TECNOLÓGICO DE ESTUDIOS SUPERIORES DE COACALCO

Mapeo y visualización de espacios utilizando el algoritmo SLAM.

PROYECTO DE INVESTIGACIÓN

PARA OBTENER EL TITULO DE

INGENIERO EN MECATRÓNICA

CON ESPECIALIDAD EN:

AUTOMATIZACIÓN Y CONTROL DE PROCESOS

P R E S E N T A

GARDUÑO GUIDO JUAN CARLOS
MATRICULA 201610148

DIRECTOR:

M. EN C. MARIO ALBERTO HERNÁNDEZ SORIANO

Coacalco de Berriozábal, Agosto de 2024

DEDICATORIAS

Dedico con todo mi ser a mi mamá y a mi papá por siempre haber confiado en mí y en mis decisiones que he tomado a lo largo de mi vida y en mi vida académica, sin ellos no podría ser la persona quien soy ahora.

Gracias, madre y padre por su tiempo y esfuerzo que dieron por mí, los amo.

RESUMEN

En este proyecto de investigación se realiza la visualización y mapeo de habitaciones mediante el algoritmo SLAM (Simultaneous Localization And Mapping) implementado en el framework ROS (Robot Operating System), para el mapeo de las habitaciones se utilizó un sensor de tipo LiDAR. El sistema de mapeo fue probado en tres habitaciones diferentes, en la sección de resultados se muestra la visualización obtenida para cada de las habitaciones probadas.

ÍNDICE

Índice de figuras

Capítulo I.
Generalidades

1.1 Introducción

El presente reporte consiste en un sistema de visualización de alta precisión, en tiempo real y remota por medio del protocolo de comunicación HTTP en una red local, gracias a un conjunto de tecnologías modernas como el sensor LiDAR (Light Detection and Ranging) que emite un láser en 360 grados, lo que permite detectar objetos de su alrededor. También están los sistemas embebidos como Nvidia Jetson nano quien se encarga del procesamiento de la información y será intermediario entre el sensor y la computadora maestra que por medio de la comunicación local podrá compartir la información mediante visualización remota del entorno mapeado.

El mapeo de un entorno no es nuevo, desde la antigüedad ha existido la necesidad de visualizar gráficamente un cierto entorno, a esto se le conoce como cartografía y a lo largo de los años la forma de plasmar un mapa ha evolucionado gracias a las múltiples tecnologías que van desarrollándose, diferentes tipos de sensores con diferentes formas de enviar información y a medida que van creciendo la información, también la forma de procesarla de una manera más rápida, eficiente y compacta, como lo son los sistemas embebidos, que se podría decir que son computadoras integradas en una placa de menos de 15cm x 15cm como lo son Nvidia Jetson nano o Raspberry Pi. Sin duda, uno de los aspectos más interesantes de este proyecto es el uso de ROS (Robot Operating System) que es un framework muy usado en todo el mundo y más en los países donde la robótica es muy avanzada como Alemania, Estados Unidos, India, China, etc., y la razón es porque facilita, optimiza y mejora el desarrollo de la robótica además de que contiene librerías para sistemas de visualización con infinidad de sensores que son compatibles con el sistema.

Hoy en día, el mapeo mediante el uso de un robot puede ser fundamental para muchas empresas, como lo es en las minas, en entornos cerrados, tanto como en el mar, tierra e incluso en el aire. Requerir de un mapa del entorno donde se está trabajando es muy importante para algunas personas, pueden ayudar en la automatización de ciertas tareas que requieran una navegación, o que simplemente visualicen el entorno en tiempo real de lo que está pasando y para esto existen tecnologías como Simultaneous Localization And Mapping o mejor conocido como SLAM.

En este proyecto se concentra únicamente en la visualización SLAM y se deja aparte el robot o la parte mecánica, sin embargo, es suficiente para poder realizar el mapeo del entorno cerrado, por ejemplo, del cuarto donde se está situado.

1.2 Justificación

Durante la exploración de cualquier entorno terrestre, aéreo o submarino implica un alto riesgo en la integridad de cualquier ser humano sin conocer previamente el entorno de exploración.

El conocimiento previo de estos entornos ayudaría bastante a la exploración, el tener un sistema que detecte cualquier objeto como muros, tubos, rocas e incluso personas o seres vivos, y representarlos en un mapa 2D en tiempo real, y así tomar mejores decisiones antes de entrar o durante la exploración.

1.3 Planteamiento del problema

La exploración en diferentes entornos es uno de los grandes problemas que se tienen en distintas zonas laborales, como en construcción, minería e incluso en el campo aéreo. Conocer la zona de trabajo es muy importante, ya que en algunos casos puede ser peligroso para el ser humano, por ejemplo, en ciertas zonas sin explorar, como cuevas o edificios abandonados.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar un sistema de reconstrucción de espacios implementado en un sistema embebido utilizando la técnica SLAM (Simultaneous Localization And Mapping) y el framework ROS (Robot Operating System).

1.4.2 Objetivos específicos

1. Conocer cómo funciona los sistemas computacionales y la informática
2. Realizar un mapeo simple con el sensor adecuado para este trabajo.
3. Realizar pruebas de visualización del entorno en tiempo real.
4. Implementar un sistema maestro-esclavo para visualizar remotamente el entorno por medio del protocolo de comunicación HTTP.

1.5 Hipótesis

Poder visualizar de manera remota cualquier entorno terrestre por medio de sensores que escaneen y procesen la información con el framework ROS.

1.6 Alcances y limitaciones

En este proyecto el alcance es obtener el sistema de visualización y hacer mediante la técnica Hector SLAM con la tecnología ROS, pero no será implementado en algún robot.

Capítulo II.
Marco Teórico

2.1 .- Cartografía

2.1.1. Definición

La Cartografía es la ciencia que estudia las formas y creación de los de los mapas, es una rama de la geografía encargada de la representación de cualquier entorno de manera visual y de fácil entendimiento. Estos mapas, en un inicio eran de forma bidimensional, hoy, gracias a los avances tecnológicos, podemos hacer mapas más precisos y hasta tridimensionales.

La cartografía es el estudio y la práctica de la elaboración de mapas. La creación de mapas implica la aplicación de elementos tanto científicos como artísticos, combinando talentos gráficos y conocimientos especializados de principios de compilación y diseño con técnicas disponibles para la generación de productos. Los mapas funcionan como herramientas de visualización de datos espaciales. Los datos espaciales se almacenan en una base de datos y se extraen para diversos fines. Los métodos analógicos tradicionales de elaboración de mapas han sido reemplazados por sistemas digitales capaces de producir mapas interactivos dinámicos que pueden manipularse digitalmente. [1]

2.1.2. Tipos de mapas

Existen dos tipos de mapas en la cartografía:

- Topográficos (Cartografía básica)
- Temáticos (Cartografía temática)

La cartografía básica es aquella que se obtiene por procesos directos de observación y medición de la superficie terrestre, sirviendo de base y referencia para su uso generalizado como representación gráfica de la Tierra. La cartografía básica puede ser topográfica o náutica. [2]

La cartografía temática es más específica con lo que quiere representar, esta es la principal diferencia con la general o la básica. Pueden expresar densidad, clases, dirección, entre más, por ende, cuando un mapa está expresando algo más que solo un plano en 2D se dice que es un mapa temático.

En la cartografía se distinguen entre los mapas topográficos y los mapas temáticos. En el caso de los mapas topográficos, la situación, cuerpo de agua, accidentes geográficos,

coberturas de suelos y el conjunto de otros elementos para la orientación general, así como las etiquetas, forman el contenido del mapa. [3]

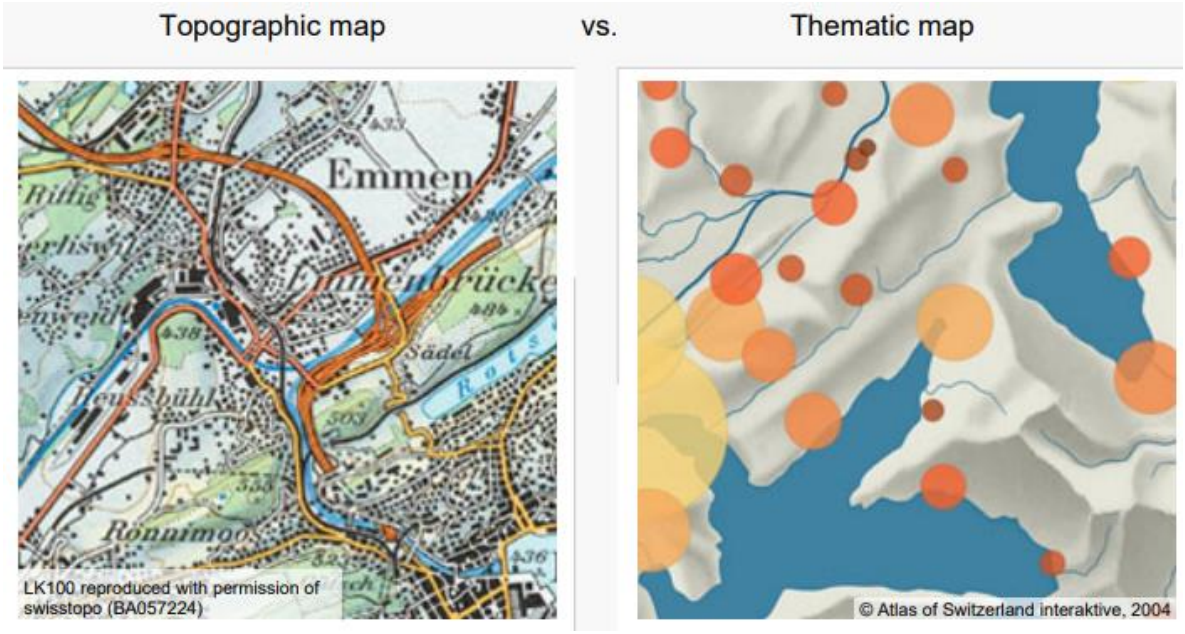

Figura 1. Cartografía general vs temática. [3]

2.2 .- Simultaneous Localization And Mapping (SLAM)

2.2.1 Definición

La localización y mapeo simultáneos (SLAM) es un algoritmo importante que permite al robot reconocer los obstáculos a su alrededor y localizarse. Cuando es combinado con algunos otros métodos, como la planificación de rutas, es posible permitir robots para navegar en entornos desconocidos o parcialmente conocidos. [4]

La navegación autónoma hoy en día es la clave de la robótica contemporánea, ayuda al robot a moverse, prácticamente, por cualquier entorno, sin la intervención de un humano. Permite hacer tareas que el humano no podría o sería riesgoso, como la exploración y así mismo investigarlos mediante mapas. Para hacer estos mapas altamente precisos se recurre muchas veces a una técnica en la robótica llamada SLAM.

En el ámbito de la robótica móvil, las técnicas de navegación basada en mapas se mantienen como uno de los enfoques más utilizados, debido a su factibilidad, pero hace necesario el conocimiento de un mapa de navegación esto puede resolverse utilizando técnicas de localización y mapeo, conocidas por sus siglas en inglés como SLAM. Estas técnicas permiten conocer y ubicar al robot en su entorno al mismo tiempo que se desplaza. En el presente reporte se describe los pasos necesarios para generar un mapa a través de técnicas de SLAM, utilizando un sensor laser y las herramientas provistas por el meta-

sistema operativo ROS el cual en años recientes se ha convertido en un estándar en el desarrollo de software de robótica. [5]

La técnica SLAM se basa en la odometría. La odometría es una forma de localización que utiliza datos de sensores como codificadores para obtener una posición estimada en relación con un punto de partida. La localización es un medio para poder situar la posición del robot en un momento dado. La odometría es especialmente útil en programas autónomos porque permite realizar más fácilmente distintas tareas sobre el terreno gracias a la comprensión de la propia posición. [6]

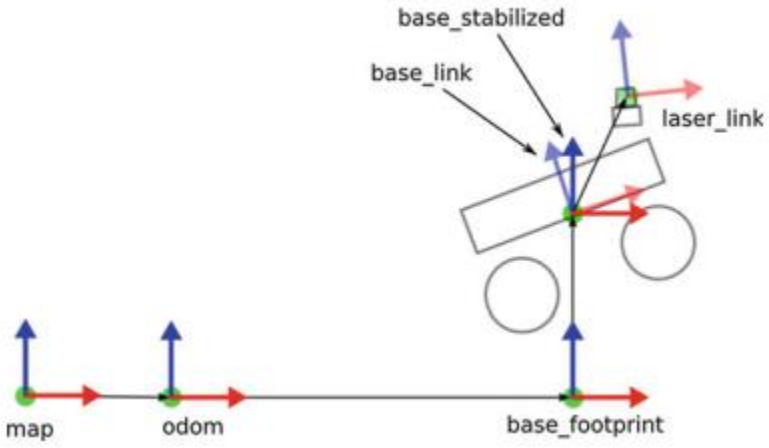

Figura 2. Odometría en SLAM. [4]

2.2.2 Ventajas y aplicaciones

Sus principales ventajas es la detección de objetos en tiempo real y así evadirlo, como rocas, personas, animales, etc., haciéndolo versátil y adaptable al uso en cualquier tarea. También es el uso de sus mapas para la exploración y conocer el entorno que antes era desconocido, permitiendo así salvaguardar la integridad de los humanos para dichas tareas de exploración.

A pesar de tener muchas ventajas, tiene varias limitaciones tales como, al ser un plano en 2D no puede identificar objetos fuera del alcance del láser, así como identificación de objetos en específicos ya que solo detecta las caras que el láser alcanza y, a veces, esta información no es suficiente para la identificación de los objetos.

