo RTK hasta que se entra debajo del porche. Pueden observarse dos situaciones que podrían explicarse como:

- Interferencias en la señal GNSS de la antena *rover* que aumentan el error aun en modo RTK. Estas interferencias podrían deberse tanto a reflexiones con el edificio y el porche de la señal como de pérdida de satélites visibles.
- Pérdida de conexión UHF con la base, que hace que la antena *rover* produzca datos en modo SPS. Estos datos presentan un elevado error probablemente debido tanto a las reflexiones como a la falta de satélites de los que recibir señal en esas circunstancias.

Hay que resaltar que en ambos casos aunque el error estimado aumenta, la elipse de error sigue manteniéndose coherente con la trayectoria realizada. Una vez pasados los portones la señal se pierde completamente, tanto la señal UHF como la señal GNSS y la antena deja de producir datos de posición.

En la figura 5.15 puede verse en detalle la situación que se da al salir del edificio con la consiguiente reconexión entre ambas antenas. Puede verse como inicialmente la antena no es capaz de proveer datos por si misma ni en comunicación con la

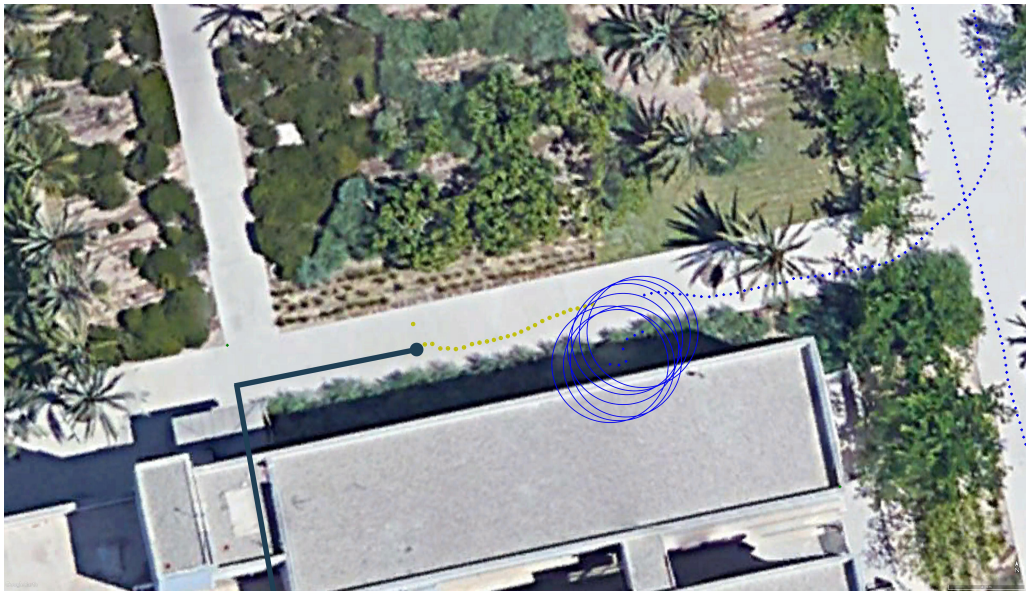


Figura 5.15: detalle reconexión entre las antenas *rover-base* al salir del edificio
Fuente Imagen: Puntos GNSS interpolados sobre imagen

antena *base*, se debe a que aun no tiene señal GNSS. La antena empieza a producir datos directamente en modo DGPS para, posteriormente, saltar a modo RTK. Como se ha visto en otros experimentos el salto de un algoritmo a otro produce una cierta oscilación en la covarianza resultante, en este caso probablemente se sume a la posibles reflexiones con el edificio así como pérdida de satélites y pérdida de señal al pasar por debajo de vegetación. Finalmente recupera la covarianza esperada del modo RTK para finalizar la trayectoria. En esta figura se pueden apreciar diferentes situaciones que podrían afectar a una posterior localización, que conviene resaltar:

- Estando ambas antenas conectadas y produciendo datos en modo RTK se puede ver un aumento de la covarianza con varios puntos que se salen de la trayectoria realizada. Puede verse como en general la elipse dibujada en base a la covarianza podría ser coherente con la trayectoria, aunque utilizar estos datos incluiría bastante error.

- Donde la situación es más crítica es en el primer punto obtenido una vez iniciada la producción de datos de posición. Puede verse como este punto se produce en modo DGPS, con la covarianza esperable en este modo, pero el punto está bastante desviado de la trayectoria realizada. Este tipo de situaciones incluirán datos erróneos en los algoritmos utilizados induciendo bastante error al llevar una covarianza asociada muy baja.

Puede concluirse que en cercanía de edificios así como en potenciales pérdidas de comunicación y reconexión pueden producirse interferencias de distinta índole que podrían afectar gravemente a cualquier algoritmo que vaya a consumir dichos datos. Será importante filtrar este tipo de situaciones para evitar en lo posible depender de estos datos que no harán más que propagar dichos errores a los algoritmos que utilicen esta información.

5.1.6. Pérdida de conexión antena rover-base por distancia

El objetivo de este experimento es comprobar en las condiciones de uso el rango de distancia en el cual ambas antenas son capaces de comunicarse. Puede verse en la figura 5.16 la proyección de los resultados sobre una imagen por satélite. En color azul se ven los puntos obtenidos en modo RTK, con ambas antenas comunicándose, y en rojo los puntos obtenidos de la antena *rover* trabajando en modo SPS, sin comunicar con la *base*.

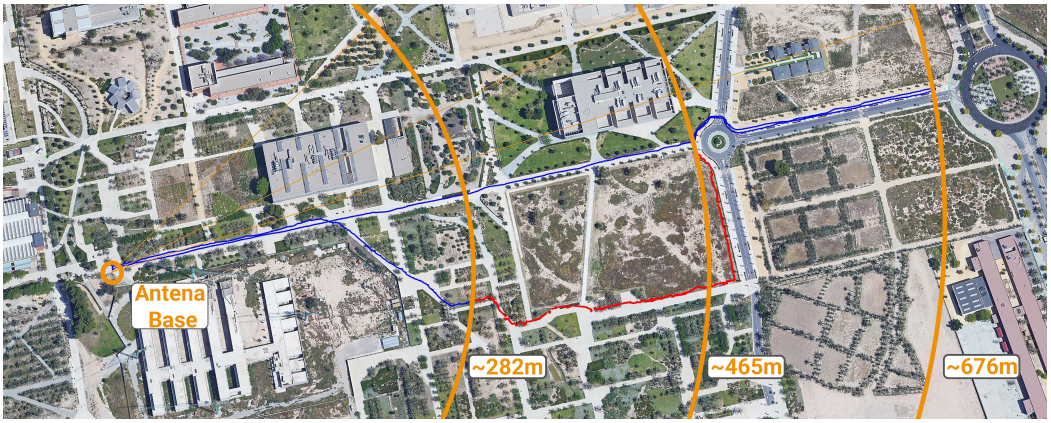


Figura 5.16: trayectoria de larga distancia con pérdida de conexión y reconexión entre las antenas *rover-base*

Fuente Imagen: Puntos GNSS interpolados sobre imagen

Como puede verse, en línea recta, ambas antenas son capaces de mantener la comunicación hasta aproximadamente 676 m, perdiendo la comunicación cuando sale de la línea recta a unos 465 m de distancia. Aun no teniendo línea directa de visión, la conexión se restablece entre ambas antenas a 282 m de distancia. Nótese que las tres distancias han sido tomadas de Google Maps, uniendo en línea recta la posición de la antena *base* con el punto en el que se produce el cambio en la conexión entre ambas antenas. Aunque en la figura 5.16 no se aprecia claramente, entre el modo SPS (en rojo) y el modo RTK (en azul), hay un espacio en que ambas antenas son capaces de comunicar y funcionar en modo DGPS. Debido a limitaciones de resolución de la imagen, las elipses de error se confunden también con los puntos, aunque si puede verse que estando en una zona despejada (sin edificios) el error se mantiene relativamente bajo.

En la figura 5.17 pueden verse más claramente las variaciones de covarianza. En modo RTK y DGPS la covarianza se mantendría igual que en el experimento visto anteriormente (figura 5.10) mientras que aumentaría notablemente cuando se produce la pérdida de comunicación UHF con la antena *base*. La covarianza en modo SPS es bastante sensible a la variación del número de satélites disponibles así como a la vegetación atravesada. Aun así se mantiene en los valores esperables. Puede apreciarse además en la línea que baja el paso por el modo DGPS antes de volver al modo RTK.

