ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESTUDIO, CONTROL E IMPLEMENTACIÓN DE SISTEMAS ROBÓTICOS AVANZADOS

APLICACIÓN DE ALGORITMOS ITERATIVOS DE PLANIFICACIÓN DE RUTAS (PATH PLANNING) EN UN SISTEMA MULTI-AGENTE DENTRO DEL AMBIENTE DE ROS-GAZEBO

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN

EDISON ANDRÉS SANTILLÁN PULLUTASIG

edison.santillan@epn.edu.ec

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PHD.

patricio.cruz@epn.edu.ec

DMQ, agosto 2022

CERTIFICACIONES

Yo, Edison Andrés Santillán Pullutasig, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

EDISON ANDRÉS SANTILLÁN PULLUTASIG

Certifico que el presente trabajo de integración curricular fue desarrollado por Edison Andrés Santillán Pullutasig, bajo mi supervisión.

ING. PATRICIO JAVIER CRUZ DÁVALOS, PHD
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

EDISON ANDRÉS SANTILLÁN PULLUTASIG

ING. PATRICIO JAVIER CRUZ DÁVALOS, PHD

DEDICATORIA

Este trabajo va dedicado a Dios por darme la fuerza para culminar esta nueva etapa en mi vida.

A mi madre que con su amor incondicional y su lucha constante me permitió cumplir mi sueño de ser un gran profesional, este logro es para usted madre querida.

A mi abuelita que fue mi segunda mamá y ahora es un angelito que guía mi camino desde el cielo. Llegaré muy lejos mamita Lolita.

A Ruby por ser la persona que me complementa y que con su sonrisa ilumina mi vida.

A mi pequeña Eira que es mi inspiración y fortaleza para dar lo mejor de mi cada día.

Con mucho amor, **DísonAndres.**

AGRADECIMIENTO

Agradezco de todo corazón a mi madre por apoyarme en cada decisión tomada, por motivarme a seguir adelante y jamás rendirme; por arriesgarse y darlo todo para que culmine con éxito mi vida universitaria.

A mi Ruby por ser mi compañera y amiga durante todo el trayecto de la universidad; por ser la persona más tolerante y comprensible en la tierra, sin tu ayuda no lo hubiese logrado mi compañera de vida.

A mi Eira que son sus pasitos y sus risitas han ayudado como inspiración.

A Mafer, Arturo y Heidy por todo el apoyo que me han brindado, muchas gracias.

A Daniel Santacruz por ser un gran amigo, testigo de boda y testigo de las experiencias vividas a lo largo de mi vida universitaria.

A mis tios que jamás dudaros que lo lograría y a mis queridos primos que sin su apoyo no hubiese sido esto posible.

Al Ing. Patricio Cruz Ph.D. por ser una guía y brindarme su apoyo a lo largo de este trabajo de titulación.

Al MSc. Diego Maldonado por ser el concejero excepcional durante cada semestre, un gran tutor y amigo, muchas gracias.

Un Agradecimiento especial a la Escuela Politécnica Nacional por la acogida durante estos años, por permitirme avanzar académica y profesionalmente; considerándose, así como mi querida alma mater.

INDICE DE CONTENIDO

RESUMEN

En el presente trabajo de integración curricular se diseña un sistema multi-agente (SMA) y se ponen a prueba algoritmos de planeación de rutas. Esta aplicación es de fundamental importancia, por ejemplo, dentro de la exploración y reconocimiento de áreas a través de robots móviles. Como primer enfoque se hace un estudio bibliográfico del sistema operativo robótico (ROS), que en la actualidad ayuda a muchos desarrolladores de software a crear aplicaciones robóticas empleando plataformas de código abierto. Además, posee una serie de librerías y complementos como, por ejemplo, Gazebo y Rviz que son herramientas de simulación y visualización 3D. En sí, en este proyecto, la creación del entorno virtual se lleva a cabo en Gazebo y este a su vez se mapea desde Rviz con la técnica de Localización simultanea y mapeo (SLAM). Terminado el diseño del entorno se implementa el algoritmo de árboles aleatorios de expansión rápida (RRT) y el algoritmo de mapas de rutas probabilísticas (PRM). La generación de rutas de ambos algoritmos es de manera diferente, ya que el primero traza caminos en forma de ramas, mientras que el segundo toma muestras del entorno. Las pruebas de estos algoritmos dentro del SMA se llevan a cabo variando sus parámetros y añadiendo obstáculos al entorno que, como resultados, se obtienen diferentes rutas adecuadas, comparándolas, por ejemplo, a través del cálculo de las distancias de recorrido y sus tiempos de simulación.

PALABRAS CLAVE: Sistema Multi-agente, Planeación de Rutas, Algoritmos RRT, Algoritmo PRM, ROS, Gazebo, Rviz.

ABSTRACT

In this project, a Multi-agent System (SMA) is designed and path-planning algorithms are tested. This application is critical, for example, within the exploration and recognition of areas through mobiles robots. As a first approach, a bibliography study of the robot operating system (ROS) is carried out. It helps currently many software developers to create several robotic applications by employing open-source platforms. Furthermore, it has many complements and libraries, for example, Gazebo and Rviz that are tools for 3D simulation and visualization, respectively. In this project, the creation of a virtual world is developed in Gazebo and then this is mapped from Rviz with the SLAM (Simultaneous Localization and Mapping) technique. Finishing the designing of the virtual world, the rapidly exploring random trees (RRT) algorithm and the probabilistic roadmap (PRM) algorithm are implemented. The generation of paths by both algorithms are different; the first one traces the paths in form of branches while the second takes samples from the virtual environment. The tests of these algorithms with the SMA are carried out by modifying their parameters and including several obstacles to the virtual environment. As results, different paths are found and compared, for example, through the computation of the navigation distance and their simulation time.

KEYWORDS: Multi-agent System, Path Planning, RRT Algorithm, PRM Algorithm, ROS. Gazebo, Rviz.

1 INTRODUCCIÓN

La robótica con los avances tecnológicos e innovadores de los últimos años no deja de impactar al ser humano en todos los campos donde ha sido implementada. Por ejemplo, la planificación de rutas de robots móviles se ha involucrado en varios ámbitos de la vida cotidiana, en campos científicos y domésticos; presentándose desde aspiradoras robotizadas hasta mecanismos complejos de exploración [1]. El interés de estudiantes, e investigadores por el control de robots móviles ha impulsado el desarrollo de aplicaciones para la planificación de rutas, la localización y la evasión de obstáculos [2].

Sin embargo, la implementación de sistemas robóticos requiere grandes inversiones económicas, ya que cuentan con tarjetas de desarrollo electrónico, tarjetas de adquisición de datos, dispositivos de monitoreo, sensores, carcazas, entre otros elementos electrónicos; adicionalmente se debe tomar en cuenta el tiempo de elaboración y calibración. Por lo cual la implementación física limita la interacción con sistemas robóticos, si no se cuentan con los recursos necesarios; sin embargo, existen otras alternativas como entornos de simulación con las mismas precisiones, características y limitaciones físicas.

Por esto, para mejorar la experiencia e interacción con sistemas robóticos se han desarrollado entornos de simulación, como la plataforma de Robot Operating System (ROS), que cumple un papel muy importante en la búsqueda de mecanismos óptimos y eficientes, creando sistemas embebidos sin la dependencia física de hardware [3]. El entorno de simulación de ROS contiene una serie de herramientas y librerías que corren en plataformas de software libre como Linux. Al ser un software de código abierto ha habido un sin número de contribuciones alrededor del mundo; como el caso de Gazebo, que es un entorno de visualización 3D multi-robot que se enfoca en simular la dinámica de robots, sensores y objetos en diferentes mundos virtuales [3], [4].

En ROS-Gazebo es posible elaborar entornos virtuales, donde los escenarios son modelos estacionarios mientras que los robots y otros objetos son dinámicos; por ejemplo, los sensores son separados del simulador dinámico, ya que únicamente recolectan o emiten información si existe alguna actividad del sensor [4]. La simulación de los diferentes escenarios en ROS-Gazebo tiene como finalidad evaluar el comportamiento dinámico de robots y mejorar los tiempos de ejecución de tareas mediante código. Es así como, la planeación de rutas para robots móviles en ROS-Gazebo se integra cada vez más en el campo de la investigación y educación,

permitiendo interactuar con modelos, algoritmos, estrategias, planeación o exploración [1], [4].

El presente proyecto de titulación tiene como producto final demostrable una plataforma multi-robots (multi-agentes) homogéneos, donde se cuenta con un escenario virtual con obstáculos estáticos en el entorno de simulación de ROS-Gazebo. Dentro del escenario interactúan dos robots móviles de tracción diferencial, para ello, se mapea a los robots y el entorno con ayuda de la herramienta de visualización 3D Rviz; donde se obtienen las vistas de los modelos de los robots, el entorno, la planeación de trayectorias y su seguimiento. Para llevar a cabo con la planeación de rutas se implementan los algoritmos Probabilistic Roadmap Method (PRM) y Rapidly-exploring Random Trees (RRT*).

1.1 OBJETIVO GENERAL

Aplicar algoritmos iterativos de planificación de rutas (path planning) en un sistema multi-agente dentro del ambiente de ROS-Gazebo.

1.2 OBJETIVOS ESPECÍFICOS

- Realizar una recopilación bibliográfica de robots móviles de tracción diferencial, métodos de planeación de rutas con algoritmos iterativos, escenarios con obstáculos y técnicas de mapeo en la plataforma de simulación de ROS-Gazebo con Rviz.
- Diseñar un escenario virtual con obstáculos estáticos en ROS-Gazebo e integrar al menos dos robots virtuales homogéneos de tracción diferencial.
- Implementar algoritmos iterativos con al menos dos métodos de planeación de rutas a los robots integrados en el escenario virtual.
- Desarrollar un sistema de monitoreo para la navegación de rutas de los robots integrados con los algoritmos de planeación de rutas en el entorno de visualización 3D Rviz.
- Evaluar el funcionamiento de los algoritmos iterativos implementados y la integración del sistema completo; sobre todo, respecto al desempeño en la generación de rutas.

1.3 ALCANCE

- Se realiza una recopilación bibliográfica enfocada a robots móviles de tracción diferencial que son aplicados dentro de la plataforma de ROS, donde se revisa el funcionamiento y ejecución dentro de ROS-Gazebo; además, se estudian las aplicaciones y herramientas disponibles para ROS como es el caso del entorno de visualización 3D Rviz.

- Se realiza un estudio sobre el mapeo de robots móviles en la plataforma de visualización 3D Rviz y el desarrollo de entornos virtuales para sistemas multi-agentes en ROS-Gazebo, donde se establece la comunicación mediante nodos y tópicos de los robots que se deben identificar para llevar a cabo la aplicación de planificación de rutas.
- Se realiza una recopilación bibliográfica de algoritmos iterativos con varios métodos de planeación de rutas como, por ejemplo, Probabilistic Roadmap Method (PRM), Rapidly-exploring Random Trees (RRT), Expansive Space Trees (EST), SPArse Roadmap Spanner algorithm (SPARS), entre otros.
- Se diseña e implementa un escenario virtual con obstáculos estáticos en el software de simulación de ROS, donde se incluye al menos dos robots móviles homogéneos de tracción diferencial; y se implementa un mecanismo de comunicación dentro de ROS para mapear a los robots móviles en Rviz y el escenario virtual diseñado en Gazebo.
- Se implementan dos algoritmos de planeación de rutas a cada robot móvil que se integró en el escenario virtual de ROS-Gazebo y se visualiza en la herramienta 3D Rviz a los robots móviles que tan bien siguen las rutas sin colisionarse dentro del escenario.
- Se realiza una integración de todas las aplicaciones diseñadas e implementadas en ROS, como la etapa del sistema de monitoreo en Rviz, la aplicación del entorno en ROS-Gazebo y los algoritmos para la planeación de rutas, con la finalidad de formar un sistema integral.
- Se evalúa el funcionamiento del sistema implementado mediante pruebas de los algoritmos, cambiando sus parámetros, como por ejemplo el número de iteraciones requeridas hasta encontrar una ruta libre de colisiones; y se pone a prueba al sistema de monitoreo en Rviz y el entorno en ROS-Gazebo.
- Se evalúa el desempeño de la aplicación de planeación de rutas mediante el cambio de posiciones de los robots móviles, variación de puntos de referencia inicio-llegada y la colocación de obstáculos que ingresó el usuario en puntos arbitrarios para posteriormente iniciar la navegación de las rutas de cada robot.

1.4 MARCO TEÓRICO

1.4.1 SISTEMA OPERATIVO ROBÓTICO (ROS)

ROS es una plataforma robótica de código abierto que posee una serie de herramientas, librerías y complementos como frameworks que ayudan a desarrolladores de software a crear aplicaciones robóticas [3]. Esta plataforma no es una estructura que corre en tiempo real, sin embargo, hace posible la integración de código vivo, es decir, corre el código en tiempo real. ROS tiene como objetivo primordial soportar el reúso de códigos (archivos

ejecutables) durante el desarrollo de las aplicaciones robóticas, ya que al ser una estructura distribuida de procesos permite la ejecución individual de códigos; por lo cual, a esta estructura se la denomina nodos y pese a que se ejecutan de forma individual, la integración con el proceso principal es de manera sencilla [5].

Los nodos en ROS pueden estar escritos en Python, C++ o Lips; y su comunicación se la hacen a través de tópicos, ya que un nodo puede estar publicando un tópico de una variable y otro se suscribe al mismo y recibe los datos transmitidos. Este sistema se basa en enviar y recibir datos a través de una estructura de mensajes y servicios, tal como se puede ver en la Figura 1.1 [3], [5].

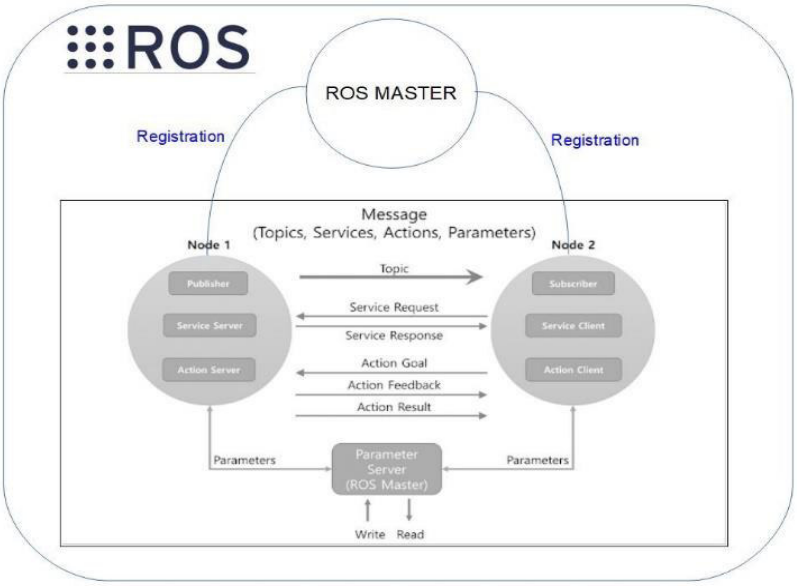


Figura 1.1. Arquitectura ROS [5]

1.4.1.1 Herramienta de simulación 3D Gazebo

Dentro de las herramientas que contiene ROS existe un framework de simulación 3D multi-robot llamado Gazebo. Este simulador 3D está diseñado para emular la dinámica y cinemática de robots en entornos complejos, adicionalmente, permite comprobar y calibrar a los algoritmos que se desarrollan [3]. La simulación se lleva a cabo en un mundo tridimensional donde se tiene un enfoque realista tanto la respuesta de los sensores como la interacción física de los robots [3], [4]. El enfoque realista de Gazebo llega gracias a las técnicas de renderizado y las librerías que contiene ROS, ya que se dividen en la interfaz de usuario, motores de física, motores de renderizado, métodos de comunicación y tipos sensores [4], [6].

La integración de Gazebo dentro de ROS es de forma independiente mediante un conjunto de paquetes que se denominan gazebo_ros_pkgs, donde se proporcionan todos los

complementos necesarios para la visualización y simulación de un robot. Para ello, Gazebo se debe de comunicar con ROS mediante mensajes y servicios para que se proporcionen las interfaces de los robots, de este modo se puede simular toda la dinámica de los robots que van desde la masa, velocidad, fricción, aceleración, etc.

Con la integración de ROS-Gazebo se permite probar algoritmos rápidamente, modelar robots, realizar pruebas de regresión y entrenar sistemas de inteligencia artificial. Además, proporcionan un alto nivel de precisión y eficiencia que tiene un grado mayor a los motores de videojuegos, tal como se puede ver en la Figura 1.2 [6].

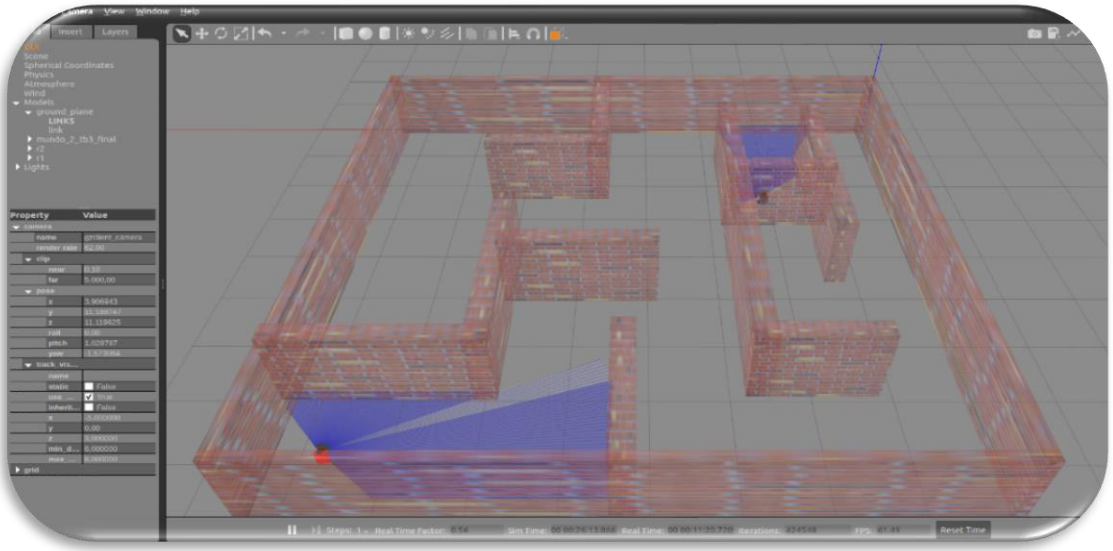


Figura 1.2. Entorno en Gazebo [Fuente Propia]

1.4.1.2 Herramienta de visualización 3D Rviz

Dentro del entorno de ROS se incluye también a una herramienta de visualización 3D llamada Rviz que tiene la finalidad de mostrar en formas gráficas a los tópicos que se comunican entre nodos. Esta herramienta cuenta con una interfaz de usuario donde se pueden observar las posiciones y las formas que van adquiriendo los robots en el tiempo [3], tal como se puede ver en la Figura 1.3.

En Rviz se pueden combinar varios modelos de robots (móviles o manipuladores) en una misma pantalla y para que funcione de manera correcta necesita de datos para poder visualizarlos. Dentro de Rviz no se generan datos, sin embargo, estos pueden ser enviados desde Gazebo o archivos ejecutables. En Gazebo se crean datos mediante los sensores de los robots cuando toman lecturas del entorno, que con ayuda de archivos ejecutables se pueden crear marcadores para enviar a Rviz. Por tal motivo, si Rviz trabaja en conjunto con Gazebo se puede registrar, analizar y visualizar la evoluciona de la aplicación robótica.

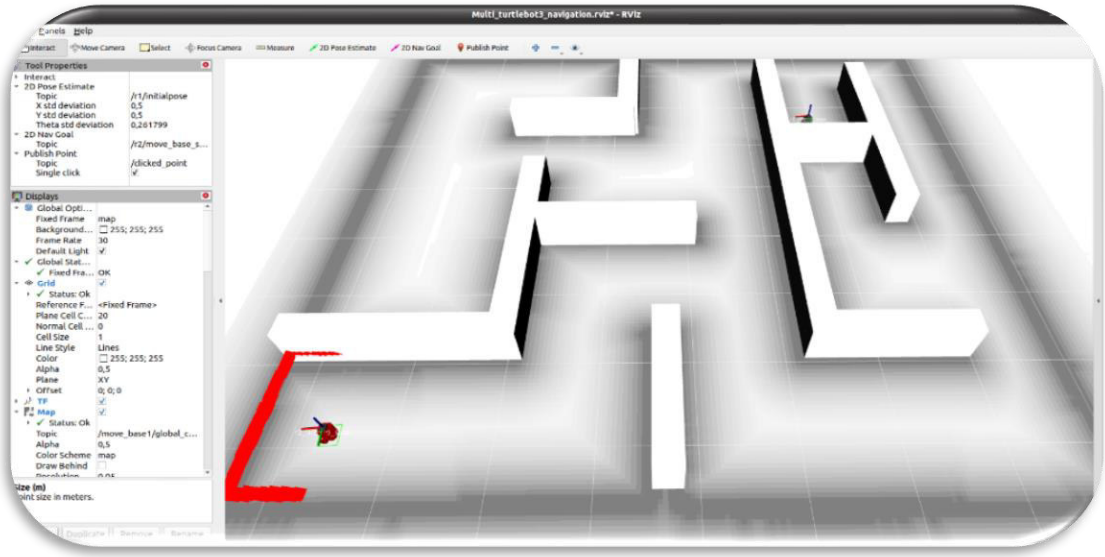
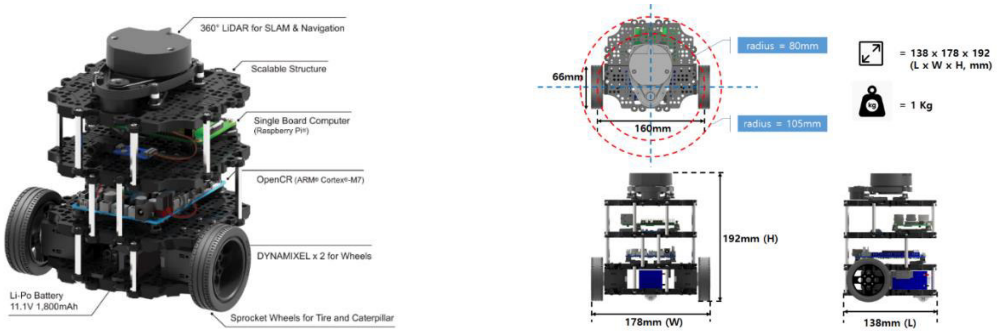


Figura 1.3. Entorno en Rviz [Fuente Propia]

1.4.2 ROBOT MÓVIL EN EL ENTORNO DE ROS (TURTLEBOT3 BURGER)

Para llevar a cabo proyectos robóticos dentro de ROS se cuenta con librerías de varios modelos de robots que ayudan a los desarrolladores de software a optimizar los tiempos de entrega se sus proyectos, estos robots ya se encuentran implementados tanto en el entorno de Gazebo como en Rviz, tal es el caso del robot TurtleBot3 Burguer.

El modelo del robot TurtleBot3 Burger se lo puede visualizar en la Figura 1.4, y este se basa en un robot móvil de tracción diferencial que es pequeño, asequible y programable. El desarrollo tecnológico de este robot se basa en la ejecución de algoritmos de navegación, manipulación, localización y mapeo simultaneo (SLAM), donde, el robot TurtleBot3 Burger puede construir mapas y conducir a través de estos autónomamente o se los puede controlar de forma remota. Este robot tiene la finalidad es contribuir en la educación e investigación [7].



(a) Robot Físico (b) Dimensiones

Figura 1.4. Características del Robot TurtleBot3 Burguer [8]

1.4.2.1 Características principales

El robot TurtleBot3 Burger está conformado por 4 niveles, tal cómo se puede ver en la Figura 1.4. En el primero se encuentra la batería Li-Po y dos motores DYNAMIXEL XL430 acoplados con sus respectivas ruedas; en el segundo se encuentra una tarjeta de potencia OpenCR que proporciona la alimentación para la Raspberry PI3® y varios sensores; en el tercer nivel se encuentra la tarjeta de desarrollo Raspberry PI3®; finalmente, en el cuarto y último nivel se encuentra el sensor de distancia laser de 360 grados LDS-01. Adicionalmente, sus características generales se resumen en la Tabla 1.1.

Tabla 1.1. Características Generales, TurtleBot3 Burguer [8].

Características	
Velocidad máxima	0.22 m/s
Velocidad máxima de rotación	2.84rad/s
Tiempo de funcionamiento aproximado	2h30m
Tiempo de carga	2h30m
Audio	Pitidos programables
Batería	Polímero de litio de 11.1V
Sensor de distancia Laser (LDS)	360 grados LDS-01
Pines de conexión	GPIO 18 pines, Arduino 32 pines
Conectores de alimentación	3.3 V a 800 mA, 5 V a 4 A, 12 V a 1 A
Unidad de Medición Inercial (IMU)	Giroscopio, acelerómetro, magnetómetro
Conexión a PC	USB

1.4.2.2 Modelo cinemático [9]

El robot TurtleBot3 Burger, al ser un robot móvil de tracción diferencial, tiene restricciones holonómicas, lo que significa que el robot solo puede moverse hacia adelante o atrás, más no realizar desplazamientos laterales. Para su modelo cinemático se establece un punto P(x_0 , y_0) en un eje de coordenadas x-y, donde P(x_0 , y_0) es considerando como el centro de gravedad del robot TurtleBot3 con respecto al eje de referencia C y está ubicado a una distancia α de la línea central de las ruedas, tal como se muestra en la Figura 1.5.

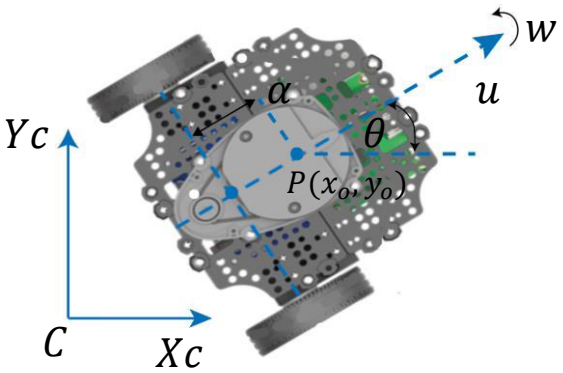


Figura 1.5. Modelo cinemático del robot TurtleBot3 Burger [Fuente Propia]

De la Figura 1.5, se definen las siguientes ecuaciones que representa al modelo cinemático del robot.

$$\dot{x} = u\,cos(\theta) - \alpha\omega\,sen(\theta) \tag{1.1}$$

$$\dot{y} = u\,sen(\theta) + \alpha\omega\,cos(\theta) \tag{1.2}$$

$$\dot{\theta} = \omega \tag{1.3}$$

Donde:

- $\dot{x}, \dot{y}$: Representa la primera derivada de la posición del robot, en este caso es el punto P, donde P es el centro de gravedad del robot con respecto al sistema de referencia C
- u : Representa a la velocidad lineal del TurtleBot3
- α : Representa al desplazamiento del punto central de las ruedas al punto P
- $\dot{\theta}$: Representa la primera derivada del ángulo de rotación del robot con respecto al eje de las abscisas
- ω : Representa la velocidad angular del robot TurtleBot3 Burger

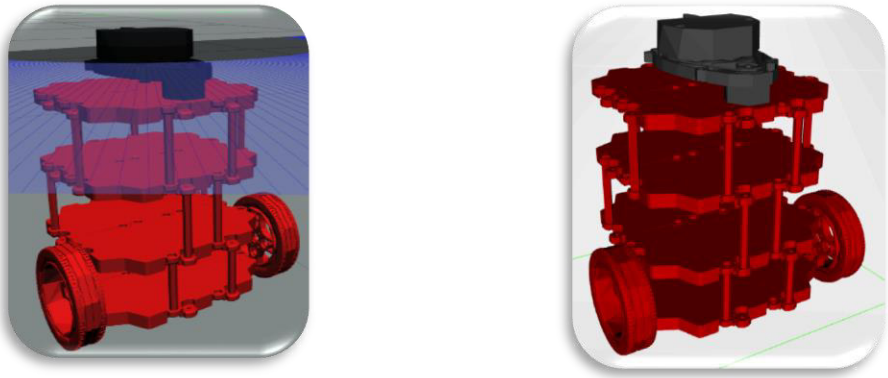
A las ecuaciones (1.1) a la (1.3) se las puede representar de forma matricial en la ecuación (1.4).

