UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de un sistema de localización y mapeo simultáneo mediante un escáner Lidar Hokuyo como un nodo en ROS para el Rover UVG

Trabajo de graduación presentado por Katharine Senn Salazar para optar al grado académico de Licenciada en Ingeniería Mecatrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de un sistema de localización y mapeo simultáneo mediante un escáner Lidar Hokuyo como un nodo en ROS para el Rover UVG

Trabajo de graduación presentado por Katharine Senn Salazar para optar al grado académico de Licenciada en Ingeniería Mecatrónica

Guatemala,

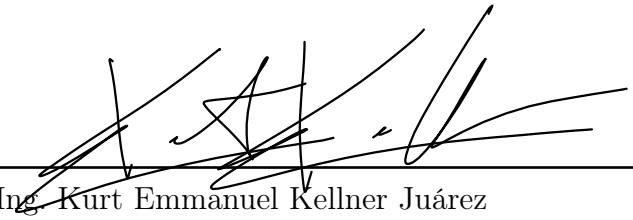
2023

Vo.Bo.:

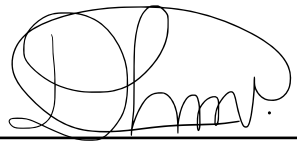
(f) 

Ing. Kurt Emmanuel Kellner Juárez

Tribunal Examinador:

(f) 

Ing. Kurt Emmanuel Kellner Juárez

(f) 

Ing. Diego Alberto Morales Ibañez

(f) 

MAEB. Pablo Daniel Mazariegos de la Cerda

Fecha de aprobación: Guatemala, 5 de enero de 2023.

Prefacio

Este trabajo presenta una solución para que robots móviles con ruedas, específicamente el Rover UVG, sean capaces de localizarse y mapear sus alrededores durante su tiempo de recorrido, aplicado como un nodo de ROS2 (Robot Operating System). Esto se desarrolló a base de herramientas del área de robótica y muchos otros conocimientos adquiridos durante el estudio de esta carrera.

Primero que nada agradezco a Dios, por todas las oportunidades de crecimiento que ha puesto en mi camino. Por otro lado, agradezco a mi mamá María José, mis hermanos Marianne y Jorge, mis abuelos y el resto de mi familia por su apoyo incondicional durante esta etapa de mi vida y siempre. Así mismo, agradezco a Fundación Educación por brindarme la oportunidad de estudiar ingeniería mecatrónica en la Universidad del Valle de Guatemala y también agradezco a los amigos que me acompañaron durante estos 5 años de estudio especialmente a Diego Mencos por todo su apoyo durante la realización de este proyecto. Por otro lado, agradezco a mi asesor Ing. Kurt Emmanuel Kellner así como a Dr. Luis Alberto Rivera y Msc. Miguel Enrique Zea Arenales por su apoyo, el tiempo invertido en este proyecto y todo el conocimiento que me compartieron a lo largo de mi carrera universitaria.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo de graduación propone una solución para realizar tareas de localización y mapeo simultáneo (SLAM, por sus siglas en inglés) utilizando un escáner HokuyoURG-04LX-UG01 como fuente de datos para el reconocimiento del entorno. Siendo a su vez compatible con el sistema de captura de movimiento disponible en la Universidad del Valle de Guatemala, Optitrack con el fin de poder implementar un módulo de SLAM en el sistema operativo ROS2 compatible con la plataforma móvil Rover UVG. La implementación de este algoritmo es presentada tanto en simulación como en pruebas físicas utilizando los sensores descritos anteriormente.

Adicionalmente se presenta una herramienta para generar archivos en formato de imagen que almacenan la información dada por los mapas generados en SLAM con la finalidad de poder utilizar esta información en procesos posteriores a la fase del proyecto que se trabaja actualmente.

Abstract

The project presented in this document proposes a solution to perform simultaneous localization and mapping (SLAM) tasks using a Hokuyo URG-04LX-UG01 scanner as a data source for environmental recognition. Being compatible with different methods for obtaining odometry data such as enconders and the motion capture system available at Universidad del Valle de Guatemala. This being able to implement a SLAM module in the ROS2 operating system compatible with the Rover UVG platform. The implementation of this algorithm is presented both in simulation and in physical experiments with the Rover UVG platform.

Additionally, a tool to generate image format files that store the information given by the maps generated by SLAM is presented in order to be able to use this information in subsequent processes to the phase that is currently being worked on this project.

CAPÍTULO 1

Introducción

La navegación en robots móviles con ruedas es integrada por tres acciones principales: localizar, planificar y mapear. Los algoritmos de *Simultaneous localization and mapping* (SLAM) nos permiten realizar localización y mapeo de forma simultánea con el fin de generar mapas de las áreas recorridas por el robot.

En este trabajo se presenta una solución compatible para realizar SLAM en la plataforma Rover UVG mediante el sistema operativo ROS2. Para la obtención de datos se utiliza un sensor Lidar: HokuyoURG-04LX-UG01 que nos permite obtener la información del entorno y por medio de odometría obtenida mediante el sistema de captura de movimiento, Optitrack:, se conoce la posición del robot. Obteniendo esta información mediante nodos de ROS2 los cuales publican los datos en el formato necesario en los respectivos canales de comunicación se utiliza un algoritmo de *Pose-Graph SLAM* mediante la librería de ROS2 *SLAM Toolbox* para la generación de mapas.

Para finalizar se presenta una herramienta adicional que permite guardar los mapas generados en cada recorrido en forma de imagen .pgm con el fin de poder utilizar estos mapas en futuras aplicaciones e investigación que se realice entorno al Rover UVG.

CAPÍTULO 2

Antecedentes

El trabajo que se propone en este documento tiene como fin ser implementado en el proyecto Rover UVG. Por lo mismo se incluyen antecedentes del proyecto en los últimos años.

2.1. Diseño mecánico, selección de motores e implementación de sensores para un Robot Explorador Modular

El trabajo de graduación de Hector Sagastume [1], señala el proceso de diseño, selección e implementación de componentes para un Robot Explorador Modular. Siendo este parte de una línea de investigación sobre robots exploradores modulares y rovers en la Universidad del Valle de Guatemala, el trabajo busca el avance, mejoramiento y actualización de un proyecto iniciado anteriormente. El trabajo se desarrolló en distintos módulos, siendo estos diseño y estructura mecánica, actualización y mejora de motores y controladores en el robot e implementación de sensores que apoyen el funcionamiento autónomo de la máquina. Uno de las actualizaciones más significativas en esta fase del proyecto fue el cambio de llantas convencionales por orugas, lo que le permite al rover ser más versátil en cuanto a los terrenos en los que puede movilizarse.

2.2. Diseño e implementación de capacidades automáticas de navegación para un Robot Explorador Modular

El proyecto de graduación de Javier Archila [2], se encarga de dotar de capacidades de navegación a un robot explorador modular. Esto se logró tanto mediante el piloto automático Pixhawk, configurado por medio del programa Mission Planner e implementado en una computadora Raspberry Pi 3 con envío de datos por medio de internet, así como de forma manual controlándolo por una palanca de mando. Como parte de los resultados de este

proyecto encontramos pruebas de funcionamiento del robot en distintos terrenos y cómo la plataforma es capaz de adaptarse a ellos, como por ejemplo subida de gradas e inclinación por banquetas.

2.3. SLAM 2d y Mapeo 3D autónomo de un ambiente utilizando un solo Lidar 2D y ROS

En el artículo [3], los autores Manuel González, Novel Certad, Said Alvarado y Ángel Terrones describen un algoritmo capaz de realizar localización y mapeo simultáneo 2D (SLAM,por sus siglas en inglés). Esto se realiza por medio de un nodo de Robot *Operating System* (ROS). De igual forma, el robot es capaz de generar mapas 3D de su entorno por medio del nodo de ROS 3D Octomap y la librería *State Machines* (SMACH) sin la necesidad de intervenciones humanas o información previa del entorno. Este mapeo 3D se logró debido a que el lidar utilizado fue adaptado a un *gimbal* impreso en 3D, diseñado por los autores. Algo de mucha relevancia en este documento es que los autores mencionan algunos de los nodos de ROS que utilizó para los distintos análisis a realizar con la información obtenida del lidar. El nodo URG se utiliza como el canal de comunicación entre el lidar y el resto de nodos en ROS. Por otro lado, el nodo *Laser Filters* se utiliza para limpiar los datos obtenidos del lidar, eliminando ruidos e infomación no deseada. Por otro lado, para realizar SLAM se utiliza el *Cartographer Node*. Este es un algoritmo de SLAM desarrollado por Google que permite generar mapas de alta calidad con únicamente la información obtenida del lidar además de utilizar poco espacio de memoria comparado con otros algoritmos. En la Figura 1 se observa la comparación entre un plano del área navegada (a la izquierda) y el mapa generado por el proyecto. Es posible observar que en efecto el sistema es capaz de recrear la geometría, en dos dimensiones, de los espacios recorridos.

2.4. Comparación de Métodos de SLAM basados en ROS para un entorno interno homogéneo

En el artículo de conferencia [4], de Ilmir Ibragimov y Ilya Afanasyev se presenta una investigación en la cual son analizados y comparados distintos métodos de SLAM. En esta comparación se evalúan la factibilidad de los distintos métodos para ser implementados en un robot de ambientes en interiores. Otra característica de lo mostrado en este documento

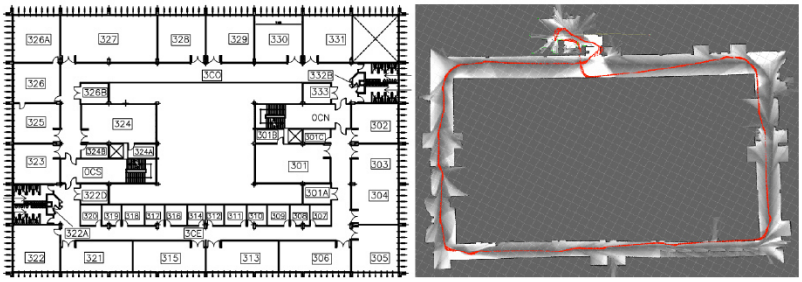


Figura 1: Comparación plano planta del área recorrida y mapa generado por medio de Lidar SLAM

es que se trabajan cada algoritmo de SLAM (Hector SLAM, Monocular SLAM, etc) con distintos sensores, entre ellos un lidar Hokuyo y distintos tipos de cámaras, con el fin de encontrar las características y beneficios de cada una de las combinaciones entre ROS y los sensores. Aunque los autores explican que no existe un mejor método que otro ya que esto dependerá de la aplicación si lograron concretar ciertas conclusiones sobre la aplicación de ciertos métodos y sobre SLAM en general. Para el algoritmo ORB SLAM, un método con bastante exactitud, se sugiere utilizarlo únicamente si el entrono de operación no tiene objetos de un solo color o se necesita un funcionamiento de alto desempeño. Por otro lado, si se necesitan crear mapas con altas profundidades, se sugiere utilizar cámaras stereo en vez de un lidar. Por último, se concluye que en general no existe un algoritmo de SLAM monocular capaz de estimar la escala absoluta de un mapa, para esto se necesita un sistema independiente de localización.

2.5. SLAM y navegación de un robot de interiores en ROS utilizando un Lidar

En el artículo [5], el autor Xiaozhuo Yang, diseña un algoritmo implementado en ROS para que un robot móvil pueda desplazarse desde un punto inicial a un punto final en un área desconocida. La toma de decisiones en cuanto al movimiento del robot se realiza mediante algoritmos A* y DWA y los puntos de inicio y fin son proporcionados por un sistema de localización independiente al sistema de SLAM. Con esta información se implementa SLAM con un lidar 2D y ROS con el fin de que el robot no tenga inconvenientes para llegar a su destino debido a la geometría o los posibles obstáculos que existan en el espacio de movimiento del robot. Un aspecto importante de este documento es que el autor no solo implementa un lidar en su robot móvil sino también implementa una cámara. Esto con el fin de poder tener información adicional que puede ser de utilidad para evitar colisiones, como lo es la profundidad e incluso pequeños obstáculos que el lidar no sea capaz de detectar. Para la verificación del funcionamiento del proyecto, el autor utilizó la plataforma de simulación Gazebo, la cual le permitió observar a su robot y al entorno por el que navegaría. Algo que llamó mi atención sobre este proyecto es que mediante simulación, al programar un punto final de movimiento, es posible observar la trayectoria que recorrerá el robot antes de que este inicie su viaje. Como se esperaba, el autor muestra una seria de pruebas con distintos puntos iniciales y finales y cómo el simulador le muestra las trayectorias encontradas para que el robot no choque durante el recorrido.

2.6. Implementación de una interfaz de datos RS-232 inalámbrica para el radar láser Hokuyo URG-04LX-UG01

En este trabajo de graduación [6] Mario Andrés Búrbano, describe el proceso de implementación de un radar laser HokuyoURG-04LX-UG01 por medio de una interfaz inalámbrica RS-232. El radar utilizado cuenta con las siguiente características: área de detección de 240°, resolución de 1mm, resolución angular de 0.35°, rango de entre 20cm y capacidad de envío de datos cada 100ms. Se implementó un anfitrión USB en un microcontrolador PIC24FJ64GB002 que es compatible con el estándar USB On-The-Go también conocido co-

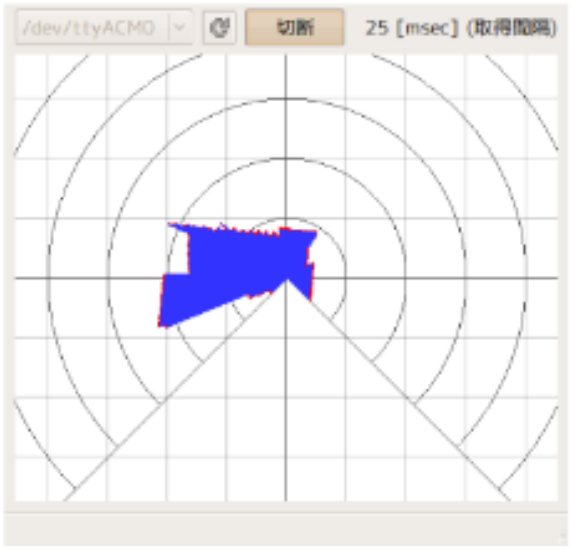


Figura 2: Mapa generado con datos obtenidos del Lidar Hokuyo URG-04LX-UG01 por Mario Búrbano

mo OTG (por sus siglas en inglés). En cuanto a la interfaz inalámbrica RS-232 se utilizaron módulos de radio frecuencia XBee Pro de 60mW. Por otro lado, para la visualización de datos, se desarrolló una interfaz gráfica en lenguaje de programación C. Dicha interfaz no solo permite visualizar los datos sino también cambiar parámetros de funcionamiento, los cuales son almacenados en una memoria no volátil del micro controlador utilizado. En la Figura 2 es posible observar uno de los mapas generados por el proyecto. Este trabajo fue realizado con el objetivo de dotar de visión a un robocóptero para que éste no colisionara con su entorno durante el vuelo.

CAPÍTULO 3

Justificación

El Rover UVG es uno de los proyectos trabajados en la línea de investigación de robots exploradores modulares en la Universidad del Valle de Guatemala. La propuesta del siguiente trabajo es desarrollar un módulo de localización y mapeo simultáneo que pueda ser implementado en la plataforma robótica móvil Rover UVG. El proyecto tiene como finalidad darle visión al robot y por consiguiente que este pueda navegar de forma autónoma espacios interiores desconocidos.

Por otro lado las herramientas propuestas para la visualización y almacenamiento de los mapas generados aportarán a la verificación del funcionamiento de la integración de SLAM con un lidar y su compatibilidad con el Rover UVG u otros robots de la misma categoría. Además que la implementación de este sistema como un nodo de ROS permite replicarlo e integrarlo en proyectos futuros. Ya sea que este se implemente nuevamente en el Rover UVG con una finalidad distinta o que se implemente en un robot distinto.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar un sistema de SLAM que utilice un escáner Lidar Hokuyo URG-04LX-UG01 como un nodo de ROS para la plataforma móvil Rover UVG.

4.2. Objetivos específicos

- Implementar nodos de ROS para leer e interpretar datos extraídos del escáner lidar HokuyoURG-04LX-UG01.
- Evaluar y comparar el rendimiento de los distintos algoritmos de SLAM disponibles en ROS para plataformas robóticas con ruedas.
- Desarrollar e implementar un método simple que permita acceder a los mapas obtenidos por SLAM para su uso posterior en planificación de movimiento.

CAPÍTULO 5

Alcance

El alcance de este proyecto abarcó tanto la simulación como la aplicación física de uno de los métodos disponibles para realizar SLAM mediante un nodo en ROS2. Esto se realizó por medio del paquete para ROS2 *SLAM Toolbox*, desarrollado por Steve Macenski. Este paquete utiliza algoritmos de *Pose Graph SLAM* para generar en tiempo real mapas 2D de los espacios recorridos por el Rover UVG.

Los mapas generados por este método se conocen como *Cost Maps*, o mapas de costos en español. Los cuales consisten en generar una imagen a la cuál se le asigna un valor a cada píxel. Para la simulación de estos algoritmos se utilizó Gazebo y para la aplicación física se utilizó un Lidar Hokuyo URG-04LX-UG01 para las lecturas de distancias en conjunto con los nodos de ROS2 realizados en otros módulos del proyecto Rover UVG para obtener la odometría, localización y orientación del robot.

En proyectos futuros o trabajos de continuación del Rover UVG será posible utilizar la información de dichas imágenes para complementar los algoritmos de navegación del Rover UVG o cualquier otro robot, por ejemplo realizar planificación de trayectorias para navegación punto a punto o incluso se puede utilizar para algoritmos de movilización autónoma.

CAPÍTULO 6

Marco teórico

6.1. Navegación en robots móviles

La navegación en robots móviles con ruedas se conforma por tres características principales:

- Localización: se encarga de identificar la posición y orientación del robot dentro de un marco de referencia definido.
- Planificación: esta se refiere a planificación de movimiento para que el robot se movilice sin necesidad de controles remoto o similares, este movimiento puede ser de punto a punto, por trayectorias o incluso puede basarse en estrategias de evasión de obstáculos, etc.
- Mapeo: esta actividad se encarga de crear *Occupancy Grids*, es decir que suponiendo que se conoce la posición del robot (por medio de la localización, explicada anteriormente) y teniendo la información de la distancia a la que se encuentran los objetos al rededor del robot es posible determinar que casillas de una cuadrícula están ocupadas y cuales están libres. Esta es una de las formas más comunes de generar mapas del entorno del robot sabiendo por que espacios le es posible movilizarse y por cuales no. [7]

6.1.1. Marcos de referencia

Para lograr las actividades mencionadas anteriormente es necesario definir al menos 4 marcos de referencia, los cuales serían:

- I = *Marco de referencia Inercial*, el cual es fijo con respecto a la tierra.