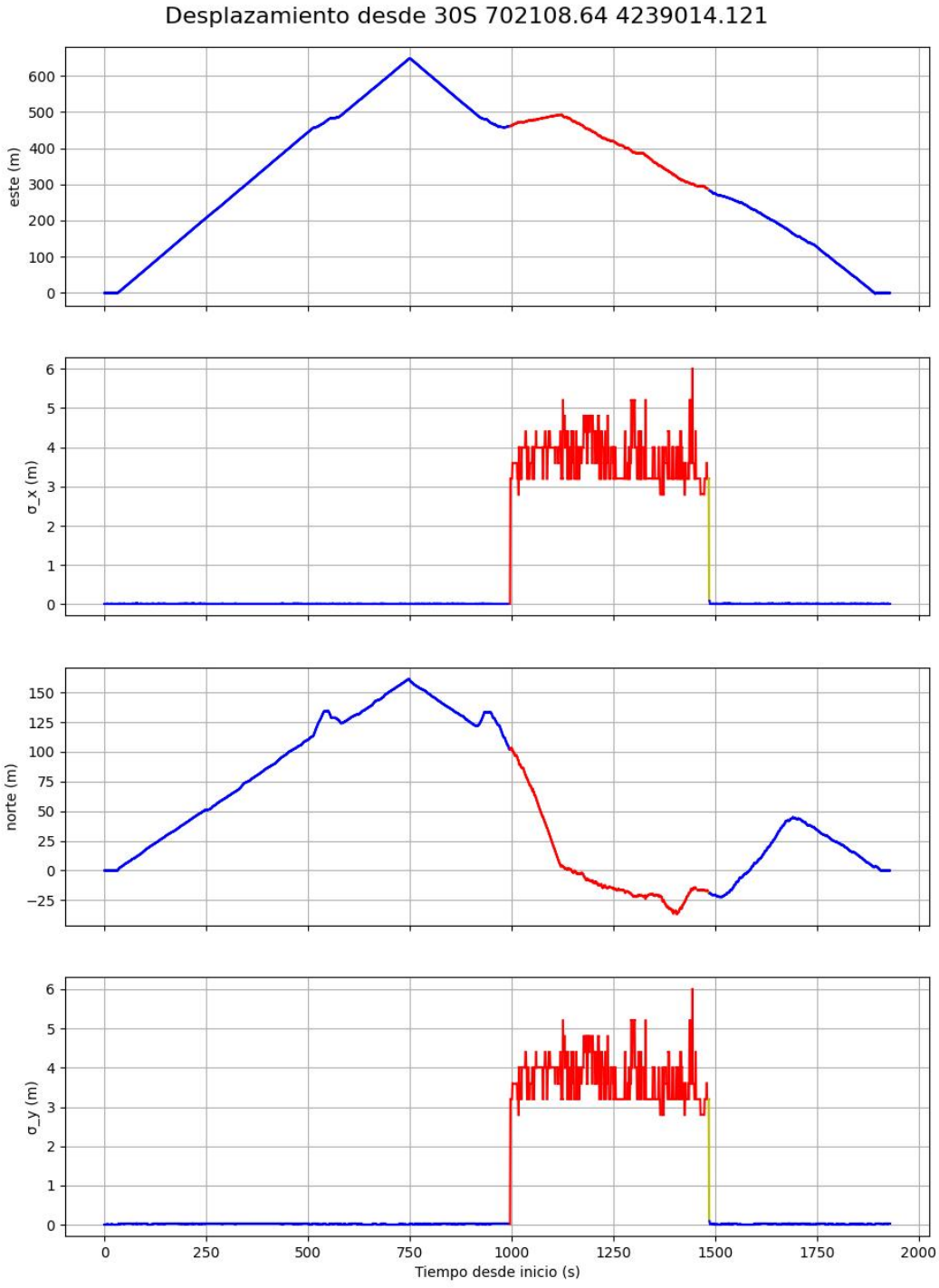


Figura 5.17: representación gráfica de los datos de desplazamiento X e Y en coordenadas UTM así como del error asociado a cada una representando la pérdida de conexión y paso a modo SPS

Fuente Imagen: Autor

5.2. Cartographer

Esta sección describe los experimentos llevados a cabo con el algoritmo de SLAM Cartographer así como los resultados y mapas obtenidos en cada caso. Se exponen experimentos en diferentes situaciones tanto simulados como con la plataforma real.

5.2.1. Procedimiento experimental

Aunque los entornos difieren, el procedimiento experimental es equivalente para todos ellos. Se ha trabajado de manera offline en *base* a datasets obtenidos previamente y almacenados a modo de rosbags.

Obtención de datos

Tanto en el caso simulado como con la plataforma real, será necesario ejecutar toda la capa funcional del robot (con la plataforma real este paquete se lanza por defecto cuando arranca el OBC).

En el caso simulado será necesario lanzar Gazebo con el escenario deseado, se irán publicando una serie de punto preestablecidos para comandar el robot a lo largo del mismo.

En el caso de la plataforma real, se utilizará el mando remoto para comandarla y moverla por el entorno.

Para guardar toda la información se ejecutará en una terminal el comando a continuación, que almacenará todos los topics que están siendo publicados en un fichero. El fichero se guardará con un *timetag* de la fecha y hora en que se empieza a grabar en la carpeta desde donde se ejecute el comando.

```
rosbag record -a
```

Una vez acabada la trayectoria deseada se interrumpirá el almacenamiento de información parando el proceso, por ejemplo con `ctrl + c`.

Procesado de la información

Antes de comenzar será necesario cargar las variables de entorno de ROS y del *Workspace* que se utilice en todas las terminales que se vayan a emplear. Para ello se invoca el *script* `devel/setup.sh` del *Workspace*.

En una terminal se lanzará el ROS Master. Nótese que aunque no es necesario re-lanzarlo de cero entre cada ejecución de cada experimento, si puede ser recomendable entre diferentes experimentos para limpiar parámetros y topics cargados.

```
roscore
```

El tiempo que utiliza ROS es el tiempo del sistema en que se ejecuta, sin embargo la información fue almacenada con el tiempo en que se ejecutó (todos los mensajes en ROS incluyen *timetag*). Para evitar discrepancia entre los tiempos de ejecución es necesario configurar el uso de tiempo simulado mediante el comando que puede verse a continuación, en vez del tiempo actual.

```
rosparam set use_sim_time true
```

Los mensajes almacenados en el rosbag incluyen referencias a la transformada que utilizan como referencia. Es necesario publicar la relación entre transformadas de nuevo para que pueda reconstruirse el árbol de transformadas y poder relacionar la información de los distintos tipos de mensajes. Las transformadas se cargan del modelo URDF con el comando siguiente:

```
roslaunch husky_manager tf_from_urdf.launch
```

Para visualizar el proceso de *mapping* puede lanzarse la interfaz rviz. Se ha creado un fichero de configuración de la interfaz (`rviz_map_cfg.rviz`) para cargar por defecto una serie de topics útiles para este caso de uso concreto. Puede ejecutarse con el comando siguiente. Nótese que rviz consume una cantidad relevante de recursos, en los experimentos más largos puede ser recomendable no utilizarlo. En caso de agotar la memoria el proceso de Cartographer podría ser cancelado por el sistema operativo a mitad de ejecución.

```
rosrun rviz rviz -d rviz_map_cfg.rviz
```

Se ha creado un fichero de configuración específico para la plataforma y condiciones de los experimentos. Incluye una serie de ficheros de configuración para el propio algoritmo así como el mapeado de los diferentes topics tal y como los empleará el software Cartographer. Se lanzará dicho módulo empleando el siguiente comando. En el fichero que aparece podrán modificarse los topics a los que se suscribe. Además hay un fichero `husky.lua` donde se configuran diferentes parámetros del Cartographer, entre ellos, la información empleada para realizar el proceso de SLAM. Ambos ficheros se irán actualizando a lo largo de los experimentos para adecuarse a las condiciones de cada situación.

```
roslaunch husky_cartographer_navigation cartographer_demo.
    launch
```

Finalmente se reproducen los datos almacenados en el rosbag, se ejecuta mediante el siguiente comando:

```
rosbag play path/rosbag_name.bag --clock
```

Para almacenar el mapa resultante puede utilizarse el servicio de ROS llamado `map_server`. El comando siguiente almacenará el último mapa publicado:

```
rosrun map_server map_saver -f path/map_name
```

Parametrización del módulo Cartographer

Como se ha explicado el código de SLAM incluye una serie de ficheros de configuración con una serie de parámetros que se podrán ajustar para optimizar los resultados. Se presenta a continuación la configuración obtenida durante las pruebas, y que será la utilizada en los experimentos presentados a continuación. Pueden verse en las tablas 5.1 y 5.2 los valores dados a los parámetros explicados en la sección 3.1.

Tabla 5.1: configuración genérica del Cartographer

Nombre del parámetro	valor
tracking_frame	os_imu
published_frame	odom
use_odometry	true
use_nav_sat	true
num_point_clouds	1

Tabla 5.2: configuración de algunos parámetros de los submódulos del Cartographer

Nombre del parámetro	valor
TRAJECTORY_BUILDER_2D.use_imu_data	true
TRAJECTORY_BUILDER_nD.imu_gravity_time_constant	2
TRAJECTORY_BUILDER_2D.min_z	0.13
TRAJECTORY_BUILDER_2D.max_z	1.5
TRAJECTORY_BUILDER_2D.min_range	0.3
TRAJECTORY_BUILDER_2D.max_range	20.
TRAJECTORY_BUILDER_2D.ceres_scan_matcher.translation_weight	10
TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_weight	15
TRAJECTORY_BUILDER_2D.submaps.num_range_data	100

5.2.2. Datos de referencia Ouster

Se ha recurrido a información provista por el fabricante del LIDAR haciendo pruebas con el propio Cartographer. Aunque las pruebas no se han replicado, si se

puede tomar como referencia los resultados obtenidos así como la información utilizada (frecuencia y tamaño de la información). Una descripción de los experimentos llevados a cabo por el fabricante puede encontrarse en [24].

Nótese que para estas pruebas el fabricante ha usado el modelo OS1-64, con menor número de líneas verticales. La resolución configurada del LIDAR para estos experimentos es de 64 líneas (vertical) x 1024 puntos en cada una (65536 puntos en total).

Las pruebas se han realizado únicamente con el sensor LIDAR disponiendo para el proceso de mapeado solamente información de PointCloud y de la IMU integrada en el sensor. Puede verse la frecuencia a la que dichos datos son producidos en la tabla 5.3.

Tabla 5.3: frecuencia media en topics relevantes en el dataset provisto por Ouster

Topic	**Frecuencia media (Hz)**	**Min (s)**	**Max (s)**	**Desviación Estándar (s)**
/os1_cloud_node/points	10.004	0.091	0.115	0.00494
/os1_cloud_node/imu	99.931	0.000	0.024	0.00276

De manera orientativa, en la figura 5.18 puede verse el resultado del mapa obtenido por el fabricante en dichos experimentos así como la trayectoria realizada, representada en color azul.