Sin embargo, hay diferentes tipos de sensores cada uno para tareas distintas entre cada uno, por lo que se puede agregar más sensores para cubrir las desventajas de otras integrándose todas a un mismo robot.

2.2.3 Sensores de mapeo

Hay una infinidad de sensores que permiten tomar la lectura del entorno, algunos son los que usualmente se usan por medio de láseres como son el caso de LDS (Laser Distance Sensor), LiDAR (Light Detection And Ranging) y LRF (Laser Range Finder), regularmente usados para mapeos en 1D o 2D. También existen otros sensores tipos de sensores más novedosos que son basados en infrarrojo como es el caso de RealSense y Kinect, usados para hacer mapeos 3D, estos tipos de sensores son más costosos y requieren procesar mucha información a alta velocidad, así como algunas librerías como OpenCV para su procesamiento, por lo que son para robots de muy alto costo. [7]

Por último, están las cámaras, que sin bien no pueden hacer un mapeo preciso como alguno de los antes mecionados, sirven para otras funciones, como detección de colores, formas, reconocimiento facial entre otras más.

Figura 3. Diferentes tipos de sensores en el mercado. [7]

2.2.4 Frameworks

Un framework es una herramienta en forma de software que permite hacer el trabajo más eficiente y rápida. Se usan para ahorrar tiempo y esfuerzo y conseguir el mismo resultado que si se hiciera sin un framework, por lo que, actualmente, se usan los framework que permiten a los desarrolladores avanzar con su trabajo de una manera más ágil.

ROS es un framework para la robótica que permite desarrollar robot de manera eficiente por la fácil interacción con los diferentes sensores y actuadores que existen en el mercado.

2.3 .- Robot Operating System (ROS)

2.3.1 Definición

Robot Operating System (ROS) es un framework que es altamente usado en la robótica contemporánea.

Fue desarrollado en 2007 por Stanford Artificial Intelligence Laboratory (SAIL) con el apoyo de Stanford AI Robot Project.

A lo largo de muchas investigaciones de las instituciones han empezado a desarrollar proyectos en ROS añadiendo hardware y compartiendo código de ejemplos. Así, las compañías fueron adaptando el hardware a sus necesidades usando ROS. [8]

Los sensores y actuadores usados en la robótica han también sido adaptados para estar dentro de ROS. Cada día hay un incremento de dispositivos con soporte en este framework.

A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. [9]

Algunas de las actividades que se pueden hacer al usar ROS son:

- Percepción
- Reconocimiento Facial
- Identificación de objetos
- Robots autónomos
- Planificación

Actualmente, ROS es un framework que se utiliza en las grandes empresas de robótica como solución a sus necesidades. [10]

ROS tiene muchos conceptos que se deben de comprender bien para poder usarse, tales como messages, topics, nodes, ROS master, services, parameters, y lunch files.

- Master: Es único, también conocido como ROS_MASTER_URI. Regularmente el que corre roscore.
- Host: Es una computadora dentro de la red local de ROS identificado por su IP. Regularmente es el que está dentro del robot obedeciendo las órdenes del master, También conocido como slave (esclavo).
- Node: Es un proceso de ROS, identificado dentro del grafo.
- Topic: Un unidireccional, asíncrono, fuertemente tipado, llamado, así como el canal de comunicación entre el mecanismo publish-suscribe.

- Message: Es una estructura de datos, usado como un tipo para los topics. Los messages pueden anidarse arbitrariamente.
- Connection: Es una conexión entre el Publisher y subscriber llevando los datos a un topic.
- Service: Es un procedimiento síncrono.
- Action: Un mecanismo de nivel superior creado sobre temas y servicios para tareas duraderas o interrumpibles con retroalimentación intermedia para la persona que llama.
- Parameters: Un diccionario multivariante compartido al que se puede acceder a través de la red API.
- Roslaunch: Es un comando propio de ros para hacer correr algún archivo XML que a su vez, corre un conjunto de nodos.

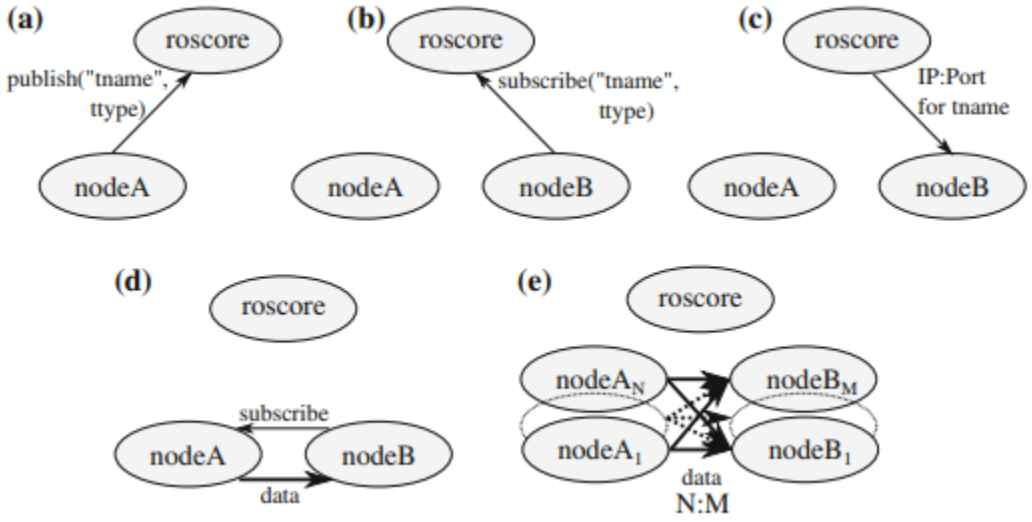

Figura 4. Representación gráfica de ROS y sus diferentes componentes.[4]

2.3.2 Versiones

ROS ha existido desde hace muchos años por lo que ha estado en constante transición, así como adaptarse a los nuevos sistemas operativos, en un inicio solo era la Ubuntu, hoy en día se puede usar hasta en Windows en sus versiones más recientes. Cada versión solo es compatible dependiendo del sistema operativo. Se puede saber que versión de ROS se debe instalar dependiendo el sistema operativo desde la página oficial de ROS https://wiki.ros.org/Distributions.

2.3.3 Módulos

ROS trae consigo muchas librerías que ayudan a implementarse de manera rápida a las necesidades del robot, a los sensores y a los actuadores, muchos de ellos son compatibles en la gran mayoría de estos componentes.

Al instalar un módulo completo, ROS ayudará para instalarlo fácilmente e inclusive instalará las dependencias, como lo son lenguajes de programación, más librerías, etc.

2.3.4 Aplicaciones

Sus aplicaciones son infinitas, desde hacer un simple robot autónomo que evada objetos, hasta hacer drones capaces de volar dentro de interiores sin necesidad de usar GPS y solo tomar decisiones mediante los sensores que lleva.

ROS es la principal herramienta que se usa en todo el mundo, se enseña desde la universidad en diferentes países como India, Alemania, Estados Unidos, China, etc.

Por su facilidad de manejo, su velocidad de desarrollo y su amplia gama de librerías, se ha convertido en el framework principal en la robótica contemporánea en todo el mundo.

2.4 .- Sistemas embebidos

Un sistema embebido (también conocido como “empotrado”, “incrustado” o “integrado”) es un sistema de computación diseñado para realizar funciones específicas, y cuyos componentes se encuentran integrados en una placa base (en inglés. “motherboard”). El procesamiento central del sistema se lleva a cabo gracias a un microcontrolador, es decir, un microprocesador que incluye además interfaces de entrada/salida, así como una memoria de tamaño reducido en el mismo chip. [11]

Estas minicomputadoras fueron principalmente creadas para personas que no tenían acceso a una computadora más cara, esta fue la solución para aquellas personas. Sin embargo, con el tiempo, se fueron implementando y dándole más usos a estos sistemas en la industria, robótica, IoT, etc.

Al tener acceso a internet, lo suficientemente poderosas para correr ciertas tareas y su pequeño tamaño los hacen óptimos para montarlos en la robótica o en los proyectos móviles, compactos y de poco consumo energético y de recursos computacionales.

Además, su sistema operativo será Linux y aquí la importancia de conocer Linux para esta área.

Los sistemas embebidos comenzaron a utilizarse antes de ser definidos y van evolucionando de manera tan rápida que provocan que las concepciones acerca de ellos cambien constantemente. Sin embargo, y a pesar de que aún se discute por qué se les llama

sistemas embebidos, se pueden distinguir cuatro características fundamentales: Hardware (embedded hardware), software (embedded software), inteligencia computacional y ejecución de una o varias tareas en tiempo real (el sistema es predecible y determinista). En este sentido, se propone definir un sistema embebido como un dispositivo electrónico que tiene inteligencia computacional, que está diseñado para cumplir una o varias tareas relacionadas que se determinan desde el diseño y por lo tanto, son predecibles al ejecutarse en tiempo real y que está integrado por componentes de hardware y software.

Los sistemas embebidos representan una evolución de los sistemas electrónicos rígidos, y su principal característica es la posibilidad de ser programados para resolver algorítmicamente un problema determinado. La programación de estos dispositivos puede ser realizada por el diseñador del sistema o por el sistema mismo, lo que da la sensación de razonamiento, es decir, de que el sistema puede aprender por sí mismo (inteligencia artificial). [12]

2.4.1 Raspberry Pi

Una de estas computadoras es la muy famosa Raspberry Pi. Este tiene su propio sistema operativo, memoria RAM, entradas/salidas, y entre otras características que lo hacen una computadora bastante funcional para casi cualquier uso. Este computador puede impresionar por lo poderoso que puede ser además de que trabaja bastante bien en un clúster de computadoras.

Son muy usados en muchos proyectos profesionales, usados como computadoras portátiles, o usados como el cerebro de un robot gracias a su pequeño tamaño y sus puertos de entrada y salida, además con el poder suficiente para procesar la información de los sensores y enviar información a los actuadores. Tiene un módulo de wifi para conectarse a internet y por esto mismo facilita en los proyectos de IoT.

Raspberry Pi es el nombre de una serie de computadoras de placa única fabricadas por la Fundación Raspberry Pi, una organización benéfica del Reino Unido que tiene como objetivo educar a las personas en informática y crear un acceso más fácil a la educación informática.

La Raspberry Pi se lanzó en 2012 y desde entonces se han lanzado varias iteraciones y variaciones. El Pi original tenía una CPU de un solo núcleo a 700 MHz y solo 256 MB de RAM, y el último modelo tiene una CPU de cuatro núcleos a más de 1,5 GHz y 4 GB de RAM. El precio de Raspberry Pi siempre ha sido inferior a 100 dólares (normalmente alrededor de 35 dólares), sobre todo el Pi Zero, que cuesta sólo 5 dólares.

En todo el mundo, la gente usa Raspberry Pi para aprender habilidades de programación, crear proyectos de hardware, realizar automatización del hogar, implementar clústeres de Kubernetes y computación Edge, e incluso usarlos en aplicaciones industriales.

La Raspberry Pi es una computadora muy económica que ejecuta Linux, pero también proporciona un conjunto de pines GPIO (entrada/salida de propósito general), que le permiten controlar componentes electrónicos para la computación física y explorar el Internet de las cosas (IoT). [13]

Figura 5. Raspberry pi, sistema embebido más usado. [13]

2.4.2 Nvidia jetson

Aunque también existen varias otras computadoras embebidas muy poderosas y famosas como es el caso del computador de la marca Nvidia Jetson Nano. A simple vista podemos decir que es muy similar a una Raspberry Pi, sin embargo, la diferencia de la Jetson con Raspberry está en su procesador.

La marca Nvidia se caracteriza por ser un proveedor de procesadores, específicamente, diseña y crea tarjetas gráficas para computadoras. Es una marca por excelencia y muy confiable a la hora de estar hablando de gráficos. Las tarjetas gráficas no solo sirven para poder visualizar en el monitor nuestras actividades o para jugar un videojuego que demanda alto rendimiento de recursos computacionales, sino también es la parte más importante en la inteligencia artificial.

Con sólo 70 x 45 mm, el módulo Jetson Nano es más pequeño que una tarjeta de crédito. Pero este sistema en módulo (SOM) listo para producción ofrece grandes resultados cuando se trata de implementar IA en el borde. Un SOM es un sistema embebido, es una computadora que tiene la capacidad de hacer muchas cosas por sus amplios módulos que tiene integrados. [14]

Jetson Nano ofrece 472 GFLOP para algoritmos de IA modernos. Ejecuta múltiples redes neuronales en paralelo y procesa sensores de alta resolución simultáneamente, lo que lo hace ideal para aplicaciones desde NVR hasta puertas de enlace inteligentes.

Potente y eficiente, visión por computadora y computación de alto rendimiento con solo 5 a 10 vatios y esto también lo convierte en un sistema embebido ideal para proyectos de

robotica, donde no están conectados a una fuente de energía y su única fuente será una batería que debería suminitrar a todo el sistema tanto como de control como de potencia, como lo son los motores o activadores.

Figura 6. Nvidia Jetson nano sistema embebido de la marca Nvidia. [14]

2.5 .- Protocolos de comunicación

Un protocolo es un conjunto de reglas: los protocolos de red son estándares y políticas formales, conformados por restricciones, procedimientos y formatos que definen el intercambio de paquetes de información para lograr la comunicación entre dos servidores o más dispositivos a través de una red. [15]

Los protocolos de comunicación es la forma de comunicarse entre sistemas. Actualmente hay diferentes tipos de comunicación para cada necesidad, algunas de estos protocolos de comunicación más famosos son:

- Http
- Sftp
- Smtp
- I2C
- UART

Como se ha dicho, cada protocolo de comunicación tiene una función en específico y para cada área, en el caso de la robótica existen algunos muy importantes como lo son el UART y I2C muy utilizados en los microcontroladores y los sensores. Por último, el rey de la transferencia de datos por internet por hipertexto es HTTP el mismo que se usa para navegar en internet entre las páginas web, aunque hay uno más seguro que es HTTPS.