- N = *Marco de referencia de Navegación*, al igual que el inercial este es un fijo pero este es un marco de referencia plano que define el entorno en el que el robot actúa.
- B = *Marco de referencia del Robot*, esta referencia es móvil y su origen suele coincidir con el centroide del robot.
- S = *Marco de referencia del Sensor*, este también es un marco de referencia móvil con respecto a I y N, pero al mismo tiempo es fijo respecto a B. Este marco de referencia se genera según la posición y orientación del sensor de medición.

6.1.2. Sensores

En cuanto a aplicaciones de navegación de robots móviles, los sensores se suelen categorizar como:

- Propioceptivos:Estos sensores son los que se encargan de indicar información propia del robot, estos con frecuencia son *encodres*, acelerómetros, etc. La información dada por estos sensores suelen estar referenciadas en los marcos B o S.
- Exteroceptivos: Por otro lado, estos sensores son los que entregan información externa al robot, es decir en vez de indicarnos las propiedades del robot en sí nos ayudan a entender su entorno. Estos pueden ser sensores lidar, gps, o cualquier otro tipo de sensor que nos entrega información sobre los alrededores del robot en movimiento. La Información dada por este tipo de sensor se referencia a los marcos I o N.

Por lo regular, en navegación se busca contar con ambos tiempos de sensores. Esto se debe a que cada tipo de sensor tiene características distintas, por lo general, los sensores propioceptivos entregan información de forma rápida, pero suelen ser señales considerablemente ruidosas. Por otro lado, los sensores exteroceptivos contra restan este efecto entregando mediciones más espaciadas en tiempo pero de mayor precisión. [7]

6.2. Light Detection and Ranging (Lidar)

Lidar es un método de teledetección óptica que utiliza un láser para obtener, por medio de puntos, muestras densas de superficies. Los sensores Lidar transmiten rayos láser hacia su alrededor mientras están en movimiento. Estos registran el tiempo preciso desde que el pulso láser sale del sistema hasta que regresa para calcular la distancia que hay entre los objetos del entorno y el sensor. Esto es posible debido a que la velocidad de la luz es una constante, entonces el sensor toma el tiempo en el que cada impulso láser enviado tarda en llegar hasta el objeto más cercano y se refleja en dirección opuesta hasta retornar al sensor.[8] Esa medición se puede representar mediante la siguiente expresión:

$$distancia = (velocidad * tiempo)/2 \tag{1}$$

En donde la distancia es el dato que el algoritmo necesita para poder generar las nubes de puntos, la velocidad es la constante conocida de la velocidad de la luz, el tiempo es el

tiempo transcurrido desde que el láser sale del sensor y regresa al mismo, esta es la medición que realiza el sensor y por último la división entre dos indica que el láser recorre 2 veces la distancia real, pues como se explica anteriormente este viaja del sensor al objeto, se refleja y luego viaja de regreso al sensor. En el mercado existen tres tipos de Lidar de a cuerdo a su aplicación: Pueden ser medidores de distancia (1D), medidores de área (2D) o pueden medir volúmenes (3D). Para este proyecto se utilizará un Lidar 2D modelo URG-04LX-UG01 de la marca Hokuyo como el que se muestra en la Figura. 3

Figura 3: Sensor Lidar Hokuyo URG-04LX-UG01

6.2.1. Lidar Hokuyo URG-04LX-UG01

El Hoyuko URG-04LX-UG01 es un sensor láser para escaneo de áreas. La luz utilizada por este sensor es un láser (clase 1) infrarrojo con una longitud de onda de 785nm. El espacio detectado es la de 240° de un círculo de hasta 4000mm de radio como se observa en la Figura 4. Este sensor es alimentado por 5V y tierra mediante el cable de conexión USB-mini de 5 pines y se comunica a computadoras por medio de un puerto serial por el mismo cable de alimentación. Una de las aplicaciones de este sensor son los robots móviles, debido a su diseño compacto el cual permite mayor libertad en el diseño del robot además de no demandar altos niveles de potencia para tener un funcionamiento correcto, sin embargo su uso suele restringirse a aplicaciones educativas o de investigación debido a que su desempeño es mejor en ambientes controlados, es decir espacios cerrados y con pocas perturbaciones externas. [9]

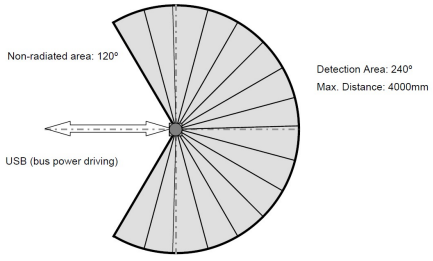


Figura 4: Rango de detección sensor Lidar Hokuyo URG-04LX-UG01

6.3. Raspberry Pi

Raspberry Pi es una marca de computadoras reconocida por sus modelos tipo "sistema en chip"(SoC, por sus siglas en inglés) lo que vuelve a estos aparatos de bajo costo y dimensiones compactas en comparación a la mayoría de computadoras en el mercado. Este tipo de computadoras son capaces de realizar cualquier tarea que se espera poder realizar en computadoras comerciales tanto de escritorio como portátiles, con la diferencia que al ocupar un espacio reducido y no verse obligadas a operar en conjunto con una pantalla y otros elementos que ocupan espacio son comúnmente utilizadas en proyectos *Maker* y de desarrollo e investigación de tecnología. En este proyecto se pretende utilizar el modelo Raspberry Pi 4 como el que se puede observar en la Figura 5. [10]

Figura 5: Computadora Raspberry Pi 4

6.3.1. Raspberry Pi 4

Este modelo de computadora cuenta con distintas características favorables para este proyecto. Algunas de esas características son las siguientes: Para iniciar esta computadora no necesita ventiladores lo que la vuelva más silenciosa y a la vez tiene un consumo energético menos que otros modelos de computadoras. Por otro lado, el *board* cuenta con un puerto de conexión de Ethernet, así como con antenas para bluetooth y Wi-Fi y 4 puertos USB. Por último este modelo está disponible con cuatro tamaños de RAM distintos (1GB, 2GB, 4GB y 8GB). [10]

6.4. Arduino

Arduino inició en 2005 como un proyecto de Instituto de Diseño e Interacción de Ivrea, IIDI por sus siglas en inglés, en Italia. EL cual buscaba proveer, tanto a profesionales como principiantes, con dispositivos de bajo costo capaces de interactuar con su entorno. Esta plataforma plataforma *Open Source* actualmente se basa en el desarrollo de *software* y *hardware* de uso "sencillo". Esto se refiere a que siempre se busca que sus diseños sean

lo más amigables con el usuario posible. A lo largo de los años, Arduino ha facilitado la realización de muchos proyectos, especialmente en el ámbito educativo, sin embargo también es utilizado en proyectos de alto nivel debido a su enfoque como herramienta de prototipado rápido. 6. [11]

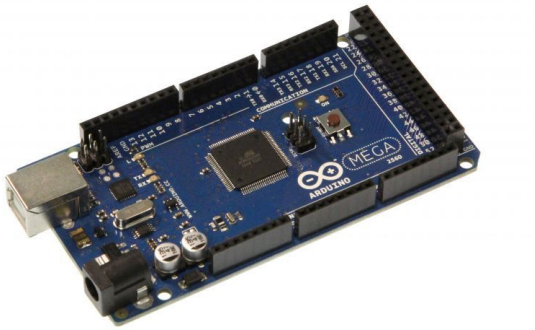


Figura 6: Arduino Mega 2560

6.4.1. Arduino Mega 2560

Para este proyecto se utilizó un Arduino Mega 2560. Este Arduino es un modelo de *board* cuyo microcontrolador se basa en un ATmega2560. Este cuenta con 54 pintes de entradas y/o salidas digitales de los cuales 15 pueden ser utilizados como salidas de PWM:. A su vez se cuenta con 4 puertos seriales en *hardware* y pines para comunicación ICSP. Este modelo de Arduino fue seccionado debido a que es capaz de realizar comunicación SCIP2.0 por medio de un *USB Host Shield*, esto era importante para el proyecto debido a este es el protocolo de comunicación utilizado por el sensor Hokuyo URG-04LX-UG01. [12]

6.4.2. USB Host Shield

El Arduino *USB Host Shield* es un accesorio de Arduino el cual permite conectar otros dispositivos mediante USB al *Board:*. Este dispositivo se basa en un controlador de *Host*/periférico USB MAX3421E. Para utilizar el mismo de forma correcta es necesario instalar la librería ÜSB Host Shield Library 2.0.[en] el *spftware* de Arduino, asi mismo el accesorio se debe ensamblar con el Arduino Mega 2560 de manera que los 6 pines ICSP de ambos artefactos queden conectados. De esta forma, al conectar un aparato pro medio de la entrada USB del *USB Host Shield* ete se podrá comunicar con el Arduino Mega 2560. 7. [13]

6.5. Robot Operating System (ROS)

ROS es un conjunto de herramientas y librerías *Open Source* que facilitan la creación de aplicaciones para robótica. ROS inició como un proyecto de 2 estudiantes de Standford en

Figura 7: USB Host Shield

2007 para el proyecto del Robot de Inteligencia Artificial de Stanford. Uno de los objetivos principales de ROS es minimizar el tiempo que toma implementar la infraestructura de software para poder construir algoritmos de robótica complejos y así poder dar más tiempo a la programación del robot como tal. [14]

ROS es conformado por dos partes principales: El sistema operativo y los paquetes de ROS (ROS-pkg, por sus siglas en inglés). Este último es un repositorio de librerías aportadas por la comunidad de los usuarios de ROS las cuales se pueden implementar como plantillas para realizar tareas utilizadas comúnmente en robótica como: simulaciones, localización y mapeo, planificación de tareas, etc.

Este sistema operativo logra volver la programación de robots más eficiente mediante estructuras creadas por nodos (ROS NODES) y tópicos (ROS TOPICS). Todo proyecto en ROS debe ser creado a partir de un ROS Máster, que es el programa que se encarga de definir los nodos que tendrá el proyecto, configurar la comunicación entre dichos nodos, la actualización de la base de datos, etc. Luego de esto se define la estructura de los nodos ROS y como se puede observar en la Figura 8 la forma en la que estos se comunicarán,mediante los tópicos ROS, entre ellos y a la base de datos también conocida como servidor de parámetros. Los distintos nodos definidos en un proyecto de ROS pueden realizar tareas tales como envío y recepción de datos, lectura de sensores, control, planificación de movimiento u otras tareas, activación de actuadores, entre otros. [15]

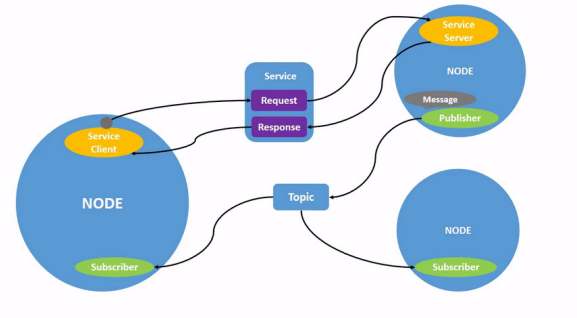


Figura 8: Estructura de nodos y tópicos ROS

6.6. Localización y Mapeo Simultáneo (SLAM)

Desde hace muchos años atrás en navegación marítima y cartografía se tenía el problema de como extender los mapas de lugares nuevos mientras se utilizaba ese mismo mapa para navegar. En robótica, este problema se conoce como *Simultaneous localization and mapping* (SLAM).Conociendo conceptos básicos de robótica este problema puede analizarse de la siguiente manera:

Siendo la configuración del vehículo en movimiento compuesta por el vector de estado y las coordenadas del punto de referencia M

$$\hat{x} = (x_v, y_v, \theta_v, x_1, y_1, x_2, y_2, ...x_M, y_M)^T \epsilon \mathbb{R}^{2M+3\times 1} \tag{2}$$

la covarianza estimada tendría la estructura de una matriz (2M +3) x (2M + 3)

$$\hat{P} = \begin{bmatrix} \hat{P}_{vv} & \hat{P}_{vm} \\ \hat{P}_{vm}^T & \hat{P}_{mm}^T \end{bmatrix} \tag{3}$$

en donde

$$\hat{P}_{vv}$$

es la covarianza de la pose del vehículo,

$$\hat{P}_{mm}$$

la covarianza de las posiciones del punto de referencia del mapa y

$$\hat{P}_{vm}$$

es la correlación entre el vehículo y el mapa.

Cuando se detecta un cambio en el vector de estado este se actualiza por medio del Jacobiano, siendo Gx es distinto a 0.

$$G_x = \frac{\partial g}{\partial x} = \begin{bmatrix} 1 & 0 & -rsin(\theta_v + \beta) \\ 0 & 1 & rcos(\theta_v + \beta) \end{bmatrix} \tag{4}$$

entonces siendo la referencia depende del vector de estado, él ahora contiene la pose del vehículo. Para realizar SLAM, el Jacobiando Hx describen como el punto de observación del punto de referencia cambia con respecto al vector de estado. Entonces, la observación de la referencia dependerá tanto de la posición del vehículo como de la posición de la referencia y se describe como:

$$H_x = \frac{\partial h}{\partial x}|_{w=0} = (H_{x_v} \cdots 0 \cdots H_{p_i} \cdots 0) \in \mathbb{R}^2 \times (2M + 3) \tag{5}$$

en donde Hpi es el punto correspondiente a la posición de la referencia pi.

Para poder realizar SLAM, tomando los términos básicos mencionados anteriormente, existen distintos métodos. La distinción más importante entre estos métodos según Peter Corke [7] es si el algoritmo se basa en filtrado o por *smoothing* (suavizado) de información.

- Filtrado: En este método el estado actual del robot y del entorno se estima sobre la marcha con base en los últimos datos tomados por los sensores. El algoritmo se basa en una constante predicción de los datos y a su vez una etapa de corrección de dichas predicciones. Los algoritmos más comunes de este tipo son el EKF SLAM (SLAM mediante Filtro Extendido de Kalman) y SLAM mediante filtro de partículas.
- Suavizado: En este método el estado del robot y el mapa se estiman en base a trayectorias cerradas "completas" utilizando todos los datos tomados con anterioridad y realizando correcciones de odometría según lo observado por los sensores de distancia al cerrar una trayectoria. El algoritmo de *Pose-Graph SLAM* (SLAM mediante gráficos de pose) utiliza esta metodología.

6.6.1. Pose-graph SLAM

Este método es una alternativa que propone separar el problema en dos componentes. Una parte frontal (Front End, en inglés) y una trasera (Back End) conectados a un gráfico de pose, como se muestra en la Figura 9. La forma de operación de este tipo de sistema de SLAM divide las tareas entre los componentes mencionados anteriormente con el fin de hacer un sistema más eficiente. La parte frontal es la encargada de crear nodos según navegue el robot y bordes según la información dada por el sensor. Por otro lado la parte trasera se encarga de ajustar la posición de los nodos con el fin de minimizar el error. [7]

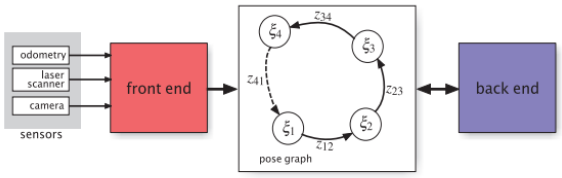


Figura 9: Diagrama algoritmo Pose-Graph SLAM

6.7. Mapas de costos

Los mapas de costos, mayormente conocidos en robótica como *occupancy grid maps* son imágenes generadas para representar el ambiente en el que se moviliza un robot con ruedas. Esto se logra por medio de una numeración de casillas, píxeles en este caso, en donde a cada casilla se le asigna un valor específico según su estado generando una estructura de valores similar a una matriz, pero representado en formato de imagen. Los estados posibles de una casilla en un *occupancy grid* son: vacío, ocupado o desconocido. Regularmente a las casilla vacías, es decir espacios en donde el robot puede movilizarse libremente, se les suele asignar el valor 0. Por lo contrario las casillas ocupadas, es decir espacios en donde hay obstáculos,

paredes o cualquier cosa física que impida el paso del robot, suelen tener asignado el valor máximo esto puede ser 255, 1, 100 o cualquier valor elegido según el sistema numérico que se utilice en el proyecto. Por último otro valor es asignado a las casillas no recorridas por el robot, o espacios no captados a estas se les suele asignar un valor, regularmente -1. Esto puede observarse en la Figura 10

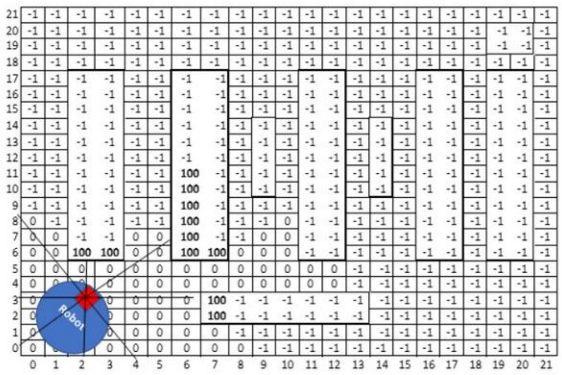


Figura 10: Representación de asignación de valores en Occupancy Grid Map

En el *toolbox* utilizado estos valores se pueden ver representados por colores en las imágenes generadas por el programa en donde los espacios vacíos se ven representados con color blanco, los espacios ocupados con color negro y los no recorridos en color gris, siendo el resultado algo similar a lo que se puede apreciar en la siguiente Figura 11. [16]

Figura 11: Ejemplo mapa generado por el SLAM Toolbox en ROS

6.8. Odometría

En robótica, la odometría es un sistema mediante el cual se puede estimar, mas no determinar, la posición de un robot relativa a su posición inicial. Basándose en la suposición de que las revoluciones de las ruedas pueden ser traducidas a desplazamiento lineal esto puede interpretarse entonces como movimiento a lo largo del tiempo. Esta información se obtiene regularmente mediante *encoders*. Los *encoders* son sensores situados en los ejes de las ruedas, encargados de traducir la cantidad de vueltas que un eje ha dado en un tiempo

específico. Conociendo esto y otros datos como el radio de las ruedas de nuestro robot es posible calcular un aproximado de la posición actual del robot. [7]

CAPÍTULO 7

Lectura e interpretación de datos extraídos del escáner Lidar Hokuyo URG-04LX-UG01

7.1. Funcionamiento del sensor Hokuyo URG-04LX-UG01

7.1.1. Características generales

El sensor láser URG-04LX de la marca Hokuyo es un escáner 2D, es decir de áreas. Este cuenta con un láser infrarrojo con longitud de onda de 785nm. El área de escaneo es una sección circular de 240º y radio de 4m. Este posee un motor eléctrico integrado el cual gira en sentido anti-horario moviendo así el punto de detección del láser. A continuación se presentan algunas de las características más relevantes del sensor: [17]

Tipo de Salida	Digital
Exactitud	±3
Rango mínimo	2cm
Rango máximo	50.96cm
Resolución lineal	1mm
Resolución angular	0.35°
Tiempo de barrido	100ms
Frecuencia	10 Hz

Cuadro 1: Características generales del sensor láser Hokuyo URG-04LX-UG01

Como se puede observar el la Figura 12, el sensor se encuentra colocado en el lado opuesto a la muesca rectangular que tiene la carcasa del sensor en su parte externa. Por otro lado, es importante tomar en cuenta que la parte trasera del sensor coincide con la parte negativa del eje vertical y la parte delantera del sensor con la parte positiva de este. Por lo mismo el

punto ciego del sensor se encuentra con vista al eje negativo.

