ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESTUDIO, CONTROL E IMPLEMENTACIÓN DE SISTEMAS ROBÓTICOS AVANZADOS

CONTROL COOPERATIVO DE UN SISTEMA MULTI-AGENTE PARA LA RESOLUCIÓN DE TAREAS COMPLEMENTARIAS

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

EMERSON JAVIER ALDÁS LÓPEZ

emerson.aldas@epn.edu.ec

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PHD.

patricio.cruz@epn.edu.ec

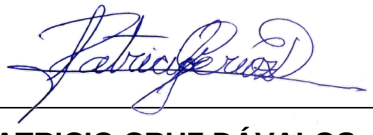
DMQ, agosto 2022

CERTIFICACIONES

Yo, EMERSON JAVIER ALDÁS LÓPEZ 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.

EMERSON JAVIER ALDÁS LÓPEZ

Certifico que el presente trabajo de integración curricular fue desarrollado por EMERSON JAVIER ALDÁS LÓPEZ, bajo mi supervisión.



ING. PATRICIO 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 (los) producto(s) resultante(s) 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.

EMERSON JAVIER ALDÁS LÓPEZ

ING. PATRICIO CRUZ DÁVALOS, PHD.

DEDICATORIA

Para mi padre, quien es la primera persona en creer en mí, todos mis logros son el reflejo de su esfuerzo.

AGRADECIMIENTOS

A mis padres Silvio y Lorena, quienes me apoyan desde siempre y me brindan todo su amor para que pueda cumplir mis sueños.

A mis hermanos, demás familiares y amigos, gracias por guiar mi camino para ayudarme a ser mejor persona y superarme.

A Patricio Cruz, por su atención, ayuda y esfuerzo para el desarrollo del presente proyecto.

ÍNDICE DE CONTENIDO

1 INTRODUCCIÓN 1
1.1 OBJETIVO GENERAL 2
1.2 OBJETIVOS ESPECÍFICOS 2
1.3 ALCANCE 2
1.4 MARCO TEÓRICO 3
1.4.1 SISTEMAS MULTI-AGENTE 3
1.4.1.1 Quadrotor 6
1.4.1.2 TurtleBot 3 7
1.4.2 TAREAS COMPLEMENTARIAS 7
1.4.2.1 Mapeo Autónomo Simultaneo y Localización (SLAM) 7
1.4.2.2 Navegación Autónoma 8
1.4.3 HERRAMIENTAS Y ESCENARIO DE SIMULACIÓN 9
1.4.3.1 ROS 9
1.4.3.2 Gazebo 12
1.4.3.3 RViz 13
2 METODOLOGÍA 14
2.1 PAQUETES Y CONFIGURACIÓN 14
2.1.1 PRECONFIGURACIÓN DE ROS 14
2.1.2 HECTOR QUADROTOR PACKAGE 17
2.1.3 TURTLEBOT 3 PACKAGE 19
2.2 ESCENARIOS DE SIMULACIÓN 22
2.2.1 CREACIÓN DE MUNDOS EN GAZEBO 3D 22
2.2.2 ARRANQUE DEL MUNDO 25
2.3 SLAM 27
2.3.1 APARICIÓN DEL QUADROTOR 27
2.3.2 MAPEO BIDIMENSIONAL 29
2.4 NAVEGACIÓN AUTÓNOMA 30
2.4.1 APARICIÓN DEL TURTLEBOT 3 30
2.4.2 NAVEGACIÓN AUTÓNOMA 32
2.5 INTEGRACIÓN DEL SISTEMA MULTI-AGENTE 35
2.5.1 LANZAMIENTO DEL MAPEO BIDIMENSIONAL Y LOCALIZACIÓN (SLAM) 35
2.5.2 LANZAMIENTO DE LA NAVEGACIÓN AUTÓNOMA 39
3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES 44
3.1 PRUEBAS Y RESULTADOS 44
3.1.1 PRUEBAS Y RESULTADOS BAJO ESCENARIO HOUSE 44

3.1.1.1 Creación de mapa bidimensional a través de SLAM 44
3.1.1.2 Pruebas de tarea complementaria de Navegación Autónoma 47
3.1.2 PRUEBAS Y RESULTADOS BAJO ESCENARIO LABERINTO 50
3.1.2.1 Creación de mapa bidimensional a través de SLAM 50
3.1.2.2 Pruebas de tarea complementaria de Navegación Autónoma 51
3.2 CONCLUSIONES 54
3.3 RECOMENDACIONES 55
4 REFERENCIAS BIBLIOGRÁFICAS 56
5 ANEXOS 58

RESUMEN

La finalidad de este trabajo es realizar la simulación y ejecución de un control cooperativo de un sistema multi-agente heterogéneo para la resolución de tareas complementarias por medio del framework de desarrollo ROS (Robot Operating System), en conjunto con las herramientas de simulación Gazebo y RViz. Las acciones que se realizan son el Mapeo Autónomo Simultaneo y Localización (SLAM) por medio de un agente robótico aéreo, del tipo Quadrotor, y la Navegación Autónoma a través de la plataforma móvil TurtleBot 3. Previo a esto se detallará el proceso para la obtención de las librerías y paquetes necesarios de ROS para una correcta aplicación. Esta aplicación cooperativa es de vital importancia, por ejemplo, cuando sucede algún tipo de accidente o inconveniente al ingresar a un escenario desconocido.

A su vez se estructurarán e integrarán las diferentes acciones realizadas por los agentes robóticos ayudándose de las funcionalidades de ROS, y así obtener dos archivos de lanzamiento en código, uno que ejemplifique la tarea de SLAM y otro para la tarea de Navegación Autónoma. Además, se probará el control cooperativo del sistema multi-agente en dos distintos escenarios creados por medio de Gazebo, logrando verificar y garantizar la Navegación Autónoma por parte del TurtleBot 3, teniendo previamente como referencia el plano bidimensional obtenido por la tarea de SLAM ejecutada por el Quadrotor.

PALABRAS CLAVE: SLAM, Navegación Autónoma, Control Cooperativo, ROS, Gazebo, RViz.

ABSTRACT

The purpose of this work is to perform the simulation and execution of a cooperative control of a heterogeneous multi-agent system for the resolution of complementary tasks through the ROS (Robot Operating System) development framework, in conjunction with the simulation tools Gazebo and RViz. The performed actions are the Simultaneous Autonomous Mapping and Localization (SLAM) by means of an aerial robotic agent, of the Quadrotor type, and the Autonomous Navigation through the mobile platform TurtleBot 3. Prior to this, the process for obtaining the necessary libraries and packages of ROS for a correct application is detailed. This cooperative application is of vital importance, for example, when some kind of accident or inconvenience occurs when entering to an unknown scenario.

At the same time, the different actions performed by the robotic agents are structured and integrated with the help of the ROS functionalities in order to obtain two code release files. One exemplifies the SLAM task, while the other for the Autonomous Navigation task. In addition, the cooperative control of the multi-agent system is tested in two different scenarios created by Gazebo, in order to verify and guarantee the Autonomous Navigation by the TurtleBot 3. This is carried out by, having previously as reference the two-dimensional plane obtained by the SLAM task executed by the Quadrotor.

KEYWORDS: SLAM, Autonomous Navigation, Cooperative Control, ROS, Gazebo, RViz.