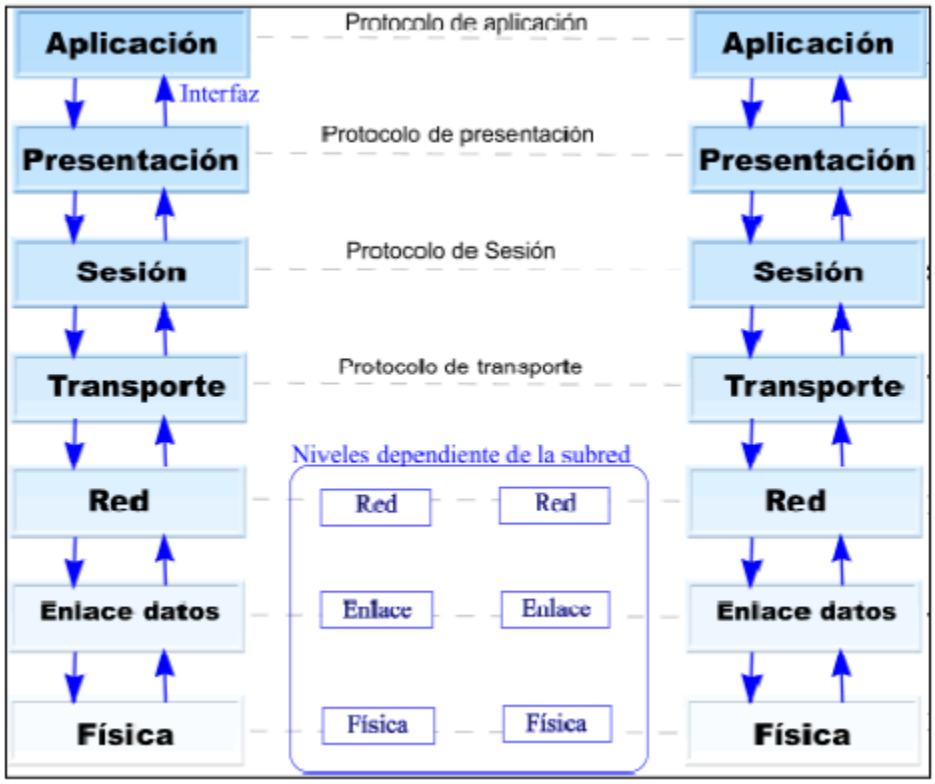

Figura 7. Niveles y servicios OSI. [16]

Los protocolos gestionan dos niveles de comunicación distintos. Las reglas de alto nivel definen como se comunican las aplicaciones, mientras que las de bajo nivel definen como se transmiten las señales.

El modelo de interconexión de sistemas abiertos (Open Systems Interconnection, OSI) es un marco conceptual que divide las funciones de comunicaciones de red en siete capas. El envío de datos a través de una red es complejo porque varias tecnologías de hardware y software deben funcionar de manera consistente a través de las fronteras geográficas y políticas. El modelo de datos OSI proporciona un lenguaje universal para las redes informáticas, de modo que diversas tecnologías pueden comunicarse mediante protocolos o reglas de comunicación estándar. Cada tecnología de una capa específica debe proporcionar ciertas capacidades y realizar funciones específicas para ser útil en las redes. Las tecnologías de las capas superiores se benefician de la abstracción, ya que pueden utilizar tecnologías de nivel inferior sin tener que preocuparse por los detalles de implementación subyacentes. [16]

2.5.1 Aplicación maestro-esclavo

El concepto de maestro-esclavo se usa en la informática para gestionar los recursos computacionales, así como las tareas que se tengan por hacer. Dichas tareas y recursos los

suministra el maestro, es aquel que lleva el control de todo el sistema, los que sigue las ordenes que el maestro da son los esclavos.

La forma que se pueden comunicar entre ambos es por medio de protocolos de comunicación como por puertos bus, http, uart, etc., esto permite el intercambio de información.

2.6 .- Sistema operativo

Un sistema operativo es un conjunto de programas informativos que se comunica con el hardware con el fin de gestionar los recursos computacionales.

Existen diferentes tipos de sistemas operativos está Windows donde este es el más famoso, sim embargo, tiene varios inconvenientes cuando se refiere la importancia de los recursos.

No porque sea el más usado signifique que sea el mejor. Cada uno de los sistemas operativos tienen sus ventajas y desventajas.

Hay un sistema operativo de código abierto, esto quiere decir que está gratis y cualquiera puede usarlo, llamado Linux, aunque este nombre queda muy ambiguo ya que el verdadero nombre es GNU/Linux. Linux por sí solo es el kernel.

GNU/Linux tiene muchas ventajas ante otros sistemas operativos como Windows y es por ello que es la mejor opción en muchas situaciones, sobre todo cuando se habla con un proyecto informático, además de que brinda muchas estabilidad, flexibilidad y alta seguridad.

Los sistemas operativos permiten que otros programas puedan utilizarlos de apoyo para poder funcionar. Por eso, a partir del sistema utilizado pueden ser instalados ciertos programas y otros no.

Son parte esencial del funcionamiento de los sistemas informáticos y la pieza de software central en la cadena de procesos, ya que establecen las condiciones mínimas para que todo funcione: la administración de los recursos, el método de comunicación con el usuario y con otros sistemas, las aplicaciones adicionales. [17]

El sistema operativo posee tres componentes esenciales o paquetes de software que permiten la interacción con el hardware:

- Sistema de archivos: Es el registro de archivos donde adquieren una estructura arbórea.
- Interpretación de comandos: Se logra con aquellos componentes que permiten la interpretación de los comandos, que tienen como función comunicar las órdenes dadas por el usuario en un lenguaje que el hardware pueda interpretar (sin que aquel que dé las órdenes conozca dicho lenguaje).

- Núcleo: Permite el funcionamiento en cuestiones básicas como la comunicación, entrada y salida de datos, gestión de procesos y la memoria, entre otros.

Sus principales funciones de cualquier sistema operativo son:

- Gestionar la memoria RAM.
- Gestionar los recursos de la CPU.
- Direccionar entradas y salidas.
- Gestionar autorizaciones de los usuarios.

Capítulo III.
Desarrollo

3.1 Diagrama general

El desarrollo del proyecto comienza con la instalación del sistema operativo que se va a usar junto con la ambientación del equipo, esto incluye las paqueterías, programas, etc. que se necesita para poder continuar con el proyecto. Una vez ambientado y con ROS instalado en la computadora con sistema operativo Ubuntu y que será el maestro, se procede a correr ros con el comando roscore el cual activará todos los servicios de ROS en la terminal.

Desde el lado del slave, se requiere conectar el sensor y para hacerlo funcionar y que se pueda comunicar con la computadora se requiere dar permisos de lectura y escritura en el puerto donde se conecta el sensor. Algunas veces este permiso ya viene dado, sino es el caso hay que correr un comando para dar dichos permisos.

Una vez dado los permisos necesarios, se corre el comando para hacer funcionar rplidar desde el slave y por el lado del maestro se corre el comando para hacer funcionar hector slam.

Por último, si el sensor funciona correctamente, se lanzará una pantalla del lado del maestro el mapeo en 2D en tiempo real, para esto, es importante que ambos equipos estén conectados a la misma red local ya que se comunican por HTTP.

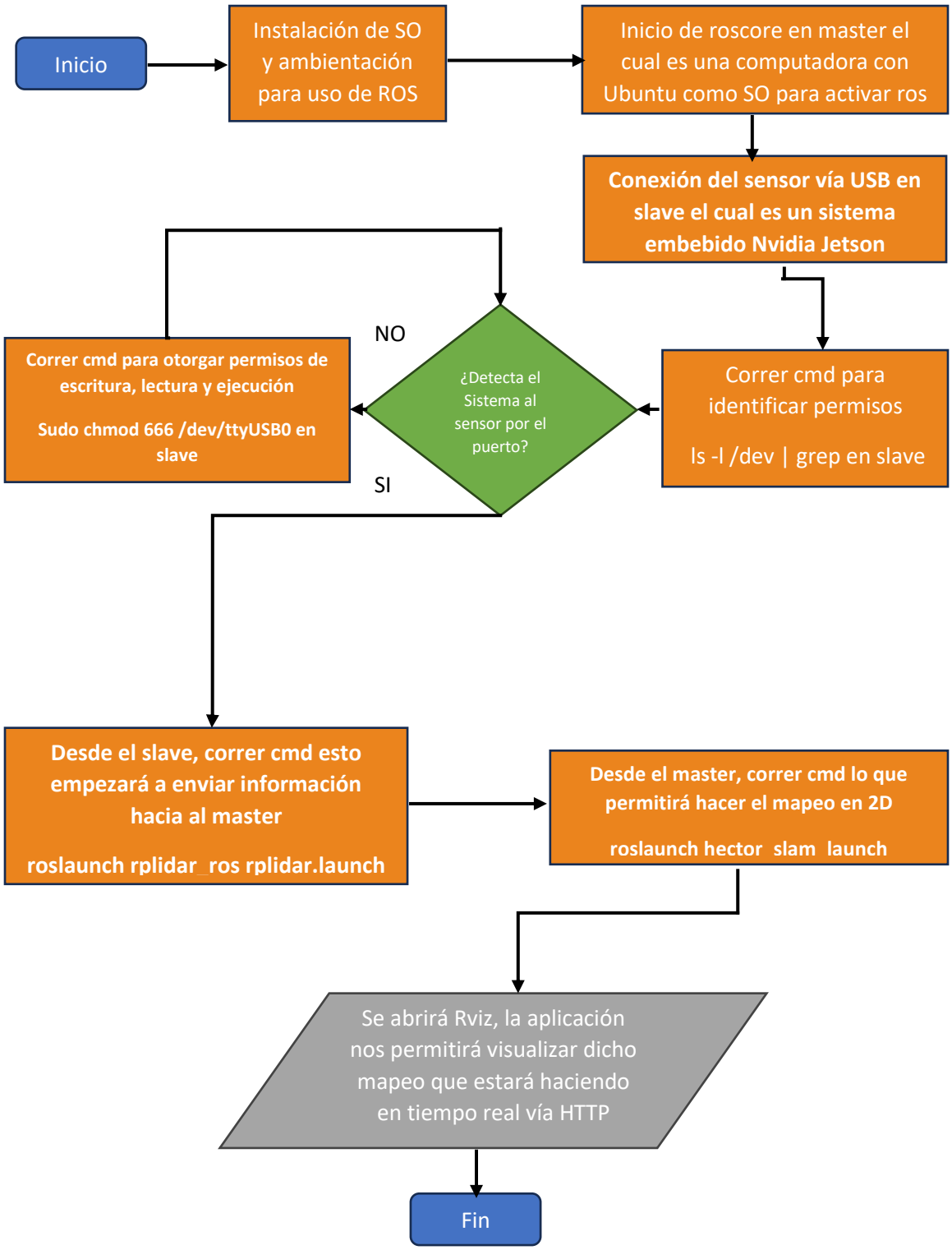

Figura 8. Diagrama de proceso de mapeo. (J. C. Garduño Guido)

3.2 Instalación de SO y ambientación para uso de ROS

3.2.1 Instalación de Ubuntu

Para este proyecto se utilizó la versión Ubuntu 18.04 LTS (Bionic Beaver). Que se puede descargar desde el siguiente enlace Ubuntu download

En ese enlace te direccionará hacia la página oficial del sistema operativo Ubuntu.

En la sección de lanzamientos pasados buscar la versión de interés.

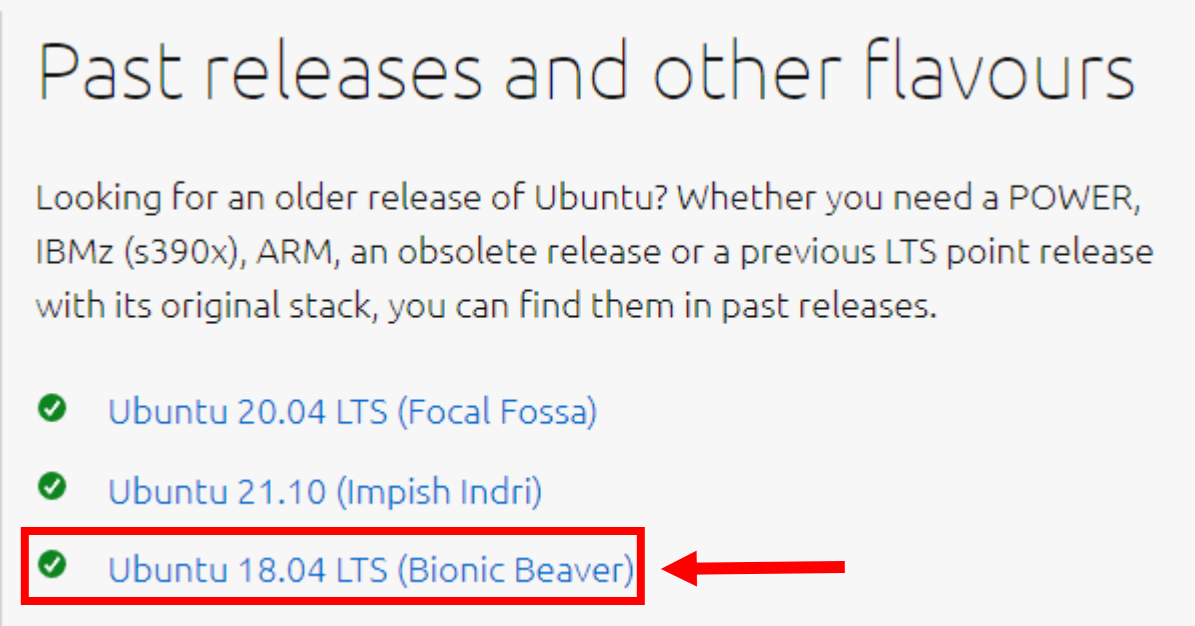

Figura 9. *Opciones de descarga de sistemas operativos Ubuntu* Obtenida desde la página oficial https://ubuntu.com/download.