El fichero que almacena dicha información está nombrado como `office_demo_9_25_19.bag`, y puede obtenerse con el comando siguiente:

```
curl -O https://data.ouster.io/downloads/office_demo_9_25_19.bag
```

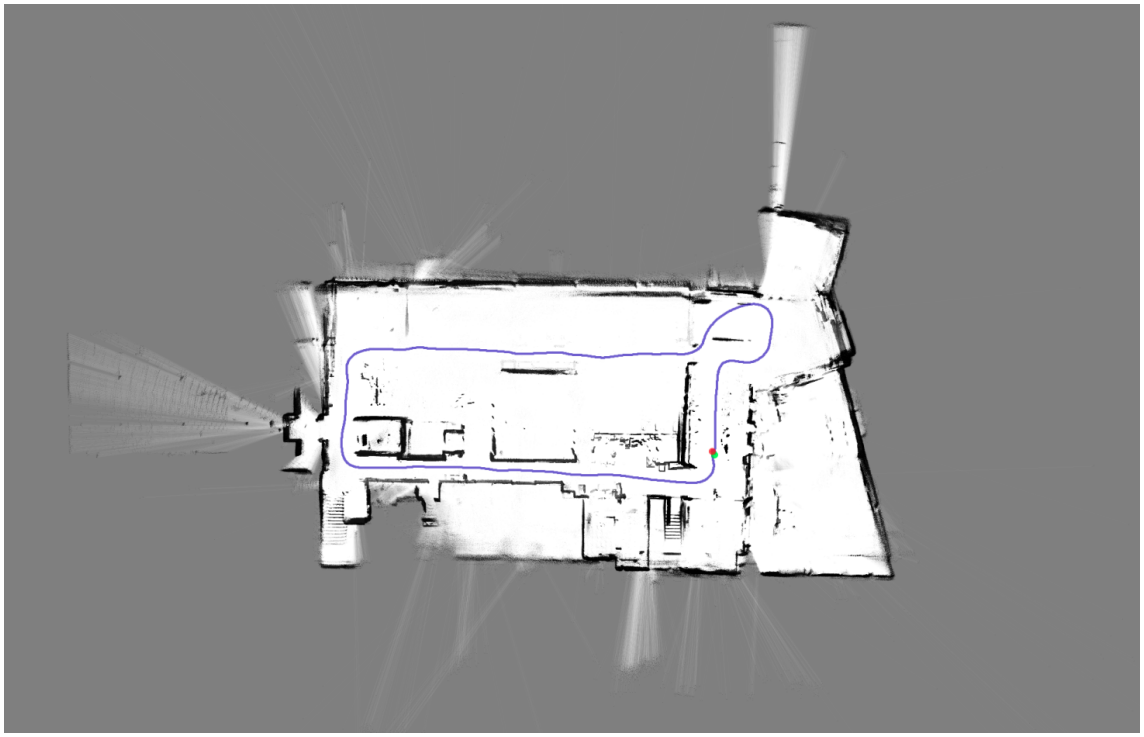


Figura 5.18: mapa de referencia provisto por Ouster generado haciendo uso de la información de PointClouds e IMU

Fuente Imagen: Mapa extraído de [24]

5.2.3. Entornos simulados

Para poner a prueba el Cartographer se ha utilizado en primera instancia, y mientras se terminaba de poner el robot a punto, un entorno simulado, tanto de exterior como en el interior de un edificio.

Se describen a continuación los diferentes entornos así como los experimentos realizados en cada uno de ellos.

Entorno de Interior

Se ha recurrido a un entorno de tipo oficina para los primeros experimentos. Sobre este entorno se ha ejecutado la plataforma Husky emulando los sensores que integrados en la plataforma real. Una imagen de planta del entorno utilizado puede verse en la figura 5.19. El experimento consistirá en realizar dos vueltas circulando por los pasillos para acabar de vuelta en el punto de inicio.

Nótese que los sensores son todos simulados. En este caso cada PointCloud incluye 56364 puntos, aunque no simula la distribución vertical y horizontal del sensor real. La frecuencia para cada tipo de dato disponible puede verse en la tabla 5.4, conviene tener en cuenta:

- La información de GNSS en el topic de /navsat/fix no será utilizada en este caso al tratarse de un entorno de interior (nótese que aunque se publica

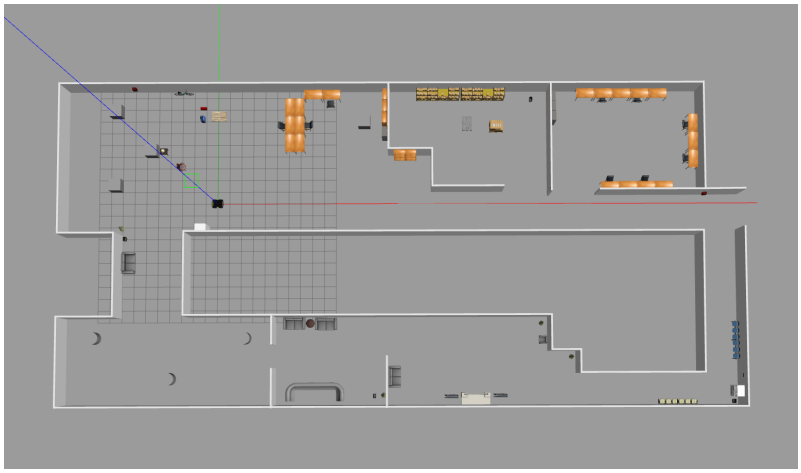


Figura 5.19: entorno de simulación interior

Fuente Imagen: Captura del entorno simulado

información, esta información solo contiene un código de error).

- La información de odometría contenida en el topic /odometry/filtered proviene de un filtro de Kalman extendido que combina información de la odometría de las ruedas como de la IMU. No será utilizada al no estar disponible en las pruebas con la plataforma real.

- La información de odometría contenida en el topic /husky_velocity_controller/odom se corresponde con la odometría de las ruedas.

Tabla 5.4: frecuencia media en topics relevantes en las pruebas simuladas en interior

Topic	**Frecuencia media (Hz)**	**Min (s)**	**Max (s)**	**Desviación Estándar (s)**
/imu/data	50.000	0.016	0.024	0.00071
/navsat/fix	10.000	0.097	0.103	0.00058
/os1/pointCloud	7.000	0.132	0.154	0.00209
/husky_velocity_controller/odom	49.988	0.000	0.049	0.02149
/odometry/filtered	50.011	0.000	0.049	0.02141

Para comprobar la influencia que tiene el uso de cada uno de los sensores se han hecho diferentes pruebas variando la información utilizada. La figura 5.20 puede verse el mapa resultante utilizando toda la información disponible: PointCloud, IMU y odometría. El mapa resultante presenta muy buenos resultados aun habiendo una pequeña discrepancia en el Close-loop de ambas vueltas, que se ha resaltado en la figura.

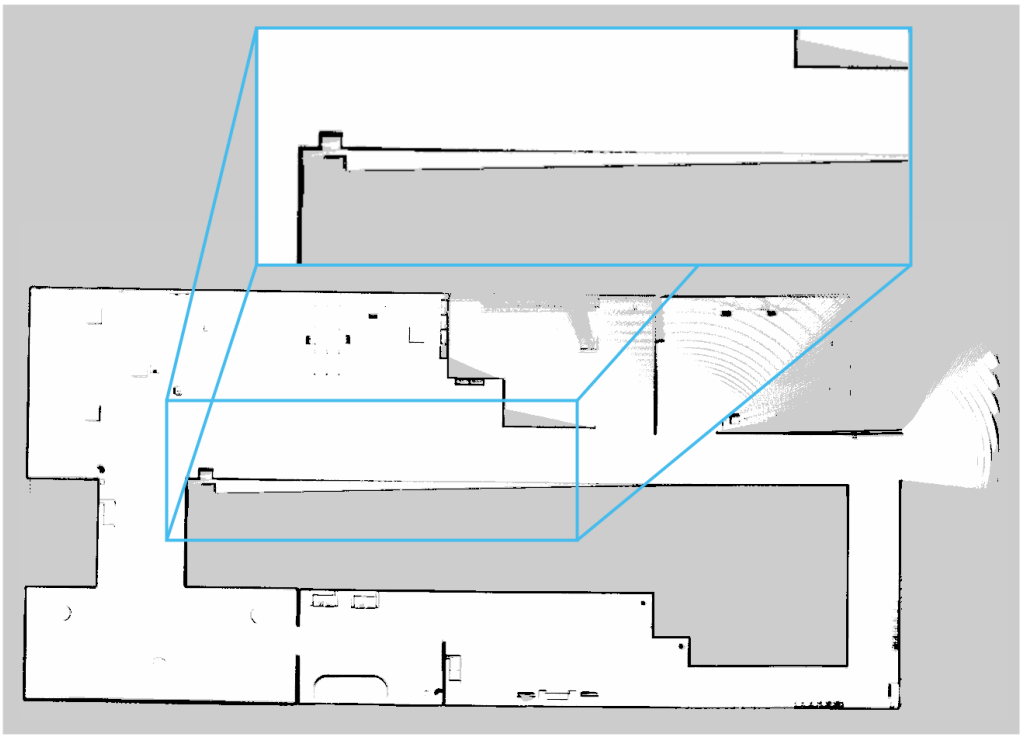


Figura 5.20: mapa de interior simulado generado haciendo uso de toda la información disponible

Fuente Imagen: Mapa guardado, resultado del experimento

Figura 5.21: mapa de interior simulado generado haciendo uso de la información de PointClouds y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

La figura 5.21 contiene el mapa resultante del experimentos utilizando la información deaquiPluralPointCloud y odometría. Puede verse como la pérdida de precisión en la estimación de los giros realizados por la plataforma impiden que el mapa resultante converja correctamente quedando dos versiones superpuestas. Como se ha dicho el experimento consiste en dar dos vueltas a la estancia, la divergencia es tal que el Close-loop no es capaz de cuadrarlas quedando dos versiones del mismo entorno, con giro y desplazamiento con respecto a la otra.