1 INTRODUCCIÓN

Existen diferentes escenarios o situaciones de riesgo donde es difícil tener acceso completo a un área específica, esto puede deberse a distintos factores, como pueden ser el desconocimiento del lugar, poco espacio para el ingreso y análisis, daños ambientales o de infraestructura que pueden poner en riesgo principalmente el bienestar humano [1]. Dicho esto, se realiza un control cooperativo de un sistema multi-agente robótico, el mismo que ayuda a la resolución de tareas complementarias ofreciendo ventajas ante los posibles riesgos mencionados.

El efectuar un trabajo en conjunto tiene diferentes beneficios en cuanto a tiempo de ejecución y rendimiento, buscando siempre cumplir los objetivos finales propuestos de una manera más factible. Además de desarrollar un trabajo con menor dificultad al tener la asistencia de varios participantes, un sistema multi-agente puede realizar tareas de manera secuencial o paralela, dependiendo de la aplicación y alcance de los agentes [2].

En el presente proyecto se realiza la ejecución y simulación de dos robots móviles heterogéneos con diferentes características, uno terrestre UGV (Unmanned Ground Vehicle) y otro aéreo UAV (Unmanned Aerial Vehicle), los mismos que efectúan diferentes acciones complementarias en un escenario desconocido. Las tareas consideradas son el Mapeo Simultaneo y Localización (SLAM) y la Navegación Autónoma, las cuales son ejecutadas secuencialmente buscando obtener provecho y ventaja de un control cooperativo [3] [4].

Como primera acción, la plataforma robótica aérea Quadrotor realiza el reconocimiento del escenario a través del Mapeo Simultaneo y Localización (SLAM), siendo esta acción esencial para la ejecución y despliegue del sistema multi-agente. Un reconocimiento previo ayuda de manera general en el desarrollo del proyecto, ya que se tiene una noción global del ambiente de trabajo y posibles problemas a enfrentar cuando se prosiga a ejecutar una tarea cooperativa. Una vez realizado el mapeo bidimensional, el robot terrestre TurtleBot 3 procede a efectuar la Navegación Autónoma tomando como referencia el plano en 2D proporcionado por la plataforma aérea, y así poner a prueba la navegación por medio de la elección de diferentes puntos destinos, analizando las rutas que toma la plataforma robótica móvil y el cumplimiento del objetivo final, que es llegar al sitio de manera autónoma.

Al ser tareas realizadas independientemente por cada agente robótico, se desarrolla, diseña y ejecuta un modelo de paquete compacto a través de ROS (Robot Operating System) [4], misma aplicación que es visualizada por medio del simulador tridimensional

Gazebo en conjunto con RViz [6]. A su vez se integran las acciones complementarias para obtener un componente colaborativo entre el grupo de robots heterogéneos de diferentes características. Adicionalmente las labores complementarias del presente proyecto pueden ser utilizadas en aplicaciones donde se busque conocer medios no explorados o desconocidos, guiado autónomo en situaciones de emergencia, asistente para personas invidentes, o a su vez en alguna aplicación donde se requiere recopilar información de un escenario para un próximo tratamiento de datos y emplearlos en diferentes ámbitos.

1.1 OBJETIVO GENERAL

Diseñar y simular un control cooperativo de un sistema multi-agente para la resolución de tareas complementarias enfocado al mapeo y navegación.

1.2 OBJETIVOS ESPECÍFICOS

1. Investigar las características de sistemas multi-agentes heterogéneos, la aplicación de tareas cooperativas de mapeo y localización simultánea (SLAM), así como la navegación autónoma, además del uso de ROS como ambiente de desarrollo y diseño, y Gazebo en conjunto con RViz como entorno de simulación.
2. Desarrollar una escena en ROS-Gazebo para un sistema multi-agente heterogéneo conformado por una plataforma robótica aérea Quadrotor y una terrestre TurtleBot 3.
3. Diseñar e implementar la tarea de mapeo bidimensional por parte de un Quadrotor, y la tarea de navegación autónoma por parte de un TurtleBot 3, empleando Gazebo como entorno de simulación y RViz como herramienta de visualización.
4. Integrar las funciones del Quadrotor y del TurtleBot 3 para poner a prueba su funcionamiento como un sistema cooperativo ante diferentes escenarios para un posterior análisis de resultados.

1.3 ALCANCE

- Se presenta una revisión bibliográfica sobre el framework de desarrollo ROS, comprendiendo todo lo que el software ofrece, tanto como sus ventajas y sus diferentes funcionalidades.
- Se realiza un estudio y análisis del entorno de simulación ROS-Gazebo y RViz, se ejecuta la simulación de diferentes paquetes ya creados, ejemplificando y efectuando algunas funciones que son compactadas para la realización del proyecto.

- Se efectúa un estudio relacionado al sistema multi agente a ser simulado, en este caso es de dos plataformas robóticas heterogéneas con diferentes cualidades, un terrestre UGV y otro aéreo UAV.
- Se ejecutan tareas de manera coordinada y cooperativa entre el grupo de robots autónomos a partir de pruebas de simulación independientes para cada uno, buscando agrupar las diferentes herramientas y archivos en ROS para su próximo tratamiento, modificación y ejecución.
- Se ejemplifican diferentes funcionalidades en cuanto al método de mapeo simultáneo y localización, previsto para el Quadrotor a través de ROS-Gazebo, de manera que recepta información del entorno para el respectivo guardado del plano bidimensional.
- Se ejemplifica la recopilación de información proporcionada por el mapeo bidimensional, la misma que es utilizada por el robot terrestre TurtleBot 3, para la realización de una navegación autónoma por medio de sus controladores en el mapa adquirido.
- Se desarrolla un modelo de paquete que contiene archivos que efectúa y compacta el entorno de simulación, ejecuta a la par las plataformas robóticas y realización de las tareas complementarias.
- Una vez desarrollado el modelo de paquete en el espacio de trabajo en ROS, se procede a la ejecución de este por medio del entorno de simulación ROS-Gazebo y RViz, ejemplificando y verificando su funcionalidad.
- Finalmente se definen pruebas en al menos dos mapas diferentes, aplicando el paquete desarrollado que tiene como ejecutables archivos tipo launch, para la realización de tareas cooperativas entre el grupo de plataformas robóticas, y concluir a través de su comparación y operatividad, sus beneficios.

1.4 MARCO TEÓRICO

1.4.1 SISTEMAS MULTI-AGENTE

Uno de los temas de interés y aplicación de los últimos años ha sido el trabajo cooperativo entre sistemas multi-agente, donde la mayoría de los agentes o robots móviles tienen las mismas características o cualidades. Pero al hablar de un sistema multi-agente heterogéneo se explora un ente colaborativo con un campo de opciones más amplio, ya que se tiene más funcionalidades que ofrecen los distintos miembros robóticos del sistema [1]. La finalidad de realizar un trabajo colaborativo entre plataformas robóticas con diferentes características es obtener un mayor beneficio y ventajas, esto al considerar parámetros como el tiempo de ejecución y rendimiento de la consecución de tareas. Esto

se dificulta al efectuarse con un sistema homogéneo ya que disponen de menos opciones al tener robots de un solo tipo o modelo [3]. Adicionalmente, existe la opción de adaptar cada acción o tarea al agente del grupo robótico heterogéneo más conveniente para su correcto desarrollo, sin desviarse del objetivo final.