Lo que se descarga solo es la imagen ahora se necesitará bootearlo en un USB, para esto se utilizan dos software más. Para ambas páginas, ir a la sección de descargas y dar click en la versión más actual.

- Rufus
- SD Card Formatter

Formateo del USB

El primer paso es abrir el programa SD Card Formatter, introduce un USB (El USB debe ser mayor a 8GB) y seleccionar en la parte de *Select card*, después seleccionar en la parte de *Formatting options* la opción *Quick format* y por último colocar un nuevo nombre al usb para identificarlo.

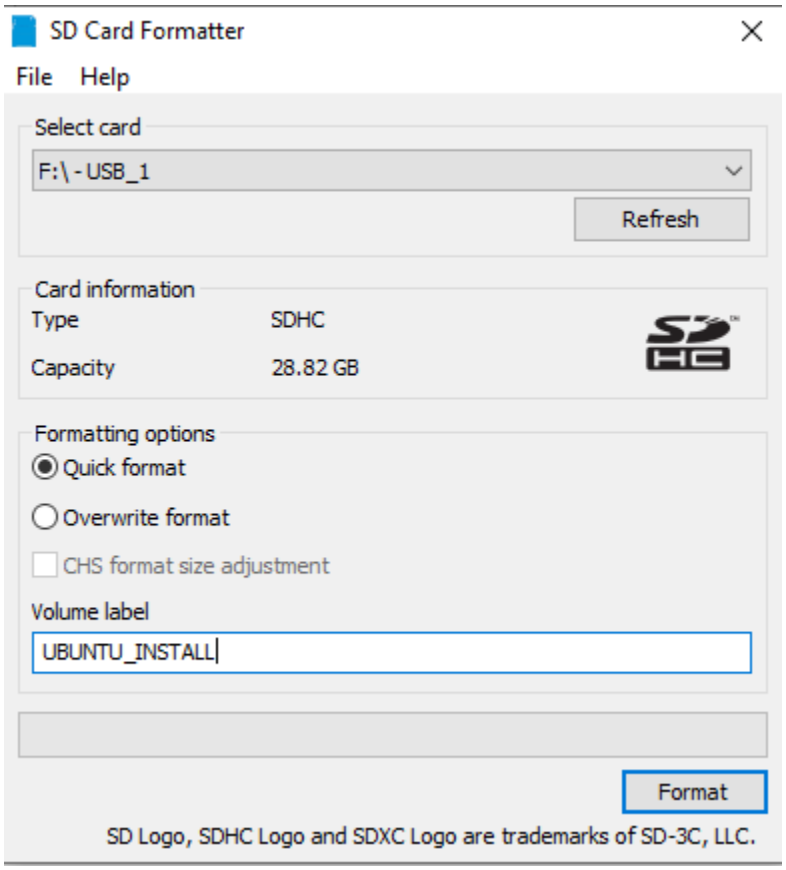

Figura 10. Screeenshot de las propiedades del software SD Card Formatter.

Una vez que se sepa que el usb esté listo con el formato deseado, el siguiente paso es flashear el usb con la imagen de Ubuntu que previamente fue descargado.

Abrir la aplicación Rufus,seleccionar nuestro usb y en la parte de *Boot selection* seleccionar *Disk or ISO image* y posteriormente la imagen de Ubuntu.

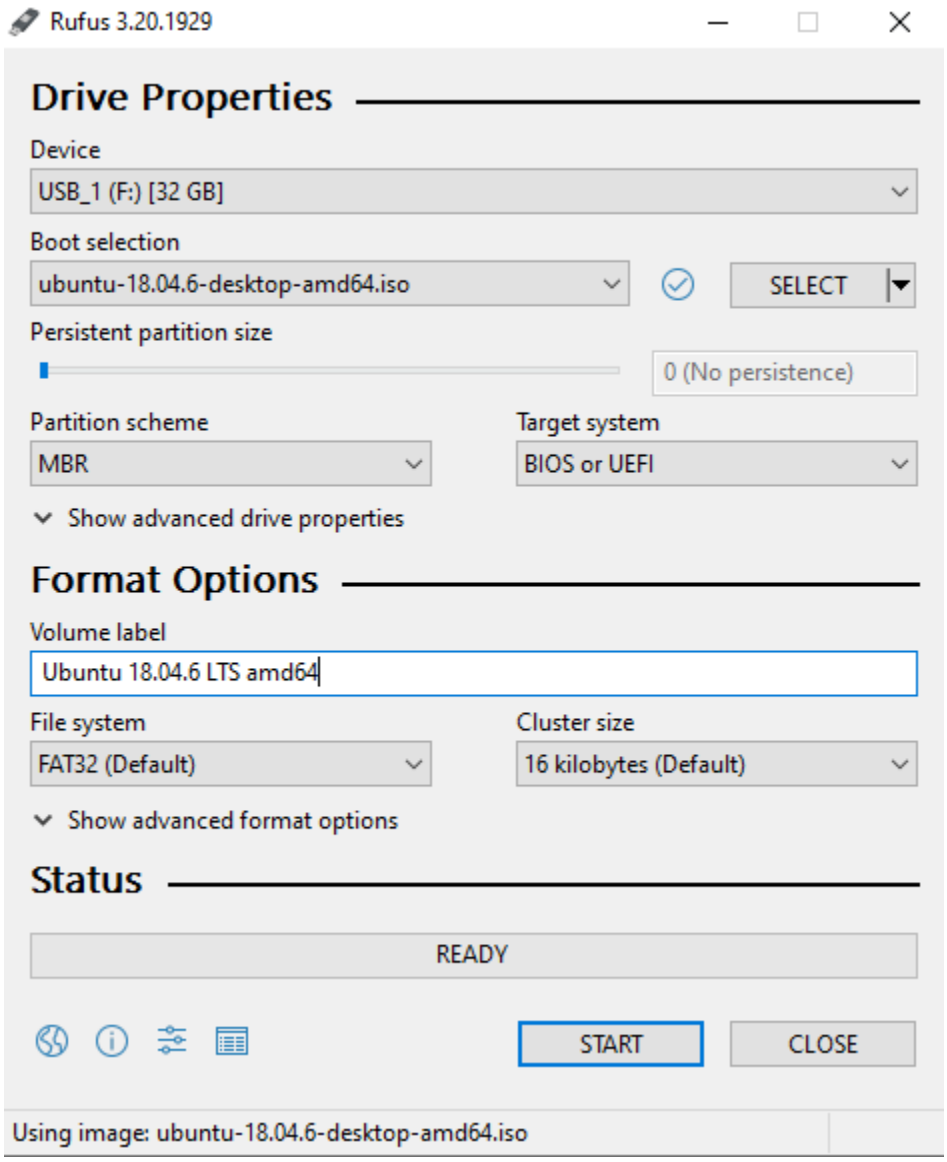

Figura 11. Screeenshot de las propiedades del software Rufus.

Se deja todos los demás parámetros tal y como viene por default y dar click en el botón de *START.* Una vez finalizado se puede desconectar su USB y seguir con los siguientes pasos.

Cuando se tenga la usb con la imagen del sistema operativo cargada con Rufus, insertar el usb en la máquina donde se instalará el nuevo sistema operativo.

Se debe considerar que al instalar un nuevo sistema operativo se borrará el otro que tenía. Si no se desea hacer esto del todo, se puede particionar la memoria o usar máquinas virtuales, en este caso, se instalará el sistema operativo desde cero y no se hará ninguna partición o máquinas virtuales, sin embargo, estas también son algunas opciones que se puede hacer.

Arrancar la máquina con el usb desde el boot mode. Cada modelo de las computadoras tiene una forma diferente de entrar a este modo y configurarlo. Si no se sabe como hacerlo, googlea como cambiar el boot mode del modelo que se tiene la máquina y una vez que seleccionar el usb, al reiniciarlo, aparecerá algo como esto:

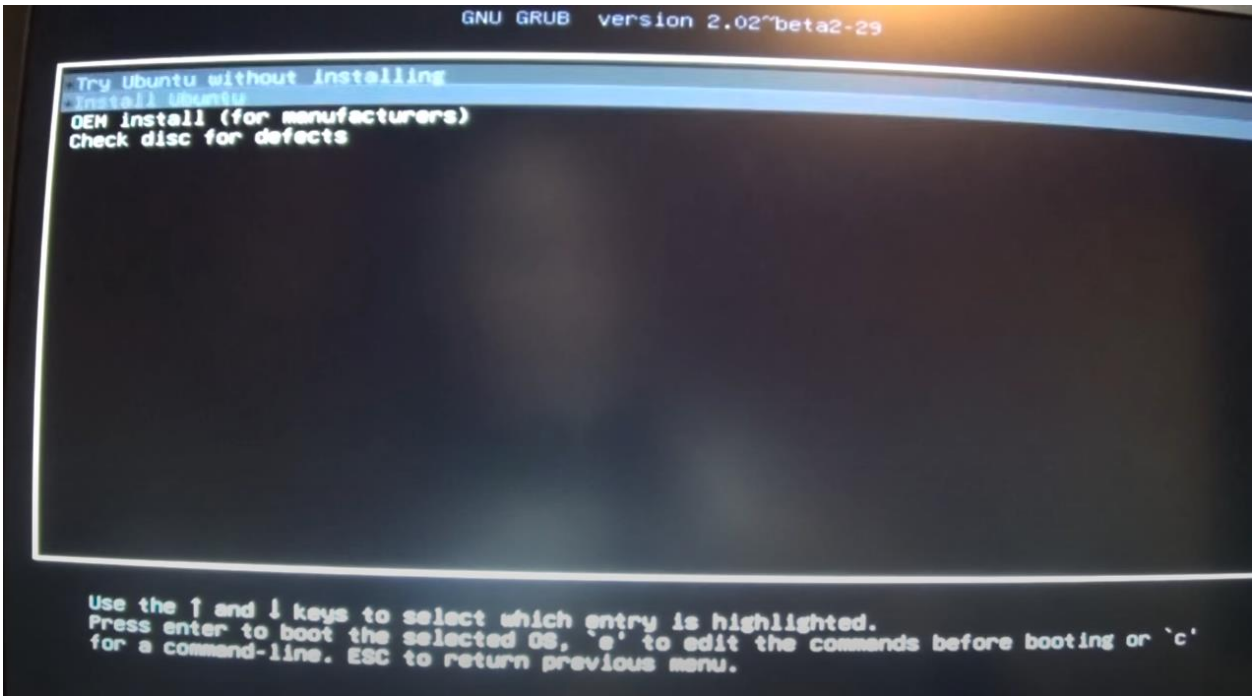

Figura 12. Screenshot al bootear e instalar Ubuntu

Seleccionar la opción *Install Ubuntu.*

Nos saldrá el idioma que queremos, la ubicación en donde esté, así como una red para conectarse, así que configuramos la máquina como deseamos y nos conectamos a nuestra red.

Cuando aparezca la siguiente ventana seleccionar las opciones como se muestra en la imagen y dar *continuar*:

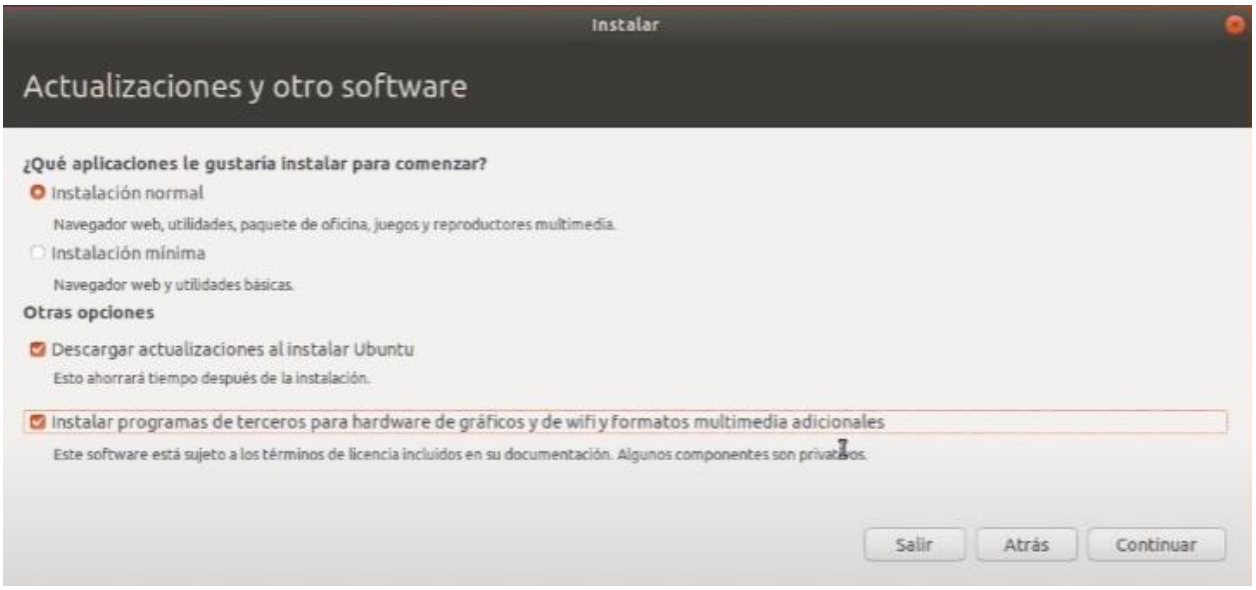

Figura 13. Screeenshot de opciones de actualización al instalar Ubuntu

Después aparecerá la siguiente ventana donde seleccionamos la opción *Borrar disco e instalar Ubuntu*:

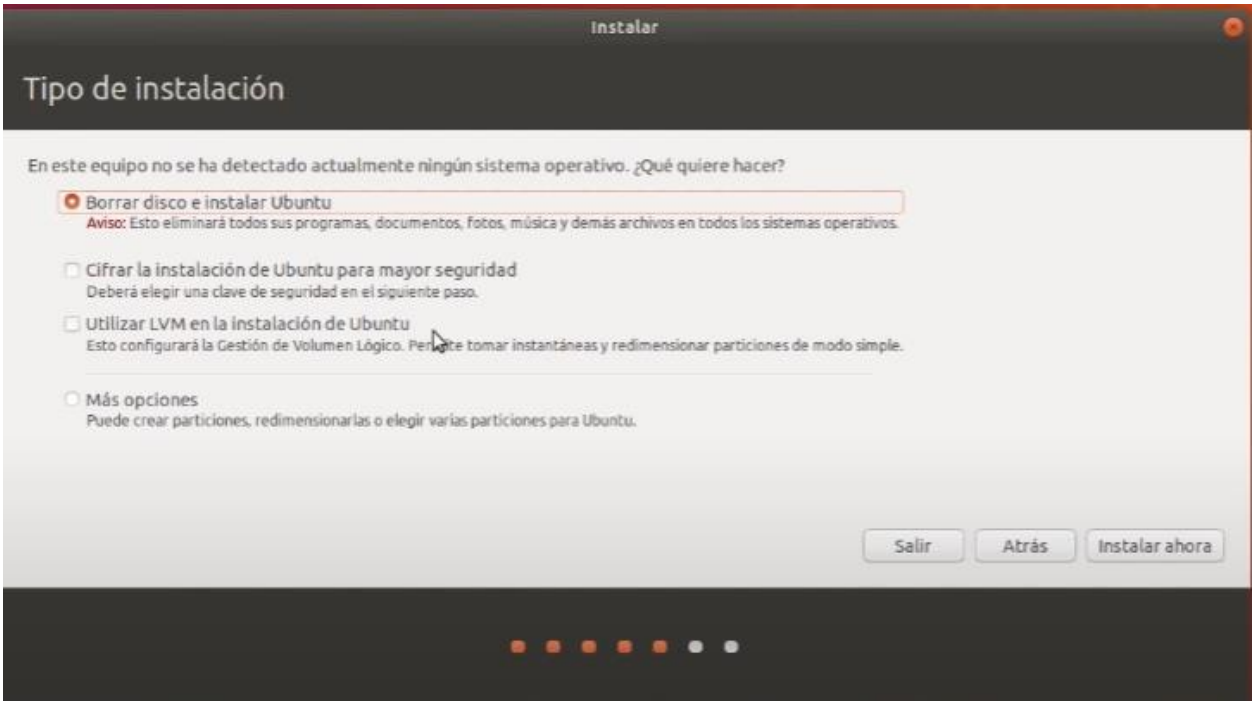

Figura 14. Screeenshot al seleccionar la opción de tipo de instalación

Seguido de eso, solo seguir las instrucciones y pronto se empezará a instalar.

3.2.2 Instalación de ROS

Se puede hacer la instalación de la paquetería desde el siguiente link ROS, es el enlace oficial de la paquetería por si se desea profundizar más. Sin embargo, solo se instalará, en este caso, lo necesario para hacer este proyecto.

Primero, abrir un pestaña de terminal.

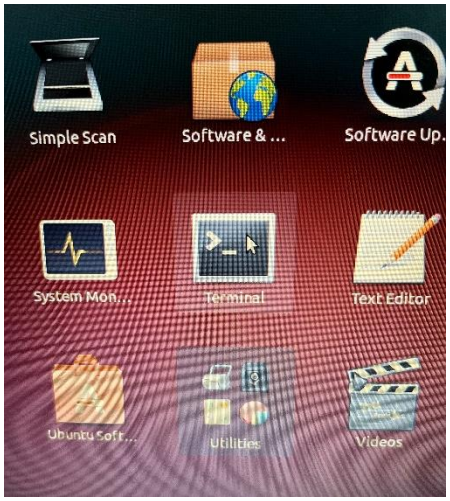

Figura 15. Icono de la terminal en Ubuntu

Una vez abierto, se recomienda siempre estar actualizando con el comando

```
sudo apt-get update
```

una vez que termine sigue el comando

```
sudo apt-get upgrade
```

si aparece una opción si se desea seguir decir “y”, esto puede tomar varios minutos.

Una vez terminado, hay que seguir con las instrucciones como viene en la página de ROS.

El primer comando que se correrá será

- sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

Esto hará que la computadora acepte los paquetes de ros.org

Seguido de eso, correr el siguiente comando

- sudo apt install curl # if you haven't already installed curl
- curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -

Estas son las llaves.

Para asegurar que no tendrá problemas en la instalación, se recomienda correr el siguiente comando

- sudo apt update

Hay diferentes tipos de librerías de ROS, pero siempre es recomendado instalar el siguiente

```
sudo apt install ros-melodic-desktop-full
```

Este contiene la mayoría de las dependencias y herramientas para desarrollar y, por lo tanto, es el más pesado, pero no debería haber problema si tienes un almacenamiento mayor de 16Gb. La instalación puede demorar varios minutos.

Una vez que termine la instalación, a partir de ese momento se puede instalar las paqueterías o dependencias independientes que se requiere. Por lo que se instalarán las siguientes

```
sudo apt install ros-melodic-slam-gmapping
sudo apt install ros-melodic-hector-mapping
```

De esta manera puedes instalar las paqueterías o depencias que desees si es que llegaras a necesitar.

sudo apt install ros-<VERSION >-<DEPENDENCIA>

Con esto se concluye la instalación y se puede corroborar corriendo el siguiente comando

roscore

Si aparece como la siguiente ventana es porque fue exitoso la instalación.

Configuración del entorno

Siempre es conveniente introducir las variables del entorno en la computadora con un bash, para eso correremos en la terminal el siguiente

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

NOTA: Es importante aclarar que cuando en el comando se vea el nombre de la versión, hay que sustituirlo con la versión que se tenga en la computadora, en este caso la versión es *melodic* pero si tienen *neotic* reemplazarlo.

Correr el siguiente commando para actualizar el archivo .bash.

```
source /opt/ros/melodic/setup.bash
```

Para poder contruir nuestros paquetes es necesario tener algunas dependencias, las más importante son las siguientes

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

También el siguiente

```
sudo apt install python-rosdep
```

Y con el siguiente podemos inicializar rosdep

```
sudo rosdep init
rosdep update
```

3.2.3 Catkin workspace

Para poder trabajar con ros se requiere tener un espacio de trabajo, se puede dar el nombre que se quiera, pero es muy usual que se le nombre *catkin_ws.*

Para eso, es necesario crear una carpeta con el siguiente comando

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
```

Ahora se debe de compilar desde la carpeta *catkin_ws*

```
catkin_make
```

Cuando acabe, se notará que se crearon nuevos archivos y carpetas, entre estos esta la carpeta *devel*

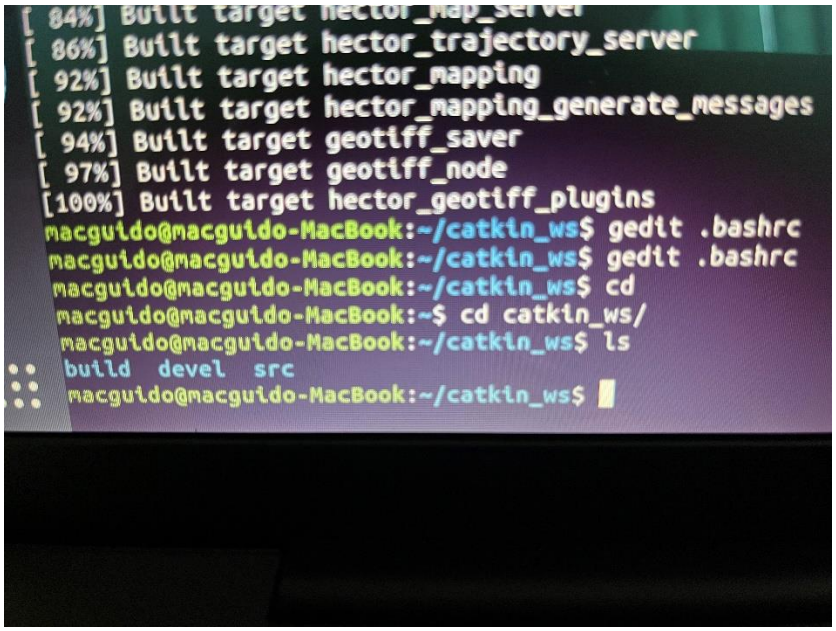

Figura 16. Rutas creadas al compilar con el comando catkin_make

Ahora colocarlo en las variables de entorno, pero se hace de una manera un poco manual. Se puede hacer con VIM pero para hacer lo más gráfico y amigable se realiza con el comando gedit pero para eso se debe de salir de la carpeta

```
cd
gedit .bashrc
```

Saldrá varias líneas de código, pero lo importante es irse hasta el final de ese archivo y observar una línea que se introdujo anteriormente, debajo de esa línea colocar lo siguiente

```
source ~/catkin_ws/devel/setup.bash
```

Se tendría que ver algo como lo siguiente

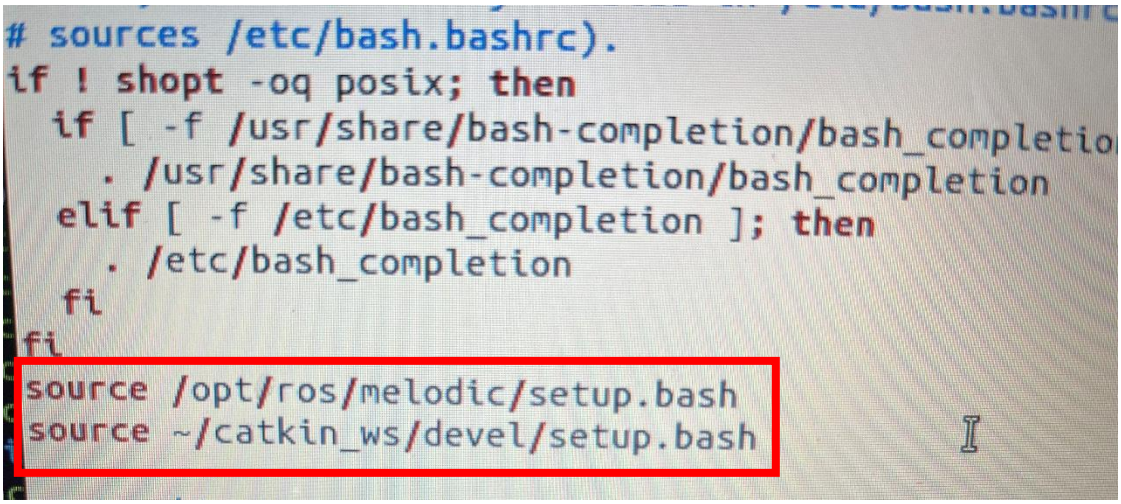

Figura 17. Líneas de Código dentro del .bashrc

Oprimir en guardar y salir.

Para actualizarlo:

```
source .bashrc
roscd
```

Y este último redirigirá a *~/catkin_ws/devel* con eso se confirma que fue exitoso y dar por terminado la ambientación.

3.2.4 Instalación de paqueterías y dependencias

Para comenzar a realizar el mapeo primero se debe instalar los paquetes necesarios de rplidar ros para posteriormente hacer HECTOR SLAM.

Los pasos para realizar esta tarea vienen en el siguiente link de Github donde detallan los pasos a seguir.

https://github.com/robopeak/rplidar_ros

Así que primero hay que clonar el repositorio del link dentro del folder src del workspace de catkin.

```
cd ~/catkin_ws/src
git clone https://github.com/robopeak/rplidar_ros.git
```

Una vez terminado hay que compilarlo. Se debe de hacer esta compilación siempre que se agregue un paquete nuevo al workspace de catkin

```
cd ~/catkin_ws
catkin_make
```

Se verá algo parecido a esto.

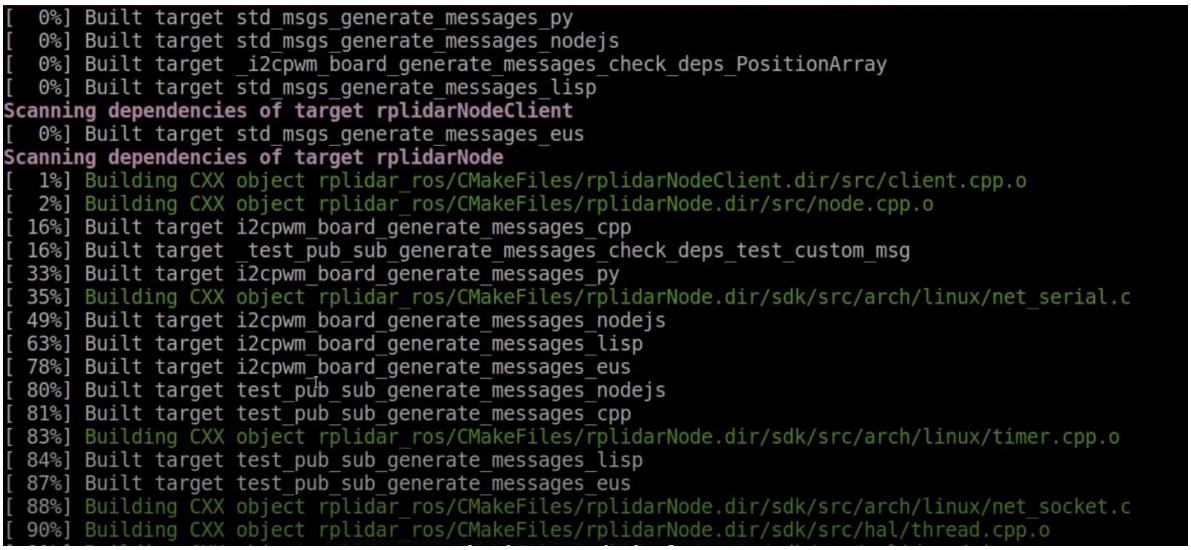

Figura 18. Screenshot de una compilación exitosa.

Una vez finalizado al 100% y satisfactoriamente, no hay que olvidar de actualizar el setup.bash con source. Para que funcione correctamente, se debe estar ubicado en la carpeta de catkin.