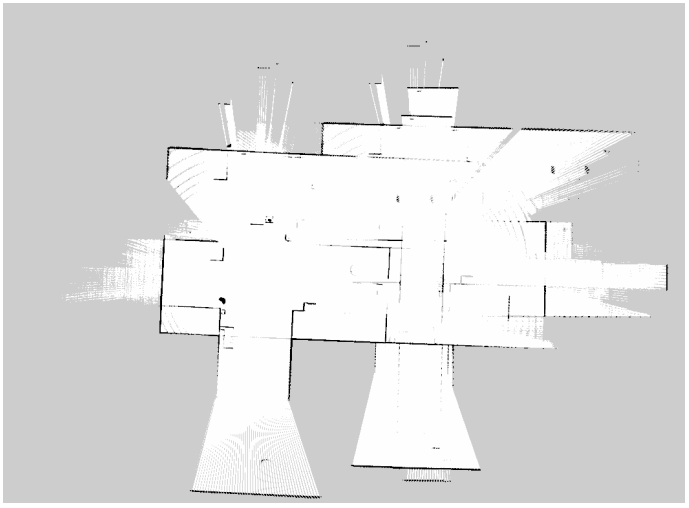


Figura 5.22: mapa de interior simulado generado haciendo uso de la información de PointClouds e IMU

Fuente Imagen: Mapa guardado, resultado del experimento

Finalmente, en la figura 5.22 puede verse el resultado de utilizar únicamente la información de PointClouds. En general el algoritmo no ha sido capaz de estimar el desplazamiento y los giros realizados por el robot, quedando todo el mapa superpuesto sobre si mismo.

La información utilizada para esta serie de experimentos puede encontrarse en el fichero nombrado como `2022_07_28/sim_interior.bag` .

Entorno en Exterior

Para poner a prueba el algoritmo de SLAM se ha utilizado un entorno simulado de exteriores, donde se incluyen una serie de vehículos así como mobiliario urbano típico. El robot simulado dará una vuelta alrededor de un pequeño parque, tal y como puede verse en la figura 5.23. En el experimento se dará una vuelta alrededor de la zona ajardinada para acabar en el mismo punto del inicio.

En este caso toda la información de los sensores es simulada, obteniendo para cada PointCloud un total de 40828 puntos. La frecuencia de cada tipo de dato empleado puede verse en la tabla 5.5, en este caso conviene tener en cuenta los puntos descritos en el caso anterior, salvo el referente a la información de GNSS, que en

Figura 5.23: entorno de simulación exterior

Fuente Imagen: Captura del entorno simulado

este caso si estará disponible.

Tabla 5.5: frecuencia media en topics relevantes en las pruebas simuladas en exterior

Topic	**Frecuencia media (Hz)**	**Min (s)**	**Max (s)**	**Desviación Estándar (s)**
/imu/data	50.000	0.017	0.023	0.00046
/navsat/fix	9.997	0.099	0.101	0.00030
/os1/pointCloud	7.020	0.115	0.157	0.00647
/husky_velocity_controller/odom	50.000	0.000	0.049	0.02154
/odometry/filtered	50.123	0.000	0.049	0.02149

El caso más completo se puede ver en la figura 5.24, en el cual se ha utilizado toda la información disponible: PointClouds, GNSS, IMU y odometría de las ruedas. Puede verse que se ha obtenido un mapa bastante claro donde pueden reconocerse los vehículos así como mobiliario urbano. Nótese que se ha configurado el Cartographer para considerar sólo puntos que supongan un obstáculo (con una altura superior a 13 cm), esto implica que parte del bordillo que rodea a la zona ajardinada (ver figura 5.23) no se ve reflejados en el mapa. Hay que tener en cuenta que el algoritmo discrimina la traversabilidad de las diferentes zonas por altura, hecho que en exteriores puede ser poco apropiado. Aun así puede considerarse un mapa muy correcto.

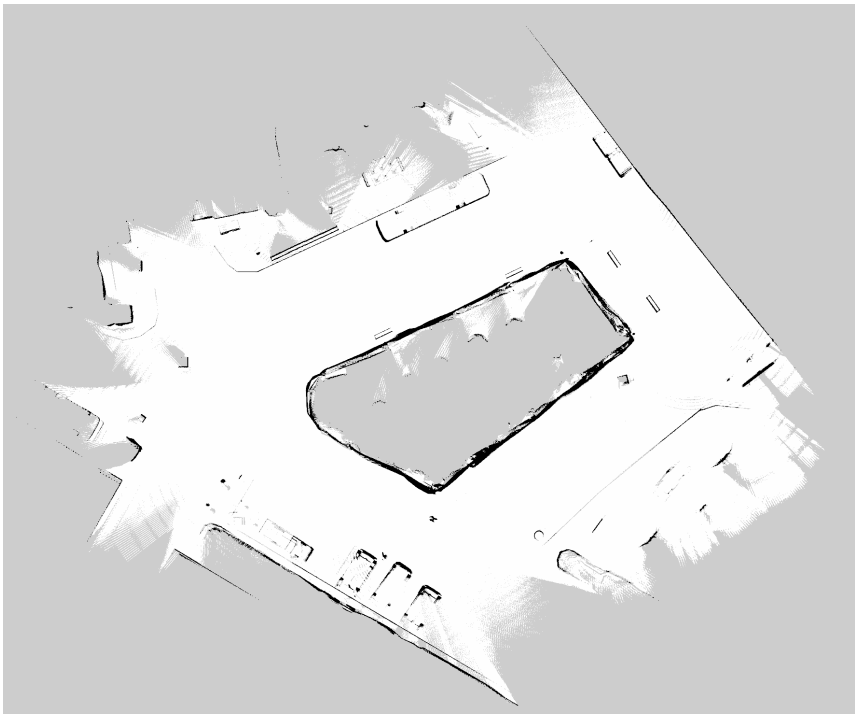


Figura 5.24: mapa del exterior simulado generado haciendo uso de toda la información disponible

Fuente Imagen: Mapa guardado, resultado del experimento

Como puede verse en la figura 5.25 eliminar el GNSS no redunda en un resultado muy diferente del anterior. Cabe destacar que el escenario y el experimento no implica una trayectoria muy larga donde se pueda acumular mayor error, además, tal y como puede verse en el mapa resultante, las diferentes zonas del mapa son bastante diferentes como para que el *scan matching* pueda converger a una solución razonable.

Aunque la estimación de los giros sin IMU es peor, el resultado visto en la figura 5.26 sigue siendo bastante correcto. Como puede verse el mapa resultante tiene zonas muy características que podrían ayudar a que converja a una solución adecuada sin necesitar datos de la IMU. Nótese que esto es debido a una característica del experimento: no es una trayectoria muy larga y que finaliza en el punto de inicio (close loop); y del escenario: no tiene zonas que sean semejantes y que puedan causar problemas de convergencia.

La figura 5.27 es el mapa resultante de utilizar únicamente PointClouds e IMU. En este caso se ve que la información utilizada no es suficiente para que el proceso de *scan matching* sea capaz de estimar el desplazamiento y el mapa resultante contiene todos los *submapas* superpuestos con diferente orientación sobre un mismo punto.

Los datos empleados en este caso de uso están almacenados en el fichero nombrado:
`2022_07_28/2022-07-28-13-00-02.bag` .

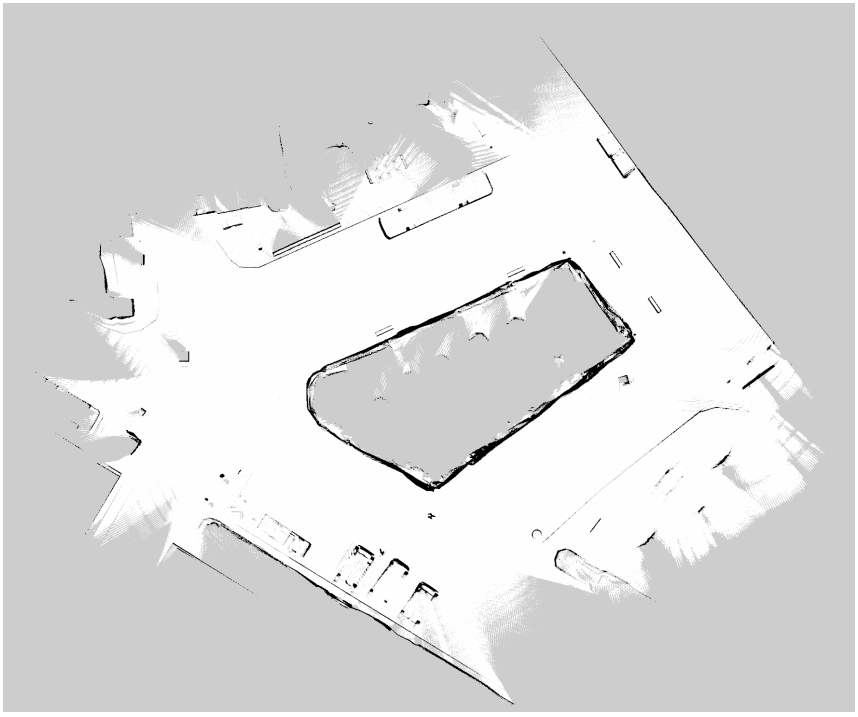


Figura 5.25: mapa del exterior simulado generado haciendo uso de la información de PointClouds, IMU y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