Al hablar de un agente robótico como elemento de un evento, se debe considerar sus aspectos para ser tomado en cuenta como parte del control de un trabajo cooperativo. Algunas de las características que debería tener un grupo de plataformas robóticas, las cuales actúen como agentes dentro de un sistema son las siguientes:

- Organización entre los robots implicados en el trabajo cooperativo para la correcta ejecución de las tareas principales, se necesita mantener un orden según el papel que desempeña cada uno de ellos dependiendo de las necesidades para un mejor desenvolvimiento dentro de un escenario o mapa.
- Cooperación para el desarrollo y objetivo del trabajo final dependiendo del resultado de las diferentes tareas efectuadas por los agentes, por lo cual dichos resultados intermedios representan un aporte importante para los otros participantes.
- Coordinación para tener un sistema multi-agente en conjunto y así lograr un comportamiento unido y colaborador, el mismo que se consigue por la comunicación entre los robots autónomos para la ejecución de cada una de las tareas.
- El control para realizar de manera autónoma una acción por medio de una marcha del sistema.
- La comunicación con la finalidad de realizar un contacto entre los agentes de un sistema robótico para alcanzar el objetivo final propuesto por el proyecto [2], [3].

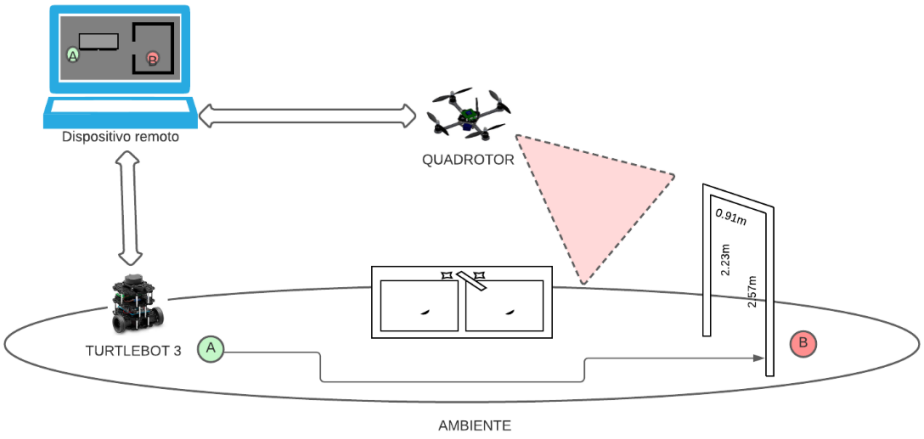


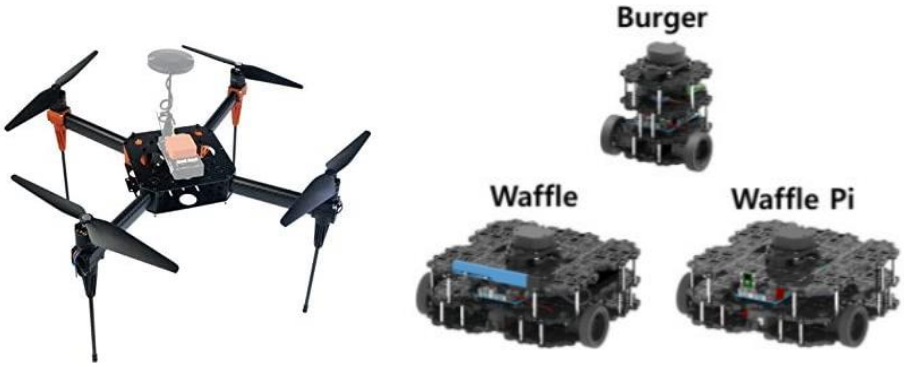
Figura 1.1 Representación de sistema multi-agente [Fuente propia]

El contexto de realizar un sistema multi-agente robótico es tomar en consideración todas las tareas o acciones que el grupo de participantes ejecuta, ya sea a la par o de manera ordenada para una meta o fin en común. En la Figura 1.1 se observa la esquematización de un sistema robótico colaborativo, donde una plataforma aérea un Quadrotor realiza la tarea de SLAM y una plataforma móvil TurtleBot 3 efectúa la Navegación Autónoma, las mismas que serán tomadas en cuenta para el desarrollo del presente proyecto y serán detalladas a lo largo del mismo.

Una de las tareas de mayor énfasis en la actualidad es el mapeo de espacios o entornos no explorados, debido a la complejidad de estos y el difícil acceso a escenarios donde no se tiene una información o conocimiento válido para algún tipo de prueba [3]. El desconocimiento de un entorno representa un grave problema, ya que ante diversas acciones principales que se deseen realizar, es necesario conocerlo con anterioridad. Algunas de las tareas que se podrían realizar a partir del mapeo de un espacio son la localización y percepción, navegación autónoma, trazo de rutas, navegación deliberativa, entre otras [3]. Las tareas mencionadas necesitan conocer el escenario o espacio donde se va a ejecutar la tarea principal, por lo que se requiere un modelo de plano fiable y claro para su próxima recopilación y tratamiento de información.

El mapeo de un entorno es una acción que poco a poco se ha venido implementando como una habilidad exigida a los robots autónomos móviles, esto con la finalidad de generar un plano o mapa bidimensional, siendo este una representación del entorno o ambiente en análisis [2]. Dicho mapeo simultáneo y de localización se le conoce como SLAM (Simultaneous Localization and Mapping), dando una solución al desconocimiento de entornos no explorados y generando una referencia para un tratamiento del plano y efectuando nuevas tareas colaborativas.

Como se indicó anteriormente dichas tareas van a ser realizadas por distintos agentes robóticos, en este caso son un robot aéreo Quadrotor y un robot móvil, los cuales tienen diferentes características tanto físicas como de software, de manera que las acciones se acoplan al sistema en general y que se presentan a continuación.



(a) (b)

Figura 1.2 (a) Estructura general de un Quadrotor [5]. (b) Familia TurtleBot 3 [4].

1.4.1.1 Quadrotor

El Quadrotor es una plataforma robótica aérea UAV (Unmanned Air Vehicle) compuesta principalmente por cuatro rotores alineados simétrica y equidistantemente alrededor de su eje de rotación, formando una cruz. En la Figura 1.2 (a) se puede apreciar la estructura general de un Quadrotor, donde sus cuatro rotores alineados simétricamente son su principal característica

Los cuadricópteros ofrecen mayor flexibilidad en cuanto al vuelo y despegue, ya que su sistema de hélices logra mantener al robot aéreo de manera controlada a través de maniobras en una sola trayectoria [5]. A su vez, presenta algo de inestabilidad y difícil operabilidad por el balance y manejo entre sus rotores. Los Quadrotores presentan ventajas que los hacen apetecibles dentro del mercado, siendo los más utilizados en implementaciones robóticas dependiendo de la aplicación en análisis, algunas de ellas son:

- Proporción de una mayor maniobrabilidad gracias a su sistema de cuatro hélices situadas de manera simétrica alrededor del centro del robot.
- Movimiento más controlado por la variación de velocidad de sus rotores, los cuales permiten aumentar o disminuir su potencia para controlar su despegue y aterrizaje.
- Al poseer un sistema de cuatro rotores tiene una mayor capacidad de carga y empuje [5].

Así como los Quadrotores tienen ventajas significativas, también tienen desventajas a considerar, como es el peso y un mayor consumo energético a comparación de otras aeronaves. El peso de sus hélices representa una desventaja, pues la cantidad de elementos que las conforman dificultan su vuelo y trayectoria.