```
source devel/setup.bash
```

3.3 Inicio de roscore, RPlidar y Hector Slam

3.3.1 Configuración de RPlidar

Ahora que ya se tiene el paquete instalado y compilados, conectar el sensor y hacer que tenga los permisos necesarios para leer, escribir y ejecutar. Para esto primero hay que estar seguros de conectar correctamente el sensor RPlidar A1 correctamente.

Figura 19. Representación de conexión entre el sensor lidar y el hub.

Una vez conectada a la máquina, hay que darle permisos a este dispositivo con el siguiente comando, pero primero hay que comprobar que está detectando el dispositivo la máquina con lo siguiente:

```
ls -l /dev | grep ttyUSB
```

Ahora, darle permisos

```
sudo chmod 666 /dev/ttyUSB0
```

Y si se vuelve a correr el comando de arriba para verificar los permisos se verá que se obtiene

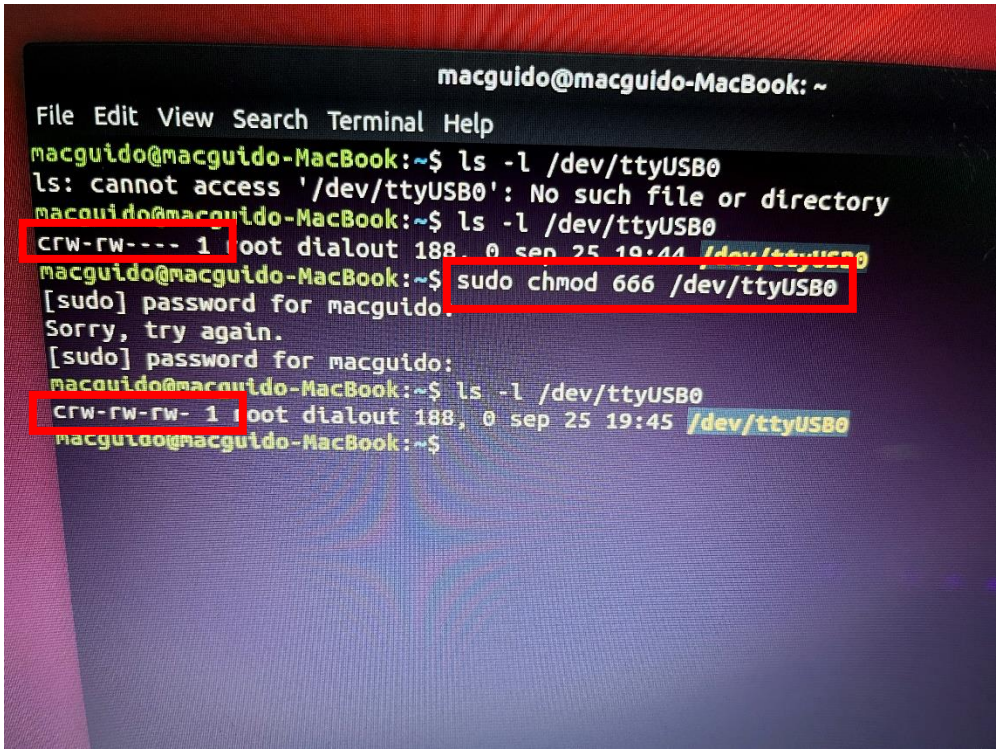

Figura 20. Demostración de permisos en el sensor.

Ahora si está todo listo para lanzar a ejecutar la lectura del rplidar, pero, siempre que se va a trabajar con ros, se debe de ejecutar *roscore* en una terminal independiente

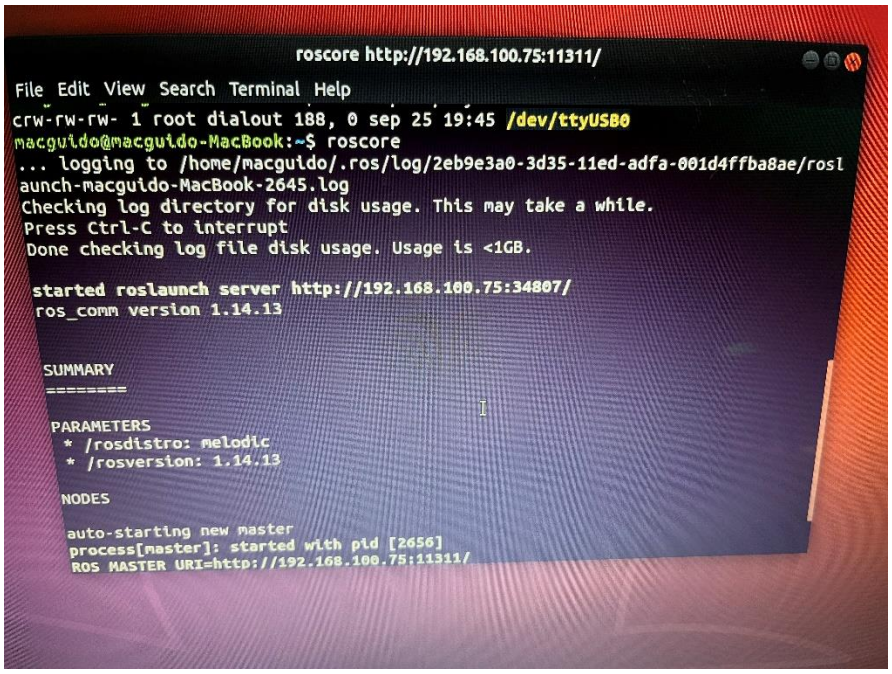

Figura 21. Comando roscore corriendo correctamente

En otra terminal, ejecutar el siguiente comando una vez que roscore esté activo

```
cd ~/catkin_ws
roslaunch rplidar_ros view_rplidar.launch
```

En seguida, abrirá la aplicación de RViz donde podrás apreciar el mapeo de tu entorno en tiempo real

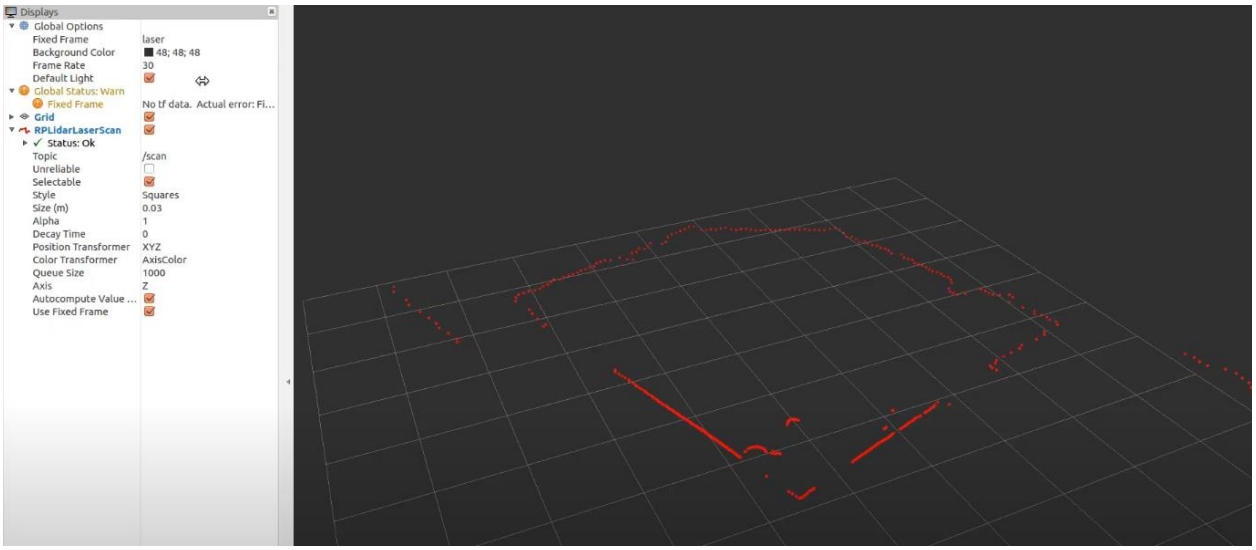

Figura 22. Mapeo del entorno en tiempo real visualizado desde el software RViz.

3.3.2 Hector Slam

Ahora que se confirma que el sensor funciona y los paquetes también, realizar Hector Slam.

Para eso se procede a descargar los paquetes correspondientes desde la de siguiente repo, y de nuevo, desde el workspace de catkin:

```
cd ~/catkin_ws
git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
source devel/setup.bash
catkin_make
```

Para poder este proceso es necesario modificar algunas líneas de código en algunos archivos. Estos archivos se encuentran en las siguientes rutas:

```
gedit ~/catkin_ws/src/hector_slam/hector_mapping/launch/mapping_default.launch
```

Y modificar las lineas que se señalan de tal manera que queden idénticas

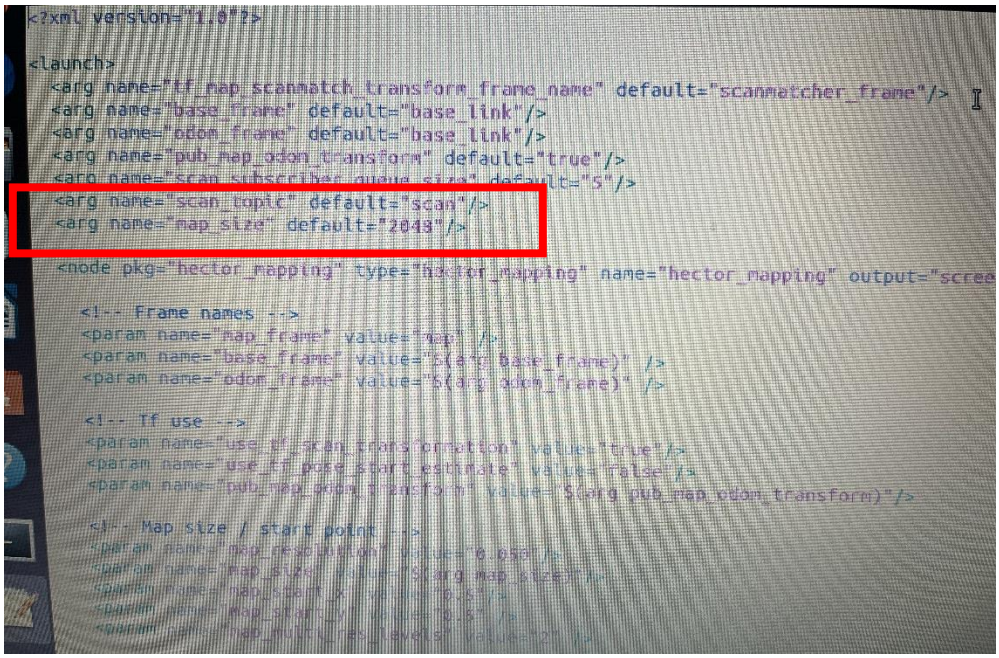

Figura 23. Líneas de Código por añadir en mapping_default.launch.

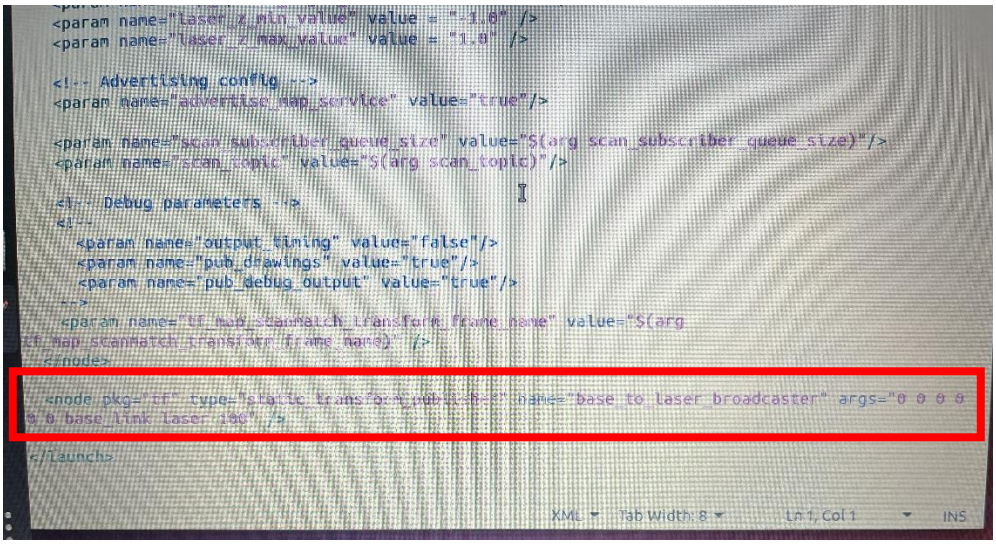

Figura 24. Líneas de Código por modificar en mapping_default.launch.

Ahora toca modificar el siguiente archivo