Figura 12: Posición de láser y área de medición.

Orientación de mediciones

En la Figura 13 se pueden observar los distintos límites y puntos importantes de la medición del láser. Iniciando con el Paso 0, este indica el punto inicial en donde el radar le sería posible efectuar la primera medición. Sin embargo, como medida de seguridad el láser no inicia las mediciones hasta el Paso A, este punto puede ser variado según los valores que se asignen en los comandos enviados al sensor. El Punto B se ubica en el centro del rango de detección, que de medir en el rango completo posible coincide con el centro físico del sensor. Por otro lado el Punto C y el Punto D son los puntos equivalentes a los puntos A y 0 respectivamente, con la diferencia que estos son los puntos finales en vez de iniciales.

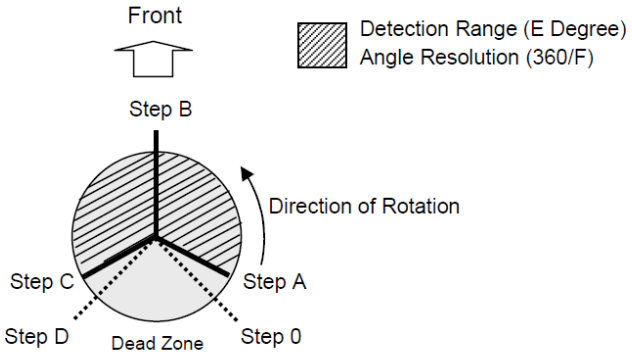


Figura 13: Puntos de medición Hokuyo URG-04LX-UG01

7.1.2. Protocolo de comunicación

El sensor utiliza un cable USB a USB mini b para comunicarse con un *Host*, así mismo el sensor utiliza el protocolo de comunicación SCIP2.0 el cual fue desarrollado en la universidad de Tsukuba, en Japón. Con el fin de comunicarse de una forma eficiente con sensores de esta gama. [18]

Para poder intercambiar información del sensor al *host* y viceversa se utilizan comandos predefinidos por el protocolo cuyas estructuras se explicarán a continuación.

Comunicación *Host* -> Sensor

La estructura general para los parámetros enviados del *Host* al sensor se describen en la Figura 14 y a continuación se describe cada una de las casillas.

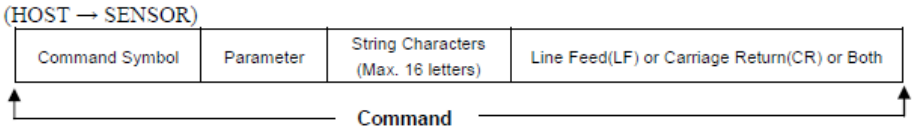


Figura 14: Comandos *Host* -> Sensor

- Símbolo de comando (2 bytes): es un conjunto de 2 caracteres no numéricos. Estos conjunto de caracteres ya están pre-establecidos por el protocolo SCIP2.0.
- Parámetros: en algunos casos los comandos requieren de información adicional para modificar algunas configuraciones solicitadas por el sensor, por lo mismo la longitud del la data puede variar según el comando.
- Caracteres en cadena (16 bytes máx.): cadena opcional de hasta 16 caracteres que permite verificar si el comando fue recibido de forma correcta, entre otros.
- Line Feed (LF) o Carr. Return (CR): Código de terminación de un comando.

Comandos sensor -> *Host*

Por otro lado, la respuesta del sensor regresa al *Host* en el siguiente formato. 15 y a continuación se describe cada una de las casillas.

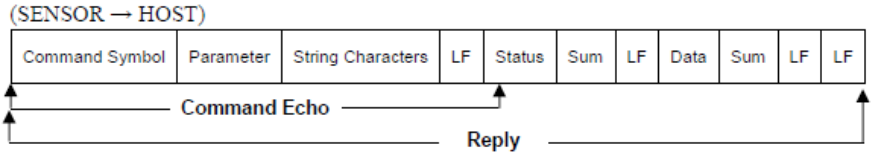


Figura 15: Comandos sensor -> *Host*

- Comando Echo: es una réplica del comando enviado del *Host* al Sensor.
- *Host* (2 bytes): código de valor entre 0 y 99 que indican el estado del comando actual, mayormente se utilizan para verificar que no existan errores como por ejemplo si el comando enviado no es válido o está incompleto.
- Sum (1 byte): caracter enviado del sensor al *Host* para autenticar los datos recibidos.
- Data (64 bytes): estos son los datos relacionados al comando actual, estos se agrupan en segmentos de 64 bytes y cada segmento está separado pro una secuencias de SUM + LF.

7.2. Lectura de datos

7.2.1. Lectura mediante Arduino y MATLAB

Arduino

Debido a que el cable que utiliza el sensor para comunicarse con un *Host* fue necesario utilizar un Arduino Mega y a este acoplarle un *Arduino USB Host Shield* así como instalar a librería de Arduino *USB Host Shield Library 2.0*. Mediante el ejemplo *acm _ terminal.ino* incluido en la librería mencionada anteriormente, se generó un canal de comunicación entre el sensor y el usuario por medio de un puerto serial. En este puerto serial se ingresaban comandos específicos desde la computadora y se obtuvieron resultados, por parte del sensor, como los que se muestran a continuación 16. Para obtener una sola corrida de datos se envió el comando:

MS0000068301001

El cual se comprende de la siguiente manera:

- El comando MS es un comando de adquisición de data, en donde la "S"señala que se desea recibir data codificada en 2 dígitos.
- Los primeros 4 dígitos *0000* expresan el punto de inicio del rango de medición deseado, en este caso se inicia en el punto 0.
- Los siguientes 4 dígitos *0683* indican que la medición final se realizará en el punto 683 de 768 puntos posibles.
- Los dos dígitos siguientes *01* son el *Cluster Count* esto se refieren a la cantidad de datos que pueden ser unidos. Por ejemplo, si el valor es 3, por cada 3 puntos de medición el sensor devolverá 1 dato del valor medio de esas 3 mediciones. En este caso el valor es 1, por lo que el sensor no unirá ningún dato y enviará todas las mediciones individuales que se realicen en cada corrida.
- Este siguiente dígito define el intervalo del escaneo, esto se refiere a la cantidad de puntos que el sensor debe saltarse cada vez que entrega una distancia. En este caso se colocó "0"debido a que se desea obtener todos los datos posibles de cada corrida.
- Para finalizar los últimos dos dígitos, en este caso *01*, indican la cantidad de corridas se quieren realizar.

Por otro lado para obtener datos continuamente el comando varía al siguiente:

MS0000068301000

Como se puede observar en , lo único que varía con el comando anterior es el último valor, pues al colocar este valor en *0* se le indica al sensor que realice una cantidad indefinida de corridas. Esto el sensor lo interpreta de forma que permanece enviando datos hasta que se le envíe un comando específico para parar. En la Figura 17 en la terminal se observa como al

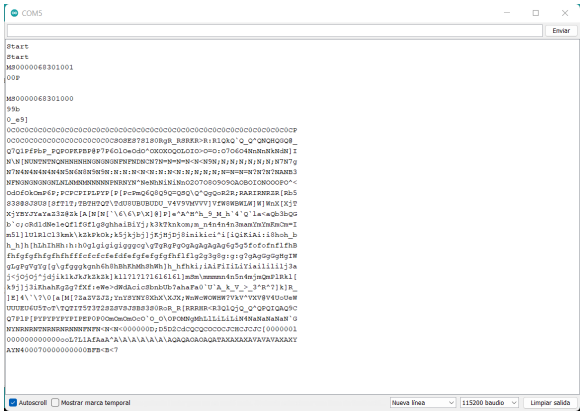


Figura 16: Resultados obtenidos en Arduino

finalizar la primera corrida de datos automáticamente se recibe nuevamente el otro mensaje completo de lectura de datos.

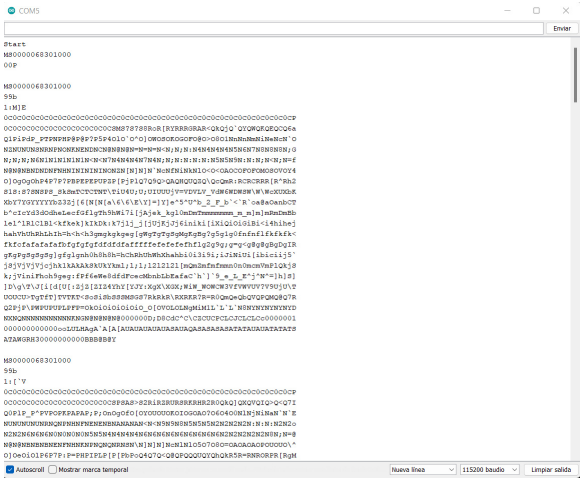


Figura 17: Resultados obtenidos en Arduino

Y, como se menciona anteriormente, para detener el envío de datos del sensor al *Host* únicamente basta con enviar el comando de 2 caracteres "QT", el cual apaga el láser al terminar la corrida de datos actual, esto se puede observar en la siguiente Figura 18.

MATLAB

MATLAB: se utilizó en esta ocasión para desarrollar un programa capaz de enviar desde una computadora los comandos al sensor. A su vez, el programa tiene la capacidad de obtener lecturas constantemente, decodificar y graficar los resultados obtenidos de dichas lecturas, actualizando la gráfica correspondiente.

Este proceso se utilizó para corroborar que las lecturas del sensor concordaban con lo esperado y al mismo tiempo desarrollar una herramienta sencilla con la cual se puedan manipular sensores de alta gama, como lo es el Hokuyo URG-04LX-UG01. Para que esto

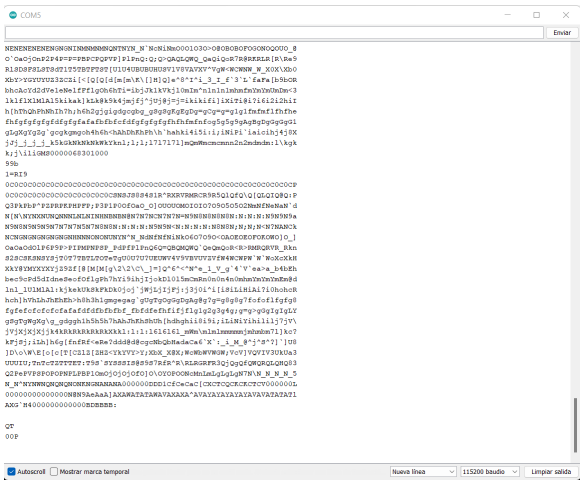


Figura 18: Resultados obtenidos en Arduino

fuera posible el código de MATLAB, como se puede apreciar en los anexos, se enfoca en la decodificación de la información obtenida.

Al recibir la respuesta del sensor, el primer paso fue identificar la data útil. Es decir, descartar de la respuesta el *Command Echo*, los bytes de verificación, etc. Al contar únicamente con los caracteres deseados fue necesario separar la cadena en grupos de 2 caracteres y así aplicar el algoritmo de decodificación 19, dado por el estándar de comunicación SCIP2.0 el cual entrega valores en milímetros.

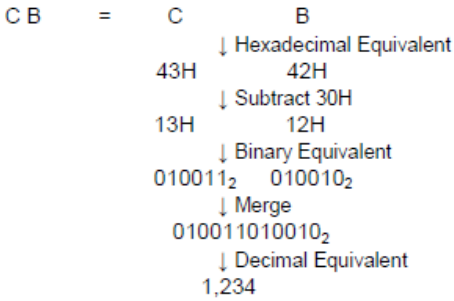


Figura 19: Algoritmo de decodificación

Para finalizar, los valores de distancias obtenidos de la decodificación se ordenaron en un arreglo de datos el cual se mostró en una gráfica de coordenadas polares en MATLAB para recrear las áreas detectadas por el sensor. Para esto se utilizaron cuatro obstáculos de prueba 20. Como primera prueba se realizó una medición de la plataforma Robotat: vacía 21, como un reconocimiento del área. Luego se fueron colocando los obstáculos en distintas posiciones dentro del rango de detección del sensor. A continuación se puede observar tanto la configuración en la que se colocaron los obstáculos en dos ocasiones distintas como los resultados de la mediciones correspondientes. 22 23.

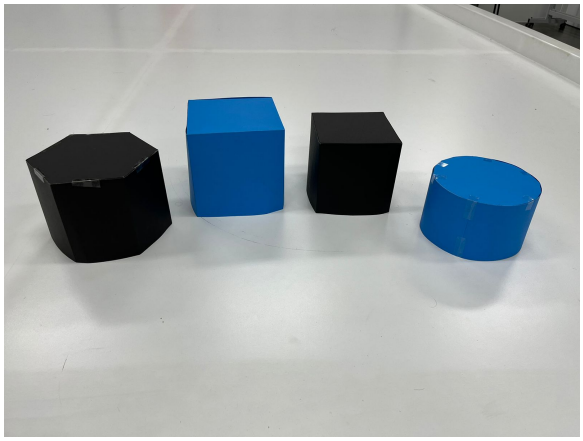
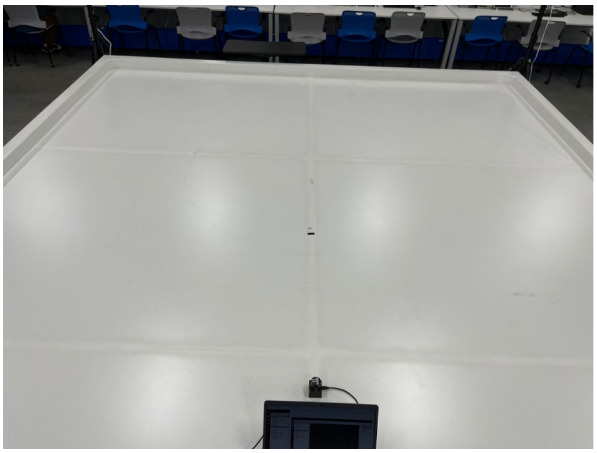


Figura 20: Obstáculos utilizados para pruebas



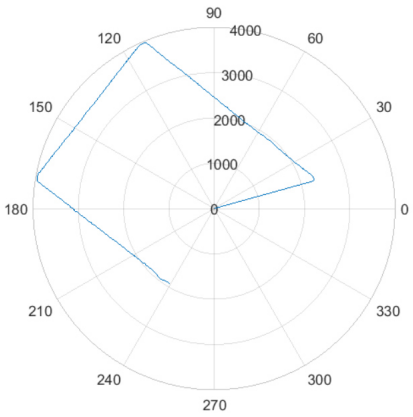
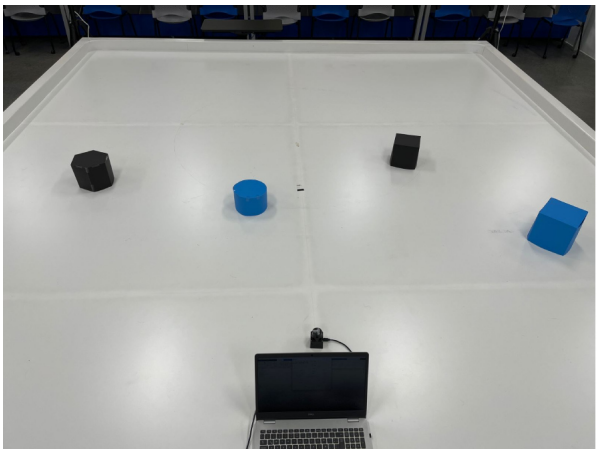


Figura 21: Ambiente vacío y lectura sensor



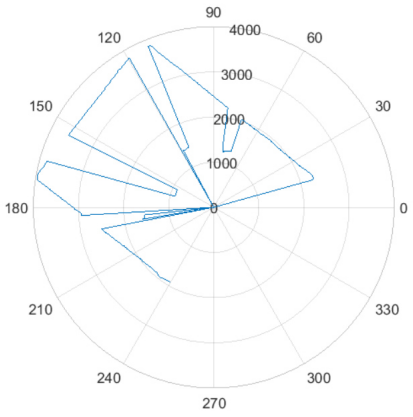
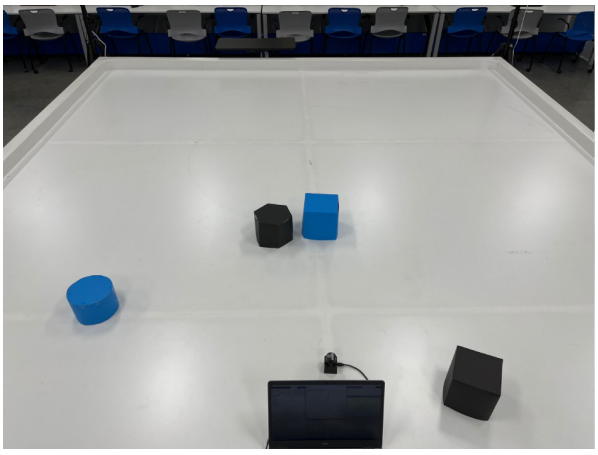


Figura 22: Configuración 1 y lectura sensor



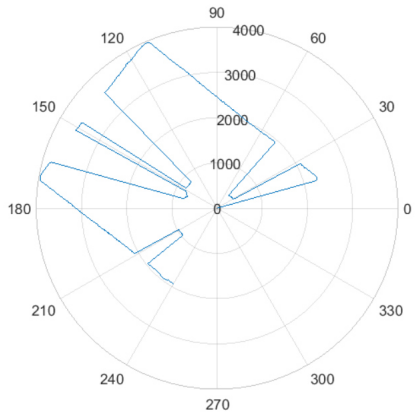


Figura 23: Configuración 2 y lectura sensor

CAPÍTULO 8

Algoritmos de SLAM disponibles para robots móviles con ruedas

El problema de SLAM en la industria cuenta con múltiples soluciones tales como EKF-SLAM, FastSLAM, Visual Odometry SLAM, entre otros. Sin emabrgo, el algoritmo a utilizar en este proyecto será *Post-Graph SLAM*, también conocido como SLAM mediante gráficos de pose. Esta selección de algoritmo se limita a las herramientas disponibles en ROS2.

8.1. Algoritmos disponibles en ROS2

ROS2 actualmente cuenta con 2 herramientas distintas para realizar tareas de navegación, localización, mapeo y similares, las cuales serán descritas a continuación.