1.4.1.2 TurtleBot 3

TurtleBot 3 es una plataforma robótica terrestre UGV (Unmanned Ground Vehicle), creada en Willow Garage (en colaboración con ROS) por Melonee Wise y Tully Foote en el mes de noviembre del 2010, utilizada frecuentemente en proyectos de investigación enfocados a tareas de diferente índole, con la ventaja de ser de software libre [4]. Dicho prototipo es de fácil implementación y uno de los más utilizados en aplicaciones de la robótica sin necesidad de tener un gran conocimiento sobre robots autónomos. La familia TurtleBot 3 se puede visualizar en la Figura 1.2 (b), la cual cuenta actualmente con tres modelos en el mercado, Burger, Waffle y Waffle Pi. Además, cada uno tiene una documentación detallada y fácil de acceder para cualquier conocedor y amante de la robótica para casos de implementación, ya sea por simulación o en campo.

ROS mantiene una relación profunda al hacer uso de este modelo, ya que es una plataforma relativamente sencilla de implementar que se acopla a aplicaciones realizadas por el framework de código abierto. Por dicha razón ROS viene precargado, integrado y ante toda prueba en el robot tipo tortuga, ofreciendo diferentes funcionalidades y herramientas para aplicaciones automáticas [3]. Adicionalmente, las principales aplicaciones que relacionan al prototipo con el entorno de desarrollo ROS son la ejecución de trayectorias, evasión de obstáculos, navegación autónoma, el mapeo de un lugar y la generación de mapas a través del reconocimiento del ambiente a partir de un sensor acoplado a la plataforma robótica [4]

Una vez dadas las características principales de las plataformas robóticas a emplear, se presenta a continuación las acciones complementarias que se desarrollan ejecutan y diseñan en el presente proyecto que son el SLAM y la Navegación Autónoma.

1.4.2 TAREAS COMPLEMENTARIAS

1.4.2.1 Mapeo Autónomo Simultaneo y Localización (SLAM)

La localización y mapeo simultaneo (más conocido por sus siglas en inglés como SLAM) es un método por el cual un robot móvil, en este caso una plataforma aérea Quadrotor, realiza la construcción de un plano bidimensional de un medio que en un principio se desconoce, a partir de los datos recibidos por sus diferentes sensores [8].

A través del entorno de simulación ROS-Gazebo y la herramienta de visualización RViz, se puede realizar la técnica SLAM para la construcción de un mapa de un ambiente inexplorado, de manera que cuando el agente robótico avance a través de una trayectoria genere el plano 2D con su sensor láser incorporado, tal como se muestra en la Figura 1.3.

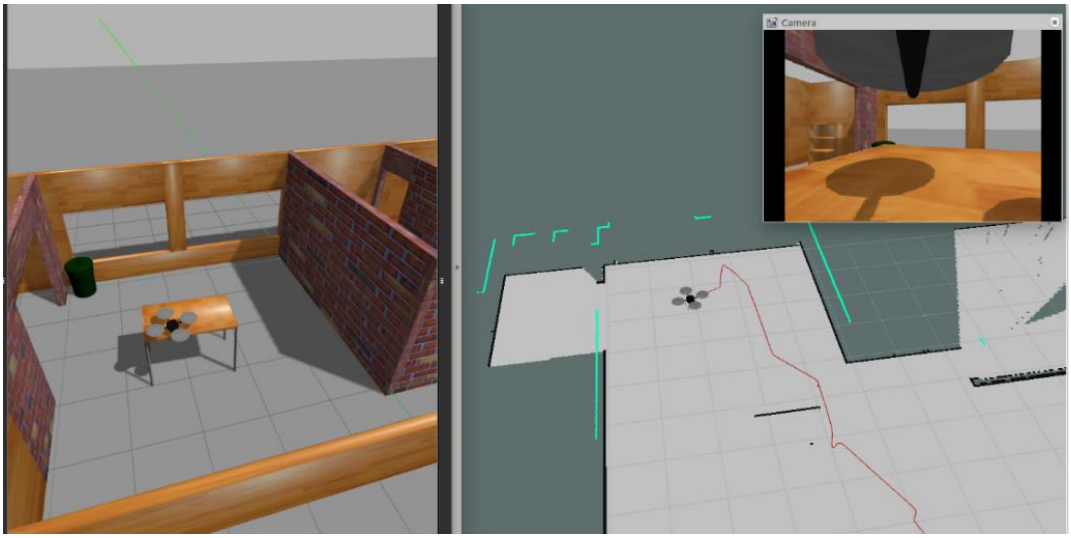


Figura 1.3 Técnica SLAM por Quadrotor. [Fuente propia]

La técnica de SLAM sirve para diversas aplicaciones en las cuales se es necesario conocer previamente el ambiente de trabajo para una ejecución de tareas complementarias, como pueden ser:

- Exploración de zonas desconocidas o de difícil acceso.
- Rescate en catástrofes ambientales o de riesgo para el ser humano.
- Navegación autónoma para guiado de personas invidentes [6].

Siendo esta última, la tarea complementaria de énfasis en el presente proyecto.

1.4.2.2 Navegación Autónoma

La navegación autónoma se define como la acción de dirigir una plataforma robótica móvil, en este caso al TurtleBot 3, desde un punto inicial hacia un punto final fijado como destino en el mapa. Cabe recalcar que para realizar la navegación autónoma es necesario que con anterioridad se conozca el entorno, el mismo que es proporcionado por el plano bidimensional a través del SLAM.

Hoy en día la navegación autónoma es una tarea que es exigida y presionada en general para los robots móviles, ya que dicha acción ofrece una optimización y un amplio campo de aplicaciones dependiendo de una necesidad en particular. La capacidad para que un robot pueda movilizarse por un medio ya conocido hasta su objetivo a través de la medida de sus sensores, se ve analizada desde su rendimiento y fiabilidad para alcanzar la meta [7], [8].

En la Figura 1.4 se evidencia como a través de la herramienta de visualización RViz se puede estimar la posición inicial del agente robótico, para próximamente darle un punto destino, siempre y cuando el robot móvil encuadre de manera correcta sus sensores en el plano proporcionado por el SLAM.

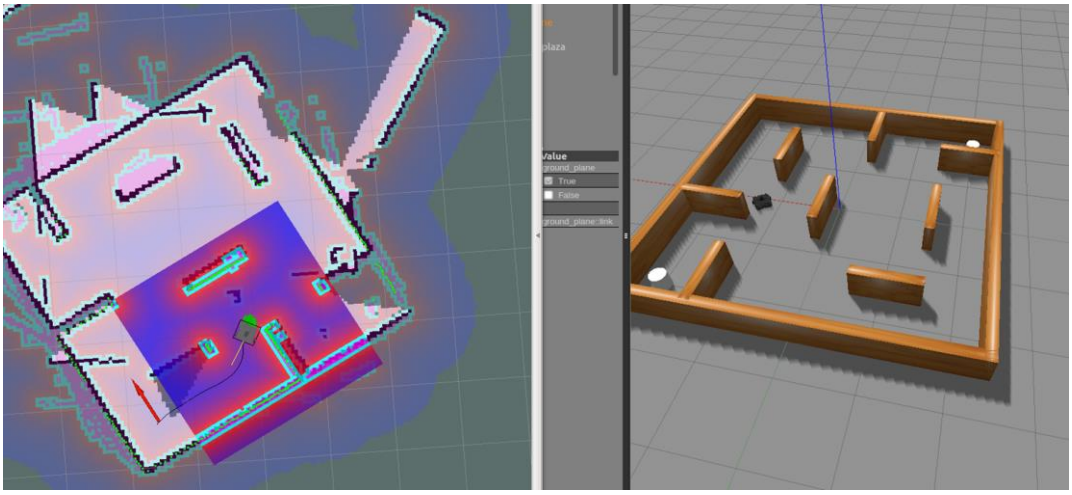


Figura 1.4 Navegación Autónoma por TurtleBot 3 [Fuente propia]

Existen diferentes y potentes herramientas para la simulación de las aplicaciones mencionadas, mejorando así su ejecución para entender el desarrollo de estas. ROS en conjunto con Gazebo y Rviz son opciones nuevas y llamativas en aplicaciones robóticas, llevando a otro nivel el avance tecnológico para el servicio del hombre.