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos(\theta) & -\alpha\,sen(\theta)\\ sen(\theta) & \alpha\,cos(\theta)\\ 0 & 1\end{bmatrix}\begin{bmatrix}u\\ \omega\end{bmatrix} \tag{1.4}$$

1.4.2.3 Modelo virtual y uso en ROS-Gazebo-Rviz [7]

Para la simulación del robot TurtleBot3 Burger en ROS, primero se debe de instalar los paquetes que conforman a los robots TurtleBot3, en estos paquetes ya se configuran todas las características físicas de los robots como la cinemática y la dinámica. Una vez que se hayan instalado todas las librerías simplemente se hace un llamado mediante un terminal, donde se especifica el tipo de robot y en que entorno 3D se va a simular, este puede ser en Gazebo, Rviz o en la combinación de ambos.

Los paquetes del robot TurtleBot3 Burger en ROS se pueden configurar mediante código (urdf o xacro) para ajustar el color, la distancia de sensado y el ángulo del sensor LDS, etc. Una vez que se guarden y se ejecuten las configuraciones respectivas para Gazebo y Rviz se puede visualizar al robot virtualmente en cada entorno, tal como se muestra en la Figura 1.6.



(a) Modelo en Gazebo (b) Modelo en Rviz

Figura 1.6. Robot TurtleBot3 Burger en ROS [Fuente Propia]

Para que el robot virtual TurtleBot3 Burguer pueda interactuar dentro de ROS es necesario ejecutar al menos un nodo, como, por ejemplo: un algoritmo de navegación autónoma que se le llamará navegación_robot1. La aplicación de este algoritmo se basa en tomar lecturas del entorno mediante el sensor LDS (r1/scan) y dar acciones de control a la velocidad del robot (r1/cmd_vel), donde la finalidad es que no choque obstáculos. Para visualizar como se comunica el nodo (navegación_robot1) con los nodos del robot(/r1/scan, r1/cmd_vel) es necesario ejecutar la herramienta de grafos en ROS, véase la Figura 1.7. En esta figura, se indica que el nodo (navegación_robot1) se suscribe a la información del sensor LDS (/r1/scan), donde se obtiene los datos medidos y posteriormente se ejecuta el algoritmo de navegación autónoma. Finalmente, el nodo (navegación_robot1) publica los datos y el nodo de la velocidad (/r1/cmd_vel) se suscriba a esta información y envía al robot TurtleBot3 Burger llamado en la aplicación como r1.

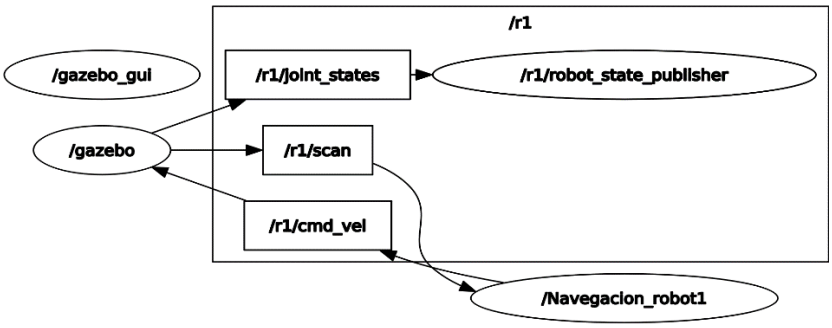
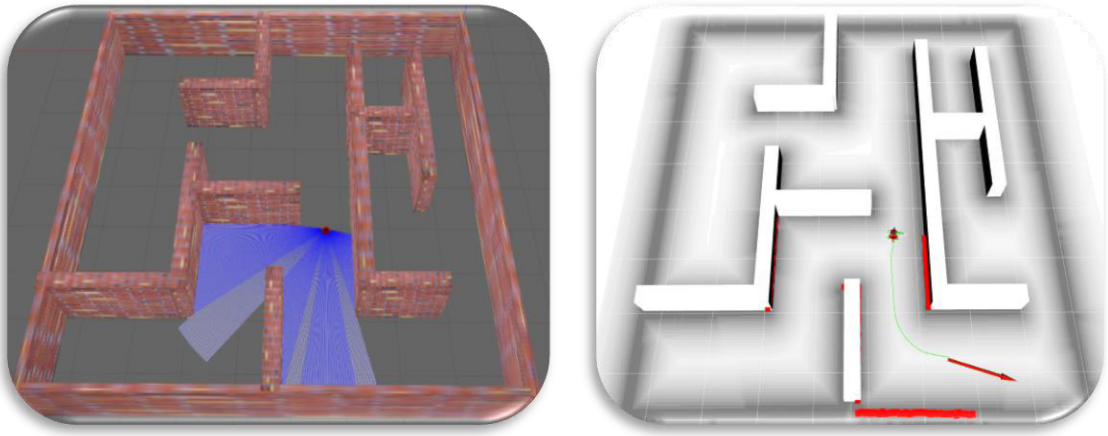


Figura 1.7. Diagrama de nodos y tópicos [Fuente Propia]

La herramienta de grafos es de gran ayuda ya que permite conocer que nodos están en ejecución y como es la comunicación entre ellos, para entender de manera visual esta aplicación se ejecutan las herramientas de Gazebo y Rviz, tal como se puede ver en la Figura 1.8. En Rviz se pueden realizar pruebas al robot y visualizar sus movimientos, sin

embargo, en Rviz no se pueden integrar los sensores del robot al entorno. En cambio, en Gazebo si se integran los sensores del robot, como: IMU, LDS, cámaras, etc. Si se juntan estas dos herramientas se puede realizar un SLAM, control remoto o navegación autónoma con una gran precisión sobre los robots.



(a) Plataforma de Gazebo (b) Plataforma de Rviz

Figura 1.8. Interacción en el entorno de ROS [Fuente Propia]

1.4.3 SISTEMAS MULTI-AGENTES (SMA) EN ROS

ROS es considerada una plataforma robótica de gran ayuda que permite conocer cómo interactúan varios robots antes de implementar un proceso o en este caso como interactuarán varios nodos para llevar a cabo una aplicación. A las aplicaciones con varios robots se las conoce como aplicaciones con Sistemas Multi-Agentes (SMA).

Un SMA es un conjunto de agentes dentro de un mismo entorno y en ROS se puede considerar como un Sistema Multi-Robot (SMR), donde los agentes pueden ser homogéneos o heterogéneos, tal como se muestra en la Figura 1.9. Para entender los conceptos sobre los SMA se parte desde la definición de un agente o agentes inteligentes. Un agente inteligente puede ser un robot, un programa o un sistema que sea capaz de realizar tareas simples autónomamente (sin intervención humana) [10]. Sin embargo, la única condición es, que debe existir un medio ambiente o entorno donde el agente realice sus gestiones. El agente dentro del entorno puede adquirir información para procesarla mediante software e interpretar el medio en el que se encuentra y ejecutar acciones.

Los SMA o SMR nacen tras el éxito que tienen al realizar tareas en conjunto, ya que estas tareas para un solo agente resultan imposibles o poco probable de realizarlas; por lo cual, múltiples agentes mejoran el rendimiento, eficacia, flexibilidad, tolerancia a fallos y las capacidades de un agente individual [11].

(a) SMA homogéneos (b) SMA Heterogéneos

Figura 1.9. Tipos de SMA [Fuente Propia]

Un SMR virtual en ROS trabaja de la siguiente manera, en Gazebo cada agente recolecta información del entorno de forma autónoma, esta información se fusiona con la de los demás agentes y se procesa mediante software para superponerla en un entorno aumentado como lo es Rviz. Mientras más información sea recopilada se tendrá una mejor percepción del entorno (rico en detalles) [3], [4], [12].

1.4.4 ALGORITMOS DE PLANEACIÓN DE RUTAS

Una de las aplicaciones que se puede desarrollar dentro de ROS con ayuda de Gazebo y Rviz son proyectos con algoritmos de planeación de rutas. Si a esta aplicación se la desarrolla con un SMA homogéneo se puede analizar cómo es la interacción de varios robots siguiendo diferentes trayectorias en un entorno donde existan obstáculos.

Los algoritmos de planeación de rutas se basan en buscar trayectorias libres de colisiones partiendo desde un punto inicial a un punto final en entornos con obstáculos tomando en cuenta criterios de evaluación. Estos criterios evalúan el tiempo que se demora o cual es el camino más corto; también hay que tomar en cuenta que, según la cantidad de obstáculos presentes en el entorno, más tiempo se llevará en ejecución el algoritmo [13].

Los algoritmos o planificadores más comunes están divididos en los grupos de consulta múltiple, de consulta única, y de grafos [14]. Un breve detalle de los algoritmos para cada una de estas divisiones se puede ver a continuación:

Algoritmos de consultas múltiples

- Método de mapas de ruta probabilística (PRM)
- Algoritmo SPArse Roadmap Spanner (SPARS)
- SPARS2

Algoritmos de consulta única

- Árboles aleatorios de exploración rápida (RRT)
- Árboles espaciales expansivos (EST)
- Árboles de subdivisión dirigidos por ruta (PDST)
- Árbol de marcha rápida (FMT∗)
- Árbol bidireccional de marcha rápida (BFMT∗)
- Cociente-Espacio RRT (QRRT)

Algoritmos de grafos

- Dijkstra
- A* (A-Star)

Los algoritmos de planificación de rutas que se han escogido en el presente proyecto de integración para su desarrollo son 2, uno de múltiples consultas (PRM) y otro de consultas únicas (RRT), ya que, generalmente, los demás algoritmos se derivan de estos dos planeadores. Los algoritmos de grafos no se los considera, ya que se deben cumplir ciertas condiciones como que las rutas a encontrar deben ser cortas, entre otras limitaciones.

La diferencia de los algoritmos de consulta múltiple, como el PRM, es que se basa en muestreo, mismo que contiene dos fases. La primera toma puntos aleatorios del entorno libres de obstáculos y construye los mapas o caminos; y la segunda es que realiza múltiples consultas a cada punto para buscar la ruta adecuada. En cambio, para los algoritmos de consulta única como el RRT toma puntos aleatorios del entorno para realizar una sola consulta y construir los caminos con el fin de hacer crecer el árbol hasta llegar al punto final, donde es considerada que es la ruta adecuada para la navegación [14], [15].

El algoritmo PRM tiene una eficiencia alta cuando se tienen pocos obstáculos en el entorno, a comparación del algoritmo RRT, sin embargo, cuando se tienen bastantes obstáculos en el entorno y pocos puntos de muestreo el algoritmo falla, por lo cual se dice que el algoritmo está incompleto en probabilidad. En cambio, el algoritmo RRT tiene una probabilidad completa, ya que llega a la meta tras tener un paso de búsqueda fija [13], [15].

1.4.4.1 Algoritmo de árboles aleatorios de expansión rápida (RRT)

El algoritmo RRT es conocido como un planeador de consulta única que se basa en los principios heurísticos, es decir que, bajo este principio se busca reducir las consultas a los múltiples nodos de la ruta que queda por recorrer. Esto se lleva a cabo gracias a la

predicción heurística, ya que calcula la distancia entre nodos desde el nodo inicial hasta el nodo meta, de este modo se prescinde una ruta libre de colisiones [16].

Los pasos para ejecutar el algoritmo RRT de la Figura 1.10 se explica a continuación [17]:

1. Se coloca el punto de inicio como nodo de salida (x_{init}) , de ahí se determina un nodo cercano (x_{near}) al de inicio, posteriormente el algoritmo determinará varios nodos random (x_{rand}) y escogerá cual es el más cercano a x_{near}.
2. Con el nodo x_{near} y el x_{rand} identificados, se determina un nuevo nodo (x_{new}). El nodo x_{new} esta alejado de x_{near} una distancia multiplicada por el factor sigma (σ) en la dirección de x_{rand}.
3. El punto x_{new} pasa a ser el nuevo x_{near} en la siguiente iteración del algoritmo, así se seleccionará un punto random cercano a este nodo y se dirigirá cada vez al punto de llegada.
4. Para la selección del x_{new} no deben existir colisiones entre el x_{near} y el x_{new}, si se da el caso de existir un obstáculo en medio, se selecciona otro nodo random cercano hasta que no se tenga obstáculos.
5. Finalmente, si se encuentra un nodo x_{rand} cerca del nodo de llegada el x_{new} será el nodo de llegada, para este caso se finaliza el algoritmo.

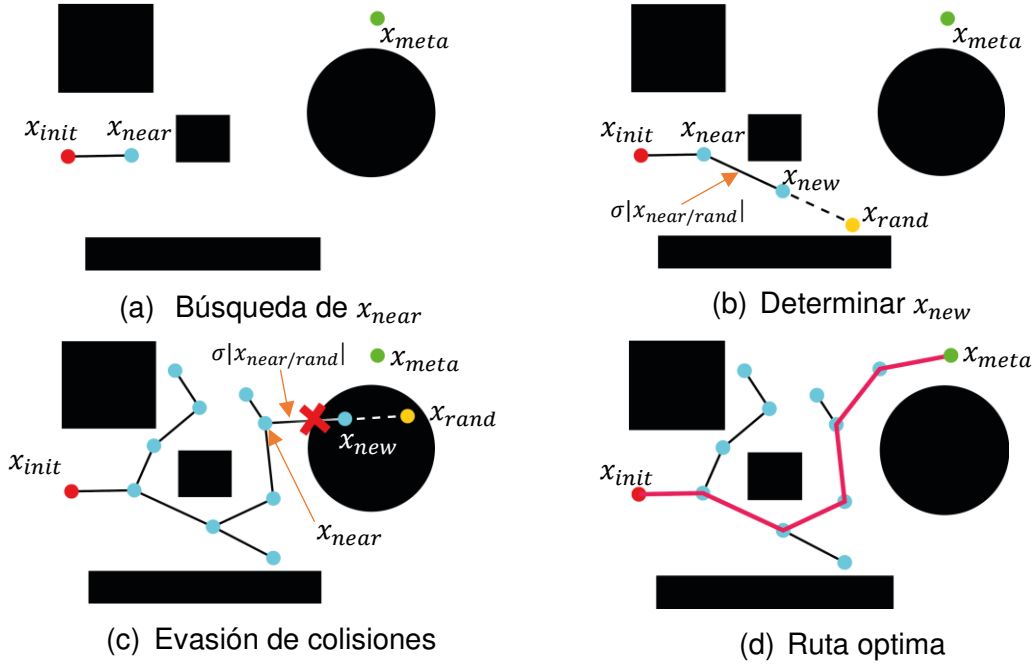
Algorithm 1: RRT Algorithm

Input: $V, \sigma, n, x_{init}, x_{meta}$
Result: $Path\ from\ x_{init}\ to\ x_{meta}$
$V \leftarrow x_{init}$, $E \leftarrow 0$
T$\leftarrow (V, E)$
For i **in range** n **:**
    $x_{rand} \leftarrow Sample(V)$
    $x_{near} \leftarrow Near(x_{rand}, T)$
    $x_{new}, u_{new}, T_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
    $E_i \leftarrow Edge(x_{new}, x_{near})$
    If $CollisionsFree(V, E_i)$ **:**
        $T \leftarrow insertNode(x_{new})$
        $T \leftarrow reWireEdge(T, x_{near}, x_{near})$
    If $x_{new} == x_{meta}$:
        return T

Figura 1.10. Algoritmo RRT adaptado de [18] , [19]

Este algoritmo generar caminos en forma de árboles o ramas. Estas ramas se extienden a lo largo y ancho del entorno y se interconectan con los nodos cercanos, tal como se ve en la Figura 1.11. Los parámetros dentro de este algoritmo son las distancias de separación

entre nodos producto del factor sigma (σ), la distancia de separación de los nodos con los obstáculos y el tiempo que les toma encontrar las rutas libres de obstáculos [20].



(a) Búsqueda de x_{near} (b) Determinar x_{new}

(c) Evasión de colisiones (d) Ruta optima

Figura 1.11. Interacción del algoritmo RRT [Fuente Propia]

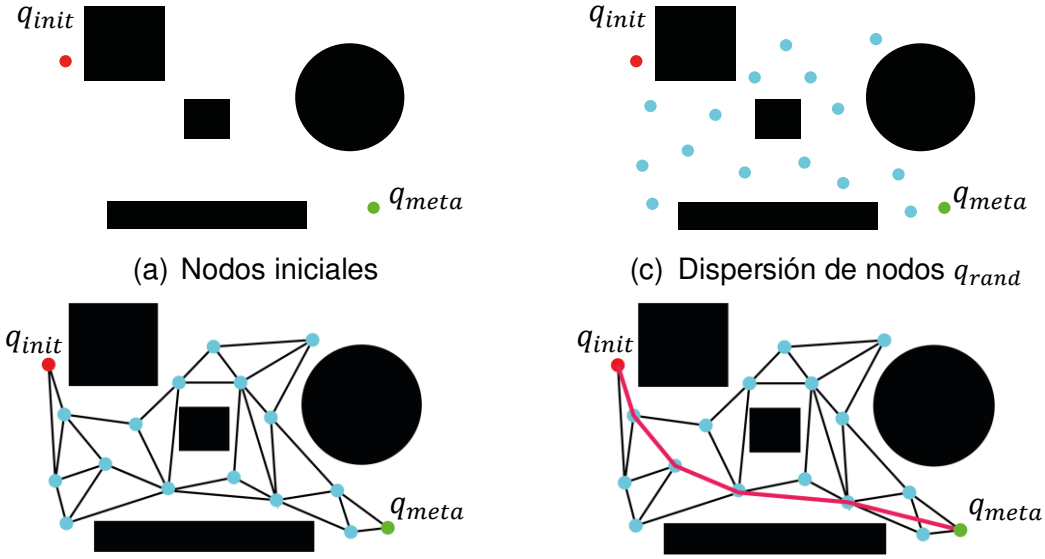
1.4.4.2 Algoritmo de Mapas de rutas probabilísticas (PRM)

El algoritmo PRM es un planeador de consultas múltiples, que la forma de implementarse es mediante dos subprocesos, donde uno construye los mapas mientras otro va consultando la ruta adecuada. Este planeador toma muestras del entorno y determina si estas muestras están en un espacio libre o se interseca con un obstáculo. Una vez que se crean suficientes muestras hasta llegar a la meta los nodos se intersecan entre sí, siempre y cuando no exista un obstáculo de por medio [20], tal como se muestra en la Figura 1.12.

Los pasos para ejecutar el algoritmo PRM, se resumen a continuación y se los presenta en la Figura 1.13:

1. Se inicializa la red de nodos , donde V es el conjunto de nodos random y E son los nodos con la ruta adecuada, si la encuentra.
2. A continuación, se dispersa una serie de nodos a lo largo del mapa y se determina si el nodo q esta en un espacio libre de obstáculos, añadiéndose así a la red de nodos V caso contrario no se añade ese nodo.
3. Los pasos cuatro al nueva se repiten hasta que se completen las iteraciones del loop o hasta que el nodo random llegue lo más cercano de la meta.

4. Dentro del algoritmo, se inicia un subproceso de consulta para la construcción del mapa de rutas, donde se toma un q' que conforma la red de nodos libres V para seleccionar y conectar con los nodos vecinos acorde a sus distancias $D(q, q')$.
5. Si las conexiones entre q y q' son exitosas y no se intersecan con ningún obstáculo, entonces corre el subproceso de consulta para la ruta de navegación añadiendo a un nuevo nodo al vector E con (q,q').



(a) Nodos iniciales (c) Dispersión de nodos q_{rand}

(d) Grafica de las rutas (e) Ruta de navegación

Figura 1.12. Interacción del algoritmo PRM [Fuente Propia]

Algorithm 1 PRM

```
1: V ← ∅
2: E ← ∅
3: loop
4:    q ← a randomly chosen free configuration
5:    V_q ← a set of candidate neighbors of q chosen from V
6:    V ← V {q}
7:    for q' ∈ V_q, in order of increasing D (q, q'), do
8:    if E = same_connected_component(q, q') < |(q, q')| then
9:          E ← E ∪ {(q, q')}
10:         update E's connected components
```

Figura 1.13. Algoritmo PRM [21]

Dentro de ROS se simulará y evaluará el comportamiento de los algoritmos en el entorno de Gazebo y Rviz, donde se visualizarán las rutas del PRM Y RRT, tomando en consideración todas las limitaciones y ventajas de cada algoritmo.

2 METODOLOGÍA

En el presente capítulo se detalla el desarrollo del diseño del entorno virtual en Gazebo y el mapeo de este con la herramienta de visualización 3D Rviz. Además, se explica cómo está conformada la clasificación de los archivos en ROS para el robot móvil TurtleBot3 Burguer y cómo llamar a estos ficheros para así visualizar a los modelos desde cada plataforma (Gazebo y Rviz). La aplicación del presente proyecto se lleva a cabo dentro del sistema operativo Linux Ubuntu (versión Focal Fossa 20.04 LTS), donde se instala y ejecuta el software libre ROS 1 en su última versión (Noetic Ninjemys) [5].

2.1 DISEÑO DEL ENTORNO VIRTUAL EN GAZEBO

Para la creación de entornos virtuales en Gazebo, primero se deben de seguir los tutorials de ROS disponibles para la versión de Noetic Nimjemys; ya que la mayoría de los proyectos inician con la creación de un Catkin Workspace y Packages [5]. A continuación, se explican estas dos funciones y su constitución se visualiza en la Figura 2.1.

- **Catkin Workspace**: Catkin es una plantilla dentro de ROS que ayuda a desarrolladores a estandarizar sus proyectos robóticos bajo el sistema Catkin (build, devel, src) y Workspace es una carpeta creada en un directorio dentro de nuestro ordenador, donde se colocarán los paquetes creados [5].

- **Packages**: El software dentro de ROS está organizado por paquetes, estos paquetes pueden contener múltiples archivos, como: nodos de comunicación, bibliotecas independientes de ROS, configuraciones, fragmentos de código o cualquier fichero que constituya una parte lógica dentro del proyecto. Estos paquetes tienen suficiente funcionalidad como para ser reutilizados siempre y cuando no sean tan pesados o difíciles de cargar en otras aplicaciones [22].

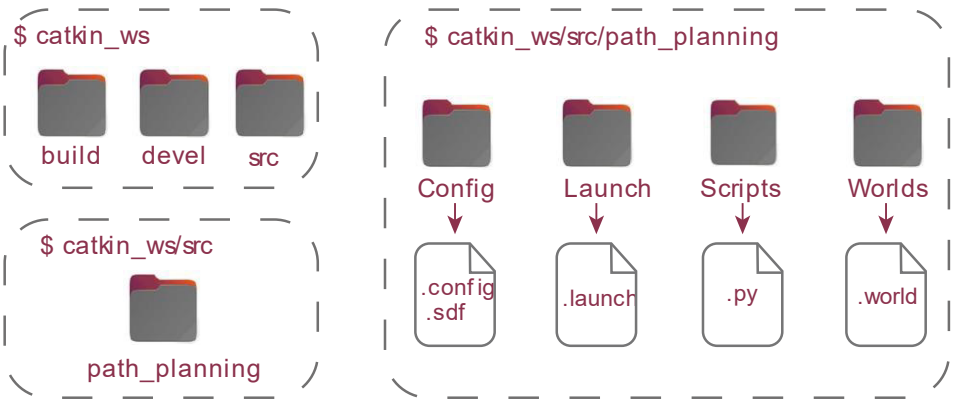


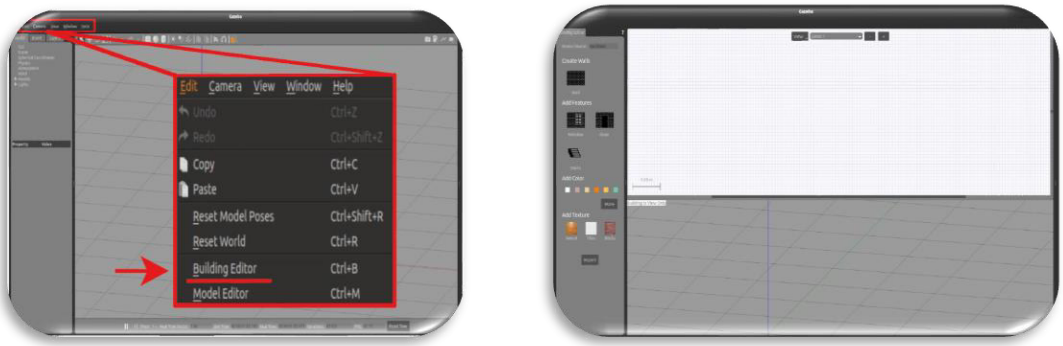
Figura 2.1. Espacio de trabajo para ROS-Gazebo [Fuente Propia]

En el espacio de trabajo (catkin_ws) se crea un paquete con el nombre del proyecto de titulación (path_planning) y se añaden subcarpetas que ayudarán a la organización y la creación de un entorno virtual en Gazebo, tal como se puede ver en la Figura 2.1. La función de estas subcarpetas se explica a continuación, ya que contienen archivos específicos dentro del proyecto actual.

- Config: En esta carpeta se encuentran las configuraciones del entorno en formatos .sdf y .config que son extensiones del formato de marcado XML. Los ficheros .sdf contienen características físicas del entorno, en cambio, el fichero .config detalla la información del desarrollador del proyecto y el directorio del archivo .sdf.
- Launch: En esta carpeta se guardan los ficheros ejecutables .launch que también es una extensión del formato XML, su función es correr múltiples nodos desde un terminal de Ubuntu y configurar los parámetros dentro del servidor de ROS.
- Scripts: En esta carpeta se añaden los archivos .py y estos pueden ser nodos de comunicación o algoritmos de control.
- Worlds: En esta carpeta se encuentran los ficheros .world que describen las propiedades físicas de los elementos del entorno, tales como: la viscosidad, las superficies de contacto, la gravedad, la fricción, etc. Este fichero también es una extensión del formato XML y son creados cuando se modela un entorno virtual desde Gazebo con la herramienta de Building Editor.

2.1.1 MODELADO VIRTUAL DEL ENTORNO

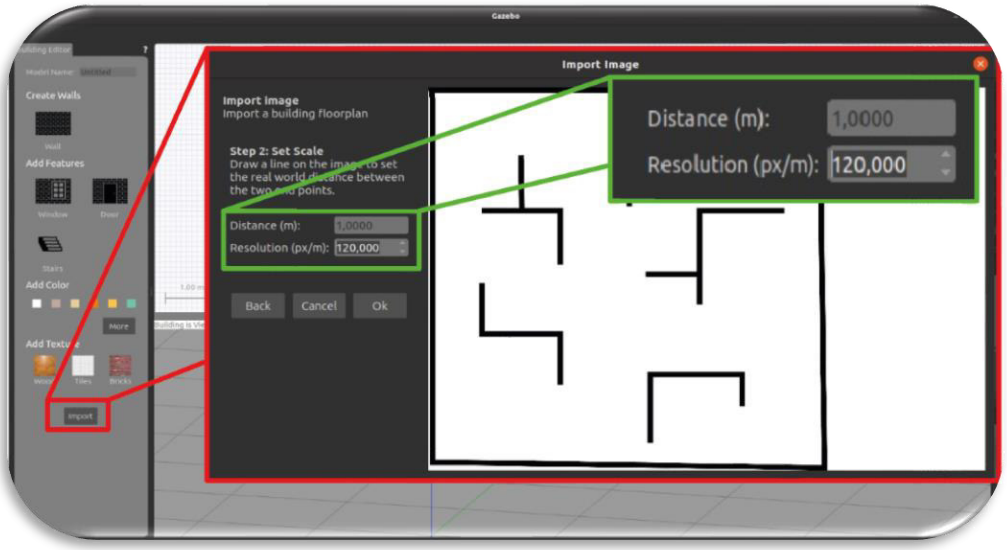
Una vez que se haya creado el espacio de trabajo (catkin_ws/path_planning) con las subcarpetas (Config, Launch, Scripts y Worlds) se inicia el diseño del entorno virtual. Para ello se corre la plataforma de Gazebo desde un terminal de Ubuntu y posteriormente se abre la pantalla de Building Editor en Gazebo, tal como se muestra en la Figura 2.2.