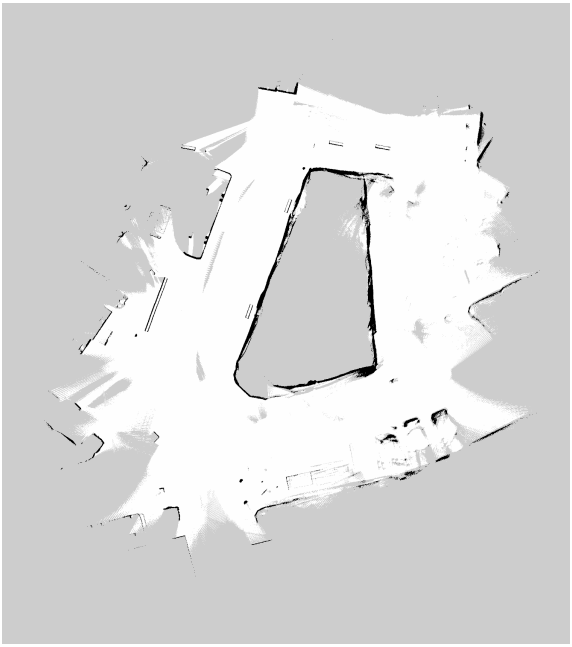


Figura 5.26: mapa del exterior simulado generado haciendo uso de la información de PointClouds y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

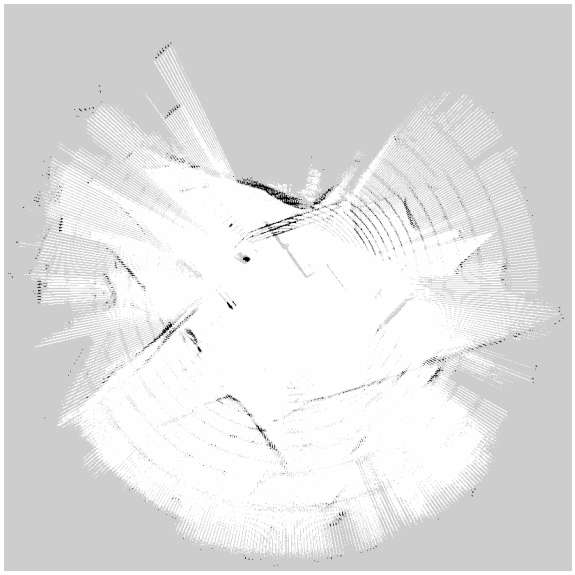


Figura 5.27: mapa del exterior simulado generado haciendo uso de la información de PointClouds e IMU

Fuente Imagen: Mapa guardado, resultado del experimento

5.2.4. Experimentos realizados con la plataforma robótica real

Se han llevado a cabo una serie de pruebas con la plataforma real en tres escenarios diferentes aumentando la complejidad en cada caso. En todos los casos experimentales se ha iniciado el experimento en el laboratorio, al lado de la estación de carga del robot, y se ha finalizado en el mismo punto. La antena *base* se ha ubicado en el mismo punto que puede verse en la figura 5.9, aunque dicha posición no saldrá en los mapas obtenidos.

La resolución del LIDAR se mantiene constante a lo largo de los tres experimentos con una resolución de cada PointCloud de 128 líneas verticales con 512 puntos (65536 puntos en total).

La IMU no quedó correctamente calibrada y los datos obtenidos de la misma no son correctos ni se han podido corregir, por lo que no se ha podido utilizar los datos en dicho topic ni en el correspondiente a la odometría filtrada en la capa funcional de la plataforma robótica mediante el filtro de Kalman extendido. EN los diferentes experimentos se habla de datos de la IMU refiriéndose siempre al sensor integrado junto con el LIDAR, nótese que al no estar calibrado no ofrece datos de orientación, si no solo de velocidad angular y la aceleración lineal.

Interior: Laboratorio y pasillo

Se ha realizado una prueba breve para poner a prueba los diferentes sensores así como las transformadas entre frames y sincronización del sistema. Se ha aprovechado

a poner a prueba el algoritmo en este entorno, las conclusiones podrán extrapolarse a las siguientes pruebas. Pueden consultarse las frecuencias a las que se publican los datos relevantes en la tabla 5.6.

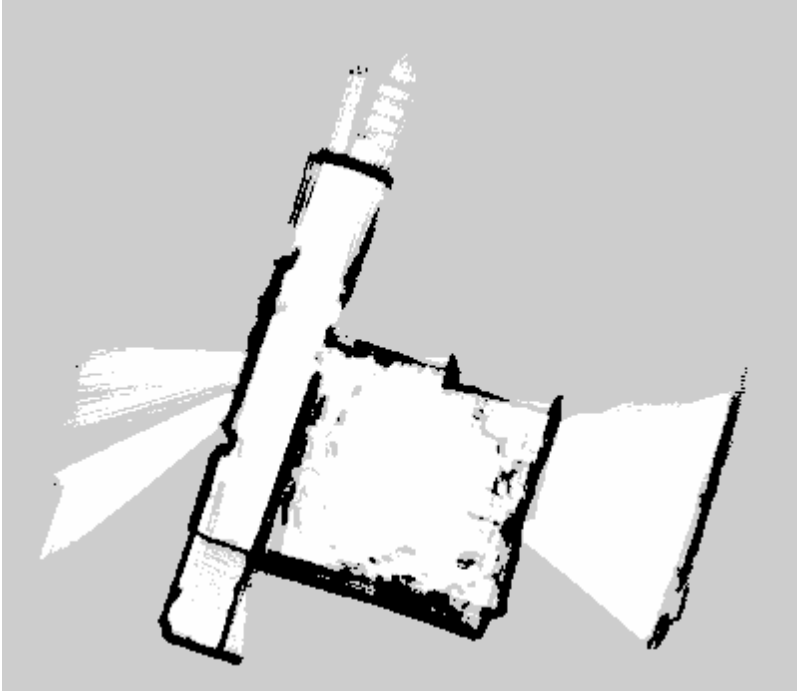


Figura 5.28: mapa de interior (laboratorio y pasillo) con la plataforma robótica real generado haciendo uso de la información de PointClouds, odometría e IMU del sensor LIDAR

Fuente Imagen: Mapa guardado, resultado del experimento

Puede verse en la figura 5.28 el resultado del mapa obtenido utilizando toda la información de los sensores disponibles: PointClouds, odometría e IMU. Se ha podido comprobar como los datos de la IMU no resultan del todo estables haciendo que la posición del frame del PointCloud con respecto al mapa sufra cierta oscilación haciendo que puntos que no deberían ser obstáculos puedan contar como tal. Puede verse como algunos bordes no están correctamente definidos así como verse ciertos obstáculos que no existen y no han sido filtrados. Debería corregirse esta situación filtrando la estimación de la dirección de la gravedad para evitar el ruido producido por el sensor IMU.

En la figura 5.29 puede verse el resultado del mapa producido sin utilizar los datos de la IMU. Aunque los bordes de objetos y entorno están mejor definidos, debido a giros muy pronunciados (en general giros sobre el propio eje del robot o *point turn*) provocan mayor error en la estimación de giro mediante odometría.

Fichero con los datos `2022_07_27/2022-07-27-14-29-10.bag` .

Tabla 5.6: frecuencia media en topics relevantes en las pruebas con el robot en interior. Se ha remarcado en rojo los datos erróneos; en azul se ven datos que no se podrán utilizar en este experimento.

Topic	**Frecuencia media (Hz)**	**Min (s)**	**Max (s)**	**Desviación Estándar (s)**
/imu/data	19.25	0.041	0.062	0.00342
/fix	1.000	0.999	1.001	0.00050
/os_cloud_node/points	10.01	0.079	0.116	0.00716
/os_cloud_node/imu	99.997	0.000	0.034	0.00602
/husky_velocity_controller/odom	10.00	0.094	0.113	0.00342
/odometry/filtered	50.102	0.000	0.051	0.02138

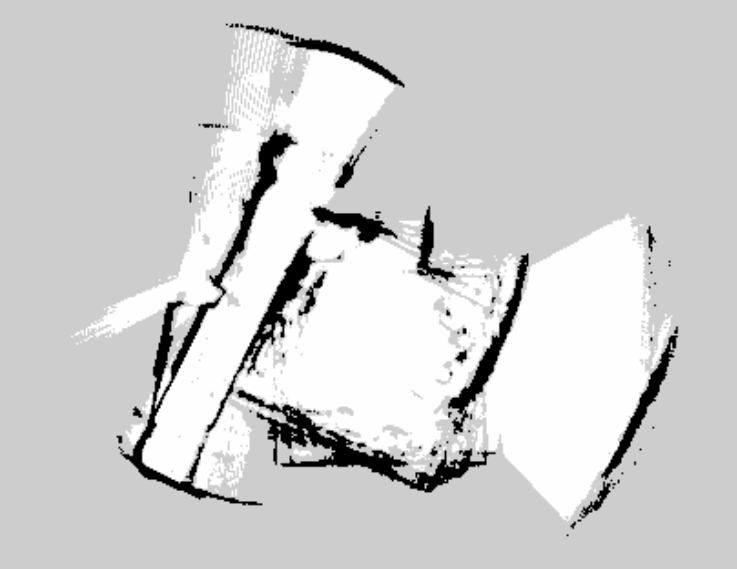


Figura 5.29: mapa de interior (laboratorio y pasillo) con la plataforma robótica real generado haciendo uso de la información de PointClouds y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

Interior: Edificio Innova

Para poner a prueba el algoritmo en entornos interiores se ha realizado un experimento más largo recorriendo el interior del edificio Innova, concretamente la primera planta, los tres pasillos sin llegar a la zona de la entrada. El experimento comienza y finaliza en el laboratorio, junto a la estación de carga. Desde ahí se avanza hasta la puerta trasera del edificio y se recorre ese primer pasillo; seguidamente se va hasta las puertas traseras previas al desnivel y se entra al tercer pasillo. Tras recorrerlo se vuelve al segundo pasillo y finalmente se vuelve al laboratorio.