1.4.3 HERRAMIENTAS Y ESCENARIO DE SIMULACIÓN

1.4.3.1 ROS

El software ROS, denominado así por en sus siglas en inglés, Robot Operating System; es un framework de código abierto utilizado ampliamente para el desarrollo de implementaciones robóticas. ROS ofrece una gran cantidad de librerías, paquetes y herramientas para el diseño, modificación y ejecución de diferentes aplicaciones [10]. La funcionalidad en ROS parte del procesamiento de sus llamados 'nodos', los cuales pueden enviar, recibir y modificar parámetros como mediciones de sensores, controladores, actuadores, entre otras cualidades del agente robótico usado.

Una característica que destaca de este framework de código abierto es el reciclaje de archivos y de investigación para futuras aplicaciones, las cuales toman como referencia los paquetes ya verificados y desarrollados por la comunidad de ROS. De esta manera se puede acoplar diferentes acciones para la distribución de procesos a través de la ejecución de paquetes previamente diseñados.

Arquitectura

La arquitectura de ROS se basa en el procesamiento de grafos, los cuales son ejecutados de manera simultánea y paralela, su puesta en marcha parte del sistema cliente – servidor. Una red de grafos es una manera versátil de comprender las cualidades de ROS y como se interconectan sus diferentes elementos para distintas funcionalidades [10].

Los elementos principales que conforman una red de grafos para la ejecución del desarrollo de aplicaciones en ROS se detallan a continuación:

- **Nodos:** Una manera fácil de entender que es un nodo, es relacionarlo con la ejecución de una acción dentro de una aplicación, al igual que en implementaciones robóticas, un nodo puede representar el control de un láser, el sensor de un agente, el control de los motores de las ruedas de un robot, una tarea de localización, una vista gráfica de una aplicación, entre otros. ROS trabaja de manera articular y paralela, esto significa que puede tener diferentes nodos ejecutando varias tareas simultáneamente.
- **ROS master:** El nodo maestro sirve para la interactuación de los diferentes nodos en ejecución, facilita la búsqueda en la red o articulación de grafos para un intercambio de mensajes o servicios.
- **Servidor de parámetros:** Este elemento forma parte del nodo maestro y permite el guardado de datos identificados en clave o código.
- **Mensajes:** La comunicación entre los nodos que se encuentran a través del nodo maestro se realizan a través de estructuras, arreglos o también llamados mensajes, los cuales pueden ser de diferente tipo, como enteros, float, booleanos, double, entre otros.
- **Tópico:** También llamados temas, se definen como una ruta por la cual los nodos pueden actuar como suscriptores o publicadores, y se usan para identificar el mensaje en proceso. Es decir, que un nodo puede ser suscriptor o publicador de varios tópicos de manera unidireccional, y al igual un tópico puede tener diferentes suscriptores y publicadores sin que los nodos que intervienen detecten la existencia entre ellos.
- **Servicios:** Son una manera de proporcionar apoyo a los nodos, pues se encargan de atender una solicitud solicitada por un nodo o a su vez enviar una petición a la

espera de una respuesta. Tienen una estructura de mensajes siguiendo el sistema cliente – servidor.

- **Bags:** Sirven para el almacenamiento de datos de ROS, como la información de sensores recopilados por un robot. Dicha información puede ser utilizada en otros nodos para su modificación y tratamiento de datos [8], [10].

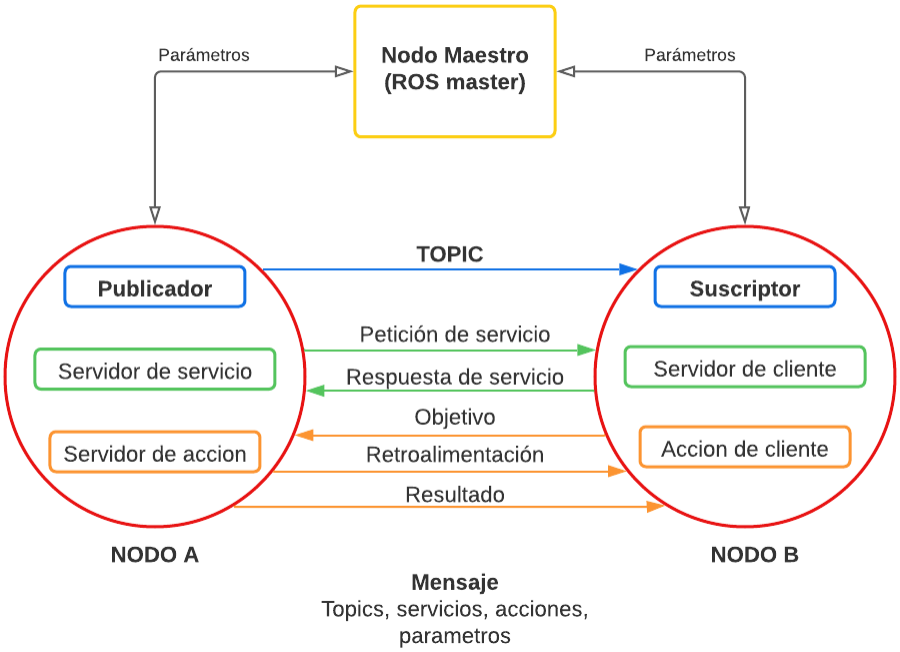


Figura 1.5 Arquitectura de comunicación ROS [Basado en 10]

En la Figura 1.5 se puede visualizar como se relacionan los nodos y su comunicación, para este caso solamente se representa la conexión entre dos nodos. El ROS máster define parámetros, los cuales establecen cualidades a los nodos asociados, los mismos que pueden recibir datos entre sí, además de facilitar su búsqueda. El nodo A actúa como publicador y el nodo B como suscriptor, a través de su tópico o tema de manera unidireccional. Al igual se puede solicitar una petición de servicio a la espera de una respuesta o alguna acción teniendo en cuenta su retroalimentación. El protocolo de comunicación en ROS es TCPROS, por lo que usa principios TCP/IP estándar [10].

Para una mejor visualización de cómo funciona la arquitectura y comunicación en ROS, se tiene un complemento GUI o interfaz de usuario donde se observa una representación gráfica de la red de grafos y los elementos activos durante la ejecución de alguna aplicación. El nodo que lanza la interfaz gráfica se ejecuta por medio del terminal desde el siguiente comando.

```
$ rosrun rqt_graph rqt_graph
```

Ventajas

- La estandarización de ROS ofrece ahorro de tiempo y una búsqueda más sencilla para el desarrollador en el uso de diferentes librerías, paquetes o archivos que a su vez pueden ser reciclados o un punto de partida para diversas aplicaciones.
- Existen paquetes ya desarrollados para diferentes plataformas robóticas, los que se pueden obtener fácilmente y navegar a través de su red de grafos para un mayor entendimiento del manejo y ejecución de ROS.
- Liviano en general, además de flexibilidad en cuanto lenguajes de programación.