8.1.1. Cartographer

Cartographer es un conjunto de librerías desarrolladas originalmente para ROS por Google en el año 2017 las cuales utilizan el algoritmo de SLAM basado en filtrado de partículas o *particle filter-based SLAM* . A pesar que sí existe y actualmente está disponible el paquete *Cartographer* para ROS2, este no funciona de manera óptima. Desafortunadamente, Google en conjunto con el equipo de desarrolladores de esta herramienta, no mantuvieron el soporte de este nuevo paquete. Debido a esto, el paquete se encontraba incompleto y desarrollado para versiones anteriores de ROS2 a la utilizada en este proyecto, operando correctamente solo para ciertos modelos de robots específicos. En abril de 2022 la *Open Source Robotics Foundation* en conjunto con algunas empresas como *WYCA Robotics* retomaron el desarrollo y soporte de este paquete y publicaron una nueva versión nombrada *Cartographer-v2* para *ROS2 Humble*, el cual se encuentra actualmente en estado de "prueba". A pesar de la nueva publicación, el paquete sigue funcionando de forma óptima únicamente para ciertos

modelos de robots de las empresas afiliadas por lo que no es posible utilizarlo con los recursos actuales como lo es el sensor Hokuyo URG-04LX. [19]

8.1.2. SLAM Toolbox

El *SALM Toolbox* es un conjunto de herramientas que forman parte del paquete de ROS2 *Navigation 2*. Este *toolbox* fue creado por Steve Macenski en el año 2021. Las 2 tareas realizadas con mayor frecuencia por medio de este paquete son *Adaptive Monte Carlo Localization*, AMCL: por sus siglas en inglés; y SLAM que es la tarea de nuestro interés. Este toolbox tiene la peculiaridad de que puede ser utilizado con cualquier tipo de sensor láser siempre y cuando se cuente con los *drivers* necesarios para enviar la información captada por el sensor en formato /LaserScan_msgs. Pues este paquete se conecta automáticamente a los *topics* /scan para la información de la medición de distancias y /odom"para los datos de la odometría del robot. [20]

Variaciones de SLAM y sus aplicaciones

SALM Toolbox utiliza los algoritmos para realizar *Pose-Graph SALM* y cuenta con 4 variaciones del mismo para realizar distintos tipos de mapeo. Estas variaciones son las siguientes:

- Offline Mapping: Esta versión no realiza mapas en tiempo real, sino que espera que se le entregue un conjunto de datos pertenecientes tanto a las distancias detectadas por un sensor como a la odometría del robot a lo largo del tiempo en un conjunto de archivos conocidos como *Bag Files* y en base a toda esta información genera un mapa. Esta variante aunque no muchos la consideran SLAM, pues no se realiza de forma simultanea, es una herramienta que, por ejemplo, permite recrear mapas para uso futuro a robots que no cuentan con la capacidad suficiente para mapear y esta información es trasladada a otra computadora que realizaría la tarea por ellos.
- Synchronous Mapping: Esta versión si realiza mapeo simultáneo y se encarga de tomar en cuenta cada una de las mediciones realizadas, mediante un buffer de datos. Esto aunque puede generar *lag*, es decir un pequeño desface en tiempo en cuanto a la posición y la vista actual del robot y lo que se puede observar en el mapa en ese mismo instante. Esta variante suele utilizarse en aplicaciones que buscan trabajar en el mismo espacio durante un periodo de tiempo largo o incluso indefinido, esto debido a que aunque exista este pequeño retraso en la generación del mapa, lo que se busca es conocer el área de la mejor manera posible. Esta modalidad sin embargo, suele utilizarse mayormente en robots que cuentan con un control remoto, pues debido a el retraso que se genera con la creación del mapa puede ser peligroso que robots de navegación autónoma utilicen esta información como fuente de datos para toma de decisiones.
- Asynchronous Mapping: Esta variante también realiza mapas de forma simultanea. Sin embargo, este no toma en cuenta los datos de todas las lecturas, pues para prevenir un retraso entre el estado actual del robot y la generación del mapa, mantiene el mapa

actualizado lo más apegado a tiempo real posible aunque se pierda información. Esta versión es bastante recomendada para robots que se movilizan a altas velocidades o que no se movilizan en la misma área recurrentemente así como los robots de navegación autónoma. Un ejemplo de esta aplicación pueden ser los carros autónomos, pues para que el funcionamiento del sistema de mapeo sea exitoso se prefiere tener una respuesta lo más apegada a la realidad posible para que el robot pueda tomar decisiones de forma rápida según la lectura actual. Los robots que utilizan este tipo de mapeo para realizar SLAM rara vez tienen la tarea de explorar un área, mas bien se enfocan en tomar decisiones según su entorno actual como evasión de obstáculos por ejemplo.

- Life Long Mapping: Esta última versión del algoritmo permite realizar mapas de forma simultánea al movimiento pero al mismo tiempo si el robot se apaga y se vuelve a encender el algoritmo es capaz de partir del mapa iniciado anteriormente para continuar con la construcción del mismo. Aunque esta herramienta es bastante útil no es posible implementarla en cualquier sistema. Esto debido a que al reiniciar la operación del robot se debe realizar un proceso previo para saber conocer la posición actual del robot, pues no siempre se iniciarán operaciones en el mismo punto y esta no es información que se pueda obtener por medio de drivers por ejemplo. Entonces es necesario contar con un sistema de localización más sofisticado y esto no siempre es posible en operaciones en ambientes no controlados o fuera de laboratorios.

Debido a que en este proyecto se pretende realizar SLAM dentro de un ambiente de laboratorio controlado y se busca que la obtención del mapa sea en el menor tiempo posible se realizarán pruebas únicamente con las variantes de *Synchronous Mapping* y *Asynchronous Mapping*.

El impacto del soporte de ROS en el desarrollo de herramientas disponibles para SLAM en ROS2

Debido a que ROS, en todas sus versiones, es un sistema operativo *Open Source* cuyo repositorio está compuesto por aportes provenientes de distintos "colaboradores individuale". Es necesario que los usuarios trabajen con la herramienta para ir equipando el repositorio general según sus intereses. Esto aunque es un modelo de desarrollo bastante ergonómico, pues la documentación de temas en tendencia o de interés en una cierta área irá aumentando según los mismos usuarios trabajen en dicha área, toma un período de tiempo extenso en lograr que el repositorio se equipe con múltiples opciones de herramientas para realizar tareas similares o una misma tarea de distintas formas.

En este proyecto se decidió trabajar con ROS2 pues es la versión más actualizada del sistema operativo y se busca que el proyecto quede vigente durante la mayor cantidad de tiempo posible. Sin embargo, se debe tomar en cuenta que esta nueva versión de ROS fue lanzada en mayo de 2021 lo que convierte a la herramienta en algo relativamente nuevo. Así mismo en necesario tener en cuenta que la versión anteriores de ROS cuenta con soporte actualmente y este finalizará hasta el año 2023. Esto genera una etapa de transición muy marcada, pues los usuarios más experimentados de ROS continúan trabajando con la versión original del sistema operativo y los usuarios nuevos, como lo es la UVG, hemos tendido a iniciar trabajos en la nueva versión. Esta tendencia ha generado un retraso en el crecimiento

de la documentación y desarrollo de herramientas para ROS2.

CAPÍTULO 9

Instalación y manejo de ROS2

Este proyecto se realizó en la versión ROS2 Foxy Fitzroy. Esta versión de ROS2 se puede utilizar dentro de los sitemas operativos Windows 10 y distintas variantes de Linux. Sin embargo, se decidió utilizar el sistema operativo de Linux Ubuntu 20.04 como un estándar en los distintos módulos a trabajar para la plataforma Rover UVG para garantizar una instalación completa y sin restricciones de los distintos módulos necesarios así como la compatibilidad de los distintos nodos a trabajar.

Para esto se utilizó una máquina virtual VMWare Workstation con las siguientes características:

- Versión: 16.2
- RAM: 4096 MB
- Almacenamiento: 25GB

9.1. Instalación de ROS2

La instalación de ROS2 se realizó por medio de *Debian packages*, esto para garantizar que todas las dependencias y herramientas tanto de simulación y visualización se instalaran de forma automática. Otro punto importante es que la instalación de esta versión de ROS2 se encuentra disponible tanto para arquitectura amd64 como para arm64, lo que permitió tanto realizar pruebas en la máquina virtual cómo la realización de pruebas con el Rover UVG en la computadora a bordo de la plataforma, RaspberryPi 4.

El proceso de instalación se realizó siguiendo la documentación oficial de ROS2 Foxy, la cual se encuentra dispoible en línea en el siguiente enlace: [21]

9.2. Creación de espacios de trabajo y paquetes en ROS2

9.2.1. Creación de espacios de trabajo

Un espacio de trabajo en ROS2 es básicamente una carpeta que dentro ordena archivos de forma específica para que el sistema operativo comprenda de forma correcta tanto la información de cada archivo como la función de los mismos. Los comandos necesarios para crear un nuevo *workspace*, el cual llamaremos *ROS-ROVER-UVG* son las siguientes:

```
$ cd ~/Workspaces
$ source /opt/ros/foxy/setup.bash
$ mkdir -p ~/ROS-ROVER-UVG/src
$ cd ~/ROS-ROVER-UVG/src
$ colcon build
```

Como es posible observar en los comandos anteriores, el espacio de trabajo se generó dentro de una carpeta ya existente nombrada *Workspaces*, esto aunque no es necesario se considera una buena práctica pues de esta forma se pueden trabajar proyectos de ROS independientes entre sí sin perder el orden.

Otro punto importante es la creación de la carpeta src dentro del espacio de trabajo. El nombre src proviene de *source*, que significa fuente u origen. En la estructura de ROS2 es imperativo que dentro de cada espacio de trabajo exista una capeta con este nombre dentro de la cual se generarán los distintos paquetes a utilizar en el proyecto.

Por último, el comando *colcon build* se utiliza para compilar el espacio de trabajo, esta acción genera tres nuevas carpetas: *build*, *log* e *install*.

9.2.2. Creación de paquetes

ROS2 trabaja con los lenguajes de programación CMake o Python y el sistema operativo es reconocido por trabajar de la misma forma sin importar el lenguaje que se utilice. Otra característica es que los distintos lenguajes pueden mezclarse dentro de un mismo proyecto sin afectar el funcionamiento del mismo.

Para la generación de paquetes en ROS2 se puede elegir si la estructura del mismo se genere en CMake o Python y como se menciona anteriormente, indiferentemente de la estructura seleccionada los archivos de programación pueden ser escritos en cualquiera de los dos lenguajes permitidos.

En esta ocasión se seleccionó la estructura CMake. Los requerimientos mínimos para el correcto funcionamiento de esta estructura se basa en los siguientes dos archivos:

- package.xml: archivo que contiene información acerca del paquete.
- CMakeLists.txt: archivo que describe cómo compilar el código dentro del paquete.

Estos archivos se generan de forma automática al crear el paquete mediante el siguiente comando:

```
$ ros2 pkg create --build-type ament_cmake <package_name>
```

Antes de ingresar el comando anterior se debe asegurar encontrarse dentro de la carpeta src de nuestro espacio de trabajo. Para crear el paquete dentro del directorio indicado.

En esta ocasión, debido a la guía que se utilizó para aprender a utilizar el *Nav2_pkg* el nombre del paquete es *basic_mobile_robot*. Por lo que los comandos para la generación de nuestro paquete fueron los siguientes:

```
$ cd ~/Workspaces/ROS-ROVER-UVG/src
$ ros2 pkg create --build-type ament_cmake basic_mobile_robot
$ cd ..
$ cd ..
$ colcon build
```

CAPÍTULO 10

Lectura de datos extraídos del escáner Lidar Hokuyo URG-04LX-UG01 mediante drivers en ROS2

A pesar que los métodos de obtención de información mencionados anteriormente son funcionales, no son la forma óptima de trabajar con el sensor para aplicaciones como el proyecto Rover UVG. Esto debido a distintas cuestiones. La primera razón se debe al tiempo de actualización de los datos, pues el método anterior tarda al rededor de 3 segundos en actualizar la gráfica, con mediciones nuevas. Por otro lado, algo crucial para trabajar de forma óptima con ROS2 es transmitir la data en formatos conocidos por el sistema operativo y el método anterior no entregaba la data en un formato estándar que trabaje ROS2.

Independientemente a que la fábrica Hokuyo únicamente provee drivers para este modelo de sensores para ROS, existe un nodo que contiene la "traducción"de los *Driver:* para los sensores Hokuyo URG de ROS a ROS2. Este paquete está registrado como *urg_ node* en el repositorio de ROS2 y nos permite obtener lecturas con actualización cada 100 ms aproximadamente. A su vez entrega los datos en el formato */LaserScan_ msgs* el cual es un formato estandarizado para ROS2 que permite intercambiar de sensores con facilidad, debido a que independientemente de la marca o modelo del sensor la data es traducida al mismo lenguaje para ser compatible con cualquier proyecto que requiera lecturas de distancia obtenidas por medio de un sensor láser. [22]

10.1. Inicialización y ejecución del *urg_ node* desde la termial de comandos

EL primer paso para poder utilizar este nodo de ROS2 es su instalación, para esto se utilizó una terminal de comandos en la cual se ingresó lo siguiente:

```
$ sudo apt update
$ sudo apt install ros-foxy-urg-node
```

Habiendo finalizado la instalación se abrió una nueva terminal en la cual se permitió la comunicación del sensor por medio de un puerto serial, ejecutando el siguiente comando no solo se indicó el puerto a utilizar sino se dieron las autorizaciones y permisos necesarios a dicho puerto.

```
$ sudo chmod 777 /dev/ttyACM0
```

Habiendo realizado esto pudo correrse el nodo de forma exitosa mediante el siguiente comando:

```
$ ros2 run urg_node urg_node_driver --ros-args -p serial_port:=/dev/ttyACM0
```

Luego de haber corrido el nodo, mediante los siguientes dos comandos se revisó primero que el *topic* al que se estaba publicando la información estuviera activo y luego se realizó un *echo* de este *topic* para revisar que la información se estuviera enviando de forma adecuada.

```
$ ros2 topic list
$ ros2 topic echo /scan
```

En la siguiente Figura 24 se puede observar la estructura del los mensajes que se están publicando en el *topic* /scan luego de correr el nodo.

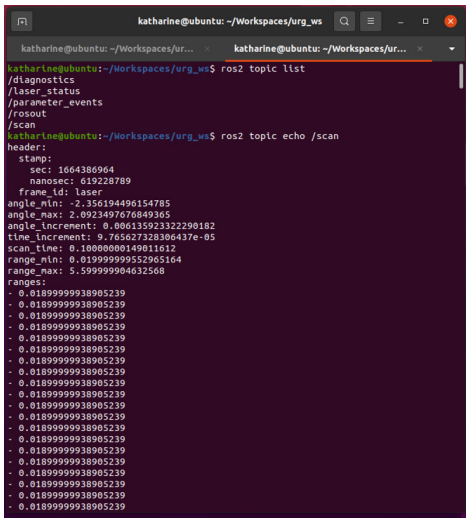


Figura 24: Estructura de mensajes /LaserScan_msg publicados en el topic /scan

10.2. Creación de archivo ejecutable para el nodo *urg_node*

La motivación de haber creado un archivo ejecutable para correr este nodo se debe a que la lectura del sensor se utiliza en la próxima fase de este proyecto como una de las entradas

para el *slam_ toolbox* por lo que era poco efectivo tener que ingresar múltiples comandos desde la terminal para poder inicial la lectura de datos del sensor y luego correr el programa encargado de realizar SLAM.

10.2.1. Creación de archivo ejecutable en lenguaje python

El archivo ejecutable, conocido como *launch file* en inglés, en este caso se creó como un archivo de python en formato py. También se creó un archivo yaml con el fin de configurar los parámetros para la comunicación y el sensor de una manera más sencilla y ordenada. Por lo mismo fue necesario generar dos nuevas carpetas dentro de nuestro paquete, se generó una llamada *params* dentro de la cual se guardó el archivo urg.yaml con los parámetros mencionados anteriormente y otra llamada *launch* donde se guardó el archivo ejecutable urg_launch.py.

Para la creación de las carpetas dentro del paquete se ingresaron los siguientes comandos dentro de la terminal:

```
$ cd ~/Workspaces/ROS-ROVER_UVG/src/basic_mobile_robot
$ mkdir launch params
```

Luego se generó el archivo de configuraciones de la siguiente manera:

```
$ cd ~/Workspaces/ROS-ROVER_UVG/src/basic_mobile_robot/params
$ mkdir urg.yaml
```

Los parámetros incluidos en el archivo se pueden observar en la siguiente imagen:

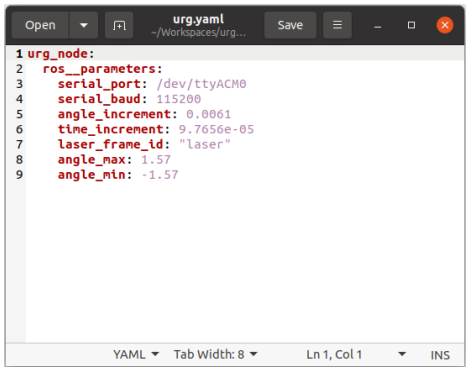


Figura 25: Parámetros de configuración urg_launch.py

Para finalizar se generó el archivo de python, urg_launch.py, dentro de la carpeta *launch* mencionada anteriormente el cual se puede revisar en los anexos y nuevamente se compiló el paquete completo ya con los nuevos archivos generados. En este archivo ejecutable, luego de importar las librerías necesarias y definir las funciones requeridas se escribió el siguiente fragmento de código, e el cual se puede observar como se define la dirección del archivo de

configuración de parámetros, el nombre y archivo específico del nodo que desea ejecutarse y el nombre del *topic* en el que se desea publicar la información obtenida.

```
param_file = os.path.join(get_package_share_directory('basic_mobile_robot'),
'params', 'urg.yaml')
return LaunchDescription([
Node(
name= 'urg_node',
package = 'urg_node',
executable = 'urg_node_driver',
parameters = [param_file],
remappings=[('scan','scan'),]
)
])
```

Para corroborar la funcionalidad de este método de lectura se realizaron pruebas similares a las realizadas con el método anterior. Iniciamos corriendo nuestro archivo ejecutable con los siguientes comandos, en una nueva terminal de comandos:

```
$ cd ~/Workspaces/ROS-ROVER-UVG/src/basic_mobile_robot
$ source install/setup.bash
$ ros2 launch basic_mobile_robot urg_launch.py
```

Como se observa en 26 luego de ingresar esta serie de comandos ROS2 nos indicó que el nodo ya estaba corriendo de forma correcta y que los datos estaban siendo publicados sin anomalías.