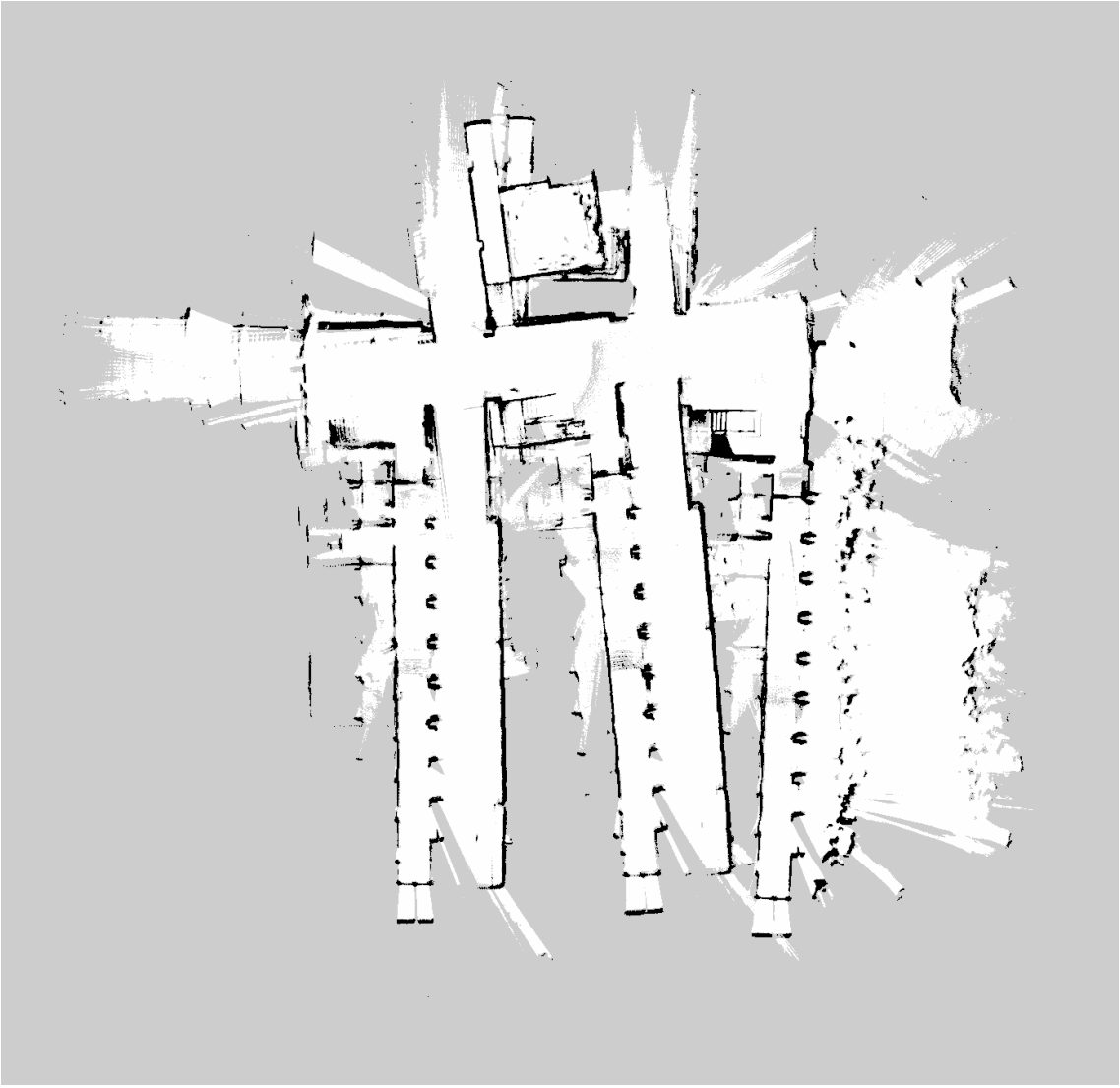


Figura 5.30: mapa de interior (edificio Innova) con la plataforma robótica real generado haciendo uso de la información de PointClouds, IMU y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

En la figura 5.30 puede verse el resultado obtenido utilizando toda la información disponible. Nótese que el error descrito antes que provoca la inestabilidad entre PointCloud y el frame del mapa sigue presente. Además puede verse como los tres

Tabla 5.7: frecuencia media en topics relevantes en las pruebas con el robot en interior en una vuelta mayor. Se ha remarcado en rojo los datos erróneos; en azul se ven datos que no se podrán utilizar en este experimento.

Topic	**Frecuencia media (Hz)**	**Min (s)**	**Max (s)**	**Desviación Estándar (s)**
/imu/data	19.609	0.045	0.061	0.00220
/fix	1.000	0.999	1.001	0.00032
/os_cloud_node/points	10.005	0.079	0.116	0.00753
/os_cloud_node/imu	99.982	0.000	0.033	0.00564
/husky_velocity_controller/odom	99.982	0.000	0.033	0.00564
/odometry/filtered	99.982	0.000	0.033	0.00564

pasillos son muy similares, cosa que complica la convergencia del mapa global. Hay que tener en cuenta nuevamente que los datos de la IMU utilizados no incluyen información absoluta del giro de la plataforma. Puede verse como parte del segundo pasillo se mezcla con el primero.

En la figura 5.31 puede verse los resultados eliminando la información de la IMU. Los errores descritos anteriormente se agravan empeorando la estimación de los giros de la plataforma.

Como puede verse por ahora el algoritmo es capaz de ofrecer buenos resultados, pero estos son muy dependientes de una buena estimación inicial tanto de la traslación como la rotación realizada. Como se ha visto tanto el proceso de scan-matching como la posterior integración de los mapas depende de un proceso de optimización local, será imprescindible contar con una buena estimación inicial para que pueda converger a la solución esperada.

Los datos empleados en este test están almacenados en un fichero con nombre: `2022_07_29/2022-07-29-13-17-50.bag` .

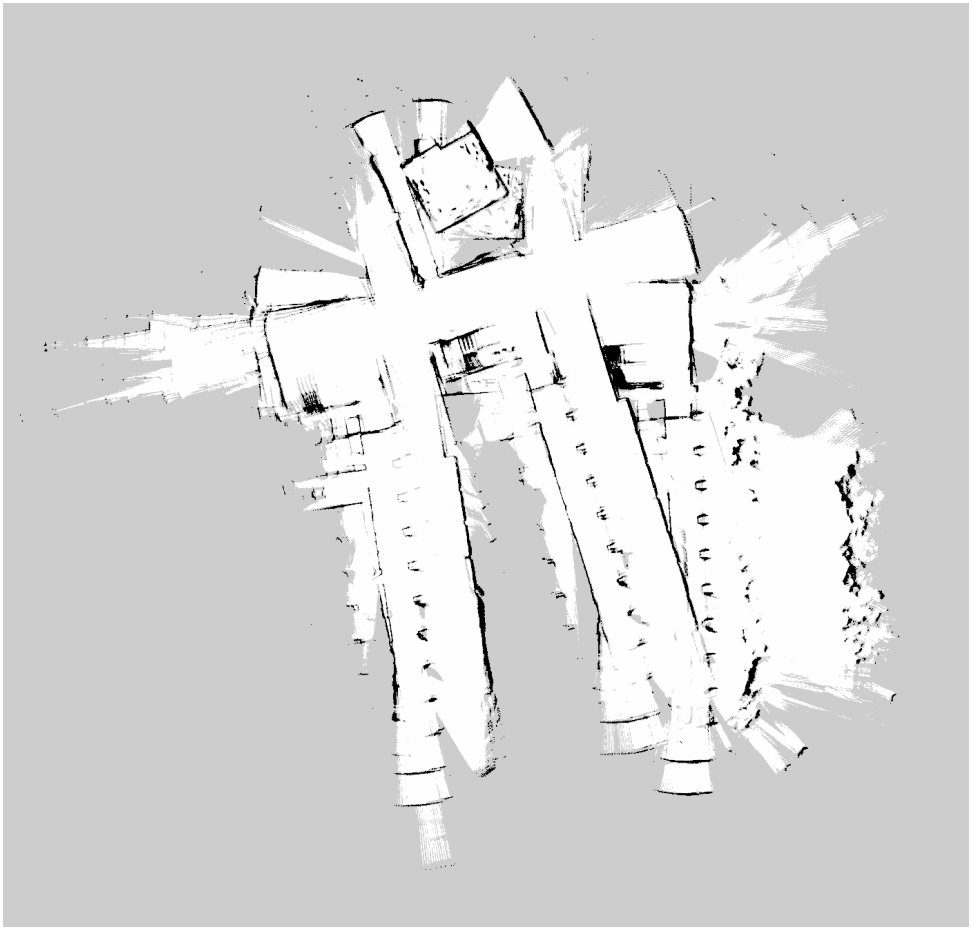


Figura 5.31: mapa de interior (edificio Innova) con la plataforma robótica real generado haciendo uso de la información de PointClouds y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

Interior y exterior: Laboratorio, edificio Innova y Rectorado

Para poner a prueba el algoritmo combinando entornos interiores y exteriores se ha llevado a cabo un experimento de mayor longitud. Nuevamente el experimento comienza y finaliza en el laboratorio, saliendo por la puerta principal se rodea el rectorado en el sentido opuesto a las agujas del reloj para volver de vuelta al edificio.