1.4.3.2 Gazebo

Gazebo es una plataforma de simulación 3D, la cual se caracteriza por su funcionalidad en cinemática y dinámica, tomando en cuenta parámetros físicos que hacen una simulación más realista en diversos entornos y ambientes. El software se enfoca en la implementación de plataformas robóticas, ya que es capaz de ejecutar paquetes y nodos relacionados con los sensores, actuadores, diseño de robots, y demás herramientas relacionadas a un sistema robótico en general [9].

Gazebo se enfoca directamente con ROS desde su creación, ya que fue desarrollado específicamente para aplicaciones en dicho framework. Existen paquetes (*gazebo_ros_pkgs*) que están implementados directamente en ROS, esto permite la interacción entre los objetos de una escena en Gazebo y la interfaz de ROS [10]. La ejecución en conjunto de los dos softwares da como resultado una potente herramienta para aplicaciones robóticas, misma que se ejecuta a través de la arquitectura de ROS y el entorno de simulación Gazebo 3D.

Considerando lo representado en la Figura 1.5. Gazebo es representado como un nodo, que puede ser ejecutado a través de un archivo tipo launch; y al ser un nodo, puede actuar como un suscriptor o publicador de cualquier tópico del modelo de la aplicación.

Además, este simulador puede ser una herramienta de diseño y creación de mundos a través de Building Editor según un análisis y semejanza requerido, el mismo proporciona diferentes modelos de elementos básicos para construir un entorno o escenario de simulación según la aplicación en análisis, tal como se muestra en la Figura 1.6.

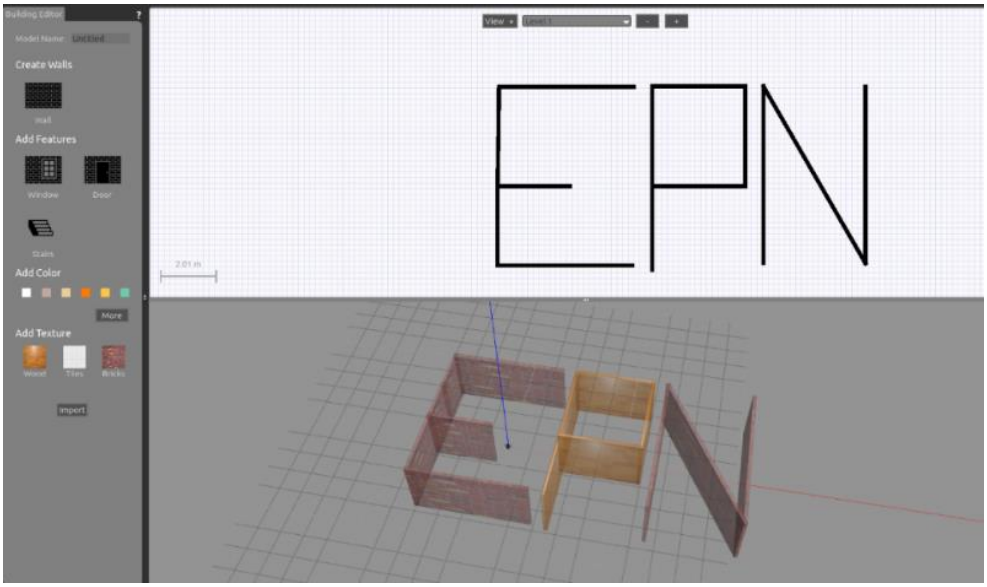


Figura 1.6 Creación de un mundo en Gazebo 3D. [Fuente propia]

1.4.3.3 RViz

RViz es una herramienta de visualización tridimensional en ROS, misma que ofrece una vista del agente robótico en implementación. Esta herramienta proporciona la adquisición de datos de diferentes sensores correspondientes al robot, y a su vez puede reproducir la información procesada, tal como se indica en la Figura 1.7.

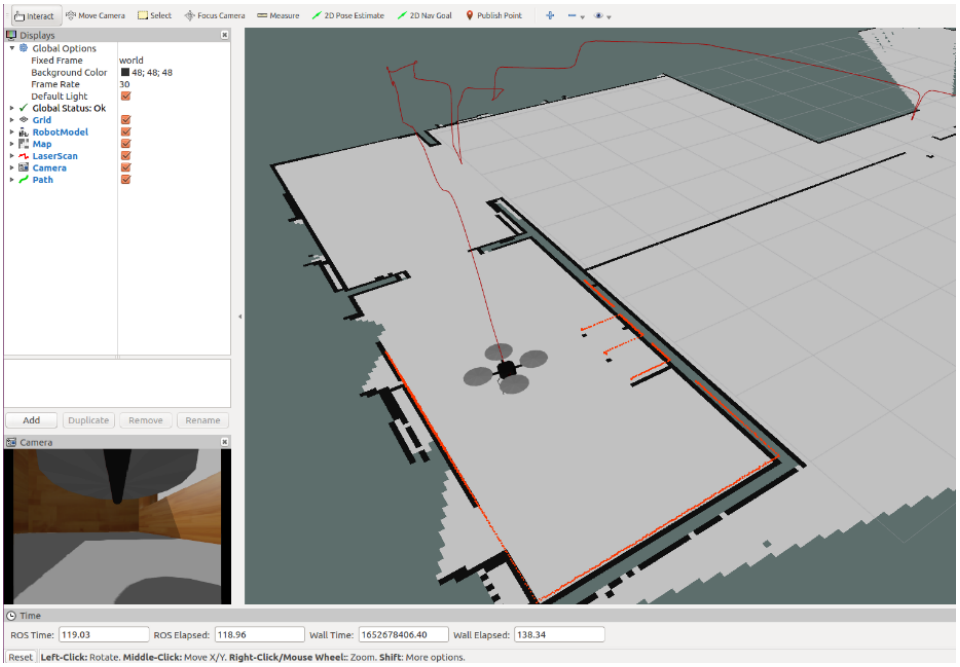


Figura 1.7 Herramienta de visualización RViz. [Fuente propia]

La visualización dinámica y recopilación de datos de un mapa o escenario de simulación puede ser ejecutada a través de RViz. El paquete de *gmaping* ofrecido por ROS y derivado

del paquete *OpenSlam* [10], se utiliza para el desarrollo de un plano bidimensional dado por la información de los sensores de una plataforma robótica. Esto quiere decir que RViz a través de los paquetes ofrecidos por ROS puede efectuar la tarea de SLAM.

El desarrollo de la aplicación viene de la mano de diferentes plataformas y herramientas de simulación, como es ROS en conjunto con Gazebo y RViz. Cada tarea complementaria es ejecutada de manera independiente para llevarlas a su integración y así obtener un modelo de paquete compacto, donde intervengan el grupo de robots heterogéneos y el análisis de estos. Justamente a continuación, se procederá a la creación, diseño y configuración de los diferentes archivos, librerías, paquetes y herramientas que se utilizan en el desarrollo del diseño, ejecución y simulación de las diferentes tareas complementarias buscando cumplir los objetivos propuestos en el presente proyecto.

2 METODOLOGÍA

2.1 PAQUETES Y CONFIGURACIÓN

Para la ejecución de cada una de las tareas complementarias y la integración de estas con el sistema multi-agente es necesario la descarga de cada uno de los paquetes que intervienen en la aplicación, mismos que se los puede obtener desde la web, páginas y repositorios de la comunidad de ROS. Además, la acción de Mapeo Simultaneo y Localización (SLAM) es realizada por el robot aéreo Quadrotor, mientras que la Navegación Autónoma es efectuada por la plataforma robótica terrestre TurtleBot 3.