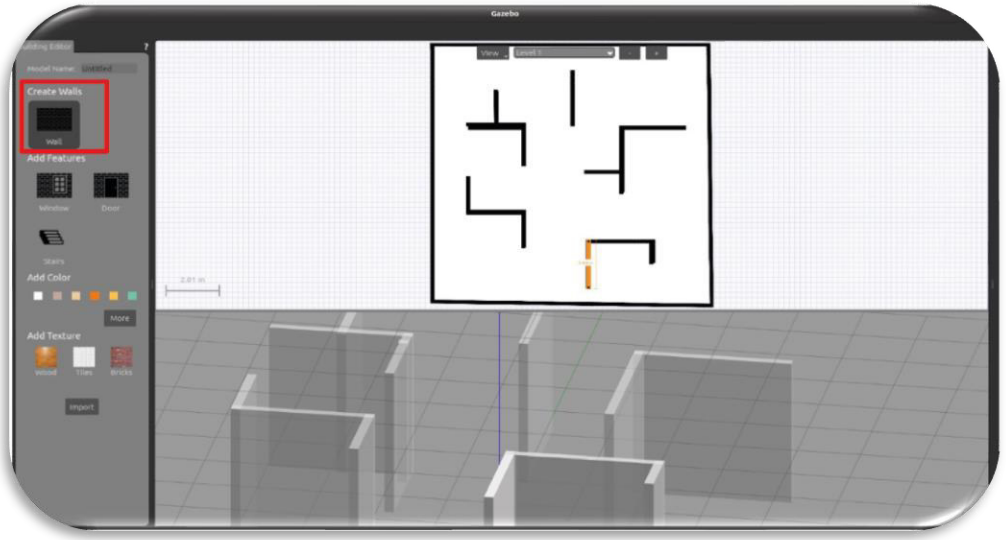
(a) Menú Edit (b) Plataforma de Building Editor

Figura 2.2. Editor de Entornos Virtuales [Fuente Propia]

En la pantalla de Building Editor se importa el mapa prediseñado del presente proyecto, este mapa se ha creado en Adobe Illustrator 24.0.2 con las siguientes especificaciones: dimensión 200x200mm, trazo de las paredes 8pt, distancia de separación entre paredes 35mm. Las dimensiones y el grosor del trazo se han establecido mediante la experimentación práctica, ya que dentro de la plataforma se debe de ajustar la resolución del mapa en píxel/metro y distancia en metros, tal como se puede ver en la Figura 2.3 (a).



(a) Importar mapa y ajustes de resolución de la imagen



(b) Creación de paredes en Building Editor

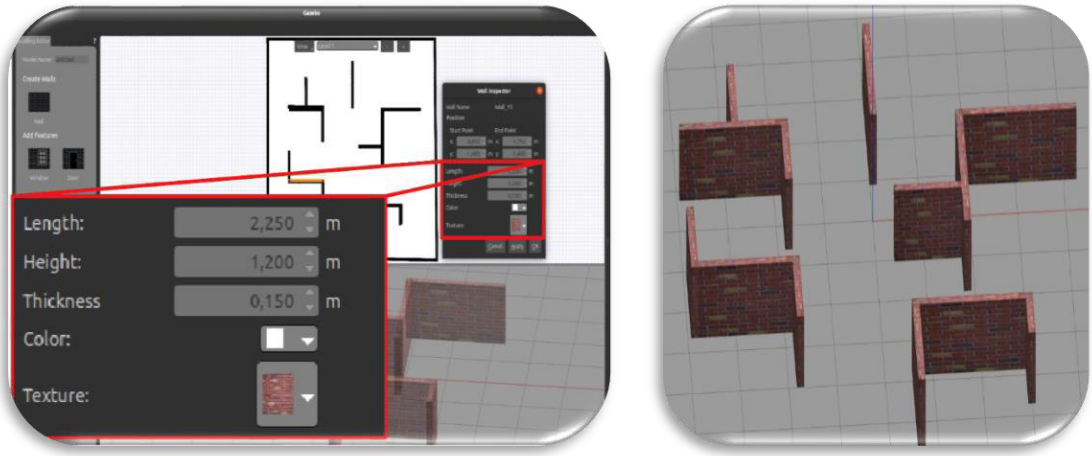
Figura 2.3. Creación de entornos virtuales en Gazebo [Fuente Propia]

En la Figura 2.3 (b) se muestra que la pantalla de Building Editor está conformada por un panel de edición y dos subpantallas (superior/inferior). En la superior se crearán las

paredes del entorno con la herramienta de create walls del panel de edición, ya que se puede visualizar el mapa importado; las paredes se crean cuando se hacen trazos sobre el mapa y este se torna amarillo. De modo que, en la subpantalla inferior estas paredes se visualizan en 3D con un ancho, una longitud y una textura por defecto.

Una vez que se terminen de crear las paredes se pueden configurar a cada una de estas con Wall Inspector, tal como se muestra en la Figura 2.4 (a). Mediante los ajustes de la longitud, la textura, el grosor y otros parámetros se mejora la presentación en Gazebo, tal como se ve en la Figura 2.4 (b). Dentro del proyecto actual a las paredes se les ha dado la textura de ladrillo y una altura de 1,20 m con el fin de visualizar a los robots que interactúen en el entorno creado.

Finalmente, para terminar con el diseño se da en salir de la pantalla de Building Editor y se guarda el fichero .world en la subcarpeta Words sin olvidar que los ficheros de configuraciones .config y .sdf se deben mover a la carpeta Config, estos archivos se encuentran en catkin_ws/default_world.



(a) Menú de Wall Inspector (b) Visualización en Gazebo

Figura 2.4. Ajuste de texturas en Building Editor [Fuente Propia]

Para correr el entorno del proyecto en Gazebo se crea un fichero .launch, donde se configuran los argumentos y parámetros del entorno, como: la ubicación del fichero .world ($(find path_planning)/worlds/mundo1.world), usar el tiempo de simulación (True), pausar al iniciar la simulación (False), etc; tal como se muestra en la Figura 2.5.

Una vez configurado el fichero .launch ya se lo puede ejecutar desde un terminar de Ubuntu con el comando roslaunch. En el ANEXO I se explica cómo está constituido este fichero y como se lo ejecuta desde un terminal de Linux Ubuntu.

```
<!-- Entorno -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" value="$(find path_planning)/worlds/mundo1.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
```

Figura 2.5. Código para correr el entorno creado en Gazebo [Fuente Propia]

2.1.2 INTEGRACIÓN DEL ROBOT MÓVIL TURTLEBOT3 BURGER EN GAZEBO

La integración del robot móvil (TurtleBot3 Burguer) al entorno se lo hace mediante la configuración del fichero .launch, sin embargo, para esta configuración se deben de contar con los ficheros .urdf o .xacro que son archivos que contienen todas las especificaciones físicas del robot, como: la descripción dinámica, cinemática, representación visual y el modelo de colisiones. Por viabilidad, en este proyecto se usan los paquetes que ofrece ROS para este tipo robot móvil con todas las configuraciones por defecto, tomando en consideración que, los ficheros .urdf son muy extensos, por esta razón se utilizan a los ficheros .xacro que son archivos simplificados de un URDF macro. Las ventajas de usar los ficheros .xacro es la simplicidad en la que se personaliza el robot, ya que solo se editan las partes deseadas y en el fichero .urdf se seguirá teniendo toda la descripción general del robot móvil [23]. En la Figura 2.6 se muestra el fragmento de código que ayuda a posicionar y orientar a un robot dentro del entorno del proyecto.

```
<!—Posicion Robot 1-->
  <arg name="ns1"  default="r1"/>
  <arg name="r1_x_pos" default="0.0"/>
  <arg name="r1_y_pos" default="0.0"/>
  <arg name="r1_z_pos" default=" 0.0"/>
  <arg name="r1_yaw"   default=" 0"/>

<!-- Grupo Robot 1 -->
  <group ns = "$(arg ns1)">
    <param name="robot_description" command="$(find xacro)/xacro  $(find pat
h_planning)/robots/urdf/turtlebot3_burger1.urdf.xacro" />

<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -mod
el $(arg ns1) -x $(arg r1_x_pos) -y $(arg r1_y_pos) -z $(arg r1_z_pos) -Y $(
arg r1_yaw)  -param robot_description" respawn="false" output="screen" />
  </group>
```

Figura 2.6. Configuración de la posición del robot [Fuente Propia]

En la Figura 2.6, el fichero .launch integra al robot móvil mediante la llamada de solicitudes de servicios al paquete de gazebo_ros, este método hace el llamado a un script de python

con el nombre de spawn_model que contiene los argumentos configurados en el fichero .launch mediante el parámetro robot_description y así agrega el fichero personalizado .xacro del robot en Gazebo con la posición y orientación (véase la **Figura 2.7**) [24].

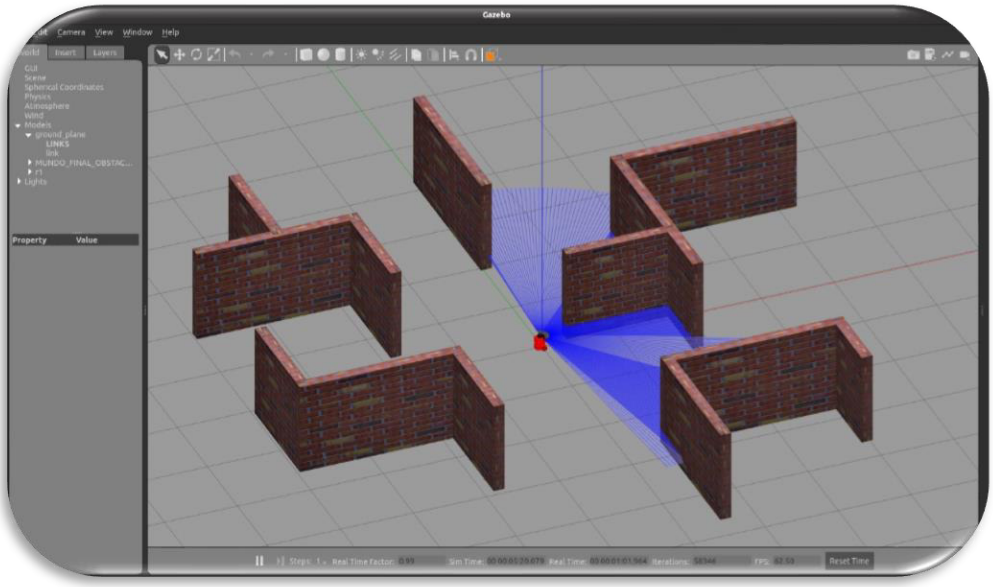


Figura 2.7. TurtleBot3 Burguer en Gazebo [Fuente Propia]

2.2 MAPEO DEL ENTORNO VIRTUAL EN RVIZ

Cuando se tenga diseñado el ambiente en Gazebo y se integre al menos un robot móvil TutleBot3 Burguer al entorno, ya se puede hacer realizar el mapeo. Para esto se deben de configurar los argumentos y parámetros correspondientes en el fichero .launch para iniciar tanto Gazebo como Rviz de la forma que se muestra en la Figura 2.8.

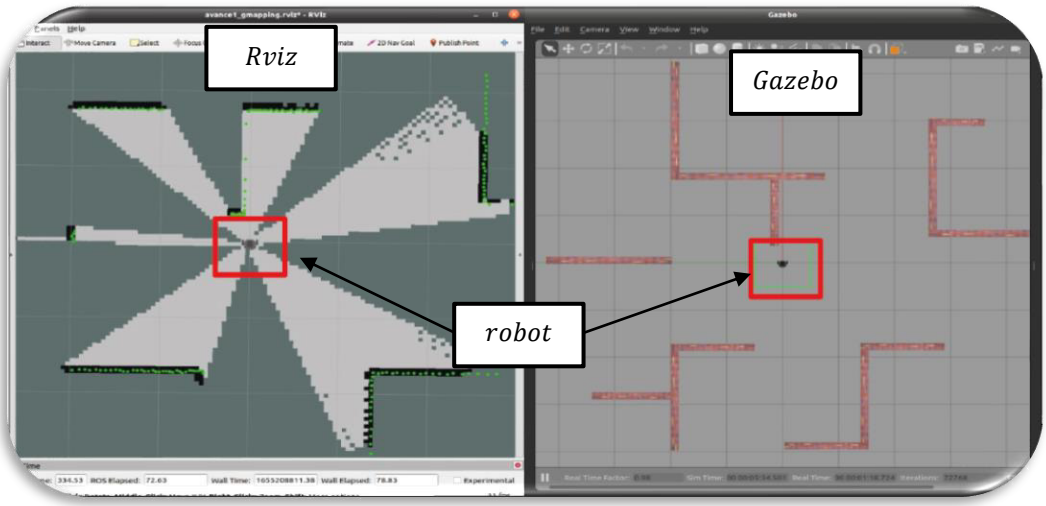


Figura 2.8. Entorno de Rviz y Gazebo antes de iniciar con el mapeo. [Fuente Propia]

Adicionalmente, es necesario añadir al paquete del proyecto (path_planning) unas subcarpetas adicionales que ayudaran con la organización del producto final demostrable, tal como se muestra en la Figura 2.9.

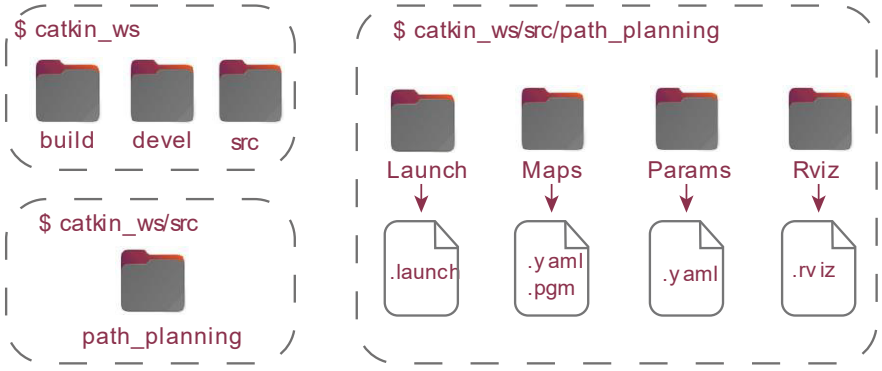


Figura 2.9. Espacio de trabajo ROS-Rviz [Fuente Propia]

Estas carpetas contendrán archivos específicos que se explican a continuación.

- **Maps**: En esta carpeta se guardan a los archivos generados después del mapeo, el archivo .pgm, que es un formato de imagen que contiene información sobre el entorno mapeado y el fichero .yaml para ser visualizado en Rviz.
- **Params**: En esta carpeta se guardan los ficheros .yaml, que son formatos de intercambio de datos, serialización y configuraciones de los parámetros de los robots o el entorno. La ventaja de este fichero es que no utiliza un lenguaje de marcado, por lo cual es ligero a comparación de los archivos XML [25].
- **Rviz**: En esta carpeta se guardan todas las configuraciones que se hagan en el entorno de Rviz en el formato .rviz.

Una vez que se han creado las subcarpetas adicionales dentro del proyecto, a continuación, se explica bajo que método se realiza el reconocimiento y reconstrucción del mapa en Rviz.

2.2.1 LOCALIZACIÓN SIMULTANEA Y MAPEO (SLAM)

El SLAM es un sistema en ROS que su principal objetivo es el reconocimiento de mapas, terrenos o espacios desconocidos. Su algoritmo se basa en sistemas con espacios de estados, donde se extraen puntos de referencias al entorno que está en Gazebo (como: paredes, esquinas, obstáculos, etc.) para posteriormente estimar el estado de este entorno en Rviz [26]. La visualización del entorno en Rviz se da cuando se actualizan los puntos de referencias que son tomados por el sensor de distancia láser (LDS) con respecto a la posición del robot móvil TurtleBot3 Burguer en Gazebo.

2.2.2 MAPEO CON GMAPPING

Gmapping es un paquete dentro de ROS que corre un nodo basado en el SLAM (slam_gmapping), donde el mapeo se lo hace bajo el mismo principio; que es publicar la posición del robot, las lecturas tomadas por el sensor LDS y mostrar los estados de estos datos en Rviz. Para el mapeo del entorno diseñado se configura un fichero .launch con los argumentos necesarios, como: la odometría (odom), la coordenada del robot sobre el piso (base_footprint), el modelo del robot (burger), entre otros, tal como se ve en la Figura 2.10.

```
  <!—Configuracion inicial de parametros para el mapeo-->
  <arg name="model" default="burger"/>
  <arg name="open_rviz" default="true"/>
  <arg name="move_forward_only" default="false"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="set_base_frame" default="base_footprint"/>
  <arg name="set_odom_frame" default="odom"/>
  <arg name="set_map_frame"  default="map"/>
```

Figura 2.10. Configuración de argumentos para el mapeo. [Fuente Propia]

A continuación, se corre el fichero de description.launch del paquete de turtlebot3_bringup, tal como se muestra en la Figura 2.11. Este fichero hace un llamado a varias instrucciones y configuraciones por defecto en ROS, cuya finalidad es controlar al robot móvil TurtleBot3 Burguer. Técnicamente a este llamado se lo denomina bringup y es considerado como el primer paso antes de ejecutar cualquier programa; ya que inicia con el rosserial para el intercambio de información y la toma de lecturas con el sensor LDS, que son importantes para el mapeo del entorno [5], [7].

```
  <!—Llamado al nodo Bringup para el robot movil TurtleBot3 Burger -->
  <include file="$(find turtlebot3_bringup)/launch/includes/description.laun
ch.xml">
    <arg name="model" value="$(arg model)" />
  </include>
```

Figura 2.11. Correr el nodo Bringup para los robots TurtleBot3. [Fuente Propia]

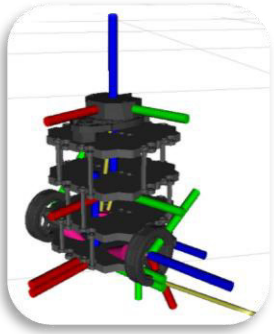
Para poder publicar el estado del robot y visualizarlo en Rviz es necesario correr los nodos de robot_state_publisher y joint_state_publisher, tal como se muestra en la Figura 2.12 (a); el funcionamiento de estos nodos se explica a profundidad en la Sección 2.2.4. Sin embargo, se dará una breve explicación de su funcionamiento, el nodo de joint_state_publisher toma como entrada los ángulos de las articulaciones (joint angles) del robot móvil en Gazebo y a la salida el nodo de robot_state_publisher publica las articulaciones en 3D mediante los enlaces (links) del robot móvil a Rviz; para ello se utiliza el modelo de árbol cinemático del robot [5], tal como se muestra en la Figura 2.12 (b). Para más información se sugiere revisar el ANEXO II.

En la Figura 2.12 (b), se visualizan varios sistemas de coordenadas, esto se debe a que las piezas en 3D que conforman al robot móvil están separadas y que gracias a los ficheros de configuración permiten unir cada articulación con sus respectivas posiciones y orientaciones, por lo cual a cada pieza independiente del robot le corresponde los ejes (x,y,z) que le ayudan a posicionarse dentro de Rviz .

```
<node pkg="robot_state_publisher"
type="robot_state_publisher" name=
"robot_state_publisher">

    <node name="joint_state_publis
her" pkg="joint_state_publisher" t
ype="joint_state_publisher">
    <param name="use_gui" value="F
alse"/>
    </node>
<param name="publish_frequency" ty
pe="double" value="50.0" />
</node>
```

(a) Código de configuración



(b) Visualización del robot

Figura 2.12. Visualización del estado del robot en Rviz. [Fuente Propia]

Posteriormente, se configuran los parámetros del robot con los argumentos que se presentaron en la Figura 2.10 para el robot móvil TurtleBot3 Burger y se llama al paquete gmapping para correr el nodo de turtlebot3_slam_gmapping, tal como se muestra en la Figura 2.13.

```
  <!—Configuracion de los parametros y el paquete Gmapping -->
  <node pkg="gmapping" type="slam_gmapping" name="turtlebot3_slam_gmapping"
output="screen">
    <param name="base_frame" value="$(arg set_base_frame)"/>
    <param name="odom_frame" value="$(arg set_odom_frame)"/>
    <param name="map_frame"  value="$(arg set_map_frame)"/>
    <rosparam command="load" file="$(find turtlebot3_slam)/config/gmapping_p
arams.yaml" />
  </node>
```

Figura 2.13. Código para correr iniciar el SLAM. [Fuente Propia]

Finalmente, dentro del fichero .launch se cargan las líneas de código para correr Rviz e iniciar con el mapeo, tal como se muestra en la Figura 2.14.

```
  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find path_planning)/rviz/gmapping.rviz"/>
  </group>
```

Figura 2.14. Código para correr Rviz [Fuente Propia]

Una vez configurado el archivo .launch se lo corre desde un terminal y se procede a realizar el mapeo del entorno mediante la navegación autónoma del robot móvil. Este método recrea el mapa que se ha diseñado en la Sección 2.1.1 para ser visualizado en Rviz. La ventaja de realizar un mapeo es que se pueden visualizar las paredes del entorno de Gazebo en Rviz, tal como se muestra en la Figura 2.15.

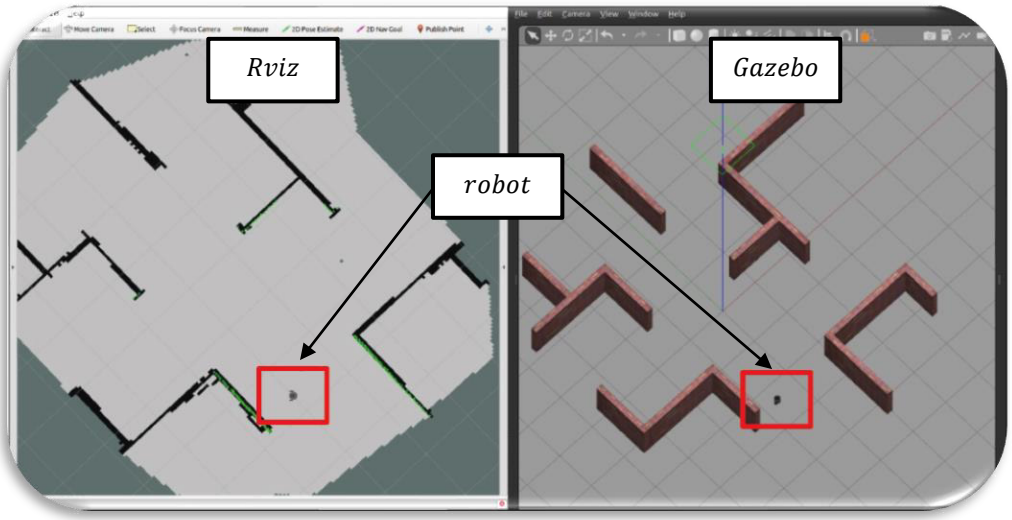


Figura 2.15. Mapeo del entorno. [Fuente Propia]

Cuando el mapeo finalice ya se podrá visualizar el entorno en Rviz, tal como se ve en la Figura 2.15. Posteriormente para guardar el mapa se debe de correr el comando que se muestra en la Figura 2.16 desde el terminal. Este comando ejecuta el nodo map_server cuya función es mostrar los datos del mapa como servicios dentro ROS, si se hace un llamado al servicio de map_saver se puede guardar el mapa generado dinámicamente en la carpeta Maps, ya que se crean los archivos .yaml y .pgm [5].

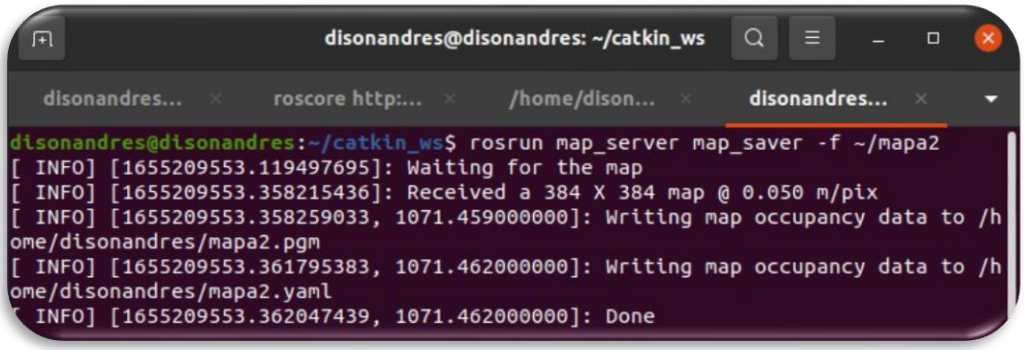
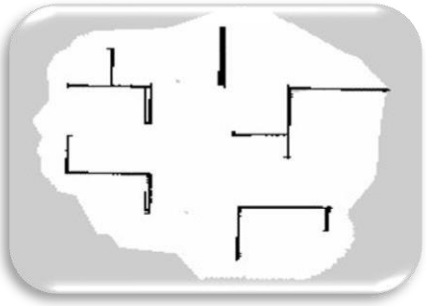


Figura 2.16. Guardar el mapa del entorno. [Fuente Propia]

Con los archivos guardados dentro de la carpeta de Maps se debe configurar el directorio del archivo .pgm en el fichero .yaml, tal como se muestra en la Figura 2.17 (a).

```
image: /home/disonandres/catkin_ws
/src/path_planning/maps/mapa2.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0
.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

(a) fichero .yaml



(b) Archivo .pgm

Figura 2.17. Archivos de configuración del mapa. [Fuente Propia]

Los campos del fichero .yaml se detalla a continuación [5].

- **Image:** En este campo se define la localización del archivo .pgm del mapa.
- **Resolution:** En este campo se ajusta la resolución del mapa en metros/pixel.
- **Origin:** se configura la posición (x,y,yaw) del mapa y se considera que el angulo yaw es cero.
- **Occupied_thresh**: Los pixeles que sobrepasa el thresh(umbral) se los considera como pixeles llenos o completos.
- **Free_thresh**: Los pixeles que no sobrepasen el thresh se los considera como pixeles libres o vacios
- **Negative**: En este campo se ajusta el color blanco/negro de la imagen o pixeles llenos o vacíos.

Para visualizar el mapa desde Rviz se lo hace mediante un fichero .launch, en la Figura 2.18 se muestra el fragmento de código de este fichero que ayuda a correr el nodo de map_server. Este nodo lee el fichero .yalm que está en la carpeta de Maps y lo ofrece dentro de ROS como un servicio, que es leído por Rviz.