Figura 5.32: mapa de exterior e interior con la plataforma robótica real generado haciendo uso de la información de PointClouds, odometría, IMU y GNSS

Fuente Imagen: Mapa guardado, resultado del experimento

En la figura 5.32 puede verse el resultado obtenido utilizando la totalidad de información de sensores disponibles. Se ha ampliado la parte correspondiente al interior del edificio Innova, pues es donde se verán reflejada la falta de correspondencia. A lo largo de la trayectoria exterior puede verse el mismo efecto descrito previamente, al filtrar el PointCloud en base solo a la altura hay muchas zonas que no se ven reflejadas en el mapa, por ejemplo, no pueden reconocerse correctamente los caminos por los que pasa el robot. La información de posición obtenida de la antena GNSS permite que la convergencia tanto en el proceso de scan matching como en el proceso de mapeado global sea muy preciso dando unos resultados bastante buenos

Tabla 5.8: frecuencia media en topics relevantes en las pruebas con el robot combinando un recorrido con interior y exterior. Se ha remarcado en rojo los datos erróneos; en azul se ven datos que no se podrán utilizar en este experimento.

Topic	**Frecuencia media (Hz)**	**Min (s)**	**Max (s)**	**Desviación Estándar (s)**
/imu/data	19.260	0.043	0.061	0.00303
/fix	1.000	0.999	1.001	0.00064
/os_cloud_node/points	10.007	0.074	0.123	0.00857
/os_cloud_node/imu	100.172	0.000	0.037	0.00537
/husky_velocity_controller/odom	10.005	0.083	0.126	0.00455
/odometry/filtered	50.151	0.000	0.048	0.02149

en exteriores, con los problemas de orientación descritos previamente en interiores. Nótese que el interior se recorre dos veces, pequeñas discrepancias en posición u orientación entre la salida y entrada del edificio pueden provocar desfases como se ven en el mapa. En este caso dicho error se ve minimizado gracias a la posición GNSS. Como se vio en la prueba descrita en la sección 5.1.4, los datos oscilarán entr RTK, DGPS, SPS y algunas zonas quedarán sin datos. De esta manera, cuando haya datos permitirá mejorar el scan-matching en la formación de cada uno de los submapas así como la posterior reconstrucción del mapa global.

Aun retirando la información de la antena GNSS los resultados se mantienen bastante correctos, tal y como puede verse en la figura 5.33. Se aprecia mayor discrepancia en el proceso de mapeado del interior del edificio, con cierto desfase, principalmente en posición. Se asume que este desfase es debido a discrepancias entre el punto de entrada y de salida basado en el error acumulado a lo largo de toda la trayectoria.

El efecto del empeoramiento de la estimación de los giros puede verse en la figura 5.34, obtenida obviando la información de la IMU. Puede verse como el efecto se amortigua en *base* a giros mucho más amplios y graduales. Puede verse el error acumulado en la falta de correspondencia entre ambos mapas superpuestos del interior, uno correspondiente al inicio del experimento y el otro al final.

Los datos empleados para dicho experimento pueden encontrarse en el fichero: `2022_07_29/2022-07-29-09-57-28.bag`

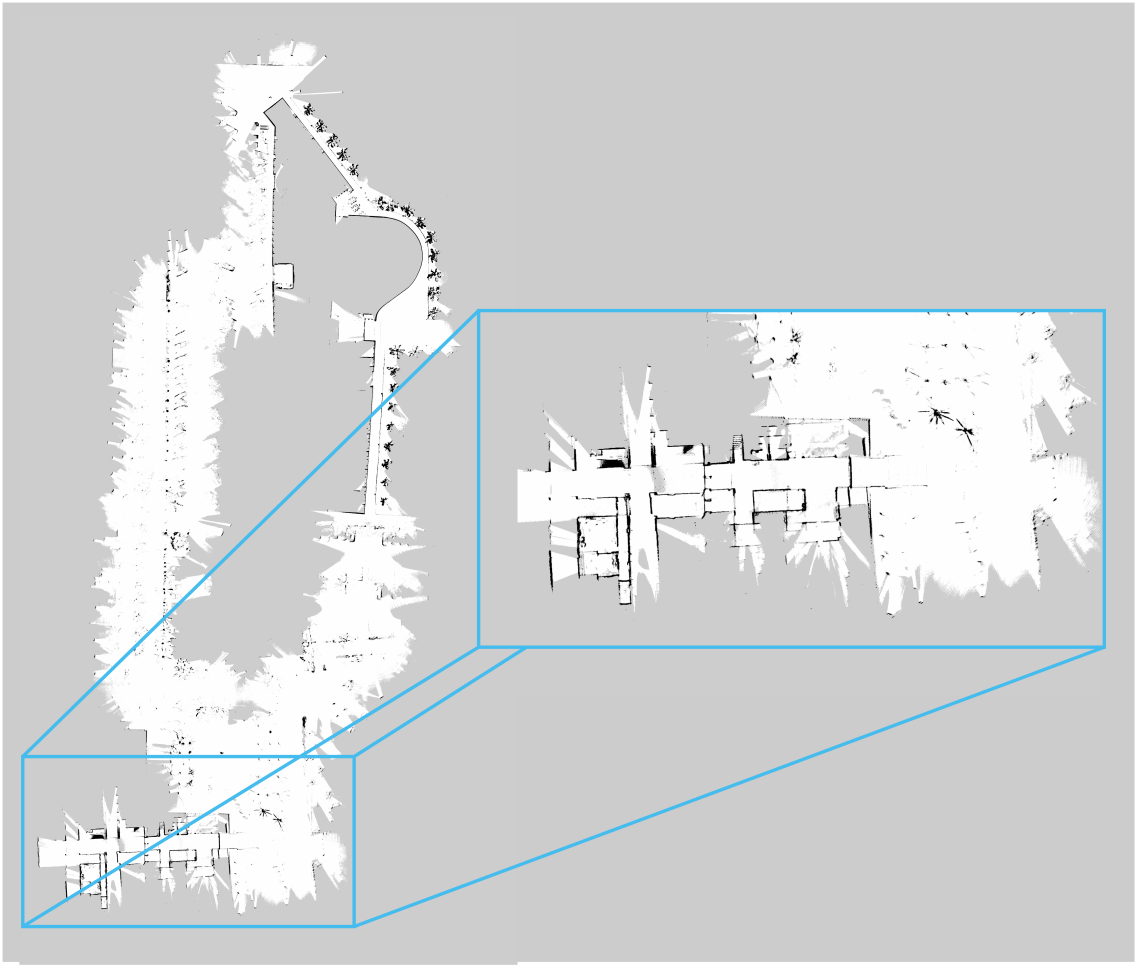


Figura 5.33: mapa de exterior e interior con la plataforma robótica real generado haciendo uso de la información de PointClouds, odometría e IMU

Fuente Imagen: Mapa guardado, resultado del experimento

Figura 5.34: mapa de exterior e interior con la plataforma robótica real generado haciendo uso de la información de PointClouds y odometría

Fuente Imagen: Mapa guardado, resultado del experimento

Capítulo 6

Conclusiones

Una vez finalizado el proyecto es conveniente revisar los objetivos para constatar su cumplimiento así como extraer las conclusiones posibles de los experimentos realizados. Se presentan a continuación dichas conclusiones así como posibles desarrollos a corto y largo plazo.

6.1. Conclusión

De los objetivos planteados en el capítulo inicial podemos concluir que:

- La plataforma robótica Husky se encuentra operativa, se ha configurado una nueva distribución de red y se han generado una serie de procedimientos de arranque para los módulos implicados en su funcionamiento.

- El sensor OS1 de Ouster ha sido integrado en la plataforma, se ha desarrollado un procedimiento automático para la sincronización del reloj interno tomando como maestro el OBC del robot. Los datos del sensor se publican en los topics correspondientes, tanto la IMU integrada como los PointClouds generados por el LIDAR. Un nuevo frame ha sido definido en la configuración de la plataforma con su transformada correspondiente, de manera que los datos publicados desde este sensor pueden relacionarse con los del resto del robot.

- Se ha integrado la IMU en la plataforma robótica y se ha seguido el procedimiento de calibración. Los datos del sensor se introducen en el ecosistema ROS mediante el topic correspondiente. Se ha añadido un nuevo frame con la transformada correspondiente a los frames existentes para poder combinar la información de este sensor con el resto de mensajes.

- Las antenas TS100 han sido configuradas y validadas en diferentes situaciones. La antena *rover* ha quedado integrada en la plataforma robótica publicándose los datos de la misma en el formato correspondiente en el ecosistema ROS. En este caso también se ha creado un nuevo frame para poder transformar la información publicada desde la antena a otras referencias del robot o el entorno.

- Se ha trabajado con el algoritmo Cartographer investigando su funcionamiento e integrándolo en diferentes entornos (simulado y real) para comprobar su correcto funcionamiento en entornos mixtos tanto de interior como de exterior. Se ha optimizado su parametrización para mejorar el rendimiento y los resultados obtenidos.

- Finalmente no se ha llegado a integrar este desarrollo dentro del *navigation stack* de ROS.

Como puede verse la mayoría de objetivos planteados se han cumplido. El último de los objetivos quedará pendiente de realizar en el futuro.