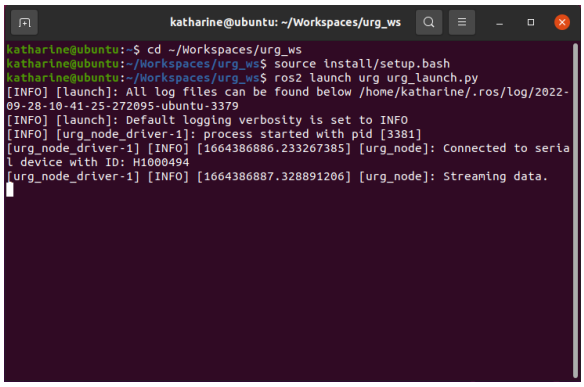


Figura 26: Terminal de comandos corriendo el archivo urg_launch.py

Se continuó ingresando los siguientes comandos con los que se revisó que el *topic scan* estuviera enviando la información deseada y que esta se enviara en el formato correcto.

```
$ ros2 topic list
$ ros2 topic echo /scan
```

Para finalizar, en una nueva ventada de la terminal de comandos se abrió el visualizador de ROS2 *Rviz* en el cual se agregó un nuevo sensor láser y se indicó de qué *topic* se necesitaba leer la información. Habiendo hecho esto fue posible realizar las mismas pruebas que con el método anterior, los resultados obtenidos fueron los siguientes:

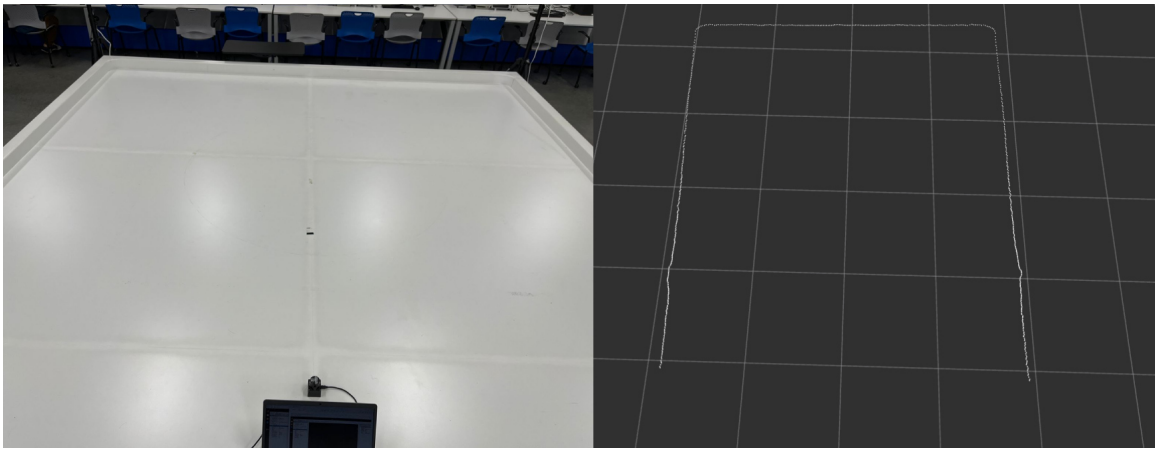


Figura 27: Ambiente vacío y lectura sensor

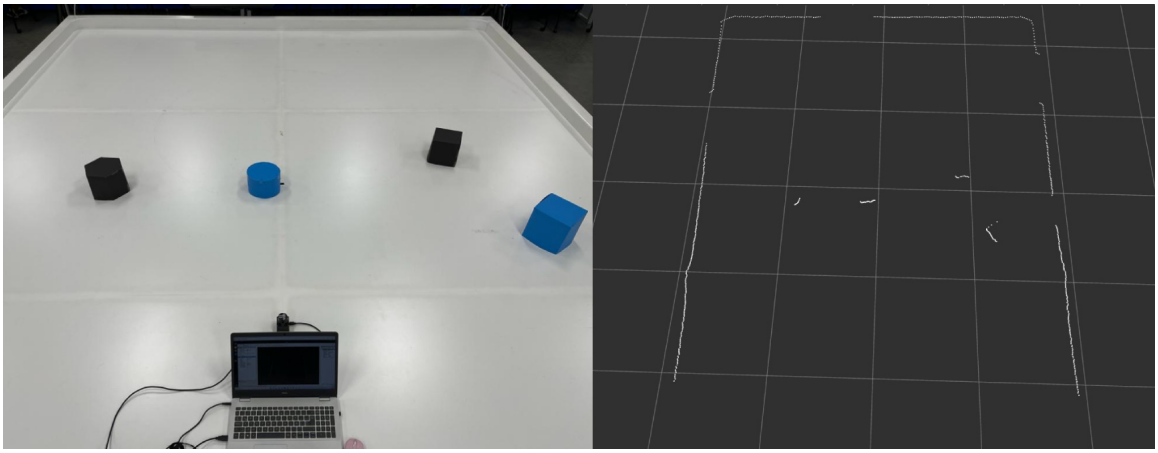


Figura 28: Configuración 1 y lectura sensor

Aunque los resultados obtenidos no se pueden observar muy claramente, esta metodología de obtención de datos presenta las siguientes ventajas ante la metodología anterior:

- Cada medida de distancia se representa como un punto individual y no como parte de una línea continua.
- El tiempo de actualización de datos es considerablemente menor.
- La información se visualiza sobre un plano de trabajo específico, el cual se define dentro del archivo urg.yaml.

—

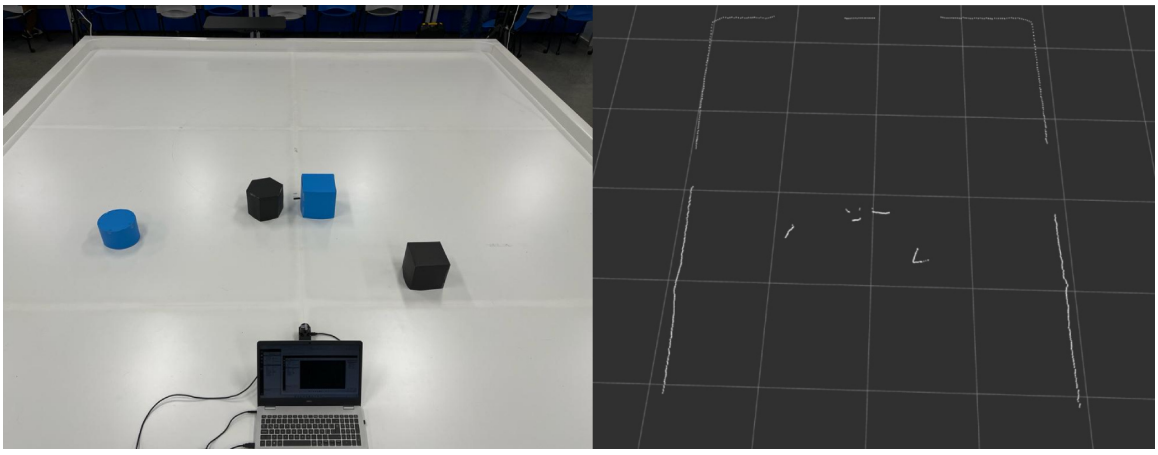


Figura 29: Configuración 2 y lectura sensor

CAPÍTULO 11

Implementación de SLAM en ROS2

Para la implementación de SLAM mediante el *slam_ toolbox* se decidió iniciar experimentando en simulación, para así comprender de una mejor manera el funcionamiento no solo de SLAM sino de todas las herramientas disponibles dentro del *toolbox*. Luego de realizar pruebas en simulación y confirmar el funcionamiento de los algoritmos y las configuraciones realizadas se realizaron las modificaciones necesarias para poder realizar una implementación física con los componentes disponibles en la Universidad del Valle de Guatemala.

Para esto es indispensable comprender la diferencia entre las 2 herramientas que incluye ROS2 que nos ayudan a realizar pruebas de este tipo. El simulador Gazebo, el cual nos permite experimentar con la mayoría de elementos y nodos de ROS2 sin contar con los elementos físicos permitiendo crear entornos virtuales completos, dentro de los cuales se pueden definir obstáculos, modelos de robots e incluso la dinámica de estos robots y las características físicas de cada uno de sus componentes. Por otro lado está Rviz, este es un visualizador exclusivamente, que permite observar la información publicada en los distintos *topics* de forma gráfica. Sin embargo esta aplicación no genera o publica información de ningún tipo, únicamente despliega la información generada por los distintos nodos.

11.1. Implementación mediante simulación en Gazebo

Para poder realizar SLAM en simulación, fue necesario realizar otras pruebas y preparar todo el entorno virtual previamente.

11.1.1. Familiarización con *slam_ toolbox* y gazebo

El proceso de la preparación del entorno virtual y configuración del paquete Nav2 se detalla a continuación.

Definición del Robot

Para la definición de robots, ROS2 utiliza el formato universal de descripción de robot, URDF por sus siglas en inglés. Estos son archivos formato Formato XML: en los cuales se describe al robot físicamente. Para esto se definen tanto las piezas que conforman al robot como las articulaciones entre dichas piezas, con el fin de definir cómo se ve el robot y también que movimientos que es capaz de realizar.

Para esta prueba se realizó un modelo básico de un robot uniciclo, similar al Rover UVG. Pues, aunque no es un modelo de iguales dimensiones y componentes, pero que su comportamiento es igual al que idealmente se espera del Rover UVG. Para esto el modelo se simplificó a un prisma rectangular con las siguientes medidas: 40cm de ancho, 70cm de largo y 20cm de altura (color rojo); y dos ruedas (color gris), una en cada lado, con un radio de 14cm. Además se especificó que el robot cuenta con un *lidar* en la parte superior (color negro). Por otro lado, para obtener los datos de la odometría también se definió una IMU en el centro del modelo virtual y *enconders* en cada una de las llantas. Para poder visualizar el robot en el visualizador Rviz se asignaron modelos 3D, en formato Formato STL:, de cada una de las partes descritas en el archivo URDF. En la Figura 30 se puede observar el resultado de la definición del robot en Rviz.

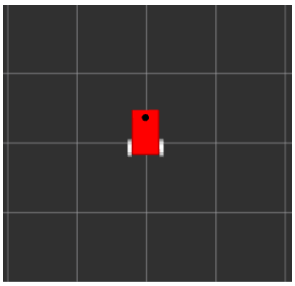


Figura 30: Modelo robot virtual en Rviz

Algo que cabe resaltar es que es en este archivo es donde se deben de definir los marcos de referencia necesarios para poder realizar cualquiera de las tareas de navegación. Por lo mismo, si no se asignan modelos a las partes del robot descritas en el archivo el visualizador desplegará únicamente las guías de los marcos de referencia. Sin embargo, este procedimiento debe realizarse tanto para las aplicaciones en simulación como para las físicas, pues la definición del robot y su funcionamiento es fundamental para poder realizar los cálculos de las tareas de navegación, control, etc.

Por otro lado, para la definición del robot específicamente dentro del simulador Gazebo, se creo un archivo SDF con el mismo contenido que el archivo URDF generado para la visualización, pero esta ves agregando características de funcionamiento cómo los parámetros

de medición del sensor Lidar. En la siguiente Figura 31 se puede observar el modelo del robot uniciclo básico Gazebo, en donde las líneas azules describen el rango de medición del sensor. En donde el rango de medición lineal y radial se configuraron para que fueran lo más similares posible a las especificaciones del sensor Hokuyo que se utilizará en las pruebas físicas.

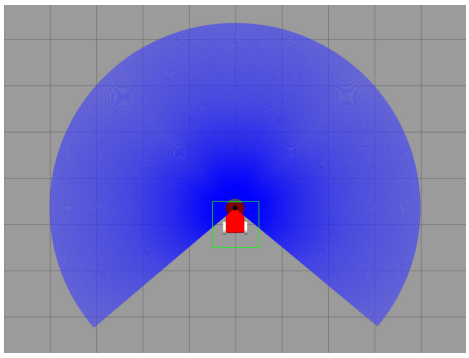


Figura 31: Modelo robot virtual en Gazebo

Creación de mundo virtual

Gazebo: es un simulador 3D que permite crear ambientes con los que interactúan robots virtuales. A pesar que este simulador cuenta con muchas distintas configuraciones para crear ambientes virtuales lo más apegados a la realidad del entorno en el que se busca trabajar de forma física que permites configurar desde la percepción de la luz solar en el espacio hasta la colocación de objetos por medio de librerías con distintos modelos tales como muebles, carros, edificaciones, entre otros; se decidió realizar las pruebas con obstáculos lo más sencillos posibles, pues en esta ocasión no se buscó evaluar el funcionamiento del toolbox sino únicamente validar que este era capaz de realizar las tareas que se necesitaban para el proyecto.

Para esto, en vez de utilizar modelos de las librerías descritas anteriormente o incluso utilizar la herramienta para construir edificaciones dentro de Gazebo, se decidió colocar únicamente cubos como obstáculos dentro del mundo en el que el robot navegaría. Estos cubos se colocaron de forma arbitraria dentro del mundo virtual, lo cual se puede observar en la siguiente imagen.

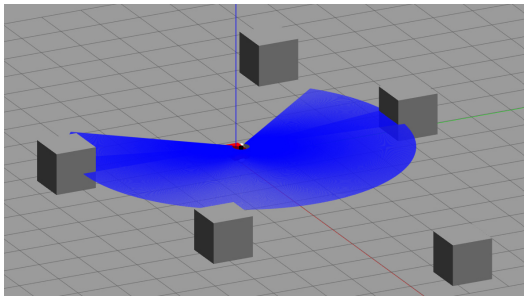


Figura 32: Vista en Gazebo del robot en un mundo con obstáculos simples

Movilización del robot dentro del mundo virtual

Debido a la forma en la que se definió el robot nos es posible utilizar la aplicación de ROS, desarrollada por Dirk Thomas, rqt_robot_steering para la movilización del robot. Esta aplicación forma parte del paquete *ROS Visualization* y nos permite enviar mensajes a las ruedas de nuestro robot por medio del *topic* /cmd_vel. Este funciona mediante una pantalla emergente 33 la cual contiene controles para indicar la velocidad lineal y radial del robot.

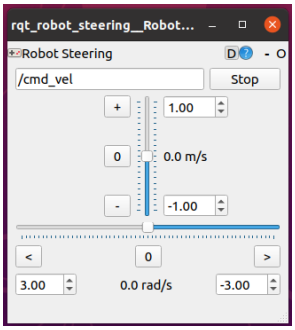


Figura 33: Pantalla aplicación rqt_robot_steering

11.1.2. SLAM en simulación

Para esta prueba se decidió realizar SLAM utilizando las metodologías de mapeo *online* síncrono y asíncrono que incluye el *slam_toolbox*, para poder después comparar los resultados obtenidos con cada una. Para esto se verificó que la información del sensor Lidar en simulación se esté publicando en el *topic* /scan y tanto la IMU: como los *enconders* estén publicando la información de la odometría en los *topics* correspondientes, pues estos son los canales de comunicación definidos en el nodo de *SLAM Toolbox* para recibir la información de odometría. El algoritmo exacto utilizado por el *toolbox*, debido a que el algoritmo de *Pose-Graph SLAM* se encuentra dentro de un nodo registrado en el repositorio general de ROS2, no es un archivo editable. Sin embargo, mediante un archivo de compilación, mejor conocidos como *launch files* en inglés, se puede asignar un archivo tipo YAML para definir los parámetros con los que correrá el algoritmo.

Los siguientes fragmentos de código indican como configurar la localización del archivo de configuraciones para realizar el mapeo del SLAM al igual que como llamar al nodo del *toolbox* para correr el algoritmo deseado.

Declaración de la localización del archivo de parámetros de mapeo dentro del paquete

```
robot_slam_file_path = os.path.join(pkg_share, 'params/mapper_params_online_async.yaml')
```

Definición e inicialización del nodo que se desea correr

```
start_async_slam_toolbox_node = Node(
       parameters=[robot_slam_file_path,
```

```
            {'use_sim_time': use_sim_time}],
        package='slam_toolbox',
        node_executable='async_slam_toolbox_node',
        name='slam_toolbox',
        output='screen')

ld.add_action(start_async_slam_toolbox_node)

return ld
```

A continuación se puede observar como el robot en simulación toma datos con el escáner láser, el rango detectado está en azul, en Gazebo. Del otro lado (derecho) de la Figura 34 se observa la pantalla de RViz con la pose actual del robot y el mapa generado hasta ese momento. Como se había mencionado anteriormente el mapa generado es un *Occupancy Grid Map* donde el área gris es el espacio no reconocido, el área blanca es el espacio libre y los pixeles negros representan los bordes de los obstáculos.

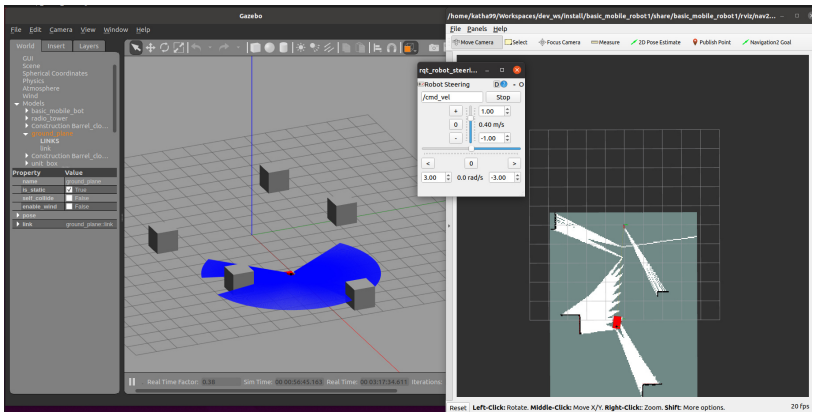


Figura 34: Pantalla de funcionamiento de SLAM en simulación.

Para realizar esta misma prueba pero con mapeo síncrono únicamente se debe cambiar el nombre del nodo que se desea ejecutar en vez de llamar al nodo *async_slam_toolbox_node* se modifica el comando para llamar al nodo *sync_slam_toolbox_node* y a su vez, según la aplicación, se podrían modificar los parámetros del mapeo en el mismo o distinto archivo, en donde la diferencia más relevante será que el *buffer* de datos de la información del sensor de distancia para construir el mapa será distinto a 1.

11.2. Implementación de SLAM en simulación para el Rover UVG

En esta etapa se modificó lo trabajado anteriormente para realizar pruebas en simulación más cercanas a las pruebas que se planificó realizar en físico. Para esto en la visualización y dimensiones de simulación se utilizó el modelo URDF de la plataforma ROVER UVG desarrollado por Diego Gerardo Mencos Caal en otro módulo del proyecto ROVER UVG. De la misma forma se verificó el rango de medición del sensor Lidar, pues el sensor en

simulación es capaz de alcanzar los 360.º, sin embargo el sensor Hokuyo URG-04LX-UG01 tiene un rango máximo de 270.º al igual que el rango lineal del sensor de físico es de 4m de radio y en simulación este dato no tiene límites. Para esta prueba, nuevamente, se colocaron obstáculos en la simulación en forma de cubos con perímetros similares a los obstáculos con los que se realizaron las pruebas físicas, esto con el fin de observar como se comportaba el algoritmo con dimensiones más cercanas a los que se planeó utilizar en las pruebas físicas acorde a lo disponible en la universidad.