```
<!-- Map server -->
    <arg name="map_file" default="/home/disonandres/catkin_ws/src/path_plann
ing/maps/mapa2.yaml"/>
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg map
_file)">
    <param name="frame_id" value="map"/>
  </node>
```

Figura 2.18. Código para cargar el mapa en Rviz. [Fuente Propia]

Cuando se termine de realizar las configuraciones de la Figura 2.18 al fichero .launch se lo ejecuta desde un terminal de Ubuntu. El mapa guardado puede visualizar dentro de la plataforma de Rviz, tal como se muestra en la Figura 2.19. Para conocer sobre las presentaciones del mapa en Rviz, por favor revise el ANEXO III.

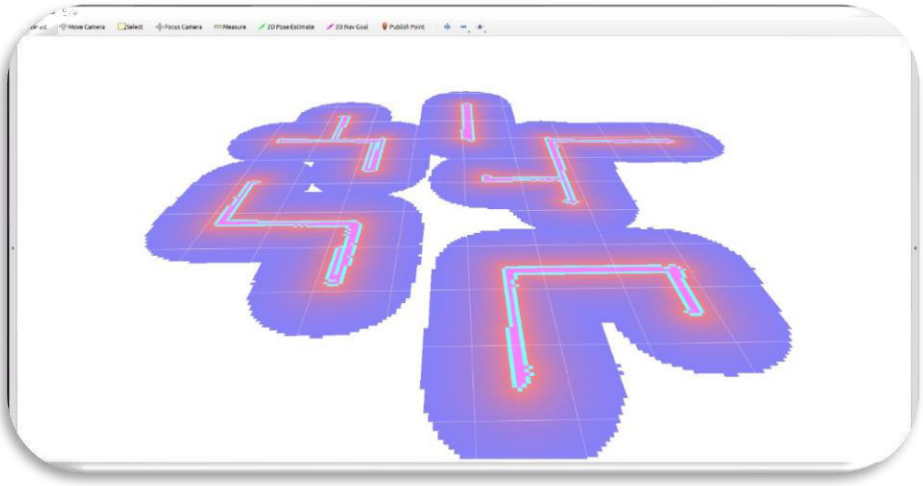


Figura 2.19. Entorno mapeado en RVIZ. [Fuente Propia]

2.2.3 CREACIÓN DE MARCADORES EN RVIZ

2.2.3.1 Tipo cubos

A Rviz se le conoce como una poderosa herramienta de visualización 3D, ya que permite integrar marcadores dentro del mapa de la Figura 2.19. Los marcadores son controles de arrastrar y soltar que muestran obstáculos o una vista previa de los movimientos del robot antes de su ejecución [27], tal como se visualiza en la Figura 2.20.

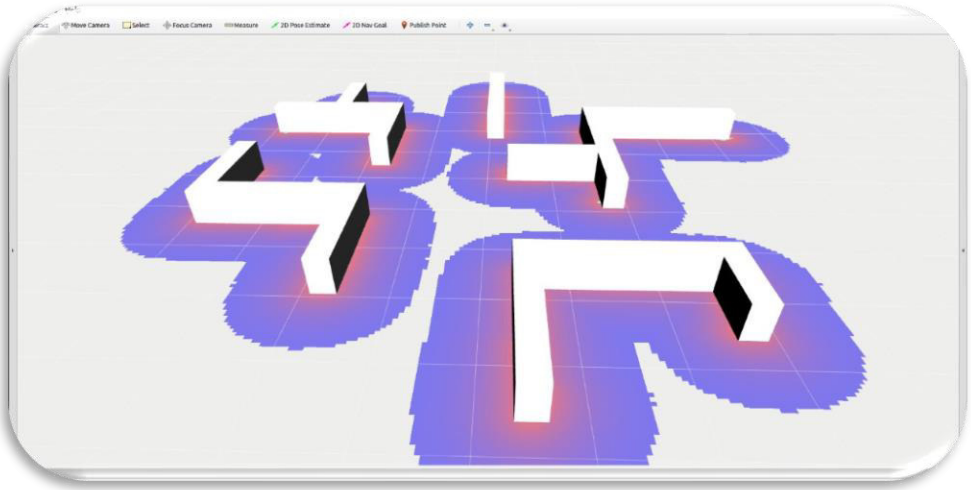


Figura 2.20. Mapa en Rviz con Marcadores. [Fuente Propia]

Estos marcadores pueden ser creados desde los lenguajes de programación C++ o Python, por lo que, para el presente proyecto se ha decidido hacerlo desde Python. En la Figura 2.21 se muestra una función que publica un marcador tipo CUBE sobre el mapa del proyecto en Rviz, la forma de este cubo se puede cambiar según la escala, por ejemplo, el

ancho de las paredes que se ha considerado en la Figura 2.20 es de 0.3 a una altura de 0.5. A continuación, se explicará los campos más importantes de la función de la Figura 2.21.

- **pared1.type:** Es el tipo de marcador (CUBE, SPHERE, LINE, etc), para el proyecto se definió un Cubo para emular las paredes del entorno.
- **pared1.id:** Es el identificador único de cada marcador
- **pared1.ns:** Es el nombre que se mostrará en Rviz y es un identificador grupal
- **pared1.lifetime:** Es el tiempo de duración de espera antes de eliminar el marcador
- **pared1.action:** Permite Añadir, modificar o eliminar el o los marcadores.

Los campos restantes se configuran de forma intuitiva, como: la escala, la posición, la orientación y el color de los marcadores.

```
Def pared(pub_marker):
    pared1=Marker()
    pared1.type =pared1.CUBE
    pared1.header.frame_id="map"
    pared1.id = 0
    pared1.ns ="pared_1"
    pared1.lifetime=rospy.Duration()
    pared1.action = pared1.ADD
    pared1.scale.x = 0.3
    pared1.scale.y = 2.5
    pared1.scale.z =0.5
    pared1.pose.position.x = 0.1
    pared1.pose.position.y = 2.9
    pared1.pose.position.z = 0.25
    pared1.pose.orientation.x = 0.0
    pared1.pose.orientation.y = 0.0
    pared1.pose.orientation.z = 0
    pared1.pose.orientation.w = 1
    pared1.color.a = 1
    pared1.color.r = 6.6
    pared1.color.g = 6.6
    pared1.color.b = 6.6
    pub_marker.publish(pared1)
```

Figura 2.21. Creación de paredes desde python. [Fuente Propia]

Si el código de la Figura 2.21 se lo repite con el mismo name space (ns), diferente identificador (id) y variando la posición y orientación de cada marcador se puede visualizar varias paredes del entorno, tal como se ve en la Figura 2.20.

2.2.3.2 Tipo esferas

Dentro del proyecto se añade nodos aleatorios al entorno y la mejor forma de representar esos nodos es mediante esferas. La creación del marcador tipo SPHERE se configura y se visualiza en la Figura 2.22. Para este tipo de marcador se ajusta tanto la escala en x como en y, pero la escala z se mantiene con cero, ya que no se dará una altura a la esfera.

```
def puntosRandom(p1,pub_markers, cont_id):
        pNuevos=Marker()
        pNuevos.type=pNuevos.SPHERE
        pNuevos.header.frame_id="map"
        pNuevos.header.stamp=rospy.Time.now()
        pNuevos.ns="puntos random"
        pNuevos.id= cont_id
        pNuevos.action=pNuevos.ADD
        pNuevos.pose.position.x=p1.x
        pNuevos.pose.position.y=p1.y
        pNuevos.pose.position.z=0
        pNuevos.pose.orientation.x = 0.0
        pNuevos.pose.orientation.y = 0.0
        pNuevos.pose.orientation.z = 0.0
        pNuevos.pose.orientation.w = 0.0
        pNuevos.scale.x = 0.1
        pNuevos.scale.y = 0.1
        pNuevos.scale.z = 0.0
        pNuevos.color.a = 1
        pNuevos.color.r = 125/255
        pNuevos.color.g = 224/255
        pNuevos.color.b = 38/255
        pNuevos.points.append(p1)
        pub_markers.publish(pNuevos)
```

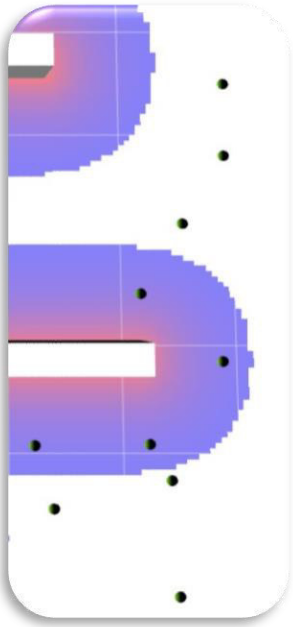


Figura 2.22. Configuración y visualización de esferas en Rviz [Fuente Propia]

2.2.3.3 Tipo líneas

Los Marcadores de tipo LINE_LIST permiten formar una arista entre dos nodos dentro de Rviz, para ello es necesario conocer las coordenadas (x, y, z) de cada nodo y se añade a un vector que posteriormente formará una línea desde la primera coordenada hasta la segunda. La importancia de este marcador dentro del proyecto es vital, ya que se podrá graficar las aristas de la ruta de cada algoritmo. La configuración y la visualización de este marcador se puede visualizar en la Figura 2.23.

```
def Rutas(p1,p2,pub_makers,cont_id):
        unirPuntos=Marker();vertices=Marker()
        vertices.type=vertices.POINTS
        unirPuntos.type=unirPuntos.LINE_LIST
        unirPuntos.header.frame_id="map"
        unirPuntos.header.stamp=rospy.Time.now()
        unirPuntos.ns="rutasPRM"
        unirPuntos.id=cont_id
        unirPuntos.action=unirPuntos.ADD
        unirPuntos.pose.orientation.w=1
        unirPuntos.scale.x=0.018
        unirPuntos.color.a=0.5
        unirPuntos.color.r=0/255
        unirPuntos.color.g=0/255
        unirPuntos.color.b=0/255
        unirPuntos.points.append(p1)
        unirPuntos.points.append(p2)
        pub_makers.publish(unirPuntos)
```

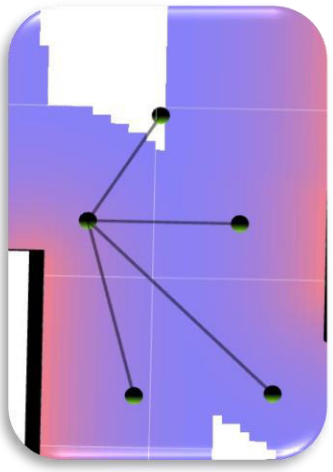


Figura 2.23. Configuración y visualización de aristas en Rviz [Fuente Propia]

2.2.3.4 Tipo Puntos

En la Figura 2.24 se puede visualizar la configuración y visualización del marcador tipo POINTS, ya que dentro de Rviz permiten visualizar únicamente la coordenada (x, y, z) de varios nodos, en el caso del proyecto se toma a dos importantes que es el punto de inicio y el punto de llegada en forma de cuadrado. Este marcador al igual que los otros también se puede ajustar su escala según la necesidad para hacerlo más llamativo dentro del entorno.

```
def Meta(p1,p2,pub_markers):
        # configuracion para el punto de inicio
        p_init=Marker()
        p_init.type=p_init.POINTS
        p_init.header.frame_id="map"
        p_init.header.stamp=rospy.Time.now()
        p_init.ns="nodos inicio/llegada"
        # configuracion para el punto de llegada
        p_meta=Marker()
        p_meta.type=p_meta.POINTS
        p_meta.header.frame_id="map"
        p_meta.header.stamp=rospy.Time.now()
        p_meta.ns="vertices inicio/llegada"
        p_init.id=0
        p_meta.id=1
        p_init.action=p_init.ADD
        p_meta.action=p_meta.ADD
        p_init.color.a = 1
        p_init.color.g = 1.0
        p_init.scale.x = 0.2
        p_init.scale.y = 0.2
        p_meta.color.a = 1
        p_meta.color.r = 1.0
        p_meta.scale.x = 0.2
        p_meta.scale.y = 0.2
        p_salida=Point()
        p_llegada=Point()
        p_salida.x = p1.x; p_salida.y = p1.y
        p_llegada.x = p2.x; p_llegada.y = p2.y
        p_init.points.append(p_salida)
        p_meta.points.append(p_llegada)
        pub_markers.publish(p_init)
        pub_markers.publish(p_meta)
```

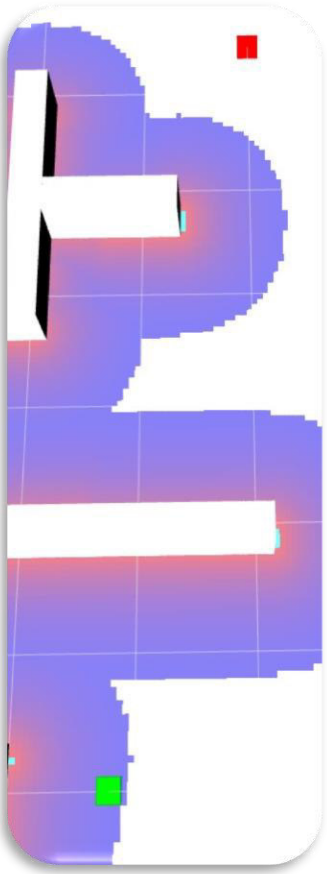


Figura 2.24. Configuración y visualización de las coordenadas de los nodos de inicio y llegada en Rviz. [Fuente Propia]

2.2.4 INTEGRACIÓN DEL ROBOT MÓVIL TURTLEBOT3 BURGER EN RVIZ

Las ventajas de utilizar Rviz es que se puede conocer como es la interacción del robot móvil con el entorno, ya que con ayuda de los marcadores se puede visualizar la ruta de navegación o el estado del robot. En el presente proyecto se integran a dos robots móviles al entorno de Gazebo los mismos que pueden ser visualizados en Rviz. En la Sección 2.1.2 se detalla el código base para la integración de un robot móvil en Gazebo. A continuación,

se explicará cómo se pueden integrar dos robots en Rviz; para ello se explica la forma en la que se envían los modelos de los robots a Rviz desde los ficheros .xacro o .urdf (véase la Figura 2.25).

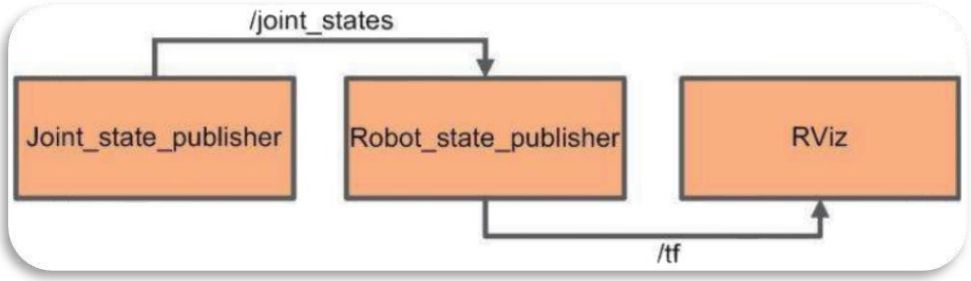


Figura 2.25. Nodos y tópicos del estado del robot en Rviz [28]

A continuación, se explica los términos presentados en la Figura 2.25 [28].

- **Joint_state_publisher**: Este nodo lee los parámetros del description_robot y publica todas las articulaciones no configuradas, información como la velocidad, posición, aceleración, frecuencia, etc.
- **robot_state_publisher:** Este nodo usa el método de cinemática directa, ya que se suscribe a la información de las articulaciones del robot según el modelo de su árbol cinemático, una vez suscrito publica los datos en un marco de coordenadas en 3D uniendo a cada articulación. Las articulaciones que conforman al robot se definen en los archivos STereoLithography (.stl) y son importados desde los ficheros .urdf.
- **tf:** Esta es la librería que estandariza el seguimiento de los marcos de coordenadas 3D y transforma los datos de todo el sistema para visualizarlos en Rviz.

Para identificar a los robots móviles en el presente proyecto se les ha asignado las siguientes abreviaturas como al robot móvil 1 con r1 y al robot móvil 2 con r2, tal como se muestra el fragmento del código en la Figura 2.26. Sin embargo, los ficheros (.xacro y .urdf) que conforman al robot móvil en la Sección 2.1.2 deben ser duplicados y nombrados de forma diferente, como: turtlebot3_burger1 y turtlebot3_burger2; de modo que durante la simulación los robots (r1 y r2) visualizados en Gazebo tengan las mismas posiciones y orientaciones que los robots (r1 y r2) de Rviz. Para ello, en la Figura 2.26 se realiza la configuración y ejecución de los nodos robot_state_publisher y joint_state_publisher que son los nodos encargados de mostrar el estado del robot en Rviz.

Dentro del fragmento de código, el parámetro más importante es tf_prefix, que es el parámetro a cuál se le asigna un nombre distintivo para identificar y visualizar el estado del robot dentro de la plataforma de Rviz, por ejemplo, r1_tf y r2_tf.

```
<!-- Grupo Robot 1 -->
  <group ns = "$(arg ns1)">
    <param name="robot_description
" command="$(find xacro)/xacro  $(
find avance1_path_planning)/robots
_avance1/urdf/turtlebot3_burger1.u
rdf.xacro" />

    <param name="tf_prefix" value=
"r1_tf" />

    <node name="joint_state_publis
her" pkg="joint_state_publisher" t
ype="joint_state_publisher">
    <param name="use_gui" value="F
alse"/>
    </node>

    <node pkg="robot_state_publish
er" type="robot_state_publisher" n
ame="robot_state_publisher" output
="screen">
      <param name="publish_frequen
cy" type="double" value="50.0" />

      <param name="tf_prefix" valu
e="r1_tf" />
    </node>
  </group>
```

(a)

```
<!-- Grupo Robot 2 -->
  <group ns = "$(arg ns2)">
    <param name="robot_description
" command="$(find xacro)/xacro  $(
find avance1_path_planning)/robots
_avance1/urdf/turtlebot3_burger2.u
rdf.xacro" />

    <param name="tf_prefix" value=
"r2_tf" />

    <node name="joint_state_publis
her" pkg="joint_state_publisher" t
ype="joint_state_publisher">
    <param name="use_gui" value="F
alse"/>
    </node>

    <node pkg="robot_state_publish
er" type="robot_state_publisher" n
ame="robot_state_publisher" output
="screen">
      <param name="publish_frequen
cy" type="double" value="50.0" />