Por lo tanto, cada uno de los agentes que intervienen en la aplicación tienen sus propios paquetes, los cuales contienen diferentes apartados como son, modelo del robot, cualidades y diseño de este, sensores, actuadores, mapas y mundos precargados, ejemplos de simulación, y un sin número de opciones para la facilidad de entendimiento del desarrollador.

Se presenta el procedimiento para la obtención de cada uno de los paquetes que servirán para el desarrollo de la aplicación, así como de las herramientas que intervienen y mejoran su simulación.

2.1.1 PRECONFIGURACIÓN DE ROS

La principal herramienta para el desarrollo del proyecto y en la que se basa la aplicación en sí, es ROS (Robotic Operating System). Siendo un framework de código abierto es una

excelente opción para aplicaciones de robótica de una manera rápida y flexible, ofreciendo distintas librerías, paquetes y funcionalidades para el desarrollo de esta.

A lo largo del tiempo ROS ha venido evolucionando y con ello actualizando sus versiones, siendo las últimas tres más recientes desde la más actual: Melodic, Lunar, Kinetic. La versión que se usa en el presente proyecto es Kinetic, esto debido a la compatibilidad y efectuación de previos trabajos garantizados sobre las diferentes plataformas robóticas de simulación a ser ejecutadas. En las últimas versiones como es Melodic y Lunar todavía no se migran todos los repositorios de los modelos de cada uno de los agentes en aplicación, por lo que se complica su ejecución y descarga causando problemas de versión.

La descarga, instalación, explicación y primeros pasos a seguir para la preconfiguración de ROS Kinetic se presenta en el Anexo I. Además, la instalación para la ejecución y simulación tanto del robot aéreo Quadrotor como del robot móvil TurtleBot 3 es basada asumiendo que se tiene la versión ROS Kinetic corriendo en el sistema operativo Ubuntu 16.04.07 LTS. En el caso que no se ejecute Gazebo se recomienda consultar la guía de instalación de cliente completo de ROS Kinetic en el siguiente enlace: http://wiki.ros.org/kinetic/Installation/Ubuntu [11].

Gazebo es una potente herramienta para la simulación y visualización 3D de aplicaciones relacionadas directamente con el framework ROS, por lo que para su ejecución es necesario de una tarjeta gráfica de alta calidad, capaz de soportar y correr de una forma óptima la herramienta proporcionada. Las especificaciones generales del equipo en uso para el desarrollo del presente proyecto se detallan en la Tabla 2.1.

Tabla 2.1 Especificaciones del equipo en uso.

Parámetro	Especificación
Modelo de equipo portátil	Dell G5 15
Procesador	Intel® Core™ i7-8750H CPU @2.2 GHz 2.21 GHz
RAM instalada	16 GB
Tarjeta gráfica	NVIDIA GeForce GTX 1050 Ti

Gazebo se descarga en conjunto con ROS, por lo que su instalación debió ser incluida en la descarga e instalación de cliente completo de ROS Kinetic, tal como está especificado en el Anexo I. Una manera de comprobar si Gazebo se encuentra presente en el equipo es ejecutando mediante el terminal el siguiente comando.

```
$ gazebo
```

Se deberá mostrar algo semejante a la siguiente ventana mostrada en la Figura 2.4.

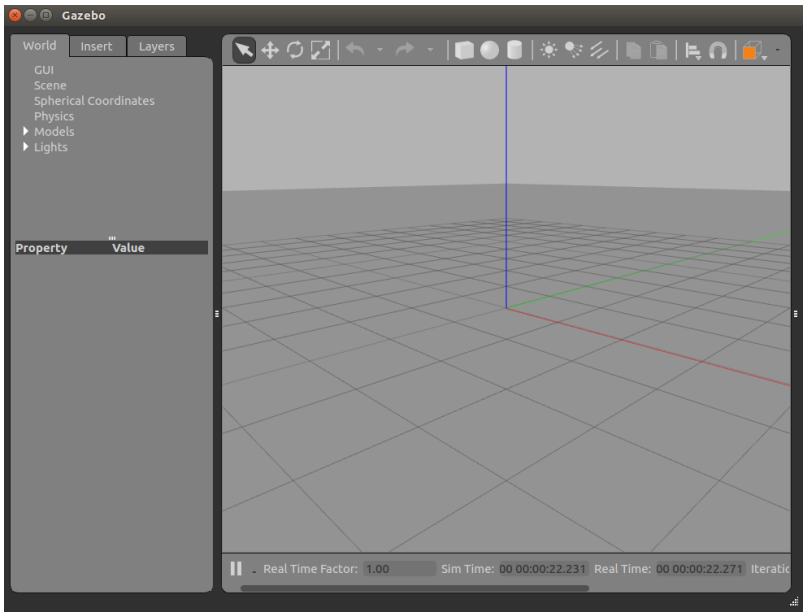


Figura 2.1 Ventana principal de Gazebo. [Fuente propia]

Otra manera de comprobar si Gazebo corre de manera efectiva es usando los archivos de lanzamientos propios de ROS – Gazebo, los mismos que contienen ejemplos y simulaciones a partir de los modelos URDF de robots y mapas de uso más frecuente. Uno de los ambientes precreados se muestra ejecutando en el terminal.

```
$ roslaunch gazebo_ros willowgarage_world.launch
```

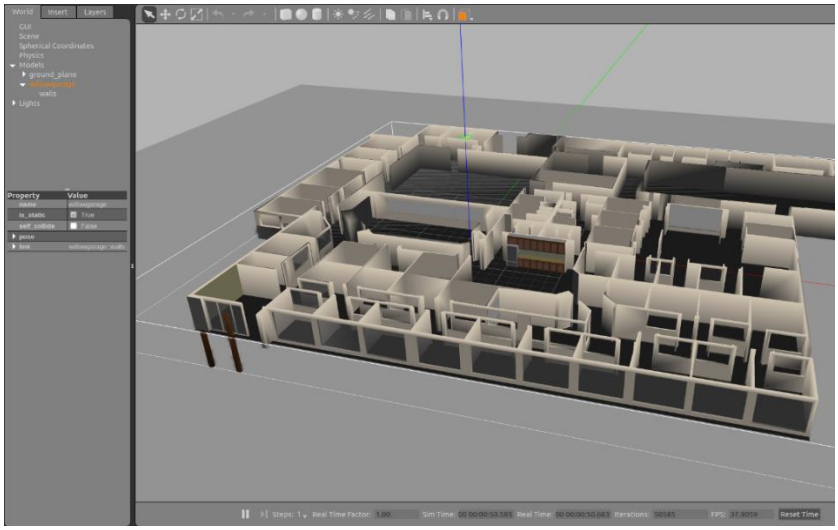


Figura 2.2 Ambiente willowgarage en Gazebo. [Fuente propia]

En la Figura 2.5 se puede observar uno de los tantos mundos predeterminados que posee la herramienta Gazebo en conjunto con ROS, en este caso willowgarage, lo que resalta su potencialidad y funcionalidad para diferentes entornos.