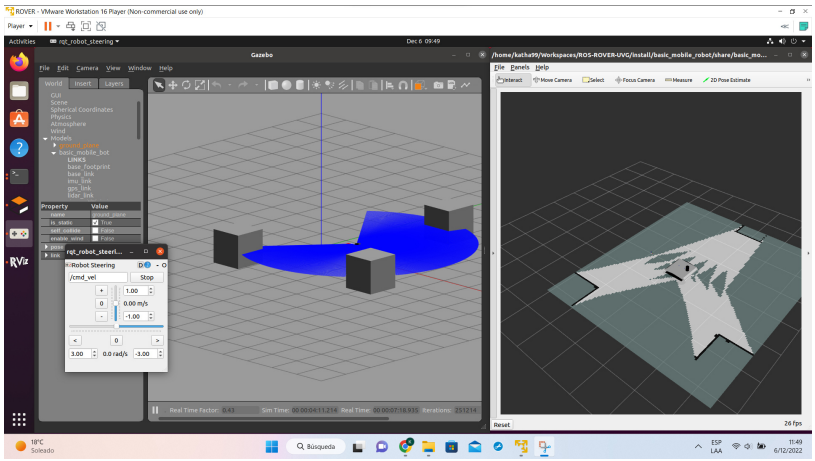


Figura 35: SLAM en simulación para el Rover UVG

En estas pruebas, como se puede observar en la imagen anterior, el robot que se visualiza en Rviz es el Rover UVG en vez del modelo simplificado que se había utilizado anteriormente. Sin embargo, este cambio de modelos no se modificó en Gazebo, pues no era necesario cambiar los modelos siempre y cuando los marcos de referencia de ambos archivos sean equivalentes, además que el Rover UVG no utiliza ruedas para movilizarse sino que tiene orugas, lo cual complica la definición de la dinámica del robot.

11.3. Implementación de SLAM física

En cuanto a la implementación física se realizaron modificaciones nuevamente con el fin de realizar pruebas más cercanas a la implementación final. Como modificación principal, para obtener la información de la odometría se utilizó el nodo de ROS2 desarrollado por Santiago Enrique Fernández Matheu, *optitrack_ odometry* el cual nos permite leer la pose de los *markers* del sistema de captura de movimiento. Entonces sobre el sensor lidar se colocó uno de estos *markers* para tener los datos de odometría requeridos. Por otro lado, pensando en la colocación del sensor en el ROVER UVG el rango del sensor se redujo a 180.º para evitar que el mismo robot interfiera en la lectura del sensor, pues para poder generar mapas del área controlada únicamente y que las lecturas del sensor lidar no salieran de la plataforma Robotat, este se colocó en la parte inferior del robot, lo más cercano al piso posible.

Luego debido a que tanto el nodo para la lectura del sensor y el algoritmo de SLAM se corrieron en la computadora Raspberry Pi que estará a bordo del ROVER UVG se generó un nuevo *launch file* únicamente para tener una visualización clara de lo que ocurría en una

computadora distinta a la que corre los nodos de la misma forma se modificó el archivo que se correría en la Raspberry Pi para no realizar la visualización del sistema en Rviz dentro de esta computadora pues no es funcional ya que esta al estar a bordo del Rover UVG no cuenta con pantalla.

Por último, como se observa en la siguiente imagen36, se diseñó y fabricó una base en donde colocar el sensor para que fuera más sencillo movilizar el sensor con el *marker* del optitrack por la plataforma Robotat sin hacer interferencia en las mediciones y que el movimiento del sensor sea más suave y continuo.

Figura 36: Montaje de Sensor Lidar Hokuyo y marker de Optitrack para realizar SLAM físico

Los modelos 3D utilizados para esto son los siguientes:

La primera pieza es un modelo original de Chris Rand, el cual cuenta con una rosca interna para poder acoplarlo a un palo de escoba. Esta pieza se observa en la Figura 37 .

Figura 37: Pieza acoplable a una bara mediante rosca

La segunda pieza, Figura 38, fue diseñada para crear una bisagra y así poder deslizar el sensor lidar a lo largo de la plataforma sin que este se incline.

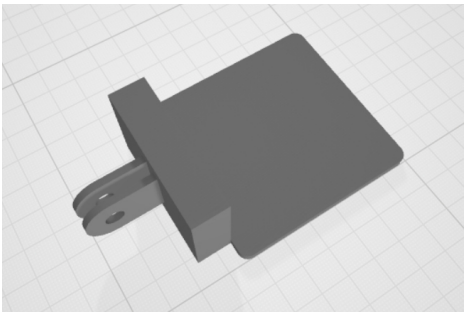


Figura 38: Base para montar el sensor lidar durante las pruebas físicas de SLAM

A continuación se puede observar, en las Figuras 39 y 40, cómo se realizaron las pruebas físicas y la información observada en la computadora independiente en donde se visualizaba la información generada.

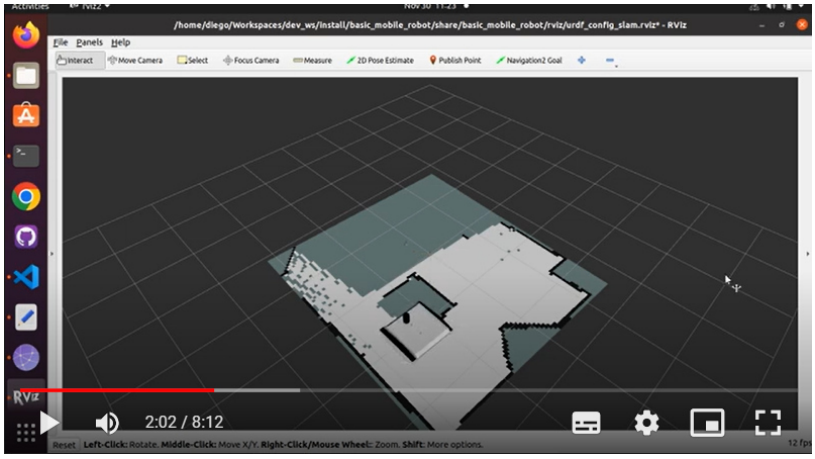


Figura 39: Toma de datos físicos en plataforma Robotat

En esta parte del proyecto se realizaron pruebas de SLAM con dos configuraciones distintas de obstáculos y en cada una se probó generar mapas con metodología de mapeo síncrona y asíncrona. A continuación se observarán los resultados obtenidos con las distintas pruebas realizadas.

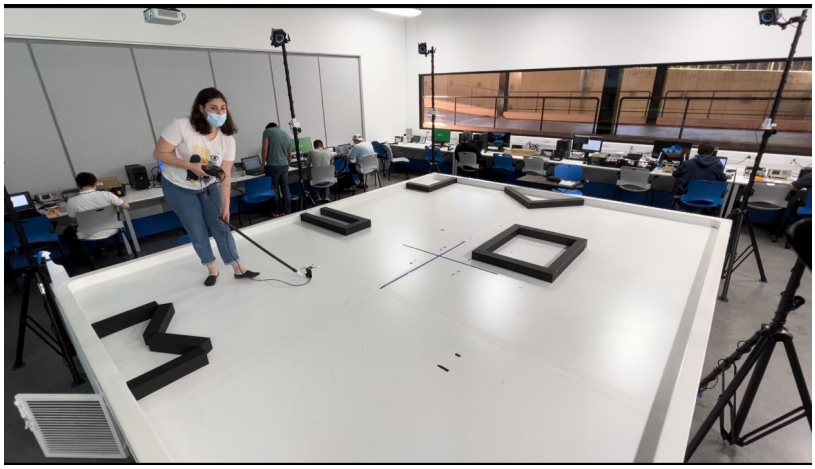


Figura 40: Visualización de datos físicos

CAPÍTULO 12

Almacenamiento de mapas y análisis de resultados obtenidos

12.1. Uso del nodo *map_ saver_ server* para generación de mapas

Con el fin de poder utilizar la data obtenida por medio de SLAM en futuros proyectos se implementó el nodo del paquete Nav2 *map_ saver_ server* que permite guardar en archivo Formato PGM: los mapas generados. A su vez se genera un documento YAML que indica los parámetros más relevante de la imagen generada, esto con el fin de poder analizar la imagen por métodos computacionales de forma más acertada.

12.1.1. Creación de *launch file* para la generación de imágenes de los mapas creados por SLAM

Nuevamente, para manejar cada acción del Rover UVG de forma eficiente y ordenada, se creó un archivo ejecutable para llamar al nodo *map_ saver_ server* al igual que un archivo YALM para predefinir ciertas configuraciones del nodo. A continuación se presenta un fragmento de código del archivo *map_ saver_ launch.py* en donde se inicializan 2 nodos distintos del paquete Nav2.

```
Node(
    package='nav2_map_server',
    executable='map_saver_server',
    output='screen',
    emulate_tty=True,
    parameters=[LaunchConfiguration('map_saver_params_file')]
    ),
```

```
Node(
    package='nav2_lifecycle_manager',
    executable='lifecycle_manager',
    name='lifecycle_manager',
    output='screen',
    emulate_tty=True,
    parameters=[
        {'use_sim_time': LaunchConfiguration('use_sim_time')},
        {'autostart': True},
        {'node_names': ['map_saver']}]
    )
```

Como es posible observar, los nodos utilizados para guardar los mapas como imágenes son los siguientes:

- *map_saver_server*: este nodo es parte del paquete *nav2_map_server*. Este nodo se encarga de traducir las matrices generadas en el *topic map* a imágenes.
- *lifecycle_manager*: este nodo forma parte del paquete *nav2_lifecycle_manager*. Este nodo se utiliza cuando es necesario administrar el ciclo de vida de todos los nodos que se están ejecutando para que no se interrumpan entre sí. Esto es necesario debido a que el nodo de *map_saver_server* no entrega resultados de forma continua sino que actúa como un servicio de ROS2 lo cual se detalla a continuación.

12.1.2. Uso de ROS2 *services*

Los servicios en ROS son otro método de comunicación entre nodos distinto a los *topics*. Como se ha explicado anteriormente, los *topics* utilizan un modelo de comunicación en donde la información es publicada en un canal de forma constante y otro nodo puede suscribirse a este *topic* para leer, la información publicada, también de forma continua. Por lo contrario, los servicios o *services* operan con una metodología basada en llamada-respuesta, esto se refiere a que aunque el nodo servidor, que es el que genera la acción, esté activo este no interactúa con ningún otro componente hasta recibir una señal departe de un nodo cliente, solicitado una acción y esta se realiza una única vez. Como se puede observar en la Figura 41.

Para la generación de imágenes esta metodología es óptima debido a que Rviz nos permite observar la generación del mapa en todo momento, y esta herramienta se utiliza únicamente cuando el usuario considera que el estado actual mapa generado es de su interés.

Para esto es recomendable correr el nodo al inicio de la generación del mapa, mediante el comando:

```
$ ros2 launch basic_mobile_robot map_saver_launch.py
```

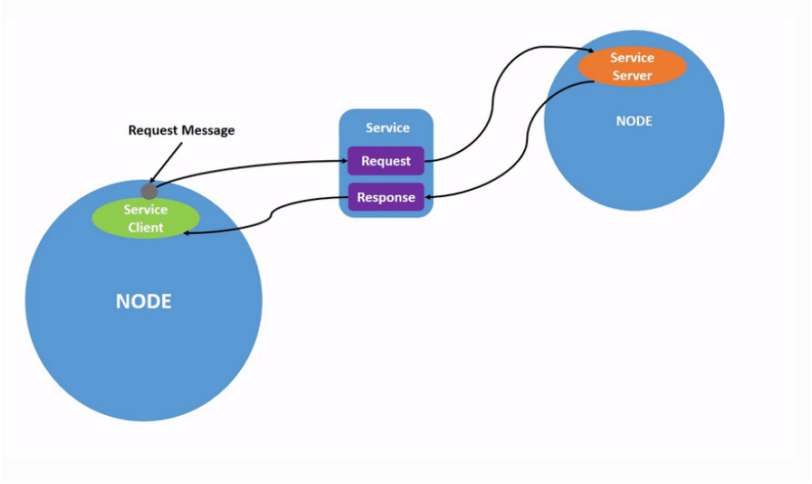


Figura 41: Diagrama de comunicación por medio de un servicio

De esta forma se puede asegurar que el servicio está disponible durante todo momento mientras se genera el mapa mediante SLAM. De esta forma es más sencillo poder solicitar el servicio en cualquier momento del proceso y capturar los mapas deseados. Esta solicitud se debe realizar por medio de una nueva ventada en la terminal de comandos y se solicita de la siguiente manera:

```
$ ros2 service call /map_saver/save_map nav2_msgs/srv/SaveMap "{map_topic: map,
  map_url: my_map, image_format: pgm, map_mode: trinary,
  free_thresh: 0.25, occupied_thresh: 0.65}"
```

En donde los parámetros detallan lo siguiente:

- map_topic: el *topic* del cual se desea leer la información.
- map_url: el nombre con el que se guardará el archivo .pgm, en el ejemplo se colocó my_map.
- image_format: se genera un archivo en escala de grises en formato .pgm
- map_mode: se indica que se desea una modalidad de mapa trinaria, es decir que solo se puede asignar 1 valor de 3 posibles a cada píxel.
- free_tresh: este valor indica la proporción máxima que debe tener ocupada una casilla para asignar el píxel como vacío. El valor de 0.25 es un valor recomendado por la documentación del *toolbox*.
- accupied_tresh: este valor indica la proporción mínima que debe tener ocupada una casilla para asignar el píxel como ocupado. El valor de 0.65 es un valor recomendado por la documentación del *toolbox*.

12.2. Comparación de mapas obtenidos con las distintas metodologías de mapeo

Como se explica en el capítulo anterior cada configuración de obstáculos colocados en el Robotat se exploró dos veces mediante SLAM. Esto para probar las dos opciones distintas de mapeo, de nuestro interés, que el *slam_ toolbox* permite realizar. En estas pruebas es posible observar que el resultado fue similar independientemente del tipo de mapeo que se estuviera realizando. Sin embargo es necesario tomar en cuenta que estas pruebas, debido al área disponible, el rango del sensor utilizado y la exactitud de la información de odometría utilizada, la capacidad del *slam_ toolbox* no fue llevada a su límite.

12.2.1. Imágenes generadas del Mapa 1

En la Figura 42 se puede apreciar pequeñas variaciones en ambos mapas, siendo las paredes del mapea síncrono un tanto más continuas que las del mapeo asíncrono. Sin embargo, se debe tomar en cuenta que los recorridos realizados por el sensor para cada prueba fueron distintas pues se realizaron de forma manual, es decir no se llevó un control exacto del movimiento del sensor pues este se movió manualmente. De igual forma es posible observar que ambas pruebas fueron exitosas, pues a pesar de las pequeñas diferencias ambos mapas captaron una geometría muy similar, por lo que si estos resultados fueran empleados para planificación de trayectoria la información aportada por cada mapa sería muy similar.

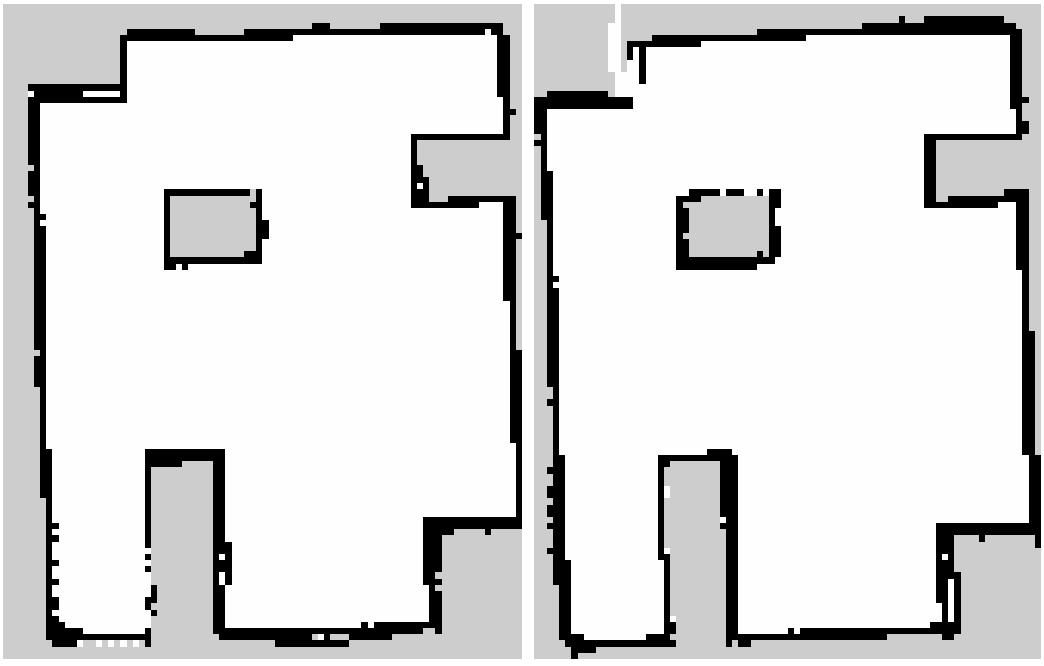


Figura 42: Comparación de mapa 1 generado con metodología síncrona (izquierda) y asíncrona (derecha)

12.2.2. Imágenes generadas del Mapa 2

Con la configuración de obstáculos utilizada para el mapa 2 se decidió colocar algunas esquinas más pronunciadas que en el mapa anterior. Esto con el fin de verificar si este tipo de geometrías también serían captadas de forma correcta por el algoritmo de SLAM. Como es posible observar en la Figura 43 la generación de los mapas es bastante similar entre sí, y como con el mapa 1, tanto utilizando la metodología de mapeo síncrona como la asíncrona se obtuvieron resultados como lo esperado. Sin embargo, con esta nueva configuración de obstáculos es posible observar cómo el método síncrono no entrega bordes tan definidos en los obstáculos esquinados como la metodología de mapeo asíncrona. Sin embargo, nuevamente es necesario tomar en cuenta que estas muestras fueron realizadas de forma manual y que el recorrido del sensor no haya sido exactamente igual en ambas pruebas agrega error a nuestros resultados. De todas formas, al momento de utilizar los mapas para procesos posteriores de planificación, con ambos mapas se obtendrían resultados similares.