      <param name="tf_prefix" valu
e="r2_tf" />
    </node>
  </group>
```

(b)

Figura 2.26. (a) configuración del r1, (b) Configuración de r2 [Fuente Propia]

Finalmente, Para añadir varios robots en Gazebo y visualizarlos en Rviz se debe configurar el argumento del name_space (ns) de cada robot (véase la Figura 2.6), como es el caso de ns1 y ns2 en el fichero .launch de la Figura 2.26. A estos argumentos se los configura con r1 y r2 como identificadores de cada robot.

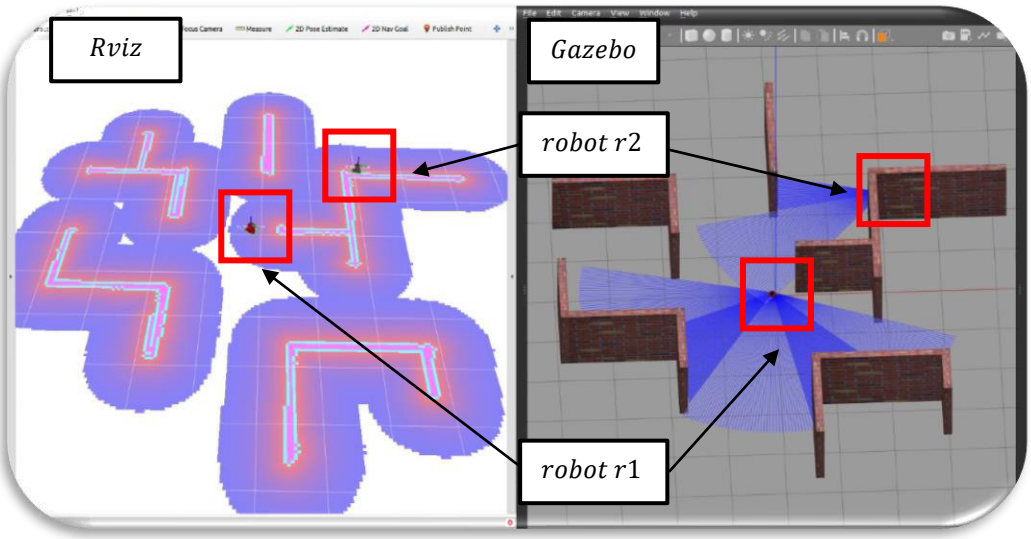


Figura 2.27. Visualización de robots en Rviz y Gazebo. [Fuente Propia]

2.3 IMPLEMENTACIÓN DE ALGORITMOS ITERATIVOS DE PLANEACIÓN DE RUTAS

Considerando lo detallado en las Secciones 2.1 y 2.2, se procede con la implementación de los algoritmos de planeación de rutas, ya que se tiene diseñado el entorno con los robots aptos para realizar pruebas y el monitoreo de su navegación. Como se indicó en las secciones anteriores, en Gazebo se visualizan a los robots y el entorno, pero el monitoreo del proyecto se lo hace desde Rviz. Como se comentó en la Sección 2.2.3, el lenguaje de programación que se emplea dentro del proyecto es python.

2.3.1 ALGORITMO DE ÁRBOLES ALEATORIOS DE EXPANSIÓN RÁPIDA (RRT)

Cabe recalcar que el enfoque de este trabajo es encontrar una ruta de navegación de manera eficiente mediante algoritmos iterativos, por esta razón en este proyecto de titulación el primer algoritmo que se desarrollo es el RRT, por su simplicidad y la vasta información bibliográfica disponible para su implementación. Al algoritmo RRT de la Sección 1.4.4.1 se añadió las posiciones de los obstáculos y otras modificaciones que mejoran la búsqueda de las rutas, dando como resultado el algoritmo de la Figura 2.28.

Algorithm 1: RRT Algorithm

Input: $\sigma, n, x_{init}, x_{meta}$
Result: $Path\ from\ x_{init}\ to\ x_{meta}$
$E_i \leftarrow x_{init}$, $path \leftarrow False$
$T \leftarrow E_i$, $Obst \leftarrow init.Obst()$
For i **in range** n **:**
    $x_{rand} \leftarrow Sample(Obst)$
    $x_{near} \leftarrow Near(x_{rand}, T)$
    $x_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
    $E_i \leftarrow Edge(x_{new}, x_{near})$
    If $CollisionsFree(E_i, Obst)$ **:**
        $T \leftarrow addNewNode(x_{new})$
        $addEdgeRviz(E_i)$
    If $x_{new} == x_{meta}$:
        $path \leftarrow True$
        $Break$

Figura 2.28. Algoritmo RRT, basado y adaptado de [18]

Antes de iniciar con la explicación del algoritmo RRT se detallará de forma resumida cuál es el aporte de cada función de la Figura 2.28, ya que se utilizarán estas funciones con algunas modificaciones dentro de la ejecución del algoritmo PRM.

Partiendo de la declaración e inicialización de obstáculos, la función de init.Obst carga todas las coordenadas de las paredes (obstáculos) y lo guarda en el vector Obst, este vector ayudará a verificar si existe un obstáculo entre dos nodos. Adicionalmente, en el vector T se guardarán todos los nodos que forman parte de la ruta hasta llegar al nodo de meta. A continuación, se listan las funciones para su previa explicación.

- **Sample:** En esta función se generan nodos random con coordenadas (x, y), el nodo de salida debe cumplir las siguientes condiciones, que se encuentre dentro del mapa y con ayuda del vector Obst se verifica si la coordenada (x, y) del nodo no se superponga con las coordenadas de los obstáculos.
- **Near:** En esta función se calculan las distancias entre el nodo que ingresa con todos los nodos guardados en el vector T y según la distancia más corta se escoge un solo nodo de este vector, el cual será el nodo más cercano al nodo ingresado.
- **Steer:** Esta función calcula el ángulo y la distancia desde un nodo a otro, la finalidad es ajustar mediante un factor esta distancia y acortarla en la misma dirección.
- **Edge:** En esta función se toma dos nodos a la entrada y a la salida se arrojan en un arreglo que se guarda en el vector E_i y este es considerado como el vector de las aristas que serán las posibles rutas para graficar en Rviz.
- **CollisionFree:** Esta es una de las funciones más importantes, ya que permite verificar si no se intersecan las aristas formadas entre dos nodos (E_i) con los obstáculos del entorno. La salida de esta función es verdadero o falso.
- **AddNewNode:** Esta Función grafica mediante marcadores un nodo como un punto circular en la plataforma de Rviz, y posteriormente lo guarda en el vector T
- **AddEdgeRviz:** Esta función grafica la arista E_i en la plataforma de Rviz con ayuda de marcadores dando la forma de ramas de árbol.

Una vez identificado el aporte de las funciones de la Figura 2.28, se puede explicar con mayor énfasis el funcionamiento del algoritmo RRT, ya que como primer paso se guarda el nodo inicial (x_{init}) en la primera posición del vector T. La variable que detendrá las iteraciones del algoritmo es $path$, por lo cual se inicializa con $False$ hasta encontrar una ruta libre de colisiones y cambiar de valor a $True$.

Se debe tomar en cada iteración del algoritmo RRT se genera un nodo random (x_{rand}) mediante la función Sample. El nodo x_{rand} ingresa en la función Near con el vector T, en

esta función como ya se explicó se calculan las distancias de x_{rand} con respecto a cada nodo guardado en el vector T y se elegirá un nodo cercano (x_{near}) al nodo x_{rand} con la distancia más corta sin considerar los obstáculos de por medio. En la Figura 2.29 se visualiza al nodo x_{rand} con su respectivo x_{near}.

Algorithm 1: RRT Algorithm
1: **Input:** $\sigma, n, x_{init}, x_{meta}$
2: **Result:** $Path\ from\ x_{init}\ to\ x_{meta}$
3: $E_i \leftarrow x_{init}$, $path \leftarrow False$
4: $T \leftarrow E_i$, $Obst \leftarrow init.Obst()$
5: **For** i **in range** n **:**
6: $x_{rand} \leftarrow Sample(Obst)$
7: $x_{near} \leftarrow Near(x_{rand}, T)$
8: $x_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
9: $E_i \leftarrow Edge(x_{new}, x_{near})$
10: **If** $CollisionsFree(E_i, Obst)$ **:**
11: $T \leftarrow addNewNode(x_{new})$
12: $addEdgeRviz(E_i)$
13: **If** $x_{new} == x_{meta}$:
14: $path \leftarrow True$
15: $Break$

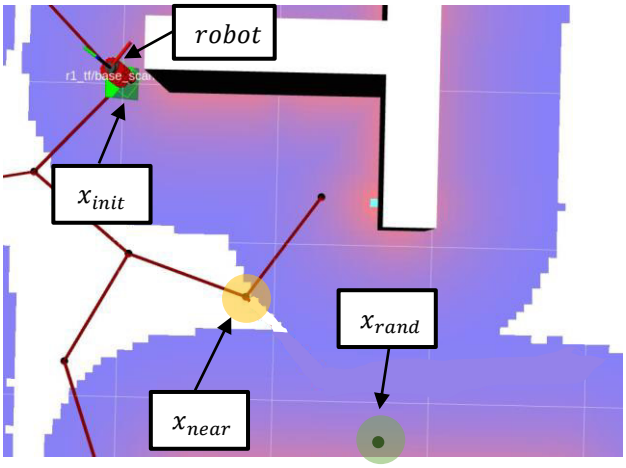


Figura 2.29. Algoritmo RRT, nodo x_{rand} y x_{near}. [Fuente Propia]

Luego de identificados los nodos x_{near} y x_{rand} se ingresan en la función Steer con el factor sigma (σ). En esta función se calcula el sentido y la magnitud desde el nodo x_{near} hacia el nodo x_{rand}, que de forma gráfica sería la línea entre cortada de la Figura 2.30. El factor σ ajusta el paso con el que se busca la mejor ruta, ya que se multiplica con la magnitud de los nodos ingresados, calculando así una segunda magnitud proporcional a la original a la que se le asocia el sentido y se determina un nodo nuevo (x_{new}), véase la Figura 2.30.

Algorithm 1: RRT Algorithm
1: **Input:** $\sigma, n, x_{init}, x_{meta}$
2: **Result:** $Path\ from\ x_{init}\ to\ x_{meta}$
3: $E_i \leftarrow x_{init}$, $path \leftarrow False$
4: $T \leftarrow E_i$, $Obst \leftarrow init.Obst()$
5: **For** i **in range** n **:**
6: $x_{rand} \leftarrow Sample(Obst)$
7: $x_{near} \leftarrow Near(x_{rand}, T)$
8: $x_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
9: $E_i \leftarrow Edge(x_{new}, x_{near})$
10: **If** $CollisionsFree(E_i, Obst)$ **:**
11: $T \leftarrow addNewNode(x_{new})$
12: $addEdgeRviz(E_i)$
13: **If** $x_{new} == x_{meta}$:
14: $path \leftarrow True$
15: $Break$

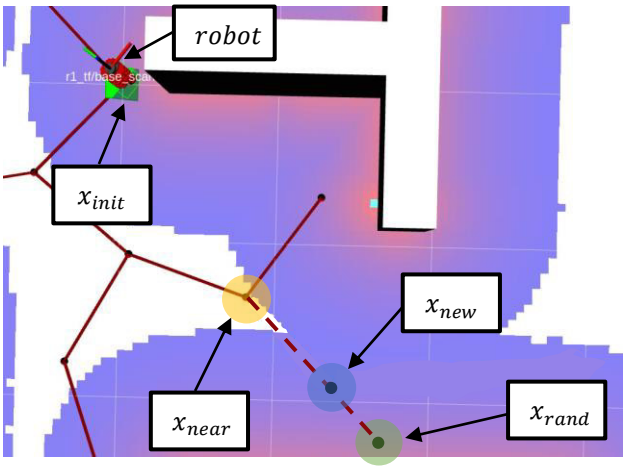


Figura 2.30. Algoritmo RRT, nodo x_{new}. [Fuente Propia]

Una vez que se determine el nodo x_{new} es un posible candidato para ser guardado en el vector T, pero antes debe verificarse si no existen obstáculos entre x_{new} y x_{near}. En la Figura 2.31 se visualiza que se cumple la condición de verificación mediante la función de CollisionsFree, por lo cual se traza la ruta y el nodo x_{new} se añade al vector T.

Algorithm 1: RRT Algorithm

Input: $\sigma, n, x_{init}, x_{meta}$
Result: $Path\ from\ x_{init}\ to\ x_{meta}$
$E_i \leftarrow x_{init}$, $path \leftarrow False$
$T \leftarrow E_i$, $Obst \leftarrow init.Obst()$
For i **in range** n **:**
  $x_{rand} \leftarrow Sample(Obst)$
  $x_{near} \leftarrow Near(x_{rand}, T)$
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
  $E_i \leftarrow Edge(x_{new}, x_{near})$
  If $CollisionsFree(E_i, Obst)$ **:**
    $T \leftarrow addNewNode(x_{new})$
    $addEdgeRviz(E_i)$
  If $x_{new} == x_{meta}$:
    $path \leftarrow True$
    $Break$

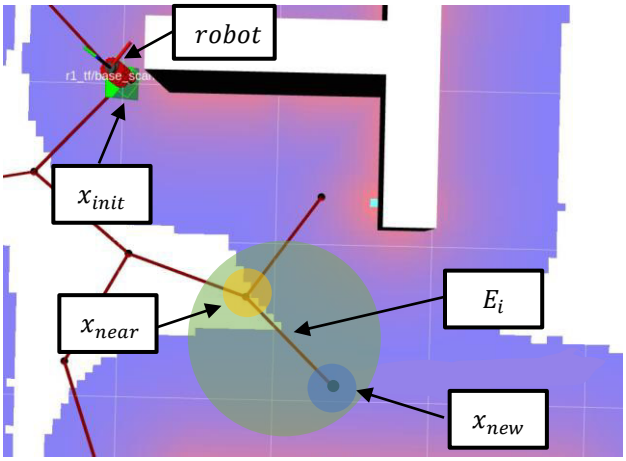


Figura 2.31. Algoritmo RRT, trazar ruta en Rviz. [Fuente Propia]

En la Figura 2.32 se puede visualizar un escenario donde se calcula x_{new} y se lo adjunta a E_i con el nodo x_{near} para su evaluación, sin embargo, esta ruta no cumple la condición de la función de CollisionFree; por lo tanto, no se añadirá este x_{new} al vector T. Además, también se puede visualizar que el nodo x_{rand} es considerado a pesar de que exista un obstáculo de por medio con el nodo x_{near}, ya que el nodo x_{new} es el de interés y depende bastante del valor que tome el factor σ. Variando el factor σ es muy probable que no existan obstáculos entre el nodo x_{new} y el nodo x_{near}.

Algorithm 1: RRT Algorithm

Input: $\sigma, n, x_{init}, x_{meta}$
Result: $Path\ from\ x_{init}\ to\ x_{meta}$
$E_i \leftarrow x_{init}$, $path \leftarrow False$
$T \leftarrow E_i$, $Obst \leftarrow init.Obst()$
For i **in range** n **:**
  $x_{rand} \leftarrow Sample(Obst)$
  $x_{near} \leftarrow Near(x_{rand}, T)$
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
  $E_i \leftarrow Edge(x_{new}, x_{near})$
  If $CollisionsFree(E_i, Obst)$ **:**
    $T \leftarrow$ [illegible]$wNode(x_{new})$
    add[illegible]$viz(E_i)$
  If x_{new} [illegible]$== x_{meta}$:
    $path \leftarrow True$
    $Break$

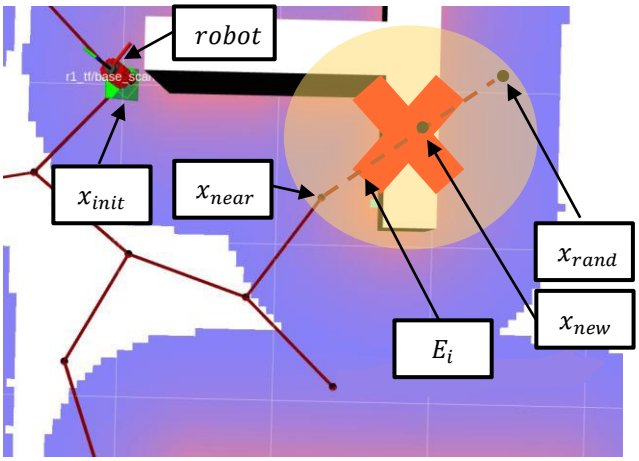


Figura 2.32. Algoritmo RRT, colisión con obstáculo. [Fuente Propia]

Una configuración necesaria dentro del algoritmo RRT para el presente proyecto es la distancia de separación entre el nodo x_{new} con las paredes del entorno (obstáculos), ya que es necesario establecer un margen de separación para que el robot móvil pueda moverse por las rutas sin rozarse con los obstáculos. Se debe tomar en cuenta que para encontrar un nodo x_{new} se parte desde el nodo x_{rand}, por lo cual esta configuración se implementó dentro de la función Sample, donde se toma las coordenadas (x, y) del nodo x_{rand} a las cuales se le suma un margen de separación en la parte superior $(y + margen)$, inferior $(y - margen)$, lateral izquierdo $(x - margen)$ y lateral derecho $(x + margen)$. De esta forma se encuentran cuatro coordenadas nuevas que serán evaluadas desde la coordenada de origen (x, y).

Una vez establecidas las coordenadas con los márgenes se hace un llamado a la función de CollisionFree, donde a la ruta E_i se lo establecerá desde el origen a cada una de estas nuevas coordenadas; si no se cumple la condición de esta función se evaluará a un siguiente nodo x_{rand} hasta que se cumpla la condición. En el algoritmo también se configuran las condiciones que delimitan el entorno, como se conoce el área del mapa no se evaluarán a los nodos x_{rand} que están fuera de este. De esta forma se garantiza que el nodo x_{new} estará siempre dentro del mapa a una distancia prudente de los obstáculos.

Finalmente, con todas las configuraciones realizadas se puede ver que las rutas de la Figura 2.33 están a un margen de separación con los obstáculos. Estas rutas crecerán de forma aleatoria hasta acercarse con el nodo de meta (x_{meta}) y una vez que se conecte con este nodo se finalizará las iteraciones del algoritmo RRT.

Algorithm 1: RRT Algorithm

1: **Input:** $\sigma, n, x_{init}, x_{meta}$
2: **Result:** $Path\ from\ x_{init}\ to\ x_{meta}$
3: $E_i \leftarrow x_{init}$, $path \leftarrow False$
4: $T \leftarrow E_i$, $Obst \leftarrow init.Obst()$
5: **For** i **in range** n **:**
6: $x_{rand} \leftarrow Sample(Obst)$
7: $x_{near} \leftarrow Near(x_{rand}, T)$
8: $x_{new} \leftarrow Steer(x_{rand}, x_{near}, \sigma)$
9: $E_i \leftarrow Edge(x_{new}, x_{near})$
10: **If** $CollisionsFree(E_i, Obst)$ **:**
11: $T \leftarrow addNewNode(x_{new})$
12: $addEdgeRviz(E_i)$
13: **If** $x_{new} == x_{meta}$:
14: $path \leftarrow True$
15: $Break$

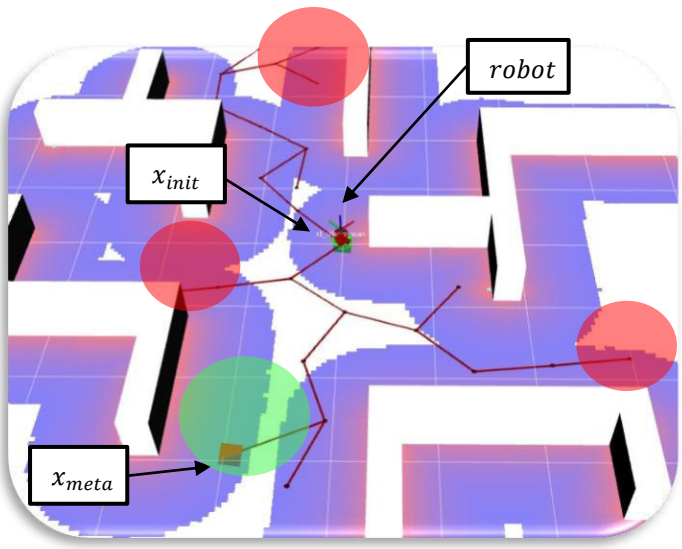


Figura 2.33. Algoritmo RRT, Ruta de x_{init} hasta x_{meta}. [Fuente Propia]

2.3.2 ALGORITMO DE MAPAS DE RUTAS PROBABILÍSTICAS (PRM)

Tal como se explicó en la Sección 1.4.4.2, la búsqueda de la ruta adecuada utilizando el algoritmo PRM se lleva a cabo mediante la construcción de caminos a partir de muestras aleatorias tomadas al entorno. En el algoritmo de la Figura 1.13 se han realizado unas modificaciones que ayudan a reutilizar funciones ya creadas con el algoritmo RRT, como es el caso de AddNewNode, CollisionFree, Sample, entre otros (véase la Figura 2.34).

Algorithm 2: PRM Algorithm

Input: n, q_{init}, q_{meta}
Result: $Path\ from\ q_{init}\ to\ q_{meta}$
$V_{q0} \leftarrow q_{init},\ \ path \leftarrow False$
$Obst \leftarrow init.Obst()$
For i **in range** n **:**
  While $not\ path$:
    $q_{rand} \leftarrow Sample(Obst)$
    If $CollisionsFree(q_{rand}, Obst)$:
      If $distance(q_{rand}, q_{meta}) < 1$:
      $V_q \leftarrow AddNewNode(q_{meta})$
      $path \leftarrow True$
      else:
      $V_q \leftarrow AddNewNode(q_{rand})$
      $Break$
if $path$:
  For q' **in** V_q:
    $E_i \leftarrow Edge(q', Vq)$
    $AddEdgeRviz(q', E_i)$

Figura 2.34. Algoritmo PRM, basado y adaptado de [21]

La diferencia del algoritmo de la Figura 1.13 con el de la Figura 2.34 es el detalle de los pasos hasta encontrar la mejor ruta, la forma con la que se encuentran a los nodos random (q_{rand}) y como estos están en un espacio libres de colisiones. De modo que se vaya avanzando con la explicación del algoritmo PRM dentro del proyecto, también se explicará cuáles han sido las modificaciones a las funciones del algoritmo RRT para que se ejecuten adecuadamente en el algoritmo actual. Además, en la Figura 2.34 se visualiza con mayor detalle las dos etapas que caracterizan al algoritmo PRM, que son: la fase de construcción y la fase de consulta para encontrar una ruta de navegación.

Como primer paso, se guarda el nodo inicial (q_{init}) en el vector de muestras V_q, en este vector se guardarán todos los nodos q_{rand} que se encuentren en un espacio libre de colisiones dentro del entorno. El nodo q_{rand} varia en cada iteración del algoritmo PRM y no

necesariamente debe estar cerca del nodo anterior, por lo cual hay una alta probabilidad de que no se llegue a encontrar una ruta adecuada de navegación.

En la función Sample de la Sección 2.3.1 se ha realizado la primera modificación, de modo que los nodos q_{rand} en la presente sección se determinan mediante una distribución aleatoria uniforme. La segunda modificación es la combinación de la función Sample con la función de CollisionFree del algoritmo RRT, donde resulta una nueva función a la cual se le nombro como CollisionFree por practicidad. La diferencia de esta función modificada con la de la Sección 2.3.1 es que no se ingresa una arista (E_i), sino que en este caso se ingresa simplemente un nodo (q_{rand}); en dicha función se obtiene la coordenada (x, y) del nodo q_{rand} ingresado y posteriormente se establece un margen de separación con los obstáculos tanto a los laterales como en la parte superior e inferior del nodo. En la Figura 2.35 se puede visualizar a todos los nodos q_{rand} que han sido verificados por la función CollisionFree, estos nodos son guardados en el vector V_q como muestras, estas muestras son espacios por donde se podrá mover un robot móvil TurtleBot3 Burguer.

Algorithm 2: PRM Algorithm

Input: n, q_{init}, q_{meta}
Result: $Path\ from\ q_{init}\ to\ q_{meta}$
$V_{q0} \leftarrow q_{init},\ path \leftarrow False$
$Obst \leftarrow init.Obst()$
For i **in range** n **:**
  While $not\ path$:
    $q_{rand} \leftarrow Sample(Obst)$
    If $CollisionsFree(q_{rand}, Obst)$:
      If $distance(q_{rand}, q_{meta}) < 1$:
        $V_q \leftarrow AddNewNode(q_{meta})$
        $path \leftarrow True$
      else:
        $V_q \leftarrow AddNewNode(q_{rand})$
        $Break$
if $path$:
  For q' **in** V_q:
    $E_i \leftarrow Edge(q', Vq)$
    $AddEdgeRviz(q', E_i)$

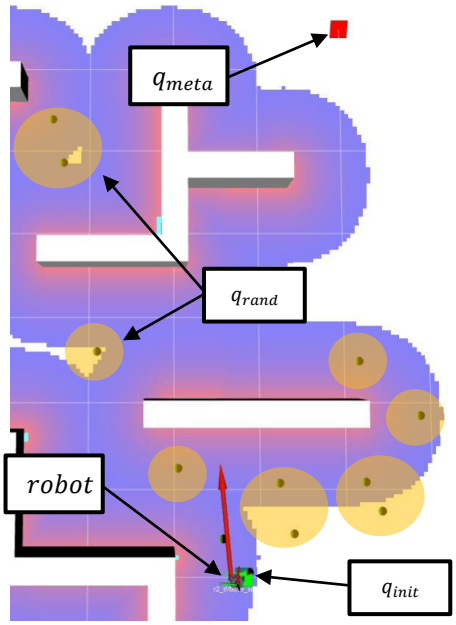


Figura 2.35. Algoritmo PRM, nodos q_{rand}. [Fuente Propia]

El número de muestras tomadas del entorno dependen de las iteraciones (n) y de la probabilidad de que tan cerca llegue a estar un nodo q_{rand} del nodo de meta (q_{meta}), ya que si el nodo q_{rand} tiene una distancia menor a $1\ px/m$ con el nodo q_{meta} se finaliza el bucle de búsqueda de muestras, tal como se visualiza en la Figura 2.36. Hay que tomar en cuenta que mientras más muestras se tomen existirá una mayor probabilidad de que se encuentre la mejor ruta, dado que la naturaleza del algoritmo PRM es aleatorio.

Algorithm 2: PRM Algorithm

1: **Input:** n, q_{init}, q_{meta}
2: **Result:** $Path\ from\ q_{init}\ to\ q_{meta}$
3: $V_{q0} \leftarrow q_{init},\ path \leftarrow False$
4: $Obst \leftarrow init.Obst()$
5: **For** i **in range** n **:**