```
gedit ~/catkin_ws/src/hector_slam/hector_slam_launch/launch/tutorial.launch
```

```
7  <!-- <param name="/use_sim_time" value="true"/> -->
8  <param name="/use_sim_time" value="false"/>
9
```

Figura 25. Línea de Código en tutorial.launch.

Cuando termine de modificar estos archivos estará lista de lanzar a ejecutar.

Primero, y como siempre será, se ejecuta en una terminal *roscore*

```
tiffo@tiffo-ubuntu16:~/catkin_ws$ roscore
... logging to /home/tiffo/.ros/log/1d667854-9075-11e9-9491-080027f7d5d7/roslaunch-tiffo-ubuntu16-7739.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

En otra terminal, ejecutar rplidar

```
tiffo@tiffo-ubuntu16:~/catkin_ws$ roslaunch rplidar_ros rplidar.launch
```

Y ahora se ejecuta hector slam

```
cd ~/catkin_ws/src

roslaunch hector_slam_launch tutorial.launch
```

Y cuando termine y si todo está correcto abrirá RViz haciendo el SLAM en la interfaz como se muestra en la siguiente imagen:

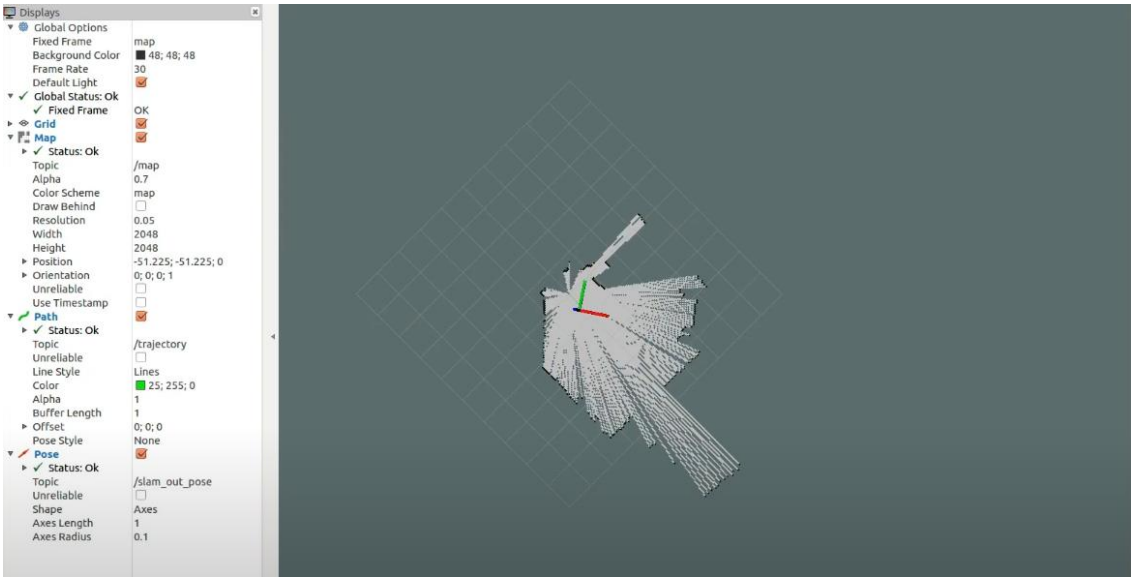

Figura 26. Demostración del SLAM en RViz.

3.4 Configuración remota master-slave

Para poder ejecutar de manera remota desde una maquina a distancia necesitamos dos máquinas, una donde va a actuar como maestro y otra como esclavo.

En la robótica actual, para estas situaciones, se usa un sistema embebido como una raspberry o nvida jetson, donde tienen los recursos computacionales suficientes para ejecutar estas tareas además de que son más accesibles y compactas, ideales para montarse en un robot. En este proyecto se usará la nvidia Jetson que actuará como esclavo.

Para tener esta configuración solo bastará con hacer unas modificaciones en amabs máquinas.

En la computadora donde actuará como master, regularmente una laptop o computadora normal, ejecute el siguiente comando para ir al archivo .bash

```
gedit ~/.bashrc
```

Y agregar lo siguiente sustituyendo tu IP.

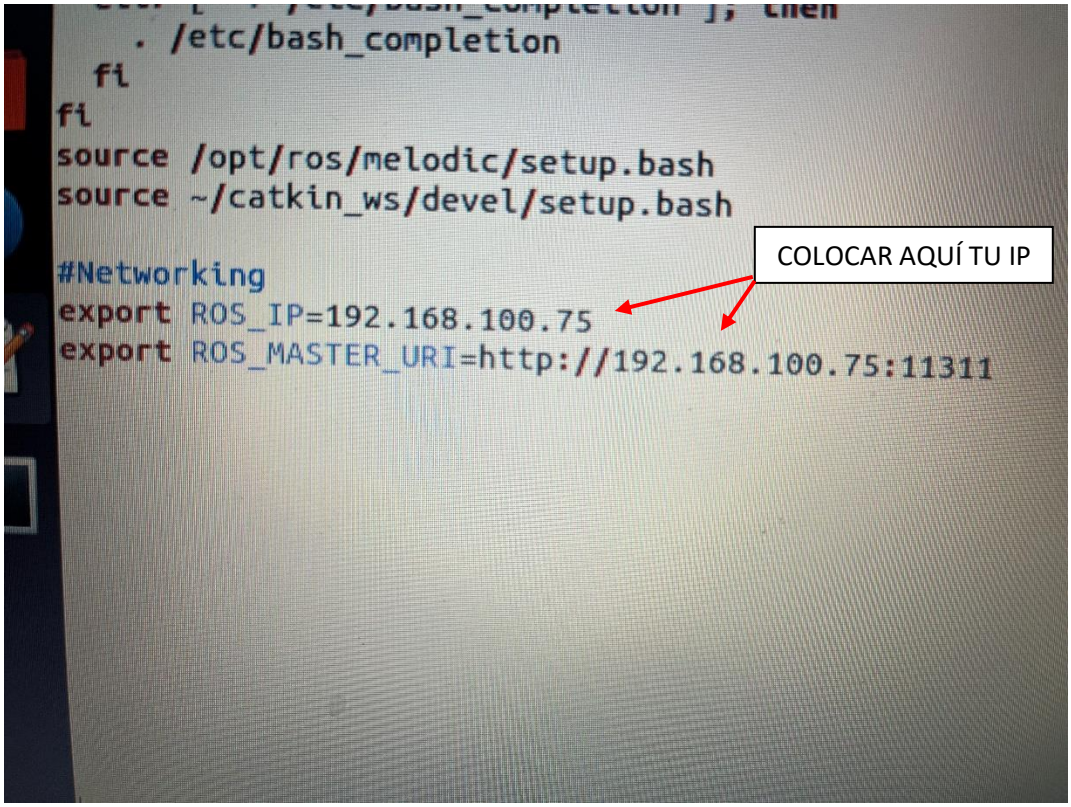

Figura 27. Configuración de IP's en el maestro.

Si se desconoce el IP de la máquina, ejecutar el siguiente comando

```
ifconfig
```

En la parte seleccionada en la siguiente imagen saldrá el IP

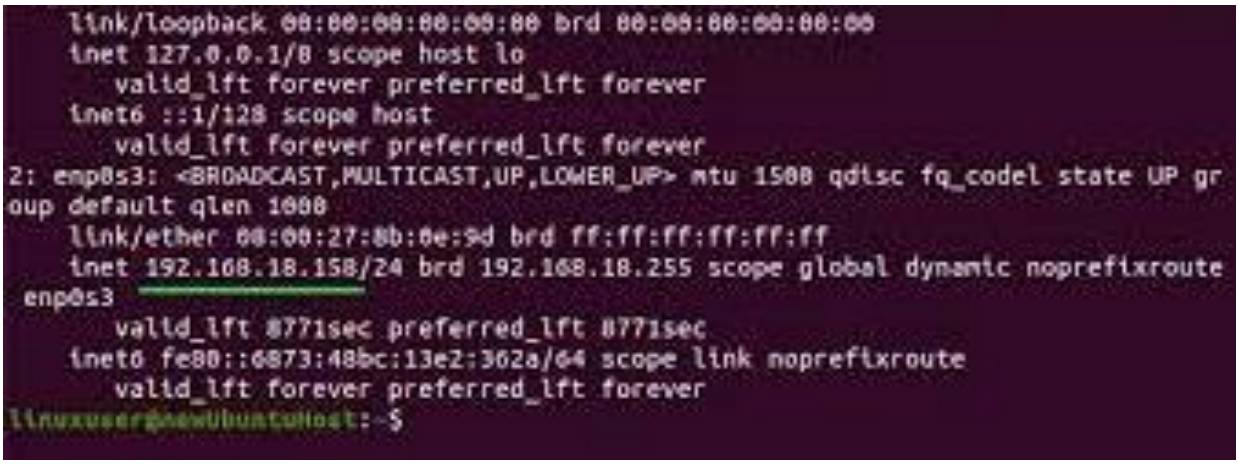

Figura 28. Ifconfig para indicar nuestra ip.

Ahora se tendrá que hacer lo mismo en la máquina esclavo, pero la diferencia aquí será que en comando *export ROS_MASTER_URI* será el mismo de la máquina master.

Una vez configurada, los pasos para ejecutar son los mismos, sin embargo, hay algunos que se tienen que ejecutar desde el esclavo y otros en el master. Se enlistará el orden de ejecución y en donde se debe de ejecutar:

1. roscore – master
2. Conectar sensor – slave
3. sudo chmod 666 /dev/ttyUSB0 – slave
4. roslaunch rplidar_ros rplidar.launch – slave
5. roslaunch hector_slam_launch tutorial.launch – master

En los siguientes segundos se vuelve abrir la pestaña de RViz en la máquina de master y se podrá apreciar el mapeo en tiempo real aún si este no está conectado al sensor directamente. Esto es provocado porque la transmisión es por HTTP y es por eso deben de estar conectados en la misma red local.

Capítulo IV.
Resultados

Para demostrar el procedimiento previamente dicho, se realizará algunos ejercicios en habitaciones diferentes donde se podrá apreciar el cambio del mapping.

Lo primero que se debe hacer es correr el comando *roscore* en la terminal tanto como la máquina *slave* como en la *master*.

Dentro de la máuina *slave* hay que conectar el sensor lidar y posterior a esto se debe correr el siguiente comando para dar permisos de lectura, escritura y ejecución a ese puerto donde está conectado el sensor.

roslaunch rplidar_ros rplidar.launch

Al ejecutar este comando habilitará al sensor y así tomar la lectura del sensor, es por eso que se debe correr desde donde está conectado el sensor, en este caso, desde el *slave* y no desde el *master.*

Por último, en a la carpeta de *catkin_ws* y se debe de ejecutar el siguiente comando para ejecutar ***hector slam*** y así lanzar la aplicación de visualización Rviz, lo que permitirá ver el mapeo en tiempo real y, si se desea, guardar en la computadora el mapeo.

roslaunch hector_slam_launch tutorial.launch

En seguida comenzará a ejecutarse la aplicación Rviz y dentro de la aplicación el mapeo de nuestro entorno. Para hacer esto lo mejor posible, se alzó el sensor a una altura que pudiera pasar de los cuerpos, así no detectaría el sensor y solo mapearía las paredes y algunos objetos de la habitación.

Demostración del cuarto Número 1.

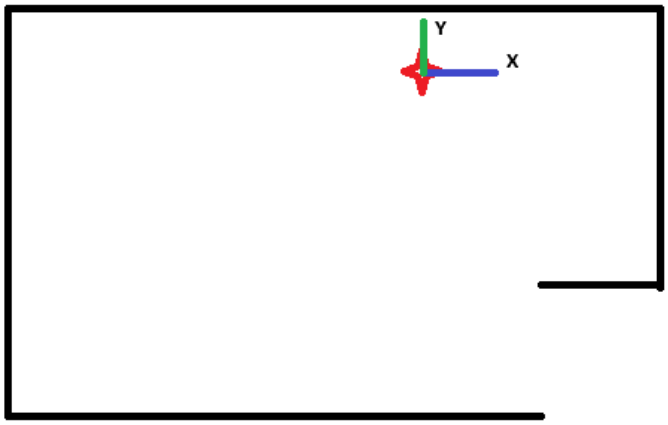

Figura 29. Layout de Cuarto No.1 y punto de partida del senseo.

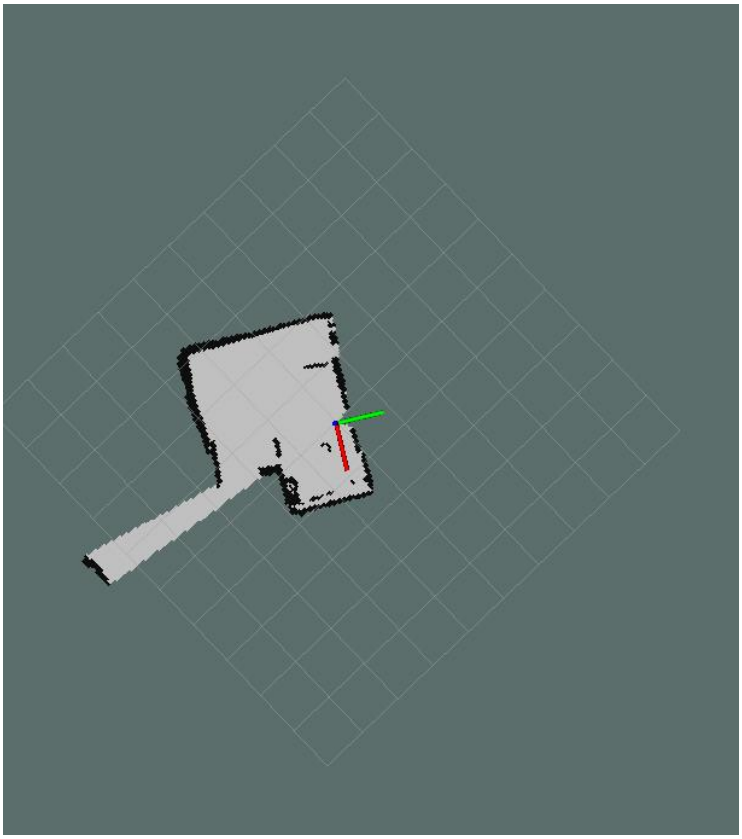

Figura 30. Toma de lectura del mapeo del cuarto No.1.