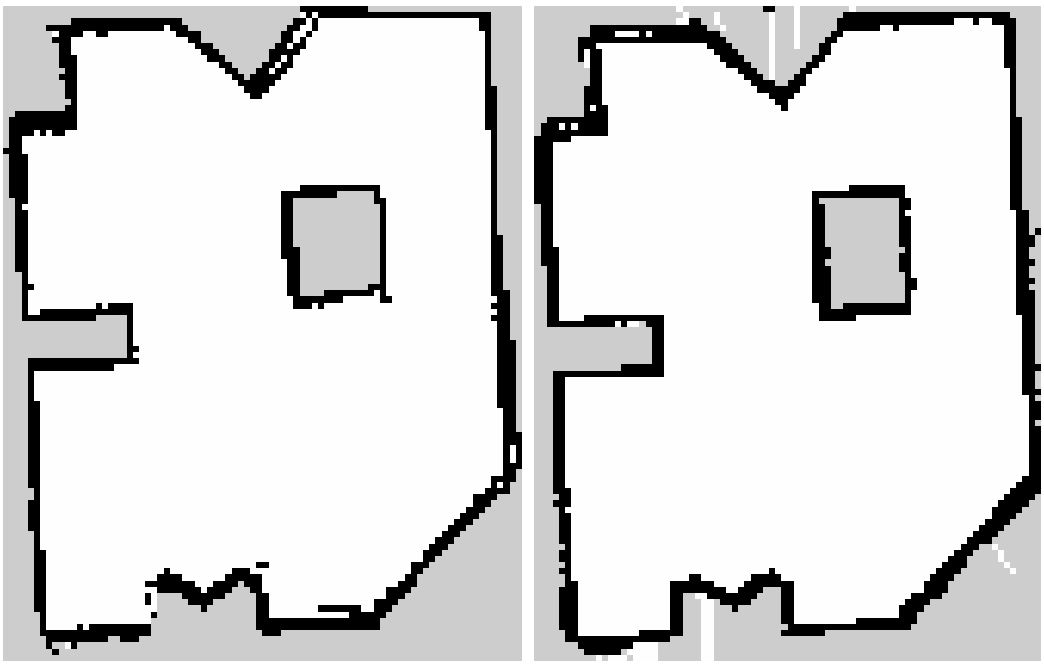


Figura 43: Comparación de mapa 2 generado con metodología síncrona (izquierda) y asíncrona (derecha)

12.2.3. Imágenes generadas de mapa del mundo virtual *small_ town*

Para finalizar se decidió realizar pruebas de SLAM en Gazebo, utilizando el mundo virtual *small_ town* provisto por la guía [23] para poder analizar el funcionamiento del algoritmo de SLAM utilizado en un área mayor, más compleja y con obstáculos distintos a los disponible físicamente con el fin de poder experimentar de mejor manera con el *SLAM Toolbox*. A continuación, en la Figura 44 se puede observar el mundo en Gazebo (lado izquierdo) así como el visualizador y la pantalla emergente de los controles del Robot. Para estas pruebas a pesar que se utilizó el modelo del robot básico uniciclo de las pruebas iniciales para poder navegar en espacios más reducidos que los que sería capaz de navegar

el Rover UVG, el rango de medición del sensor Lidar se configuró igual a las características del Hokuyo URG-04LX-UG01 que se utilizó para las pruebas físicas. Esto con el fin que la información que recibió el algoritmo de SLAM fuera lo más similar posible a la que hubiera recibido en pruebas realizadas en forma física.

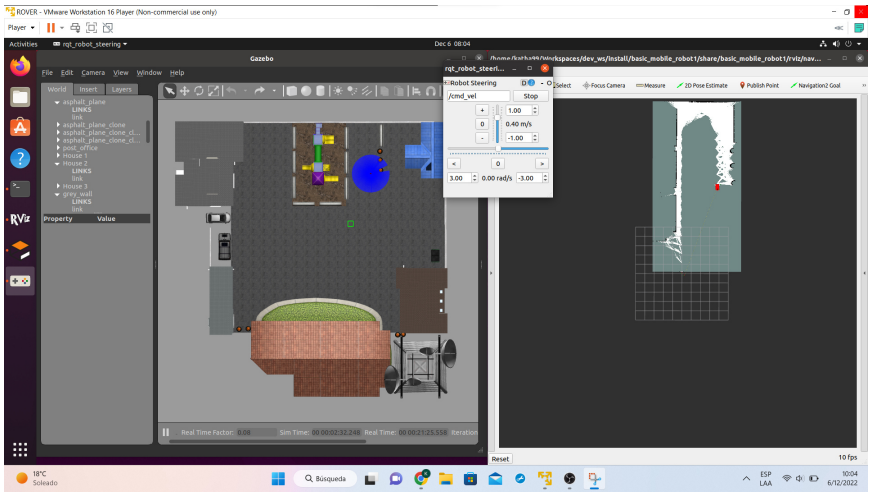


Figura 44: SLAM del mundo virtual *small_ town*

Estas pruebas se realizaron con mapeo asíncrono debido a que luego de varias iteraciones se observó que los resultados con ambos algoritmos de mapeo utilizados (síncrono y asíncrono) eran bastante similares, sin embargo por las recomendaciones de aplicación de cada forma de mapeo, debería de ser más conveniente utilizar mapeo asíncrono. En la Figura 45 se observa una imagen provista por el creador del mundo *small_ town* que presenta un mapa ideal de el área, sin embargo como ya se explicó anteriormente para la problemática de SLAM no existen soluciones exactas. Por lo que los resultados obtenidos no son idénticos al mapa proporcionado. Cabe mencionar que todos los mapas obtenidos en estas pruebas

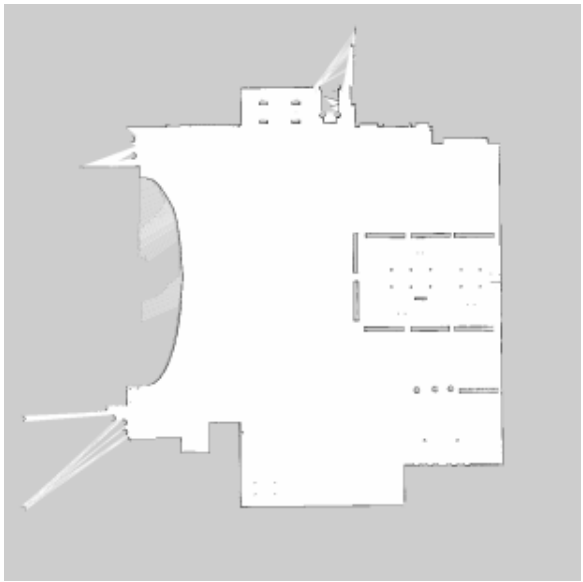


Figura 45: Mapa del mundo virtual *small_ town* ideal

muestran un área interna no reconocida (en color gris). Esto se debe a que el algoritmo de SLAM no puede decidir si un área está libre u ocupada si no tiene nada que a referenciarse, entonces debido a que el rango de medición del sensor Lidar era menor a la distancia de separación entre los obstáculos, el algoritmo no pudo asignar un valor a esta área.

En las siguientes imágenes se puede observar que la geometría general del área fue capturada de manera exitosa en todas las corridas, sin embargo debido a la constante actualización de los marcos de referencia los resultados no son exactos. Luego de realizar múltiples pruebas navegando la misma área con el mismo algoritmo de SLAM se pudo observar que según la forma de navegar el área se obtenían resultados similares. Los comportamientos cuyos resultados fueron similares en varias pruebas son los siguientes:

- Navegar siguiendo el perímetro del área una sola vez:

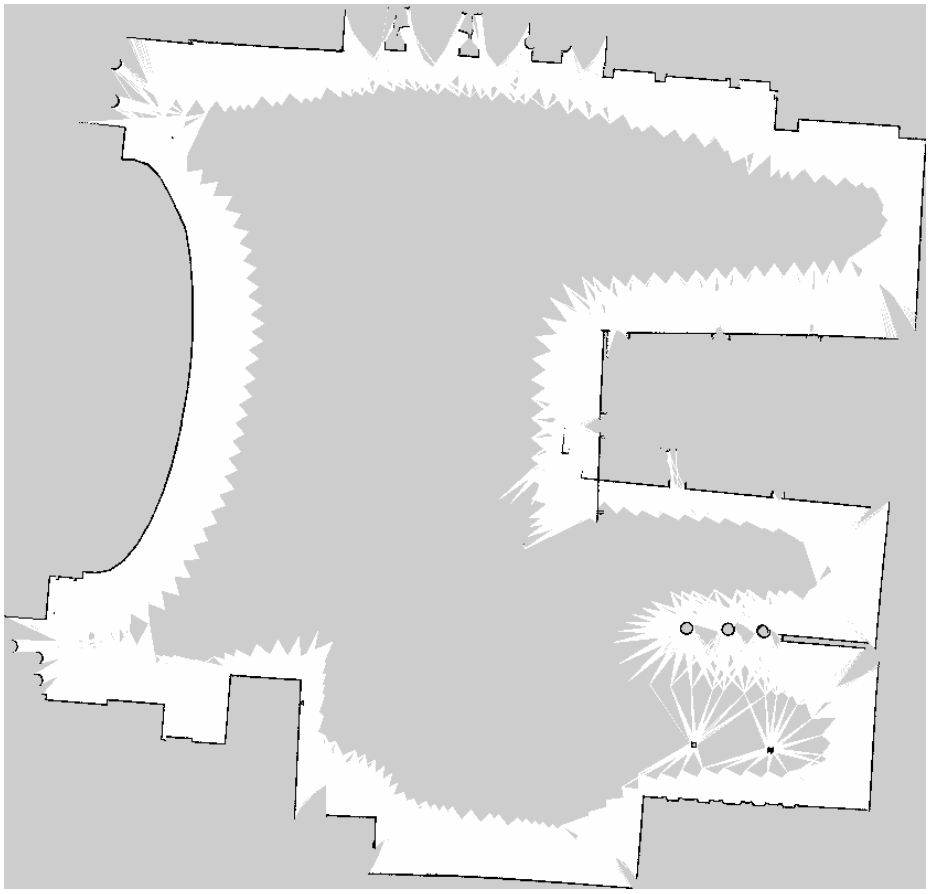


Figura 46: Mapa del mundo *small_ town* siguiendo la trayectoria del área

Como se observa en la Figura 46, al seguir esta trayectoria de navegación se logra captar una geometría bastante acertada de todo el perímetro de forma continua. Sin embargo, es posible observar que en la esquina más cercana al centro de la imagen (en donde se inicia y termina el recorrido) está traslapada, pues el error de la odometría que se acumuló durante todo el recorrido se ve reflejado en este punto. También es posible observar que la parte

superior del mapa se ve un poco inclinada, esto también se debe al ajuste de los marcos de referencia debido al error acumulado en la odometría.

- Navegar dando vueltas dentro del área sin seguir un orden en específico:

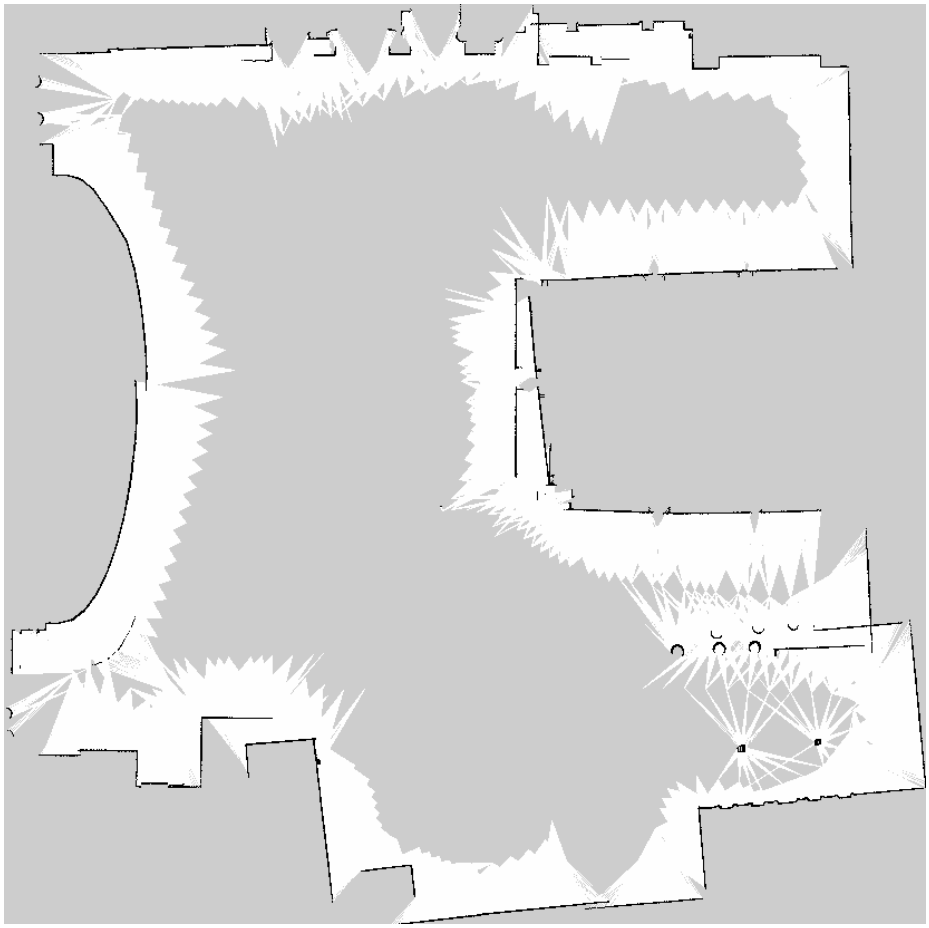


Figura 47: Mapa del mundo *small_ town* dando vueltas de forma aleatoria

Al navegar con este tipo de trayectoria, Figura 47, se obtiene un resultado más exacto en cuanto a la ubicación de los obstáculos, no tanto en la geometría de los mismos. Esto se debe a que los obstáculos se leen por tramos entonces en vez de obtener una geometría continua se obtiene una con pequeños saltos a lo largo de los perímetros establecidos por los obstáculos. Este resultado se debe a que el error de la odometría no se refleja en un solo punto del mapa, sino que cada vez que se cierra una trayectoria el algoritmo realiza una corrección de la pose del robot lo cual esparce el error en pequeñas cantidades a lo largo de todo el mapa. Este tipo de navegación entregó, en general, los resultados de mapas más apegados el ideal.

- Navegar el área recorriendo el mismo lugar varias veces seguidas antes de continuar con el resto del área:

La geometría suele verse bastante mal porque hay un corrimiento rotacional en el marco de referencia entonces cuadran menos cosas AL navegar con este comportamiento,

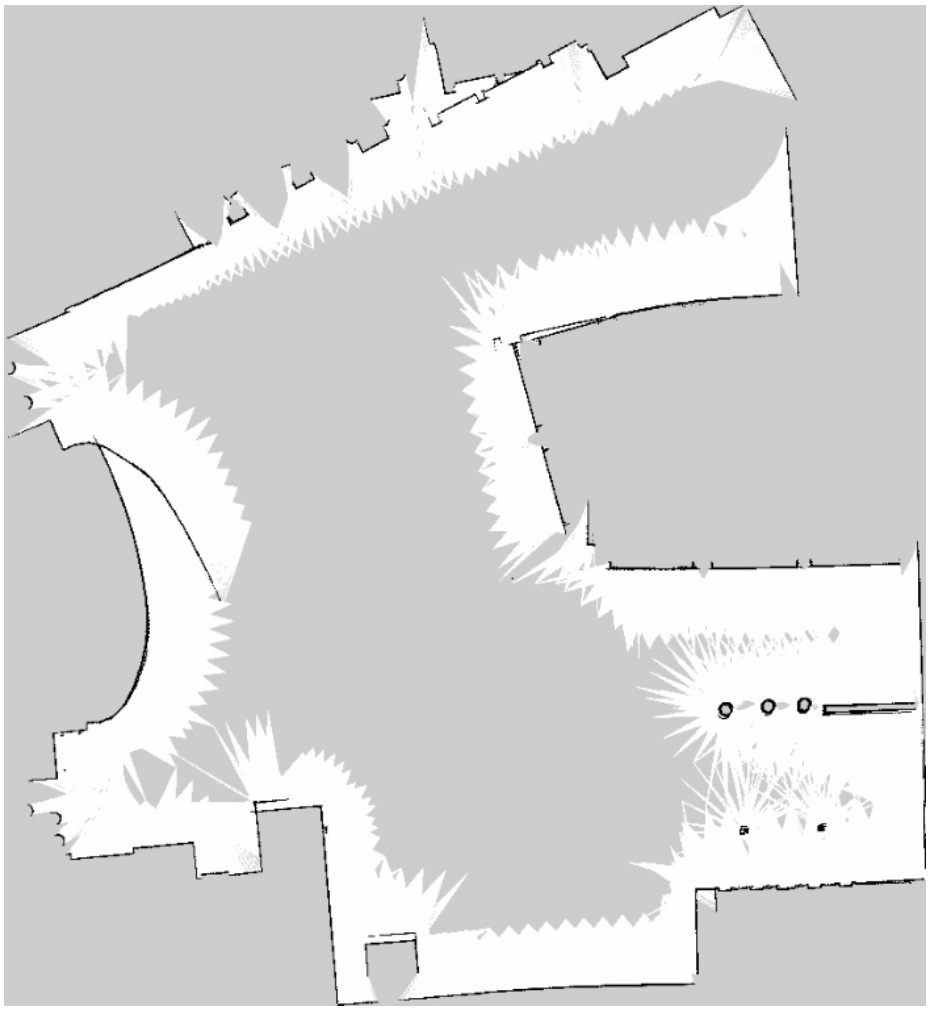


Figura 48: Mapa del mundo *small_ town* recorriendo el miso lugar varias veces hasta lograr la mayor cantidad de detalle

avanzando y retrocediendo varias veces en el mismo tramo para captar más detalles del área se pueden observar dos aspectos principales, los cuales se puede observar en la Figura 48. El primero es que en las pareas donde sí hay obstáculos en ambos lados del rango de medición del sensor sí se obtuvo mayor detalle no solo en el perímetro de los obstáculos sino también en el área que rodea los obstáculos pues no hay tantos espacios en gris (sin reconocer). Sin embargo, estos detalles se obtienen únicamente al inicio de la navegación debido a que el error en la odometría aumenta más si se realizan estos movimientos de avanzar y retroceder en el mismo lugar, entonces los resultados obtenidos no son óptimos, pues el marco de referencia sufre corrimientos drásticos tanto lineas como rotacionales al intentar corregir el error generado por los *encoders*.

CAPÍTULO 13

Integración a plataforma Rover UVG

13.1. Colocación de sensor Hokuyo URG-04LX-UG01 en el Rover UVG

Para la colocación del sensor en el Rover UVG se tomaron en cuanta distintos aspectos. Idealmente los sensores de este tipo se colocan en la parte superior del robot para que el mismo robot no interfiera en las mediciones del sensor pero esto no fue posible debido a que la plataforma Robotat, en donde se realizaron las pruebas en físico, es de un área menor que el rango de medición del sensor, entonces si este se colocaba en la parte superior los mapas no se generarían en base al área de la plataforma sino que también se mapearía todo el mobiliario del laboratorio fuera de la misma. Entonces, para evitar todo esa información que generaría ruido en los resultados, se decidió que el sensor debía colocarse en la parte inferior en Rover UVG lo más cercano al piso posible para que este sí reconociera el borde de la plataforma que es de 8cm de altura. Para evitar que las orugas del Rover UVG generaran interferencia en las mediciones del sensor, este se colocó en la parte frontal del robot y su rango de medición se redujo a 180.º, entonces las mediciones del sensor no se vieron afectadas por el volumen del robot, este fue capaz de detectar el perímetro de la plataforma Robotat y a su vez el rango de medición fue lo suficientemente amplio para generar los mapas deseados.

En la Figura 49 se observa el montaje final de los sensores en el Rover UVG. En el centro de la parte superior se encuentran los 2 sensores de nuestro interés, el *marker* que reconoce el sistema de captura de movimiento Optitrack sobre la pieza verde que se encuentra en el centro de la parte superior del robot y el sensor Lidar en la parte inferior como se detalla anteriormente.