6: **While** $not\ path$:
7: $q_{rand} \leftarrow Sample(Obst)$
8: **If** $CollisionsFree(q_{rand}, Obst)$:
9: **If** $distance(q_{rand}, q_{meta}) < 1$:
10: $V_q \leftarrow AddNewNode(q_{meta})$
11: $path \leftarrow True$
12: **else:**
13: $V_q \leftarrow AddNewNode(q_{rand})$
14: $Break$
15: **if** $path$:
16: **For** q' **in** V_q:
17: $E_i \leftarrow Edge(q', Vq)$
18: $AddEdgeRviz(q', E_i)$

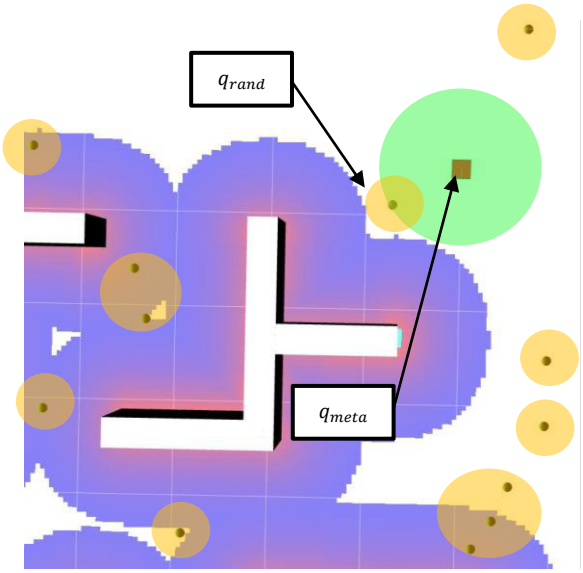


Figura 2.36. Algoritmo PRM, Fin de toma de muestras q_{rand}. [Fuente Propia]

En la Figura 2.37 se puede visualizar a la función Edge, dentro de esta se integran las funciones de Near y la de CollisionFree del algoritmo RRT. La diferencia de la función Edge de la Sección 2.3.1 con la actual es que a la salida se arrojan varios nodos del vector V_q que son cercanos al nodo q_{rand} sin colisionar con los obstáculos de por medio. El número de nodos cercanos que se ha establecido para el presente proyecto son 4, los cuales son guardados en el vector de aristas E_i; posteriormente se generará una arista entre el nodo q_{rand} con cada nodo cercano obtenido, tal como se visualiza en la Figura 2.37.

Algorithm 2: PRM Algorithm

1: **Input:** n, q_{init}, q_{meta}
2: **Result:** $Path\ from\ q_{init}\ to\ q_{meta}$
3: $V_{q0} \leftarrow q_{init},\ path \leftarrow False$
4: $Obst \leftarrow init.Obst()$
5: **For** i **in range** n **:**
6: **While** $not\ path$:
7: $q_{rand} \leftarrow Sample(Obst)$
8: **If** $CollisionsFree(q_{rand}, Obst)$:
9: **If** $distance(q_{rand}, q_{meta}) < 1$:
10: $V_q \leftarrow AddNewNode(q_{meta})$
11: $path \leftarrow True$
12: **else:**
13: $V_q \leftarrow AddNewNode(q_{rand})$
14: $Break$
15: **if** $path$:
16: **For** q' **in** V_q:
17: $E_i \leftarrow Edge(q', Vq)$
18: $AddEdgeRviz(q', E_i)$

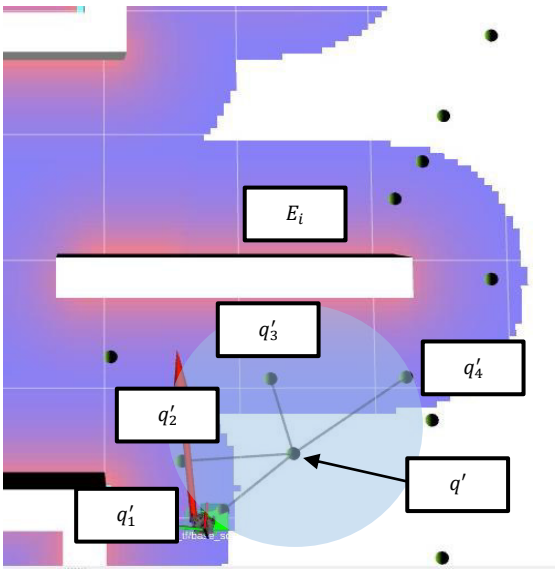


Figura 2.37. Algoritmo PRM. Creación de aristas (E_i) [Fuente Propia]

Finalmente, se concluye la primera fase del algoritmo PRM que es la toma de muestras y la construcción de las aristas, tal como se visualiza en la Figura 2.38. En la función addEdgeRviz se toma a cada muestra del vector V_q y se conecta con los 4 nodos más cercanos dentro del mismo vector, cabe recalcar que estos nodos se guardaron con anterioridad dentro del vector E_i. De este modo se visualizará en la plataforma de Rviz las posibles rutas que se podría seguir en la siguiente fase del algoritmo PRM que es la búsqueda de la ruta adecuada para la navegación del robot.

Algorithm 2: PRM Algorithm

Input: n, q_{init}, q_{meta}
Result: $Path\ from\ q_{init}\ to\ q_{meta}$
$V_{q0} \leftarrow q_{init},\ path \leftarrow False$
$Obst \leftarrow init.Obst()$
For i **in range** n **:**
  While $not\ path$:
    $q_{rand} \leftarrow Sample(Obst)$
    If $CollisionsFree(q_{rand}, Obst)$:
      If $distance(q_{rand}, q_{meta}) < 1$:
      $V_q \leftarrow AddNewNode(q_{meta})$
      $path \leftarrow True$
      else:
      $V_q \leftarrow AddNewNode(q_{rand})$
      $Break$
if $path$:
  For q' **in** V_q:
    $E_i \leftarrow Edge(q', Vq)$
    $AddEdgeRviz(q', E_i)$

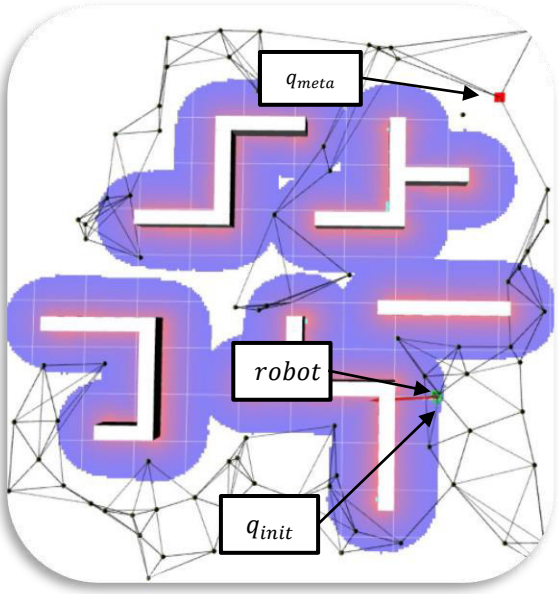


Figura 2.38. Algoritmo PRM, unión de los nodos cercanos. [Fuente Propia]

2.3.3 BÚSQUEDA DE LA RUTA ADECUADA PARA LA NAVEGACIÓN

Tanto en la Sección 2.3.1 como en la Sección 2.3.2 se explica el funcionamiento de los algoritmos iterativos implementados y la creación de las rutas en la plataforma de Rviz; sin embargo, no se hace énfasis en buscar una ruta adecuada para la navegación de los robots, por lo cual en esta sección se explicará el método que se ha implementado dentro del presente proyecto. Se parte del vector T en el caso del algoritmo RRT o del vector V_q en el algoritmo PRM, para cada caso, en estos vectores se tienen a los nodos que han sido seleccionados desde el nodo de salida hasta acercarse al nodo de llegada. Con la ayuda de la librería sklearn.neighbors en python se llaman a funciones que ayudan a ordenar a los nodos y saber cuáles son cercanos. A continuación, se explicará el funcionamiento de esta librería y que funciones son necesarias.

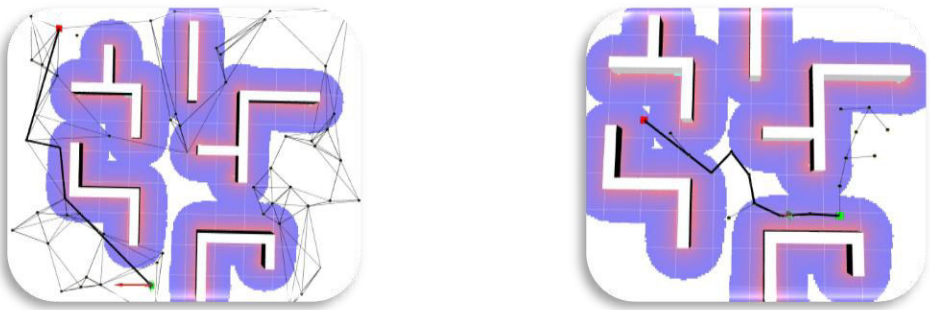
- **Sklear.neighbors**: esta librería proporciona los nodos cercanos o vecinos de un vector de muestras aleatorias, donde su aprendizaje se basa en la clasificación y la

regresión de estos datos mediante etiquetas. El número de muestras para analizar puede ser ingresada por el usuario, ser una constante definida o puede variar según la cantidad de los nodos guardados en el vector de muestras. Los nodos vecinos se determinan mediante su distancia y a esto se le asigna como etiqueta [29].

- **KDTree**: esta función es conocida como un árbol k-dimensional que permite ordenar los nodos según sus coordenadas mediante la búsqueda binaria en datos aleatorios.
- **query_radius** es una extensión del algoritmo KDTree que su función es formar un círculo de radio r en un punto (x, y) y seleccionar un nodo que este lo más cercano al punto y dentro del área del círculo.

La búsqueda de la ruta adecuada inicia con la función KDTree donde a su salida se tiene un arreglo con todos los nodos ordenados según su distancia respecto al origen $(0,0)$, pero los subíndices de este arreglo son las distancias, de modo que dificulta realizar operaciones. Por lo cual se creó un nuevo vector X, donde se guardarán los nodos ordenados partiendo desde el subíndice 0. Posteriormente, con el vector X se procede a la consulta de los nodos vecinos con la función query_radius, pero antes se crea un vector de puntos que va desde el nodo de inicio al de llegada y a cada uno de estos se creará un círculo con radio r, donde se escogerá al nodo que está dentro de su área como un posible vecino.

Una vez que se han seleccionado los nodos se hace un recorrido, que parte del nodo de Llegada y se selecciona un nodo vecino guardándolo en un vector llamado $pRutaLibre$. En cada iteración de esta verificación el nodo vecino pasa a ser el siguiente nodo al que le buscará un nodo cercano hasta llegar al nodo de salida. De modo que al momento de graficar las aristas de estos nodos en Rviz se verá como en la Figura 2.39.



(a) Algoritmo PRM (b) Algoritmo RRT

Figura 2.39. Búsqueda de la ruta adecuada para la navegación [Fuente Propia]

2.4 INTEGRACIÓN DEL SISTEMA DE MONITOREO

La integración de todas las aplicaciones desarrolladas se lleva a cabo mediante un menú desde el terminal de Linux Ubuntu. Las plataformas de simulación se organizan de tal forma que se pueda visualizar la generación de rutas en Rviz y la navegación en Gazebo. La organización de los algoritmos se lo ha hecho en 4 scripts, donde dos de ellos contienen los algoritmos RRT y PRM respectivamente, en otro script se guardan las configuraciones de los obstáculos y en el script restante se cargan las pruebas. El script de las pruebas es llamado y ejecutado desde la terminal de Linux.

Para crear un menú y correrlo desde la terminal, primero se crea un fichero .sh y se lo ejecuta mediante el comando bash [nombre-fichero]. Este fichero se crea con la finalidad de correr varios scripts de python en diferente ventada del terminal, de esta forma se garantiza que se ejecutará paralelamente los algoritmos y no existe retardo en la ejecución o interferencia de los datos durante la comunicación. El menú que centraliza la interacción de los algoritmos se visualiza en la Figura 2.40.

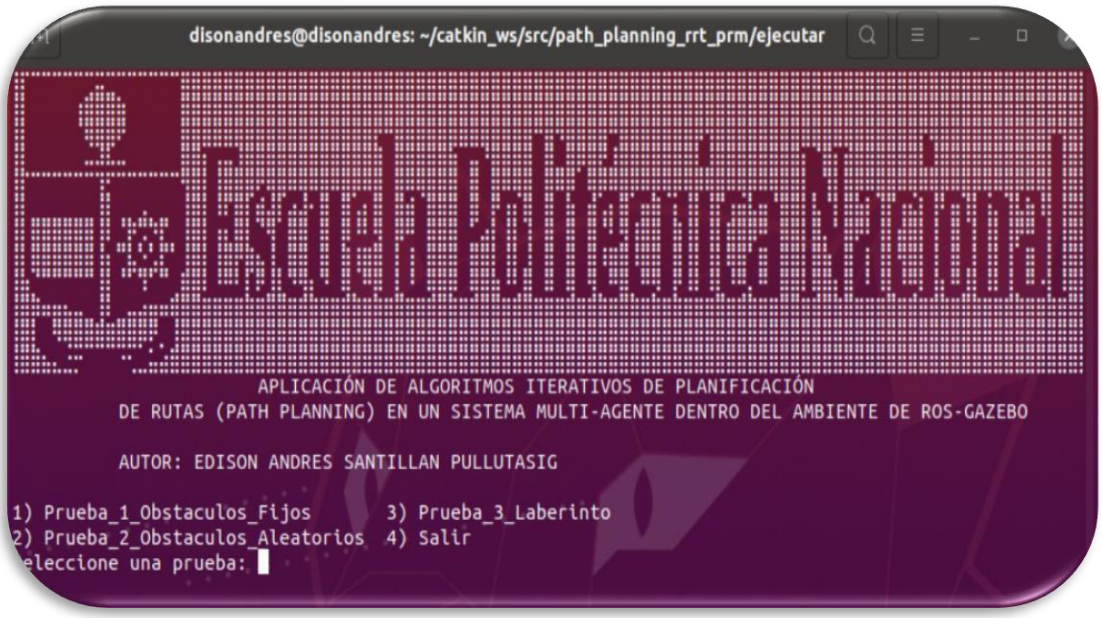


Figura 2.40. Menú de usuario. [Fuente Propia]

Los ficheros de configuración .launch también se corren durante la ejecución del fichero .sh al iniciar con la simulación, ya que primero se corre el roscore y posteriormente se abre cada plataforma (Gazebo, Rviz) con un retardo de 10 segundos en un solo terminal; y en otro terminal se corre el menú para cargar las pruebas. En los archivos .launch ya se realizan las configuraciones respectivas para visualizar un sistema multi-agente mostrando a dos robots móviles dentro del entorno de Gazebo y su visualización en Rviz.

Finalmente, la interfaz de usuario para la interacción de proyecto se observa en la Figura 2.41, en esta organización se puede visualizar la planeación de las rutas en la plataforma de Rviz y la navegación de los robots en Gazebo.

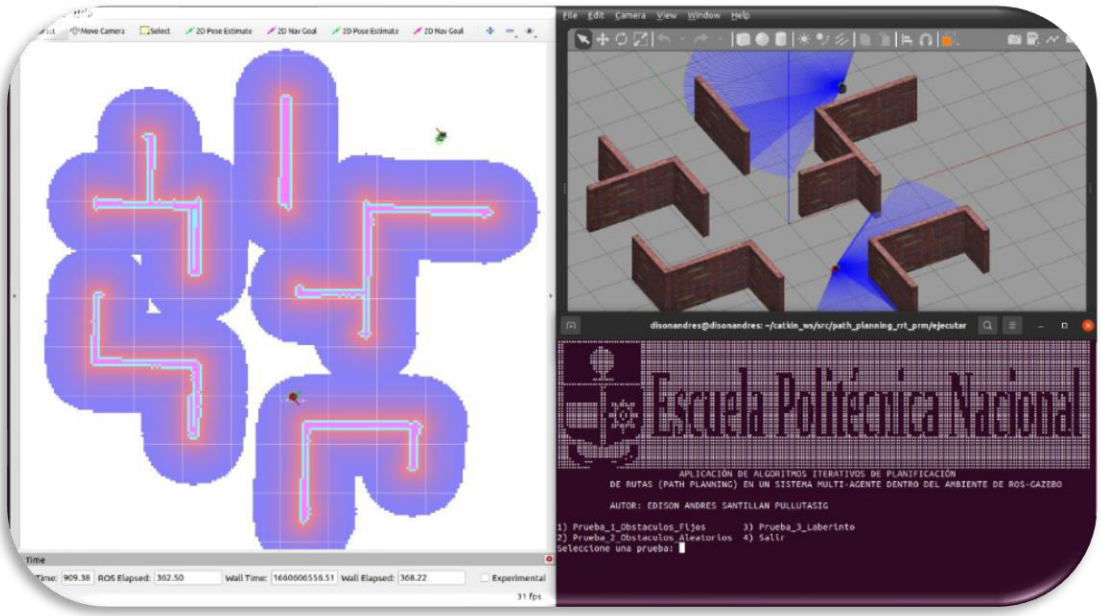


Figura 2.41. Interfaz de usuario. [Fuente Propia]

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

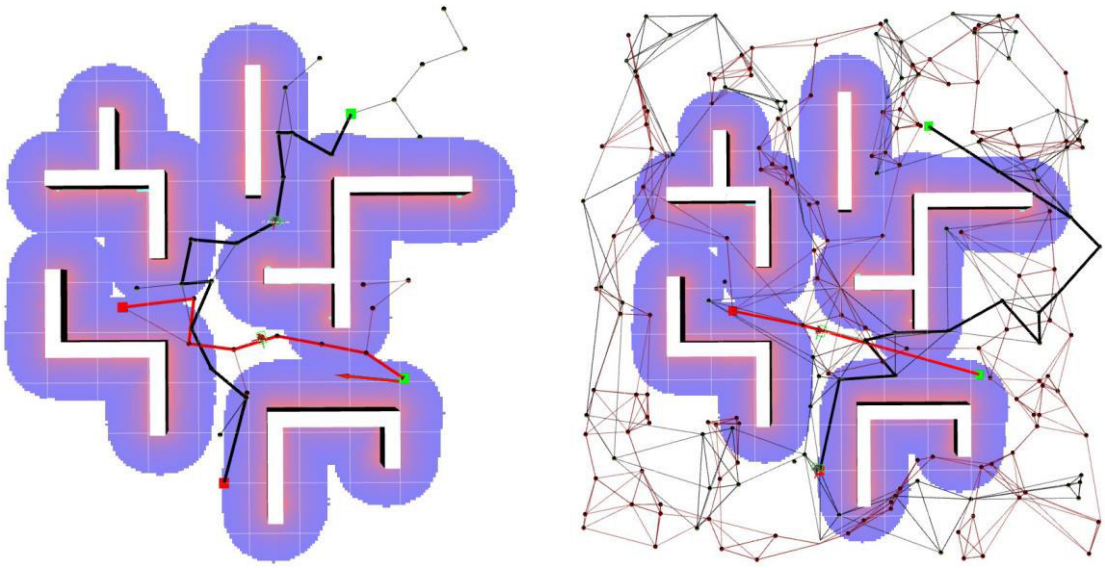
3.1 PRUEBAS Y RESULTADOS

Una vez desarrollado el sistema de monitoreo, este facilita la ejecución del entorno, los algoritmos de planeación de rutas y de los robots móviles turtlebot3 Burguer. Dentro del sistema multi-agente se ponen a prueba a los algoritmos implementados mediante diferentes configuraciones y obstáculos. Los resultados obtenidos se pueden visualizar mediante videos demostrativos donde se refleja la eficacia de los algoritmos (por favor revise el ANEXO VII).

3.1.1 RESULTADO 1, OBSTÁCULOS FIJOS EN GAZEBO

La primera prueba que se ha realizado se basa en utilizar los mismos obstáculos que se encuentran en la plataforma de Gazebo, ya que con las consideraciones de la Sección 2.1.1 las rutas se propagan a lo largo del entorno en busca de la meta. En la Figura 3.1 (a) se muestra la propagación de las rutas para el algoritmo RRT y en la Figura 3.1 (b) del algoritmo PRM. En ambos casos se realizan las pruebas con las mismas condiciones, donde el robot móvil 1 parte de la posición $P_1(3,-2)$ y la meta se ubica en la posición

$P_2(-2.5, -0.5)$; en cambio, para el robot móvil 2 parte de $P_1(2,3)$ y su meta se encuentra en la posición $P_2(-0.5, -4)$. Para ambos algoritmos se puede visualizar que cada algoritmo busca una ruta, el trazo rojo corresponde al robot 1 y el trazo negro corresponde al robot 2.



(a) Algoritmo RRT (b) Algoritmo PRM

Figura 3.1. Graficas de las rutas de navegación en Rviz [Fuente Propia]

De manera gráfica, en la Figura 3.1 se puede visualizar que cada algoritmo ofrece ventajas y desventajas, las cuales están asociadas a la búsqueda de la ruta de navegación. El robot 1 con el algoritmo PRM tiene una ruta menor en comparación al emplear el algoritmo RRT, en cambio el robot 2 del algoritmo PRM recorrerá una mayor distancia. Los datos tomados para esta prueba se pueden visualizar en la Tabla 3.1. Además, en cada gráfico es notorio el número de aristas y nodos que se presentan, por ejemplo, en la Figura 3.1 (b) se tiene un mayor número de posibles rutas que de la Figura 3.1 (a). Cabe mencionar que, en cada tabla de los resultados se sombrean filas de color morado, para identificar al robot 1 y al robot 2 con los mejores resultados en la búsqueda de la ruta adecuada con respecto a su distancia de navegación.

Tabla 3.1. Resultados de prueba 1

Algoritmo	Robot	# Iteraciones	# Nodos	Distancia Recorrida [px/m]	Tiempo de simulación [min]
RRT	R1	18	10	6.82	0.04
	R2	45	21	11.71	0.10
PRM	R1	136	136	5.89	1.90
	R2	68	68	16.43	0.59

De la Tabla 3.1 se puede concluir que el robot 1 del algoritmo PRM y el robot 2 del algoritmo RRT tienen las rutas de navegación. Un dato interesante en la prueba 1 es el tiempo de simulación con respecto al número de nodos, mientras más nodos se tengan graficados en el entorno, más tiempo de simulación tomarán los algoritmos en encontrar una ruta adecuada; tal como se muestra en la Figura 3.2. Además, en esta figura el número de iteraciones del algoritmo RRT es diferente al número de nodos, ya que se hace una selección en cada iteración de los nodos x_{new} para formar la ruta o no; en cambio en el algoritmo PRM el número de iteraciones es igual al de los nodos, porque se grafica un nodo q_{rand} libre de colisiones en el entorno en cada iteración.

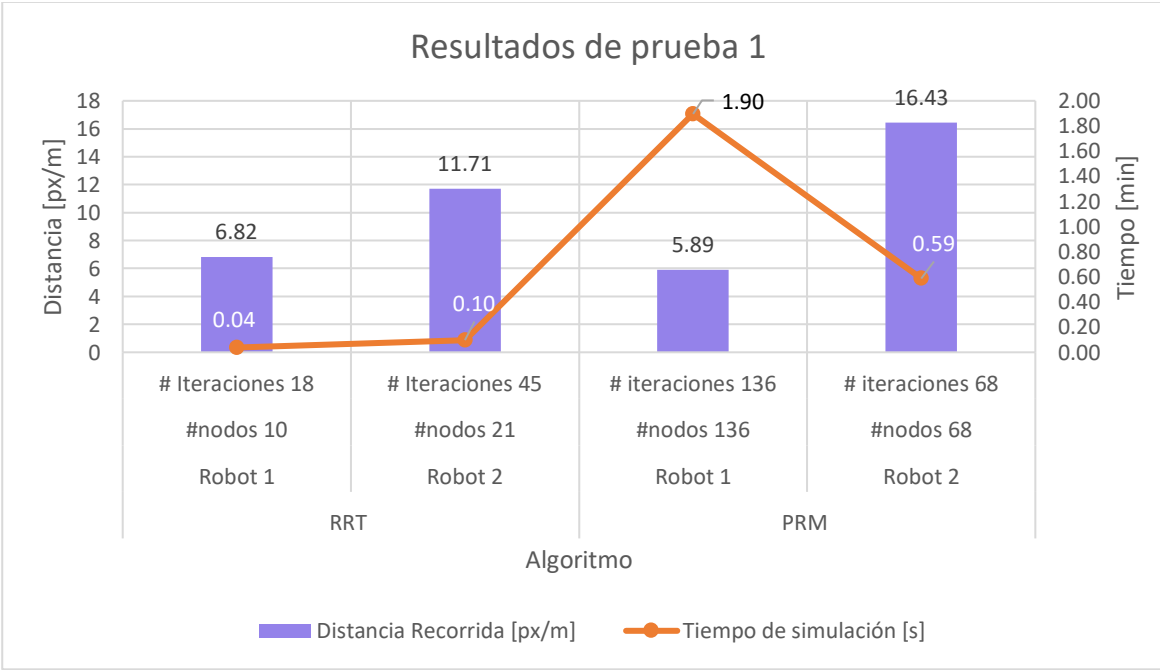


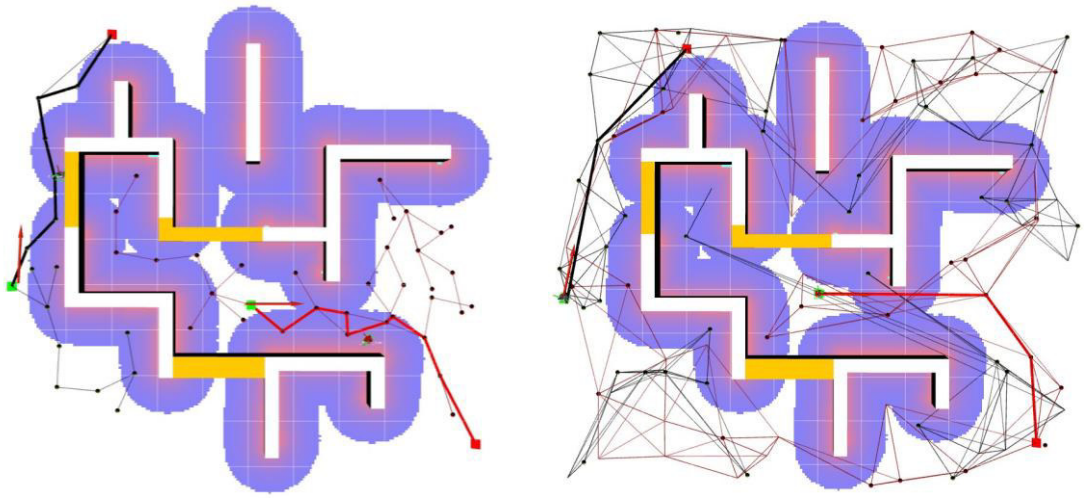
Figura 3.2. Comparación de resultados de la prueba 1 [Fuente Propia]

3.1.2 RESULTADO 2, INGRESO ALEATORIO DE OBSTÁCULOS EN RVIZ

En esta prueba se ingresan obstáculos en Rviz que no son reflejados en Gazebo, de esta forma se torna más fácil analizar el comportamiento de los algoritmos variando los obstáculos. Se conoce que los obstáculos de Gazebo vistos desde Rviz están como bloques de color blanco, por otro lado, los obstáculos ingresados de forma random serán de color amarillo, así se sabrá cual obstáculo es virtual en Rviz y cual si está modelado en la plataforma de Gazebo. Para ello se creó una base de 7 obstáculos en diferentes posiciones del mapa de Rviz y solo se mostrarán cuando el usuario inicie la simulación. Para la prueba 2 se muestran 3 de los 7 obstáculos de forma aleatoria y se procede a analizar cómo responden los algoritmos bajo las mismas condiciones. La posición de salida

del robot 1 es P_1(0,-1.5) y la posición de llegada es $P_2(5,-4.5)$; y para el robot 2 tiene como coordenada de salida $P_1(-5,-1)$ y como llegada al $P_2(-3,4.5)$.

La respuesta del algoritmo RRT busca la ruta de navegación de manera efectiva cuando se tienen presentes varios obstáculos tal como se visualizar en la Figura 3.3 (a) y la respuesta del algoritmo PRM se puede visualizar en la Figura 3.3 (b), donde se tiene trazos más rectos en la ruta. Cabe recalcar que los 3 obstáculos ingresados se visualizan de color amarillo.



(a) Algoritmo RRT (b) Algoritmo PRM

Figura 3.3. Graficas de las rutas de navegación en Rviz [Fuente Propia]

Los resultados de la Figura 3.3, a comparación de la Figura 3.1, el algoritmo PRM tiene un mayor número de nodos para los dos robots, en cambio, el algoritmo RRT genera menos nodos. Los resultados de la prueba 2 se pueden visualizar en la Tabla 3.2 y en este caso se ha determinado que el robot 2 del algoritmo PRM y el robot 1 del algoritmo RRT tienen las rutas adecuadas. Las distancias de las rutas entre algoritmos con esta prueba no son tan notables, ya que difieren por centésimas a comparación con la prueba 1, se podría decir que mientras más restricciones se colocan en el entorno se tendrá una tendencia de ruta en ambos algoritmos.

De la Tabla 3.2 se puede hacer una relación con el número de nodos, en esta prueba el algoritmo RRT tiene mayor número de iteraciones, por lo cual se demora más en acercarse a la meta; en cambio el algoritmo PRM le toma menos iteraciones en acercarse a la meta; pero el algoritmo RRT le toma menos tiempo en buscar la ruta adecuada para la navegación. Esto se debe a que el algoritmo PRM cuenta con una fase de consulta para

construir la ruta, esta consulta lo hace nodo a nodo y verifica si hay obstáculos de por medio demorándose así más tiempo de simulación.

Tabla 3.2. Resultados de prueba 2

Algoritmo	Robot	# Iteraciones	# Nodos	Distancia Recorrida [px/m]	Tiempo de simulación [min]
PRM	R1	47	47	7.51	0.34
	R2	51	51	6.71	0.44
RRT	R1	126	35	7.45	0.13
	R2	63	17	6.76	0.07

Con respecto al tiempo de ejecución se puede evidenciar que los tiempos del algoritmo PRM es mucho mayor que el algoritmo RRT pese a que se tengan pocos nodos a comparación de la prueba 1, esto se debe a la fase de consulta para la construcción de la ruta de navegación por eso le toma más tiempo de simulación. En esta prueba las rutas adecuadas de navegación son muy similares tanto para el robot 1 como el robot 2, véase la Figura 3.4.

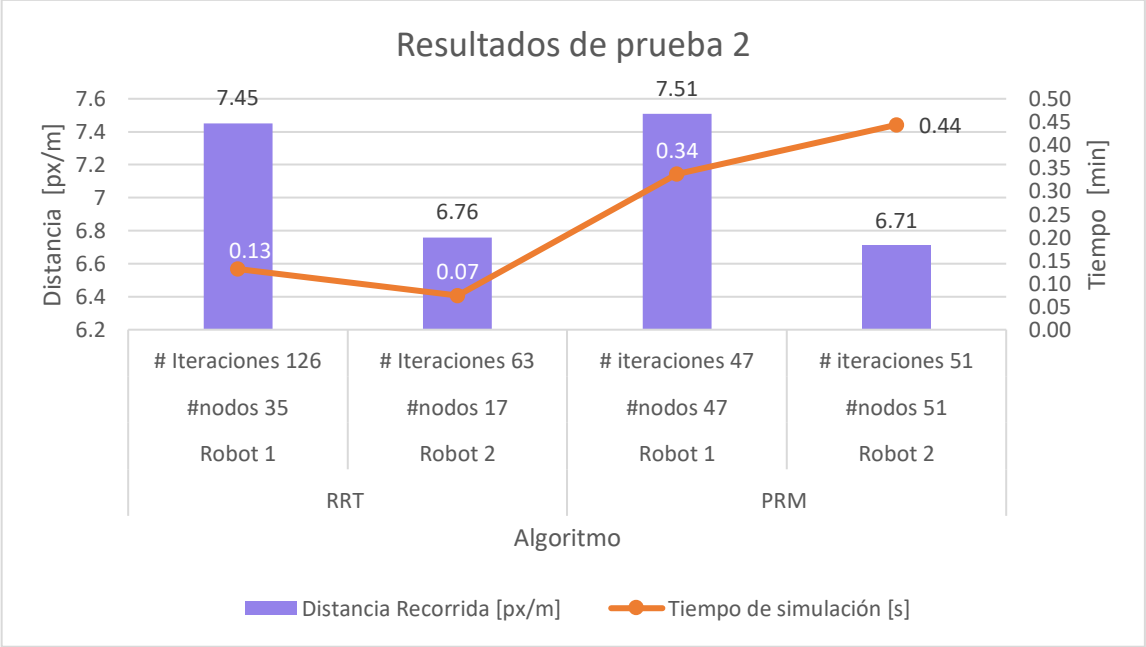


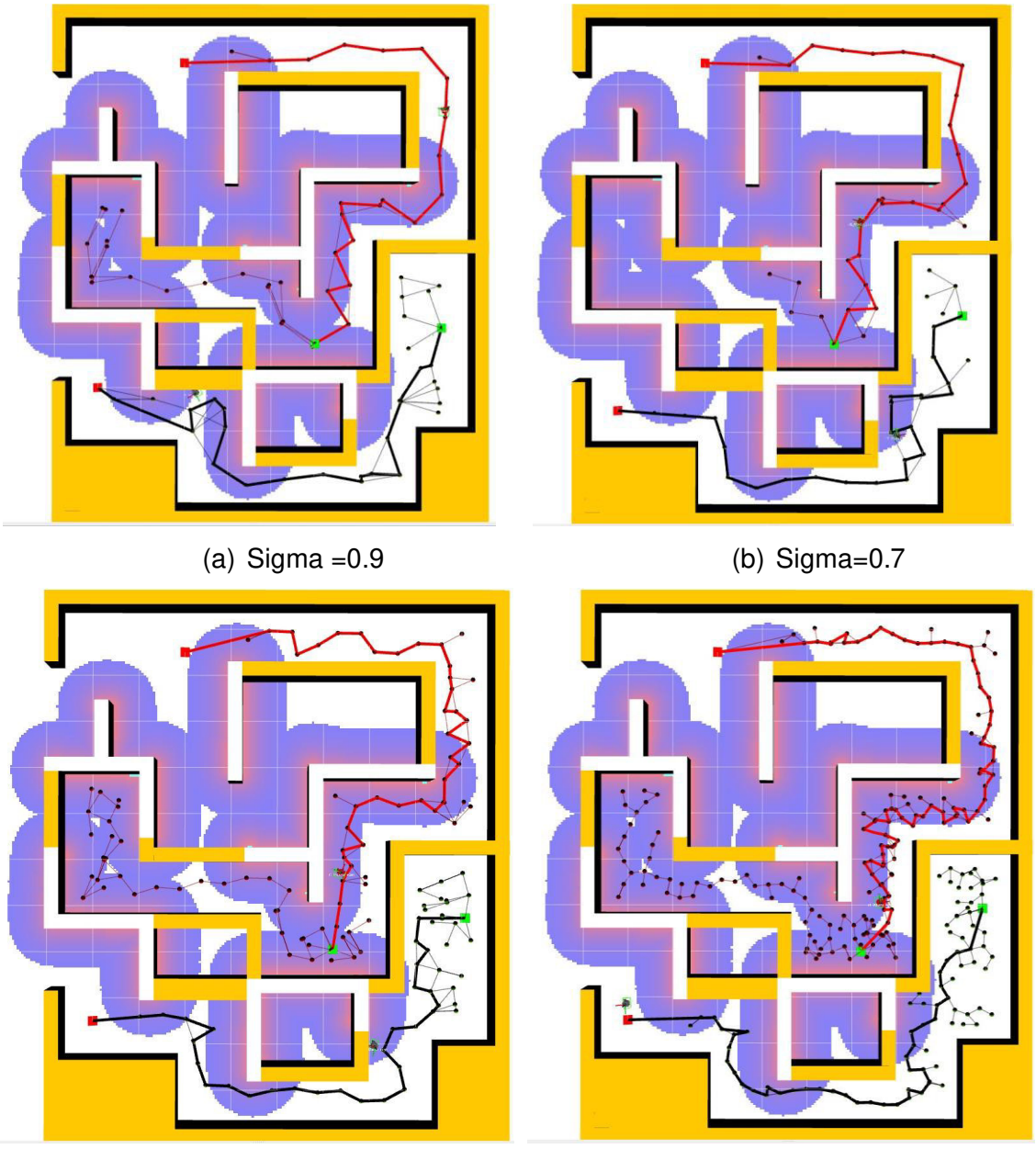
Figura 3.4. Comparación de resultados de la prueba 2 [Fuente Propia]

3.1.3 RESULTADO 3, VARIACIÓN DE PARÁMETROS DE LOS ALGORITMOS

La prueba 3 consiste en cambiar la configuración de los algoritmos, por ejemplo, en el algoritmo RRT se puede variar el paso (sigma) con el que avanza y en el algoritmo PRM se puede configurar el número de iteraciones. Por lo cual para esta prueba se ha dividido en dos secciones para explicar el comportamiento de cada algoritmo implementado.

3.1.3.1 Algoritmo RRT

Como se puede ver en la Figura 3.5, mientras se aumenta el factor sigma las ramificaciones de las rutas tienden a ser menos quebradas, pero mientras disminuye este factor se hace una búsqueda más fina sobre el entorno llegando a lugares más estrechos y de difícil acceso. La evaluación de este algoritmo se lo ha hecho en un escenario fijo tipo laberinto con las mismas condiciones, como el punto de salida del robot 1 se ubica en $P_1(1,-2)$ y el nodo de llegada en $P_2(-1,4.5)$; y el robot 2 está ubicado en $P_1(5.5,-1.5)$ y su posición de llegada está en $P_2(-3,-3.5)$.



(a) Sigma =0.9 (b) Sigma=0.7

(c) Sigma=0.5 (d) Sigma=0.3

Figura 3.5. Variación de parámetros al algoritmo RRT [Fuente Propia]

Para esta prueba se creó un laberinto con los obstáculos fijos de Gazebo y los obstáculos virtuales en Rviz. Este diseño está dividido en dos etapas, una para cada robot y se pondrá a prueba a los algoritmos cuando se tienen varios obstáculos de por medio, tal como se muestra en la Figura 3.5. El punto de partida del robot 1 se lo consideró debido a que existen dos caminos, uno sin salida y otro que lo llevaría a la meta. El algoritmo RRT tomará muestras y desplegará sus ramificaciones a lo largo de estos caminos hasta encontrar el nodo de meta o hasta finalizar con las iteraciones configuradas. Por otro lado, el punto de partida del robot 2 se lo consideró simplemente porque debe seguir el camino hasta llegar a la meta.