Desde el punto de vista experimental también pueden obtenerse algunas conclusiones interesantes del trabajo realizado:

- El cartographer está pensado para entornos sin variaciones de altitud, que pueden representarse en mapas 2D. El algoritmo realiza un filtrado de los datos de distancia en base a la altura que puede no ser tan apropiado en entornos de exterior más desestructurados. Quedan pruebas pendientes para validar si este algoritmo puede adaptarse correctamente a los entornos en los que se plantea su uso, ajustando mejor la parametrización empleada y probando diferentes versiones que puedan gestionar cambios en la altitud.

- Es imprescindible contar con unos buenos datos de partida. Cualquier error o ruido afectará a los algoritmos que se empleen. Aun con algoritmos muy robustos es necesario ajustar lo máximo posible las entradas. Para ello es imprescindible una correcta calibración de los sensores empleados. Todo el sistema debe trabajar sincronizado y las transformaciones entre los sistemas de referencia de los diferentes datos deben ser lo más precisas posibles. Dentro de lo posible será preferible aumentar la frecuencia a la que se produce la información.

- El LIDAR empleado ofrece una gran resolución en vertical, según qué tipo de algoritmo podría mejorar aumentando el número de puntos en el plano horizontal. Habría que buscar una buena solución de compromiso entre el aumento de memoria que implica el mayor número de puntos y frecuencia frente a la mejora que podrían conllevar. Una mayor frecuencia en la producción de PointClouds tiene un gran impacto en la convergencia del mapa generado, reduciendo posibles errores o faltas de información de otros sensores.

- En general cuanta más información haya disponible, mejores resultados se obtendrán. Aprovechar ambos sensores IMU puede resultar ventajoso.

- Contar con un LIDAR 3D capaz de proporcionar puntos al rededor de los 360° permite tener un gran solapamiento entre los submapas generados. Tal y como ha sido parametrizado para las pruebas el algoritmo Cartographer acumula 100 *scans* en cada submapa. A la frecuencia que se toman los *scans* implica acumular información durante 10 segundos, aún a la velocidad máxima de 1 m/s significa que se generará un submapa cada 10 m. Con el gran alcance

que permite el sensor LIDAR empleado un gran porcentaje de cada submapa coincidirá con el anterior, de manera que se podrá localizar y unir de una manera mucho más exacta. Habiendo acabado los experimentos de este proyecto se cree que esta situación aun puede ser mejor aprovechada para mejorar los resultados del SLAM.

Además de las conclusiones mencionadas siempre es interesante sacar algunas lecciones aprendidas en cuanto a la organización y el desarrollo del proyecto. Se plantean las siguientes:

- Es importante estimar correctamente los tiempos. Cuando se trabaja con hardware nuevo es normal que aparezcan errores inesperados o que el tiempo de configuración y puesta en marcha de los mismos sea más largo de lo aparente. Tratar de estimar de manera más precisa este tipo de contratiempos implica ser más realista al plantear los objetivos a cumplir y evita posibles frustraciones de no conseguir algunos objetivos o de no profundizar en algunas partes del proyecto.

- Es importante ser realista en cuanto a los conocimientos que se tienen y se van adquiriendo. Como es natural el momento en que más conocimientos se poseen del proyecto es cuando se termina, esto no invalida los experimentos y pruebas realizadas, si no que son estas pruebas las que han permitido construir ese conocimiento. Cuando se termina un proyecto es natural tener ideas sobre como avanzar o mejorar el trabajo realizado, que pueden plasmarse en los desarrollos futuros sin desmejorar ni infravalorar el trabajo realizado.

6.2. Desarrollos futuros

Una vez cerrada una primera etapa del proyecto se pueden plantear diferentes objetivos y desarrollos a realizar. A corto plazo se puede:

- Corregir la calibración de ambos sensores IMU.

- Ajustar el filtro de Kalman implementado en la capa funcional del robot para incluir la segunda IMU, la integrada con el LIDAR, en el cálculo de la odometría filtrada.

- Ajustar la frecuencia a la que se publican los datos de los diferentes sensores.

- Ajustar el filtrado de la componente de gravedad llevado a cabo por el Cartographer y mejorar la parametrización del mismo con los nuevos ajustes.

- Comprobar nuevamente las distancias de funcionamiento de las antenas GNSS trabajando en modo RTK. Se trata de comprobar como de grande se podrá hacer un mapa manteniendo la precisión RTK.

Algunos objetivos que pueden plantearse a medio plazo pueden incluir:

- Adaptar el *navigation stack* de ROS para utilizar los mapas y localización producidos por el algoritmo Cartographer.

- Integrar la planificación y control de trayectorias con el robot validándola en los entornos de interior y exterior.

A largo plazo se espera contar con una plataforma autónoma robusta sobre la que poder llevar a cabo pruebas, obtener datos y avanzar en diferentes líneas de investigación por parte de todo el equipo. Aunque el objetivo no es investigar sobre navegación, si no más bien aprovechar las tecnologías existentes, está claro que es una pieza importante y un punto de partida para el trabajo futuro.

Bibliografía

[1] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver, 3 2022.

[2] Ángela Bouzas, Silvia Albarracín, and Javier Alcantara Carrio. *Estudios de erosión en costas sedimentarias mediante GPS diferencial y ecosondas monohaz/multihaz.*, pages 105–130. 07 2009.

[3] Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Active mobile robot localization, 1997.

[4] Jens Clausen. Branch and bound algorithms-principles and examples. 2003.

[5] Clearpath Robotics Inc. *HUSKY USER MANUAL*, 12 2014. 0.20.

[6] Clearpath Robotics Inc. *HUSKY datasheet*, 01 2022. 0.20.

[7] Thomas Collins, J.J. Collins, and Donor Ryan. Occupancy grid mapping: An empirical evaluation. In *2007 Mediterranean Conference on Control Automation*, pages 1–6, 2007.

[8] Iván del Pino. *Desarrollo de un sistema completo de navegación autónoma basado en GNSS y LiDAR para robots terrestres que operan en entornos dinámicos y no estructurados.* PhD thesis, INSTITUTO UNIVERSITARIO DE INVESTIGACIÓN INFORMÁTICA. Universidad de Alicante, 2021.

[9] D. Filliat and J.A. Meyer. Map-based navigation in mobile robots:: I.a review of localization strategies. volume vol. 4,no. 4 pp, page 243–282, 2003.

[10] Arturo Gil Aparicio. *Construcción cooperativa de mapas visuales mediante un equipo de robots móviles.* PhD thesis, Departamento de Ingenier ıa de Sistemas Industriales. Universidad Miguel Hernández, Oct 2008.

[11] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2432–2437, 2005.

[12] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[13] Harxon. *SMART antenna command description*, 10 2018. Version 1.3.

[14] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[15] Uwe Jahn, Daniel Heß, Merlin Stampa, Andreas Sutorma, Christof Röhrig, Peter Schulz, and Carsten Wolff. A taxonomy for mobile robots: Types, applications, capabilities, implementations, requirements, and challenges. *Robotics*, 9(4), 2020.

[16] Luc Jaulin. Introduction. In Luc Jaulin, editor, *Mobile Robotics*, pages ix–xi. Elsevier, 2015.

[17] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. Autonomous robot navigation in highly populated pedestrian zones. *Journal of Field Robotics*, 32(4):565–589, 2015.

[18] Redshift Labs. UM7 datasheet, 10 2016. Rev. 1.6.

[19] Redshift Labs. UM7 calibration procedure. `https://redshiftlabs.com.au/support-services/um7-calibration-procedure/test_utils`, (s.d.). [Consulta Online 08 2022].

[20] Róbert Marc and Piotr Weclewski. Capabilities of long range autonomous multi-mode rover navigation system - ergo field trials and planned evolution. 05 2019.

[21] Saqib Mehmood and Bing Qiao. 2d slam and path planning for indoor search and rescue using multiple mobile robots. *International Journal of Modern Research in Engineering and Technology (IJMRET)*, 6(4), 2021.

[22] Abul Kashem Niloy, Anika Shama, Ripon Chakrabortty, Mike Ryan, Faisal Badal, Zinat Tasneem, Hafiz Ahamed, Sumaya Moyeen, Sajal Das, Md Ali, Md Robiul Islam, and Dip Saha. Critical design and control issues of indoor autonomous mobile robots: A review. *IEEE Access*, PP:1–1, 02 2021.

[23] IFR International Federation of Robotics. A mobile revolution: How mobility is reshaping robotics. *Robotics*, 06 2021.

[24] Ouster. Building maps using google cartographer and the OS1 LIDAR sensor. `https://ouster.com/blog/building-maps-using-google-cartographer-and-the-os1-lidar-sensor/`.

[25] Ouster, Inc. *Ouster OS1 Datasheet*, 04 2022. HW Rev. 06.

[26] Carlos Perez-del Pulgar, J. Ricardo Sánchez Ibáñez, Andres Jesus Sanchez Fernandez, Martin Azkarate, and Gianfranco Visentin. Path planning for reconfigurable rovers in planetary exploration. pages 1453–1458, 07 2017.

[27] Sick AG. *Sick AG whitepaper: Funcionamiento y variantes de los sensores LiDAR*, 07 2018. 0.20.

[28] Anderson Souza and Luiz M. G. Gonçalves. Occupancy-elevation grid: an alternative approach for robotic mapping and navigation. *Robotica*, 34(11):2592–2609, 2016.

[29] Robotics Technical Committee ISO/TC 299. Robotics — Vocabulary. Standard, International Organization for Standardization, Geneva, CH, 2021.

[30] Spyros G. Tzafestas. 1 - mobile robots: General concepts. In Spyros G. Tzafestas, editor, *Introduction to Mobile Robot Control*, pages 1–29. Elsevier, Oxford, 2014.

[31] H. Umari and S. Mukhopadhyay. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1396–1402, Sept 2017.