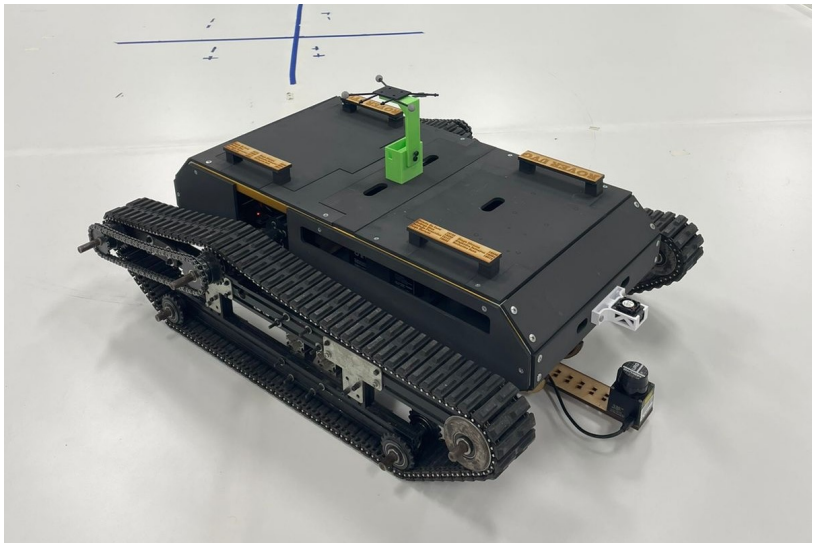


Figura 49: Montaje final de sensores en Rover UVG

13.1.1. Modificación de marcos de referencia

Como se detalla anteriormente, para poder realizar de forma correcta las tareas de navegación con robots móviles con ruedas se deben definir cuatro marcos de referencia como mínimo. En esta integración, debido a la posición de los sensores fue necesario redefinir el marco de referencia del sensor con respecto al centroide del robot. Como se observa en la Figura 50, siendo el marco negro la plataforma Robotat, el rectángulo gris el Rover UVG y las lineas azules el rango de medición del sensor Hokuyo, esta es la definición de los ejes X y Y para el sistema.

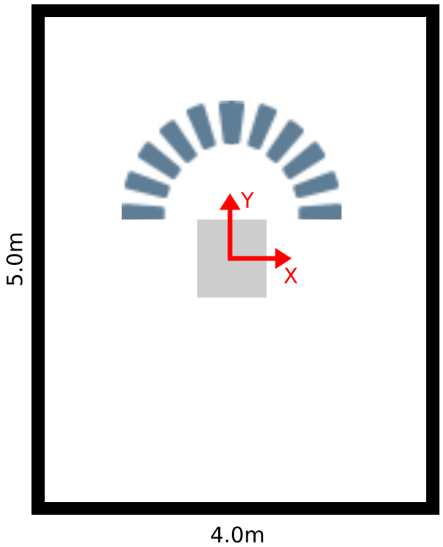


Figura 50: Definición de ejes X Y en plataforma Robotat

Debido a que el algoritmo de SLAM se está utilizando exclusivamente en 2 dimensiones, las variaciones en el eje Z no fueron tomadas en cuenta. De la misma forma, la variación entre el marco de referencia del robot y del sensor en eje X se despreció pues en teoría estos están alineados en dicho eje y de no estar alineados en su totalidad la variación es mínima, por lo que puede despreciarse. Finalmente, el corrimiento de los marcos de referencia en el eje Y si se modificó en el archivo URDF con el que se define el robot de forma virtual, pues en diferencia a las pruebas físicas realizadas anteriormente en donde el *marker* y el sensor estaban unidos entre sí, en esta prueba el corrimiento en el eje Y fue de 46cm.

13.2. Generación de archivo ejecutable

Sabiendo que la computadora *on board* del Rover UVG es una Raspberry Pi 4, se generó un nuevo archivo ejecutable, el cual nos permitiera correr todos los nodos necesarios para realizar SLAM a la vez. Utilizando como base el archivo que se utilizó para las pruebas físicas anteriores, únicamente se modificó el archivo URDF. Este archivo ejecutable corre a la vez el nodo de odometría, *Optitrack_ odometry*; el nodo de lectura de datos con el sensor Hokuyo, *urg_ node* y el nodo del *SLAM Toolbox* para correr el algoritmo de SLAM con mapeo asíncrono.

13.3. Pruebas realizadas y resultados obtenidos

Como se muestra en la Figura 51, para esta prueba los obstáculos se colocaron únicamente en la orilla de la plataforma para que el Rover UVG pudiera moverse sin problemas dentro de la plataforma. Por la misma limitante del espacio, el Rover UVG únicamente se movió hacia adelante y a hacia atrás en esta prueba. Sin embargo, se logró captar en su mayoría la geometría creada por los obstáculos.

Figura 51: Rover UVG realizando SLAM dentro de la plataforma Robotat

Durante esta prueba el archivo se corrió desde una terminal de comandos en una computadora del laboratorio, pues al estar esta computadora y la Raspberry Pi a bordo del robot conectadas a la misma red local fue posible correr el archivo de forma remota. De igual forma, los resultados de la posición del actual del robot y el mapa que se generó se visualizaron en Rviz de forma remota en la computadora externa al Rover UVG como se observa en la Figura 52. En esta prueba, debido a que el URDF se modificó, no se visualizó el modelo del Rover UVG modifico, sino que únicamente se mostraron los marcos de referencia utilizados.

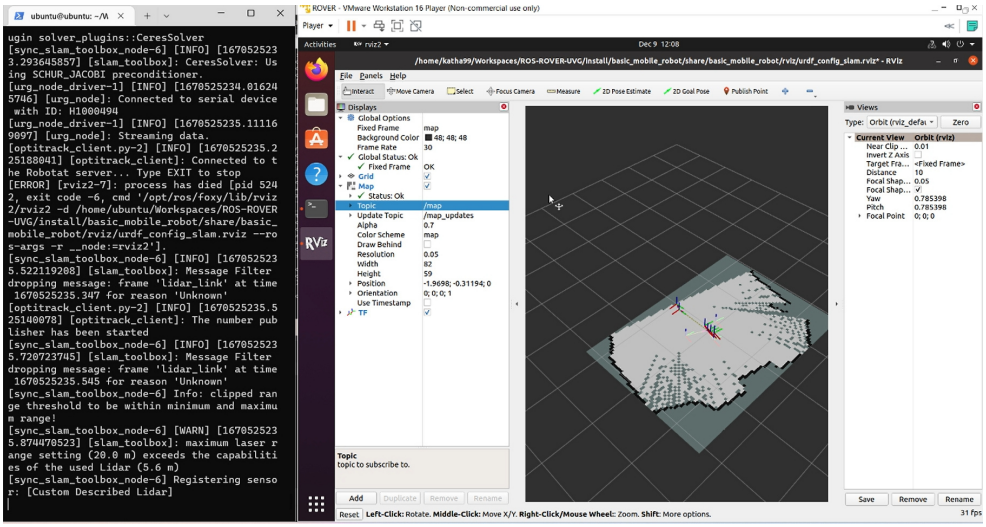


Figura 52: Archivo ejecutable de SLAM corriendo remotamente y visualización de resultados.

13.3.1. Mapa generado

A pesar que los resultados obtenidos también fueron exitosos, al estar ambos sensores sobre el Rover UVG se generó más ruido debido a que el robot al trasladarse vibra debido al sistema de movimiento que utiliza. Por esto mismo en la Figura 53 se puede observar que las líneas generadas por los obstáculos son más gruesas y tienen una mayor cantidad de puntos grises, los cuales representan espacio no navegado, en algunas áreas, sin embargo el resultado obtenido fue exitoso.

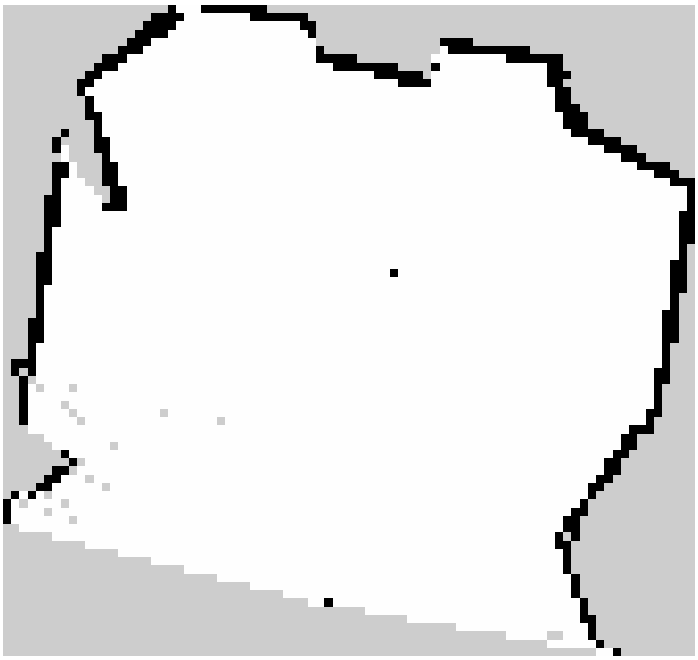


Figura 53: Archivo ejecutable de SLAM corriendo remotamente y visualización de resultados.

CAPÍTULO 14

Conclusiones

- El *slam_ toolbox* es una herramienta para ROS2 que provee algoritmos de *Pose-Graph SLAM*. Esta herramienta brinda una solución versátil para poder reconocer áreas recorridas por robots móviles con ruedas tales como el Rover UVG siempre y cuando este cuente con sensores que permitan obtener su odometría y lecturas de las distancias de los objetos de su entorno mediante un sensor láser como lo es el Lidar HokuyoURG-04LX-UG01.
- A pesar que actualmente no existen más herramientas disponibles en ROS2 para realizar SLAM con robot móviles con ruedas, los resultados obtenidos por medio del *slam_ toolbox* fueron satisfactorios tanto en las pruebas realizadas en simulación como en físico y con el Rover UVG pues fue posible identificar tanto el perímetro del área como los obstáculos que se encuentran dentro de la misma en cada una de las pruebas.
- La escala de grises provista por imágenes en formato .pgm es ideal para los *Occupancy Grid Maps* generados por SLAM, pues es una forma gráfica de obtener el valor de cada casilla del mapa mediante una tonalidad de gris para análisis durante la navegación del robot o incluso para acceder a esa información para procesos posteriores. En este caso fue el nodo *map_ saver_ server* del paquete Nav2 el que permitió acceder a la información obtenida por SLAM de forma gráfica.
- A pesar que no se pudo experimentar con distintos algoritmos de SLAM, ambas metodologías de mapeo disponibles en el *slam_ toolbox*, mapeo síncrono y asíncrono, presentaron resultados satisfactorios. Sin embargo, la recomendación del autor del *toolbox* recomienda utilizar la metodología de mapeo asíncrona para aplicaciones en donde los robots móviles no navegan en todo momento en base a controles operados por los usuarios sino también navegan mediante controles computacionales como el Rover UVG.
- Aún que no se experimentó únicamente con un algoritmo de SLAM, fue posible concluir que el tipo de trayectoria que siga el robot durante el tiempo que se corra el algoritmo puede afectar a los resultados. Si la trayectoria es bastante extensa sin ser cerrada el error se acumula y se verá reflejado en el punto en que la trayectoria se cierre, lo

cual entrega resultados bastante acertados en cuando a la geometría del área pero suele tener corrimientos en cuanto a la posición de los obstáculos. Por otro lado, si se navega de forma tal que se cierren múltiples trayectorias pequeñas en la misma corrida del algoritmo este repartirá el error en cada punto en el que se cierre una trayectoria, estos resultados a pesar de tener pequeñas discontinuidades en cuando a la geometría de los obstáculos son más acertados en cuanto a la ubicación de los obstáculos.

—

CAPÍTULO 15

Recomendaciones

- Se recomienda retomar este estudio cuando haya una disponibilidad mayor de métodos para realizar SLAM con robots móviles con ruedas en ROS2, distintos al *SLAM Toolbox* para poder evaluar el funcionamiento de este algoritmo y determinar si es el óptimo para esta aplicación.
- Realizar pruebas físicas en espacios con áreas mayores a la disponible en la plataforma Robotat de la Universidad del Valle de Guatemala para poder verificar el funcionamiento del algoritmo de SLAM utilizado y el alcance que este puede tener en aplicaciones donde se pretenda explorar áreas desconocidas mayores a la explorada en este proyecto y así poder ampliar el listado de posibles aplicaciones futuras del Rover UVG en cuanto a SLAM.
- Equipar la plataforma Rover UVG con sensores que indiquen la odometría del robot distintos al sistema de captura de movimiento Optitrack con el fin de obtener la pose interoseptiva del robot y así poder utilizar odometría real que se utiliza en los algoritmos de SLAM, es decir que los valores de pose estén referenciados al mismo robot en vez de a un marco de referencia externo como el que maneja el sistema Optitrack.
- Se recomienda realizar una mayor cantidad de pruebas de SLAM con los sensores a bordo del Rover UVG con el fin de poder definir si el ruido generado por el mismo movimiento del robot afecta de manera significativa a la creación de mapas. También se recomienda realizar pruebas en donde se realice SLAM al mismo tiempo en que el Rover UVG se esté moviendo de forma autónoma y no únicamente cuando el robot está siendo controlado de forma remota por el usuario.

CAPÍTULO 16

Bibliografía

[1] H. J. Sagastume, *Diseño Mecánico, Selección de Motores e Implementación de Sensores para un Robot Explorador Modular*, Tesis de licenciatura, 2021.

[2] J. E. Archila, *Diseño e implementación de capacidades automáticas de navegación para un Robot Explorador Modular*, Tesis de licenciatura, 2021.

[3] M. G. Ocando, N. Certad, S. Alvarado y Á. Terrones, "Autonomous 2D SLAM and 3D mapping of an environment using a single 2D LIDAR and ROS," en *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, 2017, págs. 1-6. DOI: 10.1109/SBR-LARS-R.2017.8215333.

[4] I. Z. Ibragimov e I. M. Afanasyev, "Comparison of ROS-based visual slam methods in homogeneous indoor environment," *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, 2017. DOI: 10.1109/wpnc.2017.8250081.

[5] X. Yang, "Slam and navigation of indoor robot based on Ros and Lidar," *Journal of Physics: Conference Series*, 22038.ª ép., vol. 1748, n.º 2, págs. 1-5, ene. de 2021. DOI: 10.1088/1742-6596/1748/2/022038.

[6] M. A. Búrbano, *Implementación de una Interfáz de Datos RS-232 Inalámbrica para el Radar Láser Hokuyo URG-04-LX-UG01*, Tesis de licenciatura, 2012.

[7] P. Corke, "Chapter6: Localization," en *Robotics, Vision and Control*, B. Siciliano y O. Khatib, eds., Second Edition. Springer International Publishing AG, 2017, vol. 118, págs. 151-185.

[8] *Usar LIDAR en ArcGIS*. dirección: https://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/using-lidar-in-arcgis.htm.

[9] L. Hokuyo Automatic Co, *Scanning Laser Range Finder URG-041LK-UG01 Specifications*, ene. de 2009.

[10] R. Pi. dirección: https://www.raspberrypi.com/.

[11] T. T.A. dirección: https://www.arduino.cc/en/Guide/Introduction.

[12] T. T.A. dirección: https://store.arduino.cc/products/arduino-mega-2560-rev3.

[13] T. T.A. dirección: https://docs.arduino.cc/retired/shields/arduino-usb-host-shield.

[14] R. Tellez, *A history of ROS (robot operating system)*, jul. de 2020. dirección: https://www.theconstructsim.com/history-ros/..

[15] *What is ROS?* Mayo de 2021. dirección: https://vimeo.com/639236696.

[16] AutomaticAddison, 2021. dirección: https://automaticaddison.com/what-is-an-occupancy-grid-map/.

[17] L. Hokuyo Automatic CO., "Scanning Laser Range Finder URG-04LX-UG01," vol. C-42-3635, n.º 1, pág. 5, 2009.

[18] L. Hokuyo Automatic CO., "Communication Protocol Specification For SCIP2.0 Standard," vol. C-42-03320B, n.º 1, pág. 25, 2006.

[19] G. Doisy, 2022. dirección: https://www.wyca-robotics.fr/2022/05/09/cartographer-ros2-rolling-and-humble/.

[20] S. Macenski, 2021. dirección: https://github.com/SteveMacenski/slam_toolbox.

[21] OpenRobotics, 2022. dirección: https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html.

[22] Porizou1, 2021. dirección: https://qiita.com/porizou1/items/b3cc26eb0cf168412dbc.

[23] A. Addison, 2021. dirección: https://automaticaddison.com/the-ultimate-guide-to-the-ros-2-navigation-stack/.

CAPÍTULO 17

Anexos

17.1. Repositorio Github

https://github.com/sen18012/ROS2_SLAM_ROVER_UVG

Figura 54: Enlace a repositorio de Github del proyecto

CAPÍTULO 18

Glosario

AMCL: Sistema de localización probabilístico para robots móviles en 2D. Siglas del inglés *Adaptive Monte Carlo Localization*. 32

Board: Placa de material no conductor que contiene un circuito electrónico impreso en cobre. 17

Driver: Conjunto de archivos que permiten a uno o más dispositivos comunicarse con el sistema operativo de una computadora. 39

Formato PGM: Archivo que almacena imágenes 2D en escala de grises, cada pixel contiene uno o dos bytes de información. Siglas del inglés *Portable Gray Map*. 55

Formato STL: Diminutivo de estereolitografía. Archivo que almacena la información de un modelo 3D comúnmente realizados por medio de diseño CAD. 46

Formato XML: Archivo que permite almacenar, transmitir y reconstruir data arbitraria. Siglas del inglés *Extensible Markup Language*. 46

Gazebo: Aplicación independiente pero compatible con ROS y ROS2 especializada para realizar simulaciones robótica. 47

IMU: Unidad de medición inercial, dispositivo que reporta la fuerza específica, velocidad angular y orientación de un cuerpo específico. 48

Lidar: Dispositivo que permite determinar la distancia desde un emisor láser hacia un objeto o superficie midiendo el tiempo de retraso entre la emisión del pulso láser y la detección de la señal reflejada. (acrónimo del inglés LiDAR, Light Detection and Rangign). 1

MATLAB: Plataforma de programación y computación numérica especializada en el análisis de datos, desarrollo de algoritmos y creación de modelos. 27

Optitrack: Sistema de captura de movimiento y rastreo 3D de alta precisión, líder en la industria, para aplicaciones de robótica, diseño de video juegos, animación, entre otros. 1

PWM: Técnica en la que se varía el ancho de pulso de una señal cuadrada para representar la amplitud de una señal analógica (siglas del inglés *Pulse Width Modulation*. 17

Robotat: Plataforma disponible en la Universidad del Valle de Guatemala para realizar prácticas de robótica en un ambiente controlado. 28