Los resultados de la Figura 3.5 se muestran en la Tabla 3.3, donde se puede visualizar la variación de sigma. Por ejemplo, en la Figura 3.5 (d) se puede ver un mayor número de nodos con una ruta de navegación mucho más quebrada que el grafico de la Figura 3.5 (b). El número de iteración con el factor sigma están asociados, ya que con un paso pequeño de avance se necesitará más iteraciones y por ende más nodos en el entorno hasta llegar a la meta. En el escenario tipo laberinto, el robot 1 tiene una alta probabilidad de encontrar una ruta para la navegación, de igual forma el robot 2; ya que el entorno está limitado por espacios reducidos y caminos estrechos. Con un sigma de 0.9 la propagación de las ramificaciones son pocas, por lo cual mucho mejor buscar su ruta adecuada.

Tabla 3.3. Resultados de prueba 3 para el algoritmo RRT

Parámetros	Robot	# Iteraciones	# Nodos	Distancia Recorrida [px/m]	Tiempo de simulación [min]
Sigma=0.9 C1	R1	427	46	16.56	0.89
	R2	418	31	13.67	0.82
Sigma=0.7 C2	R1	356	36	15.79	0.19
	R2	296	30	14.57	0.71
Sigma=0.5 C3	R1	802	111	16.99	2.86
	R2	638	56	14.75	1.66
Sigma=0.3 C4	R1	1107	187	19.68	7.43
	R2	842	94	14.31	3.43

En esta prueba según la Tabla 3.3 se puede comprobar que el robot 1 con un sigma de 0.9 y el robot 2 con un sigma de 0.7 tienen las mejores rutas más adecuadas; por lo cual el rango de operación del factor sigma sería de 0.7 a 0.9. Estos resultados se visualizan de mejor forma en la Figura 3.6, donde se nota que para un sigma de 0.3 el tiempo de simulación es aproximadamente 8 minutos; este tiempo es demasiado grande a comparación de los tiempos de las pruebas anteriores, por tal motivo no se realizó la prueba

con un sigma de 0.1, ya que el resultado sería similar con un tiempo de simulación mucho mayor.

En la Figura 3.6 también se puede verificar que con un sigma igual a 0.7 se tiene un menor número de iteraciones para el robot 1 y el robot 2. Un parámetro que se debe configurar al iniciar esta prueba es el número de iteraciones, ya que en las anteriores pruebas se colocó una base de 500 iteraciones, pero se encontraba el nodo de llegada en un máximo de 200 debido a que en estas pruebas se las realizaron con un sigma de 0.8. Sin embargo, en esta prueba mientras más pequeño es sigma se deben de configurar más de 1000 iteraciones, ya que por cada 30 o 50 iteraciones se avanza muy poco hacia el nodo de llegada, debido a que se generan muchas rutas dentro del mapa y la ruta adecuada de navegación será mucho más quebrada, tal como se puede visualizar en la Figura 3.5.

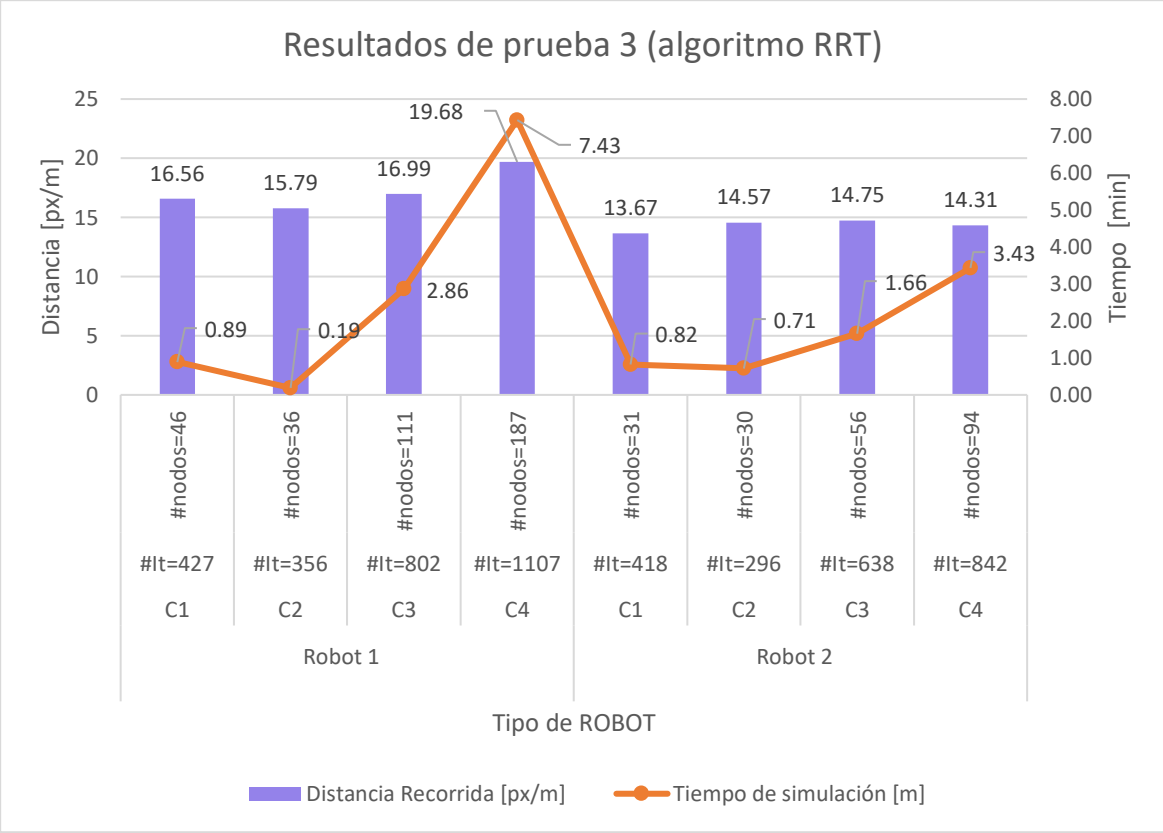
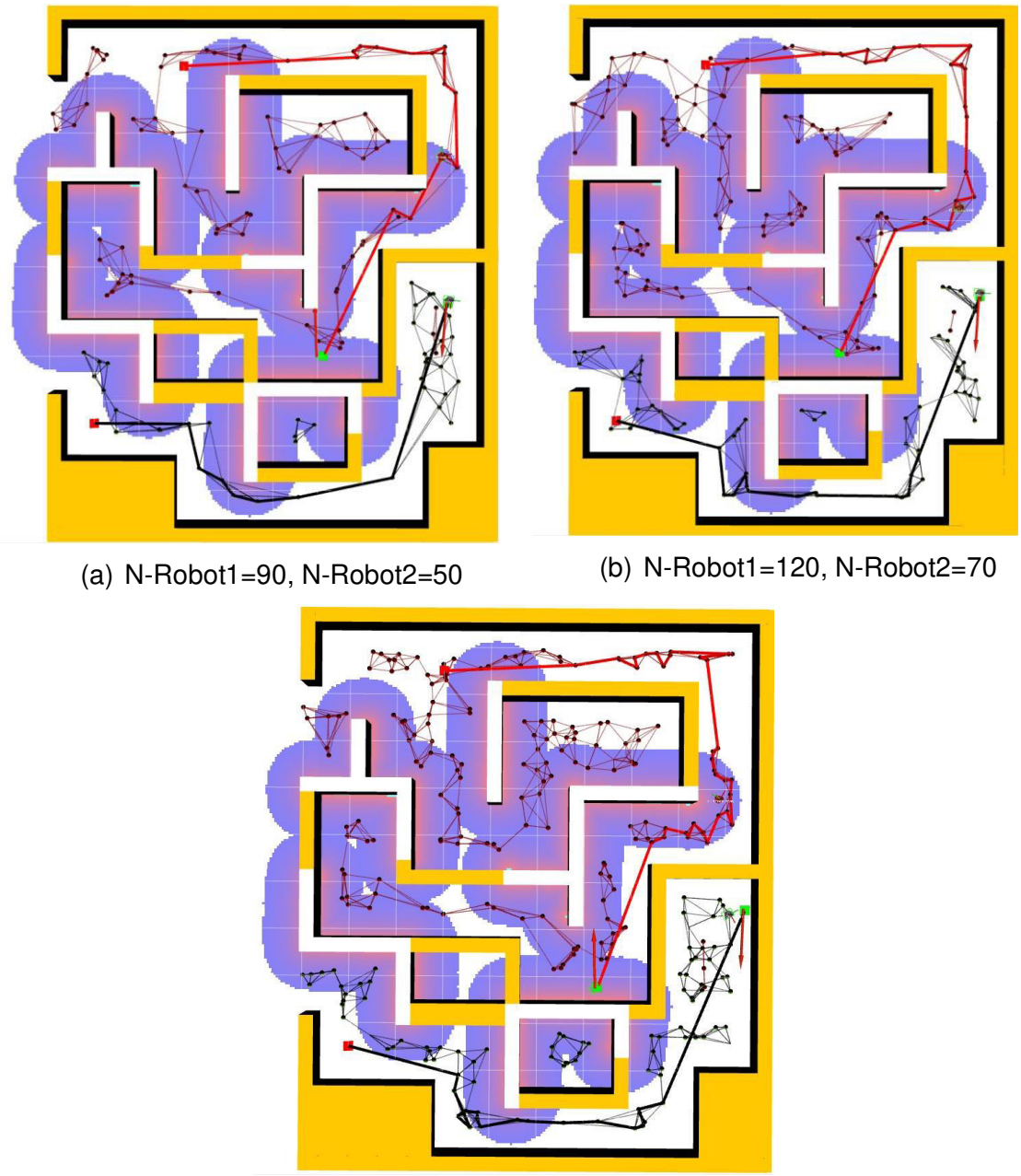


Figura 3.6. Comparación de resultados de la prueba 3 RRT [Fuente Propia]

3.1.3.2 Algoritmo PRM

En las pruebas del algoritmo PRM se aplican las mismas configuraciones que el algoritmo RRT, como los puntos de inicio y llegada. Sin embargo, en esta prueba no se varia sigma porque el algoritmo PRM no lo tiene, pero se puede variar el número de iteraciones ya que es igual al número de muestras (nodos) en el entorno. En las anteriores pruebas se ha

explicado que para este algoritmo el tiempo de ejecución es sumamente alto a comparación del algoritmo RRT, por lo cual se han realizado 3 pruebas, tal como se puede visualizar en la Figura 3.7. Para la ejecución de esta prueba se limita al mapa del laberinto por etapas según cada robot, ya que se graficarán muestras aleatorias alrededor de cada etapa. La región de operación del robot 1 está limitada por $x_{min} = -5.5$, $x_{max} = 5.5$, $y_{min} = -2.5$, $y_{max} = 5.5$; en cambio, para el robot 2 sería de la siguiente forma $x_{min} = -5.5$, $x_{max} = 5.5$, $y_{min} = -5.5$, $y_{max} = 2$; de esta forma se cubre toda la superficie del entorno.



(a) N-Robot1=90, N-Robot2=50

(b) N-Robot1=120, N-Robot2=70

(c) N-Robot1=150, N-Robot2=100

Figura 3.7. Variación de parámetros al algoritmo PRM [Fuente Propia]

Los resultados de la Figura 3.7 se pueden visualizar en la Tabla 3.4, donde se observa que la ruta adecuada de navegación para el robot 1 se determina con 90 muestras y para el robot 2 con 50 muestras. Cabe recordar que en cada iteración del algoritmo PRM grafica un nodo libre de obstáculos dentro del entorno al cual se le llama muestra. Según las pruebas realizadas, cuando se configuró un número pequeño de iteraciones el algoritmo falla, ya que no logra conectar los nodos con aristas y formar así una ruta, por lo cual en esta prueba se verifica que el algoritmo PRM tiene una baja probabilidad en encontrar un ruta de navegación. Mientras más obstáculos existan dentro del entorno más difícil le resulta al algoritmo en encontrar una ruta, como una posible solución es graficar más muestras al entorno, ya que mientras más nodos se tengan mayor probabilidad existirá en conectar las muestras y buscar una ruta adecuada para la navegación de los robots.

Si se grafican muchas muestras en un entorno con espacios estrechos la ruta de navegación tenderá a ser quebrada debido al algoritmo de búsqueda, como por ejemplo las rutas de la Figura 3.7 (c) con las rutas de la Figura 3.7 (a). En esta prueba la probabilidad de que el robot 1 pueda encontrar una ruta es intermedia (50%), ya que su nodo de Llegada está en un espacio abierto. Cómo se explicó en la Sección 2.3.3 la creación de la ruta de navegación se inicia conectando muestras desde el nodo de llegada hasta el nodo de inicio, sin embargo, se encontró el caso de que esta conexión se realizaba por los caminos que no tienen salida o no se encontraba el nodo de inicio. Para aumentar la probabilidad de búsqueda del robot 1 se debe de colocar el nodo de llegada en espacios cerrados como por ejemplo en la posición $P_2(-3, -0.5)$ ya que se conectarán los nodos del único camino.

Por otro lado, la probabilidad de encontrar la ruta del robot 2 es alta, debido a que su mapa es limitado a seguir un solo camino hasta encontrar el nodo de Llegada. Esta etapa del laberinto se diseñó con esa intención a tener un único camino a seguir y analizar la interacción del algoritmo PRM en caminos estrechos con una única probabilidad de éxito.

Tabla 3.4. Resultados de prueba 3 para el algoritmo PRM

	# Iteraciones	Distancia Recorrida [px/m]	Tiempo de simulación [min]
Robot 1	90	15.03	2.44
	120	15.79	4.09
	150	17.76	8.90
Robot 2	50	11.86	0.68
	70	13.08	1.35
	100	13.66	2.55

En la Figura 3.8 se puede apreciar con más detalle los resultados de esta prueba, donde los tiempos de simulación van aumentando acorde al número de nodos e iteraciones. Un caso particular que se encontró también en el algoritmo RRT es que cuando se el número de muestras es mucho mayor a 150 nodos graficados, ambos algoritmos (PRM y RRT) se demorarán más de 7 minutos en buscar una ruta adecuada. En este caso como el algoritmo PRM es mucho más lento en buscar la ruta, alrededor de 9 minutos, tardando un minuto más que el algoritmo RRT. Una similitud que se notó de la Figura 3.6 con la Figura 3.8 son los tiempos de simulación, tomando como ejemplo solo el número de nodos se tiene una variación de tiempo de máximo dos minutos.

Los casos de estudio para cada prueba se pueden visualizar en el ANEXO IV.

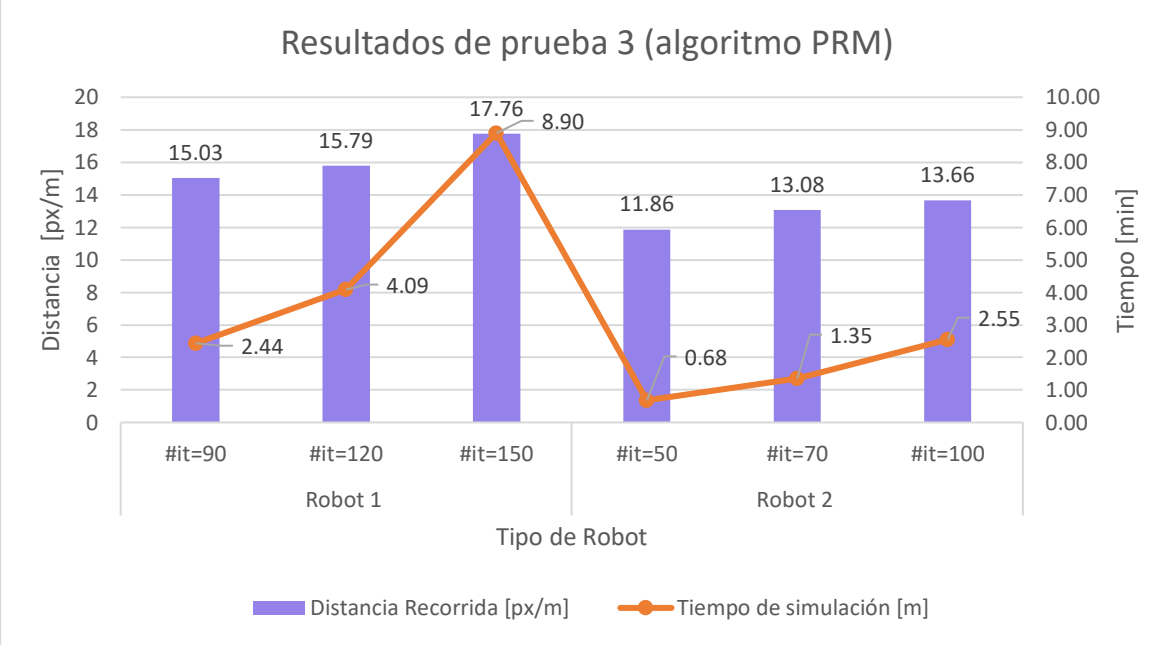


Figura 3.8. Comparación de resultados de la prueba 3 para PRM [Fuente Propia]

3.2 CONCLUSIONES

- Con ayuda de las herramientas de simulación y visualización 3D de ROS, se emuló la dinámica y cinemática de los robots en el entorno tridimensional diseñado en Gazebo con un enfoque realista, gracias a su renderizado y motores de física, en cambio, en Rviz se observaron las gráficas de las rutas y la ruta adecuada para la navegación de cada robot del entorno.
- Se determinó en ROS que los mensajes con las señales de control y las lecturas de los sensores de cada robot tienen interferencias en la comunicación cuando se

ejecutan varios nodos desde un mismo script de python, por lo cual se ejecutó individualmente a cada algoritmo como nodos independientes.

- Se determinó que la aplicación desarrollada para sistemas multi-agentes tiene mejor interacción con los algoritmos si está dentro de un sistema de monitoreo, ya que se centralizaron las pruebas en un menú de usuario desde el terminal de Ubuntu; y se distribuyó a las plataformas de Gazebo y Rviz en una sola pantalla; además, se ejecutaron varios scripts a la vez sin interferir con la comunicación o visualización de los robots.

- Con la variación de los parámetros de los algoritmos RRT y PRM, se determinó que mientras más nodos se grafiquen en el entorno se llevará mayor tiempo de simulación y un aumento en la distancia total recorrida. Esto se debe al algoritmo que busca la ruta adecuada para la navegación; ya que forma la ruta conectando los nodos vecinos, por lo cual también se concluyó que el algoritmo PRM tiene mayor tiempo de simulación que el algoritmo RRT.

- El algoritmo PRM a comparación del algoritmo RRT tiene un bajo índice de probabilidad para encontrar una ruta adecuada, ya que mediante las pruebas realizadas se concluyó que mientas más obstáculos y pocas muestras se encuentren en el entorno más difícil le resulta al algoritmo en encontrar una ruta libre de colisiones.

- De los resultados obtenidos con el ingreso de obstáculos arbitrarios y obstáculos fijos se evidenció que, el algoritmo RRT funciona sin problema con varios obstáculos, pero la ruta adecuada para la navegación del robot tiende a ser quebrada, en cambio, el algoritmo PRM cuando no se tiene muchos obstáculos, tiene una ruta menos quebrada que la del algoritmo RRT.

3.3 RECOMENDACIONES

- Se recomienda que, en trabajos futuros se enfoquen en implementar algoritmos de búsqueda de rutas como los de dijkstra o Astar que se encargan de buscar una ruta adecuada en un entorno lleno de nodos con obstáculos y como resultado se determinará la ruta más corta.

- Se recomienda que cuando se realizan las pruebas en entornos que tengan limitaciones como obstáculos, caminos sin salidas, etc, se deben de tomar varias muestras (nodos) al entorno que estén libres de colisiones para aumentar la probabilidad de búsqueda de una ruta adecuada para la navegación.

- Debido a que el proyecto fue realizado en la última versión de ROS 1 (Noetic Ninjemys) se recomienda formar una fuente bibliográfica de generación de rutas de navegación para la versión de ROS 2, ya disponible actualmente.
- Se recomienda al momento de iniciar algún proyecto robótico en ROS, primero analizar la aplicación que se va a desarrollar, segundo revisar la distribución que se va a emplear de ROS (1 o 2) y finalmente determinar si se cuentan con los paquetes necesarios para llevarse a cabo el proyecto sea con la versión actual o anteriores de la distribución escogida.
- Se recomienda en futuros proyectos, seguir los tutoriales de ROS para la versión actual que sea estable, ya que algunos paquetes o frameworks dejen de usarse en versiones finales y se actualizan con nombres parecidos que cumplen con las mismas funciones.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] F. Basoalto Jimenez, "ANALISIS DE ALGORITMOS PARA LA PLANIFICACIÓN Y SEGUIMIENTO DE TRAYECTORIAS EN ROBOTS AGRICOLAS," Universidad Andrés Bello, Chile, 2017.

[2] H. Montiel, E. Jacinto and F. Martínez, "Generación de Ruta Óptima para Robots Móviles a Partir de Segmentación de Imágenes," *Información Tecnológica,* vol. 26, no. 2, pp. 145-152, 2015.

[3] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, W. Shaokui, H. Deng, Y. Hu and J. Zhang, "Manipulation Task Simulation using ROS and Gazebo," *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014),* pp. 2594-2598, 2014.

[4] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," *Proceedings 01 2004 IEEE!RSJ International Conference on Intelligent Robots and Systems,* vol. 3, pp. 2149-2154, 2004.

[5] Documentation, "Wiki.ros.org," 2022. [Online]. Available: http://wiki.ros.org/Documentation. [Accessed 25 apr 2022].

[6] Tutorials, "gazebosim.org," 2022. [Online]. Available: https://classic.gazebosim.org/tutorials?cat=connect_ros. [Accessed 26 apr 2022].

[7] ROBOTIS, "emanual.robotis.com," 2022. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Accessed 03 May 2022].

[8] Turtlebot, "roscomponents.com," 2022. [Online]. Available: https://www.roscomponents.com/es/robots-moviles/214-turtlebot3-

burger.html#/cursos-no/turtlebot_3_burger_modelo-burger_rpi4_2gb. [Accessed 03 May 2022].

[9] L. J. Arcos Enriquez and C. A. Calala Ayala, "DISEÑO E IMPLEMENTACIÓN DEL CONTROL DE LA FORMACIÓN DE UN GRUPO DE ROBOTS MÓVILES TERRESTRES DE TAMAÑO REDUCIDO ENFOCADO AL SEGUIMIENTO DE TRAYECTORIA BASADO EN EL SISTEMA OPERATIVO ROBÓTICO (ROS)," Escuela Politécnica Nacional, Quito, 2020.

[10] Z. Obdržálek, "Mobile Agents in Multi-Agent UAV/UGV System," *2017 International Conference on Military Technologies (ICMT),* pp. 753-759, 2017.

[11] J. Gómez Roldan, J. de Leon Rivas, P. Garcia Aunon and A. Barrientos, "Una revisión de los sistemas multi-robot: Desafíos actuales para los operadores y nuevos desarrollos de interfaces," *Revista Iberoamericana de Automática e Informática Industrial 17,* pp. 294-305, 2020.

[12] S. Vorapojpisut, M. Lhongpol, R. Amornlikitsin and T. Phuapaiboon, "A Robot Augmented Environment Based on ROS multi-Agent Structure," *2019 4th International Conference on Control, Robotics and Cybernetics (CRC),* pp. 52-56, 2019.

[13] J. Xu, Z. Tian, W. He and Y. Huang, "A Fast Path Planning Algorithm Fusing PRM and P-Bi-RRT," *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan),* pp. 503-508, 2020.

[14] K. Lab, "ompl.kavrakilab.org," Department of Computer Science, [Online]. Available: https://ompl.kavrakilab.org/planners.html. [Accessed 2022 May 27].

[15] E. Díaz, A. Sanchéz, M. Serna, R. Gonzales and B. Bernabe, "Planificación reactiva de movimientos en tiempo real para robots móviles," *Research in Computing Science,* vol. 7, no. 147, pp. 115-128, 2018.

[16] Á. M. Montes Romero, "Planificación de caminos basada en modelo combinando algoritmos de búsqueda en grafo, derivados de RRT y RRT*.," Escuela Técnica Superior de Ingeniería Universidad de Sevilla, España-Sevilla, 2017.

[17] A. Reyes, A. Sanchéz, F. Guevara and A. Toriz, "Planificación de movimientos para robots aéreos no tripulados," *Research in Computing Science,* vol. 7, no. 147, pp. 99-113, 2018.

[18] W. Xinyu, L. XIAOJUA, G. YONG, S. JIADON and W. RUI, "Bidirectional Potential Guided RRT* for Motion Planning," *IEEE Access,* vol. 7, pp. 95046-95057, 2019.

[19] S. Karaman, M. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime Motion Planning using the RRT∗," *2011 IEEE International Conference on Robotics and Automati,* pp. 1478-1483, 2011.

[20] L. Ponce Cevallos, "INTEGRACIÓN DE UN SISTEMA DE CONTROL Y MONITOREO DE UN GRUPO DE ROBOTS MÓVILES TERRESTRES BASADO

EN VISIÓN ARTIFICIAL Y ENFOCADO AL MAPEO Y NAVEGACIÓN DENTRO DE UN ÁREA DE TRABAJO PROVISTA DE OBSTÁCULOS," Escuela Politécnica Nacional, Quito, 2022.

[21] J. Chen, Y. Zhou and Y. Deng, "An Improved Probabilistic Roadmap Algorithm with Potential Field Function for Path Planning of Quadrotor," *Proceedings of the 38th Chinese Control Conference,* pp. 3248-3253, 2019.

[22] Packages, "wiki.ros.org," 04 2019. [Online]. Available: http://wiki.ros.org/Packages. [Accessed 14 Jun 2022].

[23] R. MERINO LÓPEZ, "CREACIÓN DE MODELO URDF DEL ROBOT MANFRED," UNIVERSIDAD CARLOS III DE MADRID, Madrid-España, 2015.

[24] Gazebo, "classic.gazebosim.org," 2014. [Online]. Available: https://classic.gazebosim.org/tutorials?tut=ros_roslaunch. [Accessed 22 06 2022].

[25] R. Shawn, D. Jens and A. Tahir, "Laughter in the Wild: A Study into DoS Vulnerabilities in YAML Libraries," *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering,* pp. 342-349, 2019.

[26] L. García Montañés, "Navegación sin mapa y mapeado en robótica móvil para entornos no estructurados," Escuela Técnica Superior de Ingeniería Universidad de Sevilla, España-Sevilla, 2017.

[27] A. Sharp, K. Kruusamäe, B. Ebersole and M. Pryor, "Semiautonomous Dual-Arm Mobile Manipulator System with Intuitive Supervisory User Interfaces," *2017 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO),* pp. 1-6, 2017.

[28] W. Qian, Z. Xia, J. Xiong, Y. Gan and G. Yangchao, "Manipulation Task Simulation using ROS and Gazebo," *2014 IEEE International Conference on Robotics and Biomimetics,* pp. 2594-2598, 2014.

[29] "scikit-learn.org," 2022. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html. [Accessed 22 07 2022].

5 ANEXOS

ANEXO I. Estructura de ficheros .launch

ANEXO II. Articulaciones y junturas de un robot.

ANEXO III. Presentaciones del mapa en Rviz.

ANEXO IV. Casos de estudio de los algoritmo.

ANEXO V. Diagrama de nodos del trabajo.

ANEXO VI. Repositorio digital y manual de usuario.

ANEXO VII. Link de videos demostrativos.

ANEXO I. Estructura de ficheros .launch

Los ficheros .launch se usan como contenedores de mucho más archivos, como archivos de configuración del entorno (en Gazebo y Rviz), archivos de configuración de robots, etc. La estructura de estos ficheros se basa en atributos y elementos. A continuación, se explicará su funcionamiento.

Atributos

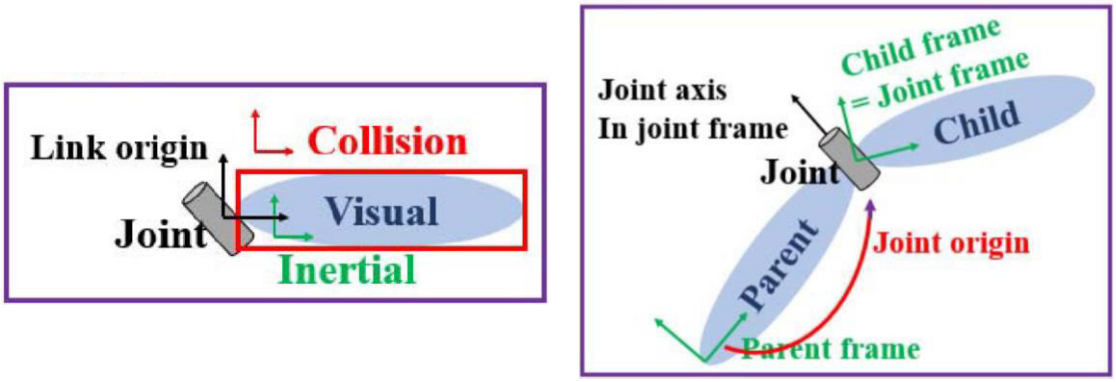
Los atributos se usan básicamente para advertir que el fichero .launch ya no está disponible o está obsoleto.