Como se aprecia en el layout de muestra, se observa un cuarto cerrado y solo con una puerta la cual se encontrada abierta al momento de la prueba.

Demostración del cuarto Número 2.

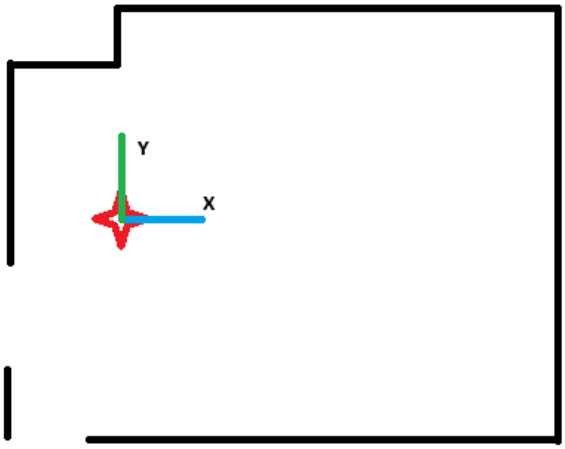

Figura 31. Layout de Cuarto No.2 y punto de partida del senseo.

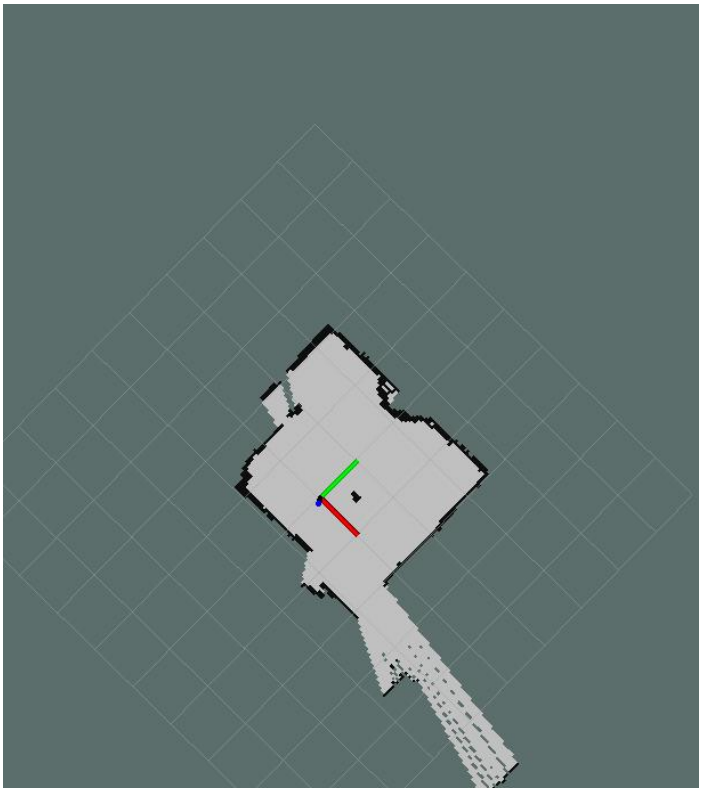

Figura 32. Toma de lectura del Cuarto No.2.

En la siguiente demostración se observa un cuarto un poco diferente, también, no solo una puerta abierta, sino dos puertas abiertas las cuales se identifican por la profundidad para localizar una pared o una superficie sólida.

Demostración del cuarto Número 2.

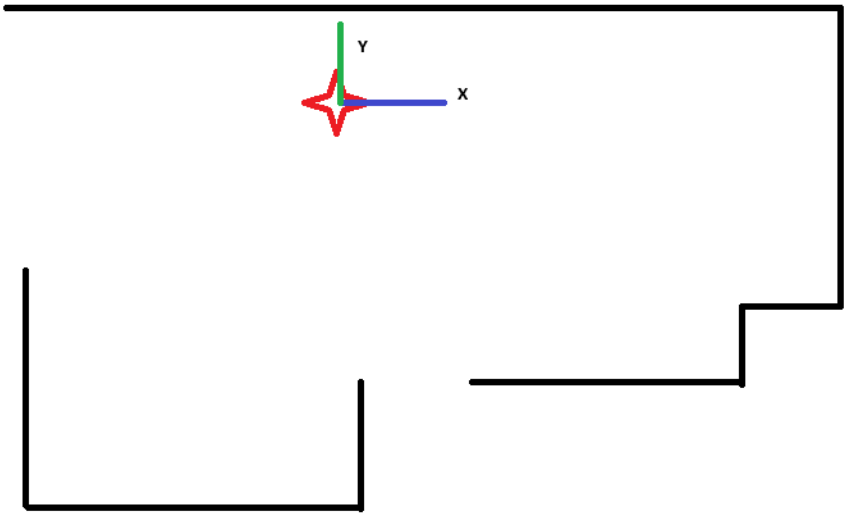

Figura 33. Layout de Cuarto No.3 y punto de partida del senseo.

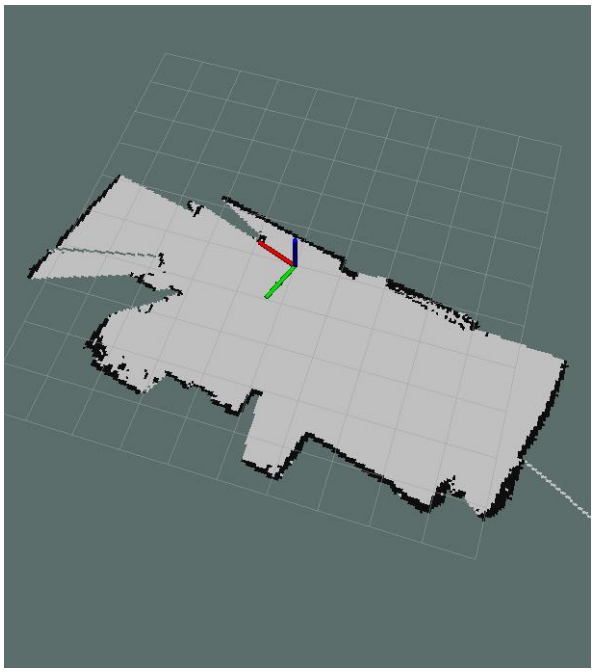

Figura 34. Toma de lectura del Cuarto No.3.

Por último, en este cuarto hay muchos objetos de por medio provocando mucho ruido, sin embargo, logramos detectar las paredes principales para darle la misma forma como está en el layout.

Capítulo V.
Conclusiones

Se logró obtener el sistema de visualización mediante el uso de la técnica Hector SLAM en conjunto con la tecnología ROS, pero no se implementa en algún robot. Se logra visualizar de manera remota cualquier entorno terrestre por medio de un sensor de tipo LiDAR que escanea y procesa la información en tiempo real.

Estos mapas no solo se visualizan en tiempo real, se pueden guardar para su análisis más detalladamente. Se puede implementar a fututo en un robot autónomo que detecte los objetos y pueda evadir cualquier obstáculo que impida llegar a su destino y concluir con la exploración del entorno por completo.

Finalmente, el tener un mapa en tiempo real de zonas inexploradas, ayuda a personas que se dedican a trabajos de exploración, puede ser útil para la prevención de posibles accidentes, ya que se puede conocer previamente el camino, tomando así, mejores decisiones posteriormente.

Fuentes de Información

[1] Department of Rural Development and Land Reform, «ngi,» 2013. [En línea]. Available: https://ngi.dalrrd.gov.za/index.php/technical-information/catography/what-is-cartography.

[2] R. d. Murcia, «Cartografía básica,» 2024. [En línea]. Available: https://sitmurcia.carm.es/cartografia-basica.

[3] Geographic Information Technology Training Alliance (GITTA), «Thematic Cartography,» 2017. [En línea]. Available: http://www.gitta.info/ThematicCart/en/text/ThematicCart.pdf.

[4] A. Koubaa, Robot Operating System (ROS) The complete Reference (Volume 1), Switzerland: Springer , 2016.

[5] I. P. N. UPIITA, «SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM) UTILIZANDO PAQUETES DEL AMBIENTE ROBOT OPERATING SYSTEM (ROS),» 2022. [En línea]. Available: https://www.boletin.upiita.ipn.mx/index.php/ciencia/971-cyt-numero-88/2014-simultaneous-localization-and-mapping-slam-utilizando-paquetes-del-ambiente-robot-operating-system-ros.

[6] Game Manual 0, «gm0,» [En línea]. Available: https://gm0.org/es/latest/docs/software/concepts/odometry.html.

[7] &. Y. Yuridia, «Sensores,» 2023. [En línea]. Available: https://sdindustrial.com.mx/blog/sensores/.

[8] A. &. F. E. Martinez, Learning ROS for robotics programming : a practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework., Packt, 2023.

[9] D. O. Delgado, «Que es ROS (Robot Operating System),» 2017. [En línea]. Available: https://openwebinars.net/blog/que-es-ros/.

[10] ROS, «Why ROS?,» 2021. [En línea]. Available: https://www.ros.org/blog/why-ros/.

[11] S. Luchetti, «Sistemas embebidos y sus características,» [En línea]. Available: https://tech.tribalyte.eu/blog-sistema-embebido-caracteristicas.

[12] M. F. Rodríguez, «Sistemas embebidos Smart: La era del Internet de las Cosas,» 2004. [En línea]. Available: https://www.infotec.mx/sistemas-embebidos-smart-la-era-del-internet-de-las-cosas.

[13] opensource, «What is a Raspberry Pi,» 2024. [En línea]. Available: https://opensource.com/resources/raspberry-pi.

[14] Nvidia, «NVIDIA Jetson Nano,» [En línea]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/.

[15] Jithin. Interserver, «Protocolos de red comunes,» 2016. [En línea]. Available: https://www.interserver.net/tips/kb/common-network-protocols-ports/.

[16] AWS, «¿Qué es el modelo OSI?,» [En línea]. Available: https://aws.amazon.com/es/what-is/osi-model/.

[17] concepto, «Sistema Operativo,» [En línea]. Available: https://concepto.de/sistema-operativo/.

[18] Alberto, «Botones.com,» [En línea]. Available: https://www.geograma.com/blog/slam-cartografia-gis/#:~:text=SLAM%20son%20las%20siglas%20de,%C3%A9ste%20es%20desconocido%20para%20%C3%A9l.. [Último acceso: 17 06 2024].

[19] A. Koubaa, Robot Operating System (ROS) The complete Reference (Volume 3), Springer, 2019.

[20] A. Koubaa, Robot Operating System (ROS) The complete Reference (Volume `2), Springer, 2017.

[21] B. G. &. W. D. S. Morgan Quigley, Programmin Robots with ROS, Sebastopol, CA: O'Reilly, 2015.

[22] L. Joseph, Robot Operating System for Absolute Beginners: Robotics Programming, apress, 2018.

[23] H. C. R. J. T. L. YoonSeok Pyo, ROS Robot Programming, Seoul, Republic of Korea: ROBOTIS Co.,Ltd., 2017.

[24] R. Fox, Linux with Operating System Concepts, CRC Press, 2014.

[25] J. Bates, Linux Mastery The Ultimate Linux Operating System and Command Line Mastery, CreateSpace Independent Publishing Platform, 2016.

[26] T. L. Warner, Hacking Raspberry Pi, Que Publishing, 2013.

[27] M. Zeiler, Exploring ArcObjects - Applications and Cartography [Vol. 1], ESRI, 2001.

[28] C. Weitkamp, Lidar: Range-Resolved Optical Remote Sensing of the Atmosphere, Springer, 2005.

[29] J. Young, LiDAR For Dummies, Wiley, 2011.

[30] A. Kurniawan, IoT Projects with NVIDIA Jetson Nano: AI-Enabled Internet of Things Projects for Beginners, apress, 2021.

COMPETENCIAS DESARROLLADAS

1. Identificar, formular y resolver problemas de ingeniería aplicando los principios de las ciencias básicas e ingeniería.

2. Aplicar, analizar y sintetizar procesos de diseño de ingeniería que resulten en proyectos que cumplen las necesidades especificadas.

3. Desarrollar y conducir una experimentación adecuada; analizar e interpretar datos y utilizar el juicio ingenieril para establecer conclusiones.

4. Comunicarse efectivamente con diferentes audiencias.

5. sus responsabilidades éticas y profesionales en situaciones relevantes para la ingeniería y realizar juicios informados, que consideren el impacto de las soluciones de ingeniería en los contextos global, económico, ambiental y social.

6. Reconocer la necesidad permanente de conocimiento adicional y tener la habilidad para localizar, evaluar, integrar y aplicar este conocimiento adecuadamente.

7. Trabajar efectivamente en equipos que establecen metas, planean tareas, cumplen fechas límite y analizan riesgos e incertidumbre.