```
deprecated="deprecation message"
```

Elementos

- **<node>:** Este elemento permite ejecutar un nodo.
- **<param>:** Este elemento es usado para configurar parámetros en el servidor de parámetros (Parameter Server).
- **<remap>:** Este elemento es usado para hacer un renombre o reasignación de nombre declarado dentro del fichero.
- **<machine>**: Este elemento declara una máquina que será usada durante la inicialización del fichero conocido como launching
- **<rosparam>**: Este apartado configura los parámetros dentro de ros permite parar e iniciar archivos de tipo rosparam
- **<include>**: Este item permite incluir varios ficheros .launch en el archivo de ejecución roslaunch que pueden contener configuraciones del entorno o de los robots.
- **<env>**: Es usado para especificar una variable de entorno para todos los nodos que son ejecutados en el fichero.
- **<test>**: Permite ejecutar un nodo de prueba (véase rostest)
- **<arg>**: Declara un argumento
- **<group>**: Agrupa varios elementos como nodos de comunicación compartiendo el mismo namespace o un mismo remap

ANEXO II. Articulaciones y junturas de un robot



(a) Links (b) Joint

II.1. Variables en el archivo URDF

Variables URDF Link

- **Collision:** Establecer la configuración para el recuento de las colisiones en las articulaciones.
- **Visual**: Establecer la visualización de las articulaciones.
- **Origin**: Establece la información de la posición de las articulaciones.
- **Mass**: Configuración del peso de la articulación en Kg.
- **Inertial**: Configuración de la inercia de la articulación.
- **Geometry**: Configura la forma de la articulación (caja, cilindro, modelo).

Variables URDF Joint

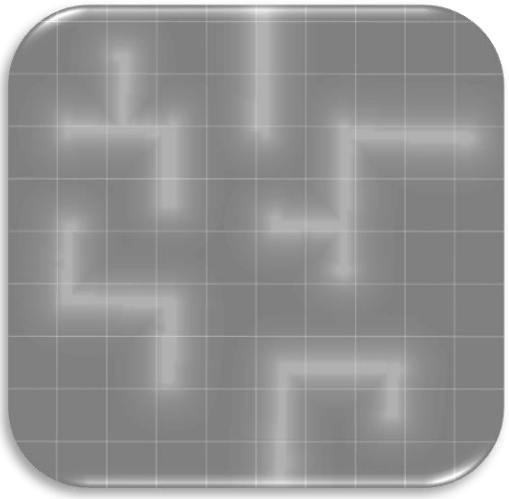
- **Parent**: Enlace principal de las articulaciones.
- **Child**: Enlace secundario de las articulaciones.
- **Origin**: Transforma el sistema de coordenadas de la articulación principal (parent) al sistema de coordenadas de la articulación secundaria (child).
- **Axis**: Configuración del eje de rotación.
- **Limit**: Configuración de la velocidad, la fuerza y el radio de las articulaciones.

ANEXO III. Presentaciones del mapa en Rviz

Las presentaciones del mapa se muestran a continuación, el que se ha escogido es el de la Figura III.1.(a) para realizar las pruebas de los algoritmos.

(a)

Topic:
/move_base/global_costmap/costmap
Color Scheme: costmap

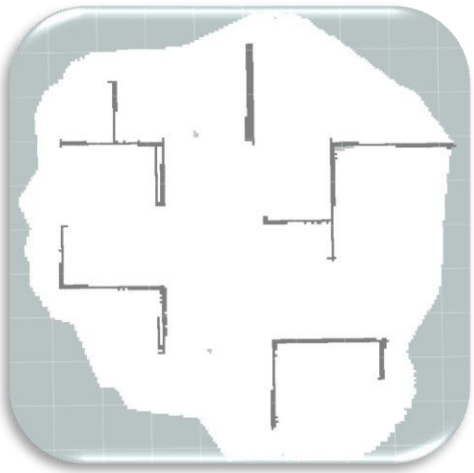


(b)

Topic:
/move_base/global_costmap/costmap
Color Scheme: map

(c)

Topic:
/move_base/global_costmap/costmap
Color Scheme: map



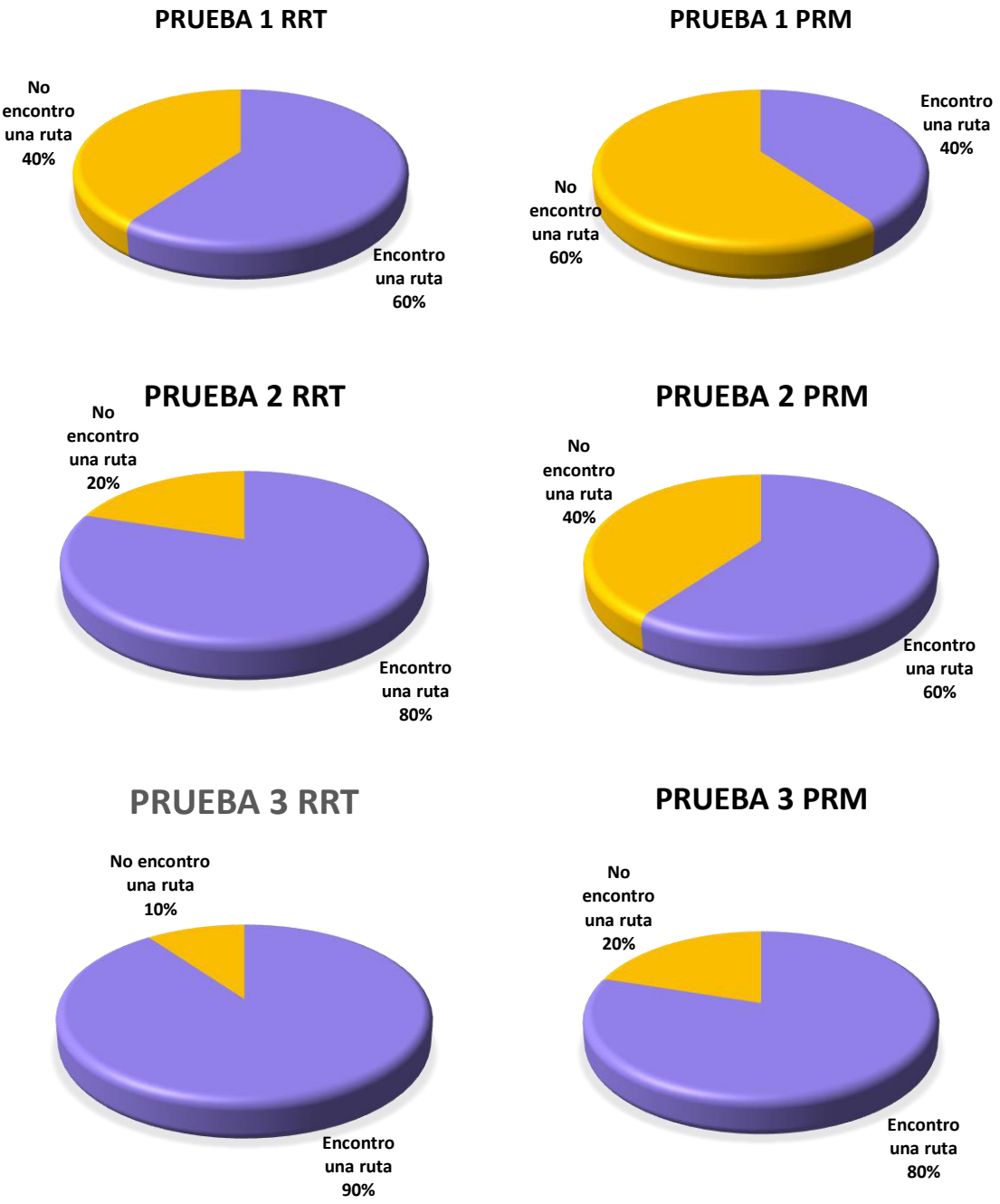
(d)

Topic:
/map
Color Scheme: map

Figura III.1. Presentación del mapa en Rviz

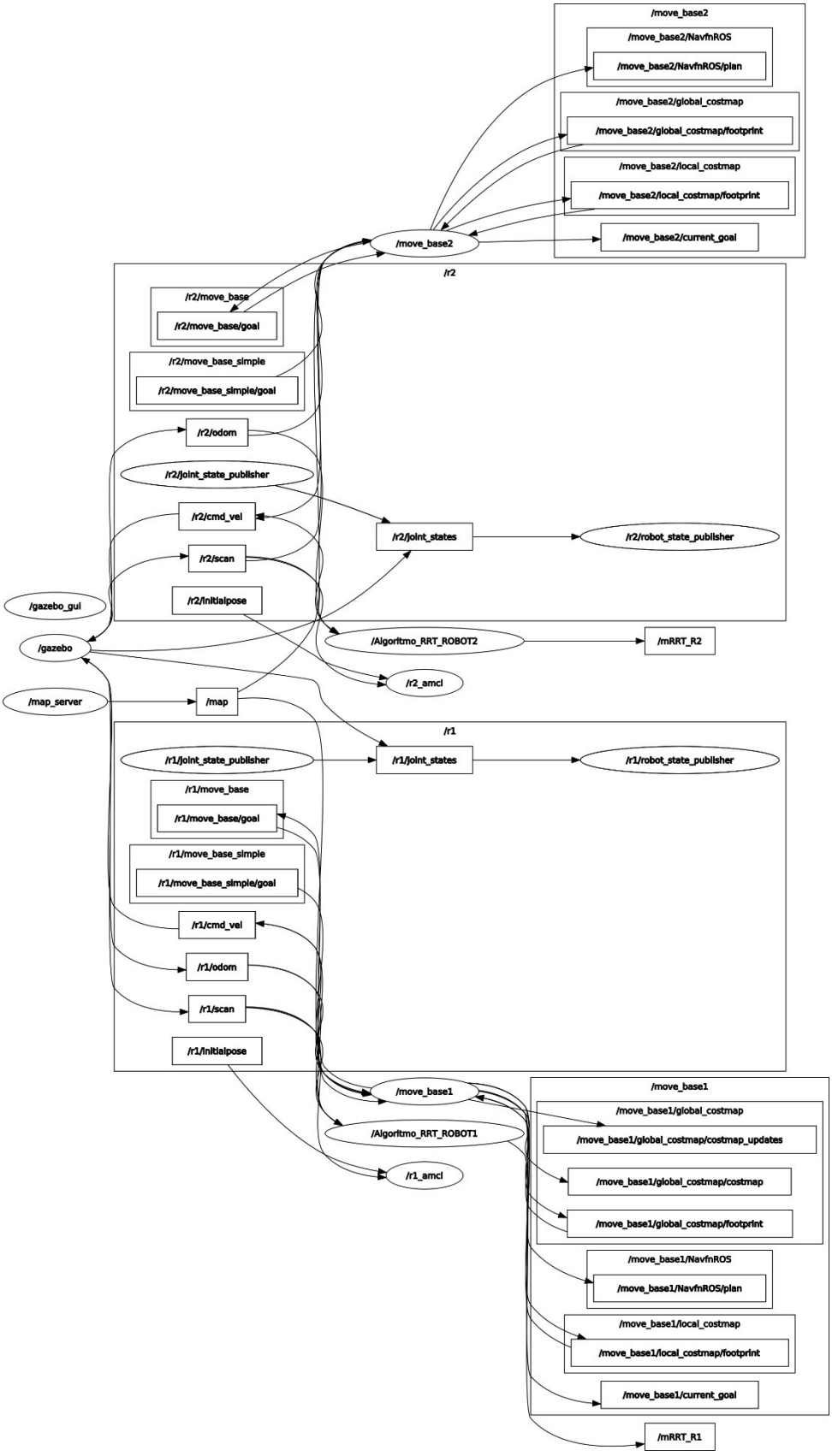
ANEXO IV. Casos de estudio de los algoritmos

Para los casos de estudio se han realizado 10 intentos a cada prueba, de las cuales se ha determinado como caso de éxito si el robot 1 como el robot 2 encuentran una ruta de navegación, caso contrario estaría incompleta la simulación. A continuación, se muestran los resultados de las pruebas realizadas.



IV.1. Caso de estudio para las 3 pruebas realizadas

ANEXO V. Diagrama de nodos del trabajo



V.1. Diagrama de nodos y tópicos.

ANEXO VI. Repositorio digital y Manual de usuario

- **Paso 1**, Se debe dirigir al repositorio digital y se visualizará como en la Figura VI.1. En este repositorio se almacenó todo el desarrollo del producto final demostrable.

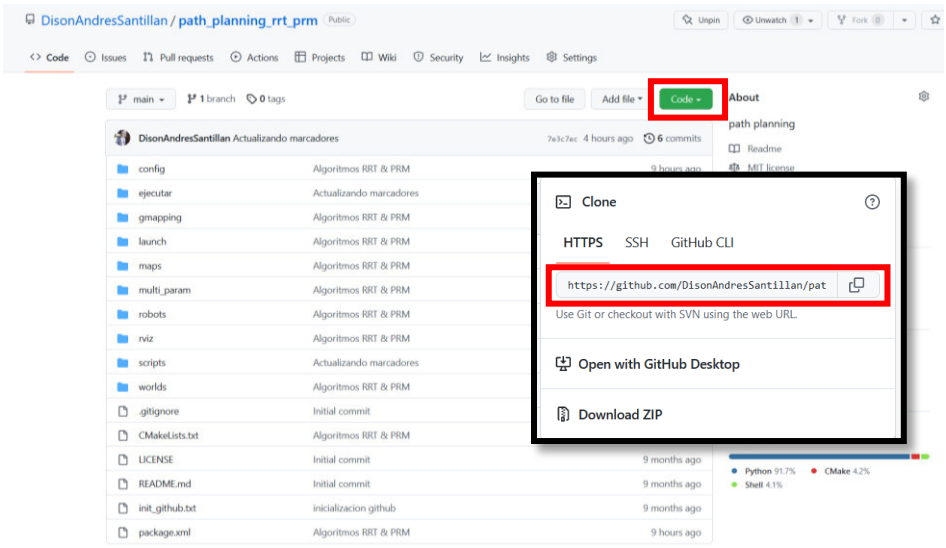


Figura VI.1. Repositorio de github.

Paso 2, se debe dirigir al botón verde de la Figura VI.1 y copiar el link del repositorio. A continuación, se debe abrir el terminal de Ubuntu y dirigirse al path del espacio de trabajo ($catkin_workspace/src) y colocar **git clone** con el link copiado, tal como se muestra en la figura Figura VI.2. Posteriormente se debe dirigir a la carpeta del espacio de trabajo y se digita **$catkin_make** y **$source devel/setup.bash** para inicializar el repositorio en el espacio de trabajo.

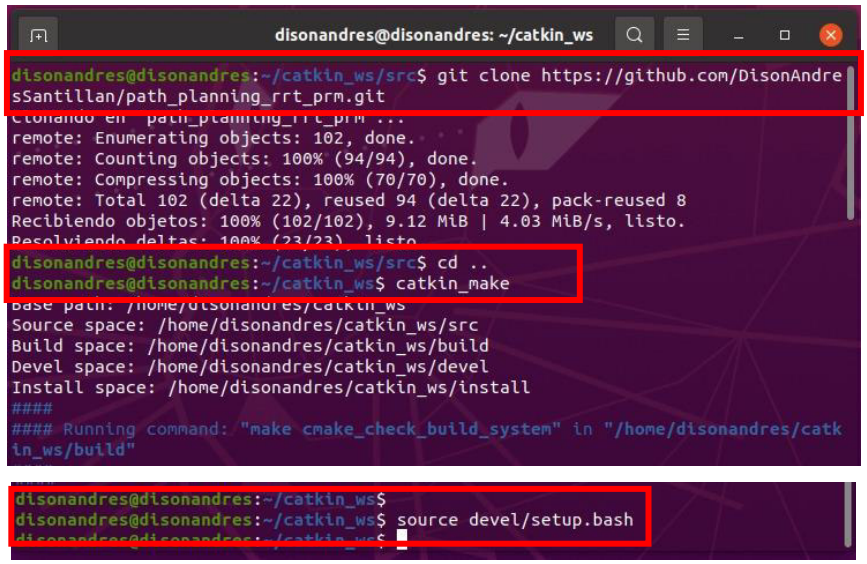
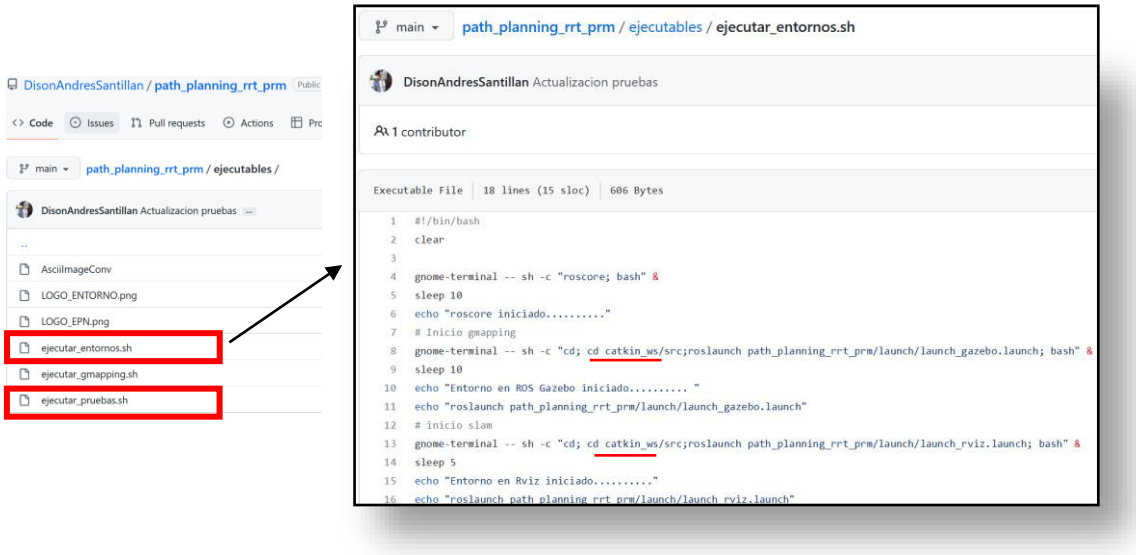


Figura VI.2. Clonación e inicialización del repositorio

Para correr el proyecto se cuenta con dos ejecutables, uno corre el roscore con las plataformas de Gazebo y Rviz; y el otro corre el menú de pruebas. Las consideraciones que se deben tomar son, el espacio de trabajo para el proyecto creado se llama **catkin_ws,** en el caso de que el usuario tenga un nombre diferente se debe cambiar en las líneas subrayadas de la Figura VI.3.



VI.3. Archivos ejecutables .sh

Paso 3, se debe dirigir a la ubicación de los ejecutables y correr el que inicia con el entorno usando el comando **bash**, después se organizan las plataformas de Rviz y Gazebo en la ventana de Ubuntu, tal como se muestra en la Figura VI.4.

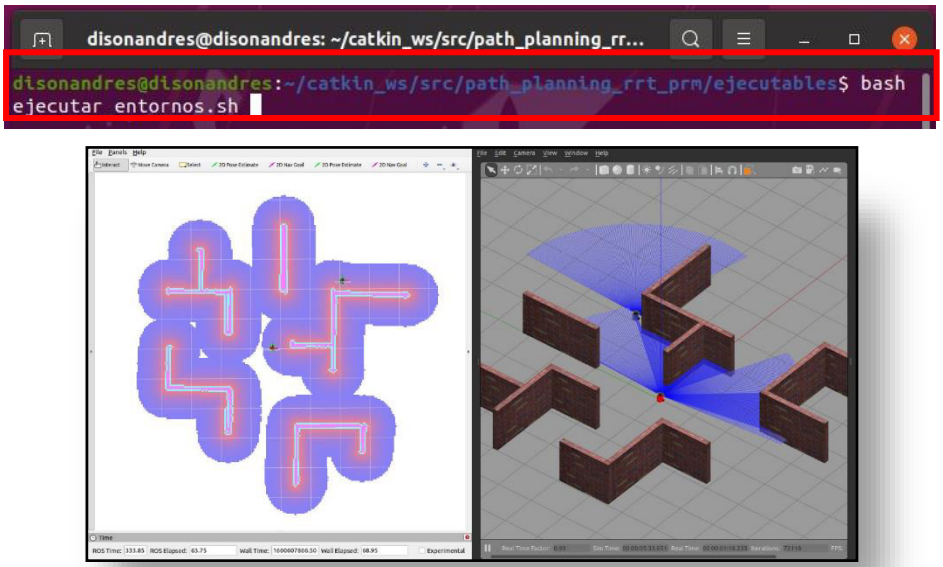


Figura VI.4. Ejecutable 1, inicialización del entorno

Paso 4, para iniciar con las pruebas se corre el segundo ejecutable con el comando **bash** (menuTesis) para desplegar en la terminal de Ubuntu el menú de las pruebas (véase la Figura VI.5.).

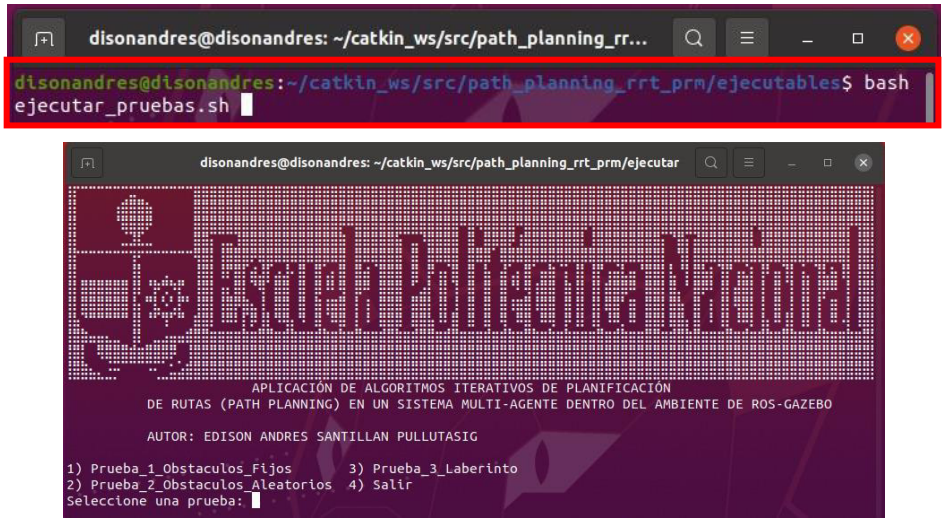


Figura VI.5. Ejecutable 2, menú de pruebas

El ejecutable de las pruebas hace un llamado a los scripts de python con los algoritmos, estos se visualizan en la Figura VI.6. Dentro del proyecto se cuenta con 2 scripts base que son **ruta_RRT.py** y **ruta_PRM.py**, En los scripts de pruebas (**pruebas_PRM** & **pruebas_RRT**) se cargan los parámetros de los algoritmos y se hacen un llamado a los scripts base, que por lo general se configura el número de iteraciones y las posiciones de salida y llegada de los robots móviles.

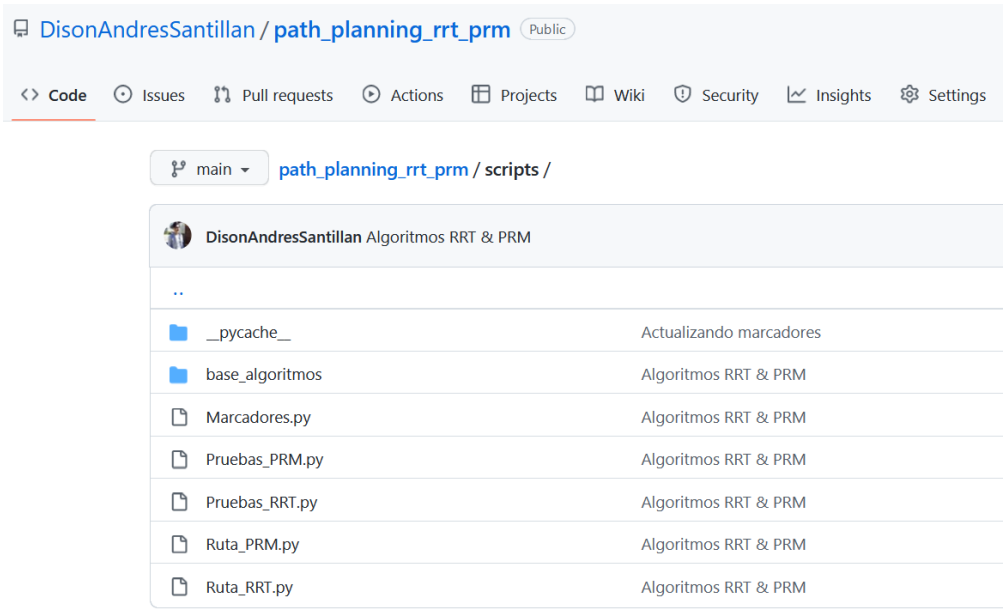


Figura VI.6. Scripts de python

El menú de las pruebas se visualiza a continuación, como un breve resumen, en la prueba 1 se configura la salida y llegada de los robots móviles con obstáculos fijos. En la prueba 2 con ayuda de la función random los obstáculos son arbitrarios. Finalmente, en la prueba 3 se configuran el número de iteraciones y otros parámetros dentro de un entorno laberinto.

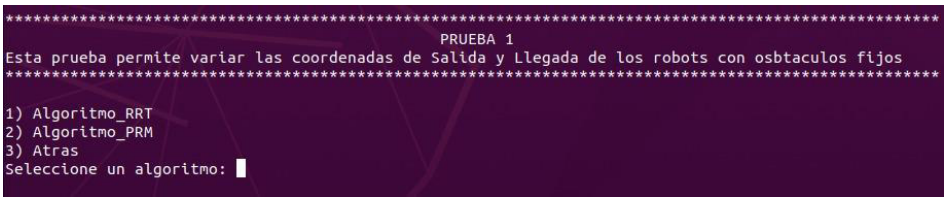


Figura VI.7. Visualización del menú, prueba 1

Figura VI.8. Visualización del menú, prueba 2

Figura VI.9. Visualización del menú, prueba 3

ANEXO VII. Link de videos demostrativos

En el siguiente anexo se tiene enlaces para acceder a las pruebas realizadas a los algoritmos de forma visual en la plataforma de YouTube. El Link del canal es algorithms path planning ros-YouTube y a continuación se hace una pequeña introducción y se adjunta un enlace para cada prueba.

Prueba 1, en los siguientes videos se analiza el comportamiento de los algoritmos con los obstáculos fijos o predefinidos desde gazebo que se visualizan también en Rviz.

- Video 1, prueba 1 - algoritmo PRM
- Video 2, prueba 1 - algoritmo RRT

Prueba 2, en esta prueba se visualiza como se seleccionan obstáculos arbitrarios en la plataforma de Rviz, en cambio en Gazebo se continúa teniendo los obstáculos fijos, por convención de las pruebas y la rapidez del movimiento de los robots móviles al llegar a las coordenadas configuradas por el usuario.

- Video 3, prueba 2- algoritmo RRT
- Video 4, prueba 2- algoritmo PRM

Prueba 3, en esta prueba se hace dentro de un laberinto cambiando los parámetros de los algoritmos como el número de iteraciones entre otros.

- Video 5, prueba 3- algoritmo RRT
- Video 6, prueba 3- algoritmo PRM