2.1.2 HECTOR QUADROTOR PACKAGE

El metapaquete de pila *hector_quadrotor* fue desarrollado específicamente para el entorno ROS – Gazebo por Team Hector of Technische Universitat Darmstadt [11]. Contiene todo modelado relacionado al agente robótico aéreo usado en simulación, mismo que abarca el control, características, sensores y actuadores, ejemplos o archivos de lanzamiento, entre otros paquetes, siendo los más importantes detallados a continuación:

- **hector_quadrotor_description:** Contiene el modelo genérico URDF del robot aéreo, además de variantes con distintas características de sensores y actuadores.
- **hector_quadrotor_gazebo:** Proporciona los archivos ejecutables o de lanzamiento .launch para la simulación de diferentes ejemplos y aplicaciones del robot.
- **hector_quadrotor_teleop:** Proporciona un nodo el cual maneja al agente robótico establecido usando el teclado mediante las teclas predeterminadas por configuración.
- **hector_quadrotor_gazebo_plugins:** Contiene características específicas para la simulación y ejecución del modelo genérico aéreo y sus derivaciones.
    - gazebo_quadrotor_simple_controller: Nodo que se suscribe directamente al tópico o tema cmd_vel para calcular la fuerza y torque ejercido por el robot.
    - gazebo_quadrotor_propulsion y gazebo_quadrotor_aerodynamics: Pluggins que simulan la aerodinámica, propulsión y mensajes de vuelo de los motores del Quadrotor para su despegue.
- **hector_quadrotor_controllers**: Contiene librerías y el nodo principal para el control de la plataforma robótica aérea usando *ros_control* [11].

A continuación, se detalla el procedimiento para la descarga e instalación del paquete completo desde la web, mismo que es utilizado para la simulación de la plataforma robótica aérea y la tarea de Mapeo Simultaneo y Localización (SLAM).

Como primer paso se debe crear un directorio o carpeta donde se desee almacenar todo el paquete de pila, se recomienda sea una dirección corta y fácil de acceder o a su vez en la carpeta personal del equipo.

Abrir un terminal y ejecutar el siguiente comando.

```
$ mkdir ~/catkin_ws/src
```

Una vez creado la carpeta de instalación, se ingresa en la misma.

```
$ cd ~/catkin_ws
```

Se elige instalar el paquete desde la fuente directamente.

```
$ wstool init src https://raw.github.com/tu-darmstadt-ros-pkg/hector/kineti
c-devel/tutorials.rosinstall
```

De esta manera se inicializa una carpeta *src* dentro de *hector quadrotor* y procederá a realizar la descarga o clonación del paquete completo.

Para obtener una instalación completa del paquete *hector_quadrotor* y no tener ningún problema próximo de ejecución, se realiza la instalación de algunos paquetes adicionales de simulación. Se efectúa la instalación desde el terminal aplicando por delante siempre el comando *sudo*, nos solicitara la contraseña personal de equipo, misma que debemos ingresar para dar permiso de administrador. En el caso de que algún paquete ya este instalado no existirá ningún problema ya que el equipo lo detectara y nos mostrara un mensaje de que no hubo ninguna actualización, esto se da debido a la fuente de donde se clono el metapaquete principal.

```
$ sudo apt-get install ros-kinetic-geographic-info
$ sudo apt-get install ros-kinetic-ros-control
$ sudo apt-get install ros-kinetic-gazebo-ros-control
$ sudo apt-get install ros-kinetic-joy
$ sudo apt-get install ros-kinetic-teleop-twist-keyboard
```

Una vez se tenga la instalación de pila completa procedemos a ingresar al directorio principal y ‘construir’ el mismo mediante los siguientes comandos.

```
$ cd ~/catkin_ws
$ catkin_make
```

Se empezará a ejecutar la revisión y construcción del paquete, siente este el último paso para la obtención del mismo. Además, se corre por medio de un archivo de lanzamiento uno de los ejemplos ya precargados para el robot aéreo para garantizar su funcionamiento, mismo que se muestra mediante el ingreso del siguiente comando en el terminal.

```
$ roslaunch hector_quadrotor_gazebo quadrotor_empty_world.launch
```

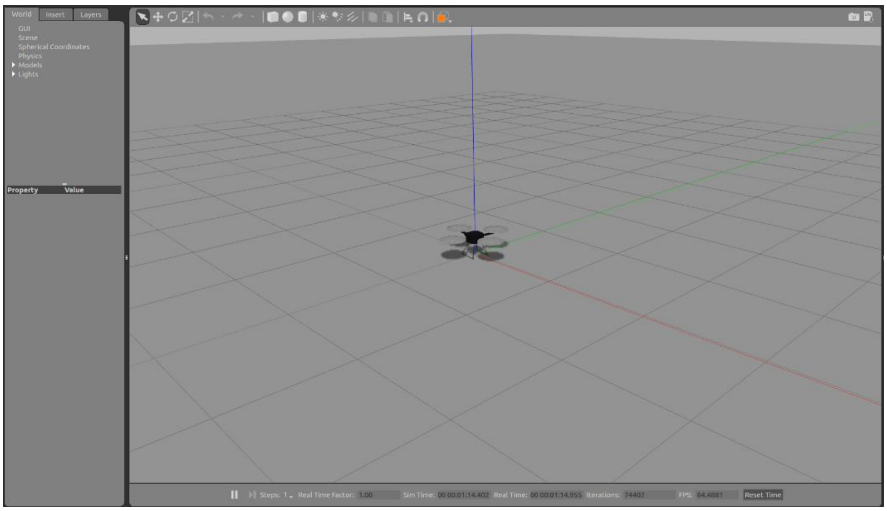


Figura 2.3 Simulación del Quadrotor. [Fuente propia]

En la Figura 2.1 se muestra el correcto funcionamiento de uno de los ejemplos de ROS – Gazebo proveniente del paquete que contiene el modelo aéreo Quadrotor, en él se visualiza a la plataforma robótica aérea en Gazebo en un mundo vacío.

2.1.3 TURTLEBOT 3 PACKAGE

Como se indicó en la sección anterior, para la descarga e instalación del metapaquete *hector_quadrotor* se creó un directorio o carpeta principal llamada *catkin_ws*. Este espacio de trabajo puede ser utilizado de igual manera para la descarga e instalación de todos los paquetes necesarios para la simulación y ejecución de la plataforma robótica terrestre TurtleBot 3 sin ningún inconveniente. Dicho esto, se procede a ingresar a la carpeta fuente desde el terminal.

```
$ cd ~/catkin_ws/src
```

A continuación, se procede con la descarga de los paquetes dependientes que contienen el modelo, características, simulación, entre otras funcionalidades del robot.

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
```

Una vez que se tengan los diferentes paquetes descargados en la carpeta fuente, se ingresa al directorio principal o espacio de trabajo y se los instala o construye a través de los siguientes comandos.

```
$ cd ~/catkin_ws
$ catkin_make
```

La descripción de los variados paquetes instalados se detalla a continuación.

turtlebot3_msgs: Contiene tipos de mensajes y servicios para la plataforma móvil TurtelBot 3 como son:

- Panoramalmg.msg: Mensaje que contiene características como id, latitud, longitud, y tema de imagen.
- SensorState.msg: Mensaje que contiene parámetros de valores constantes y activaciones para los sensores de la plataforma robótica.
- Sound.msg: Sonido para alguna acción del TurtleBot 3.
- SetFollowState.srv: Servicio de seteo de una acción y resultado
- TakePanorama.srv: Servicio de estado del progreso de una acción de robot.

turtlebot3: Metapaquete principal que contiene paquetes para la ejecución, operación, simulación y arranque del robot móvil, además de su modelo y características. Los paquetes descargados se listan a continuación:

- turtlebot3
- turtlebot3_bringup
- turtlebot3_descripcion
- turtlebot3_example
- turtlebot3_navigation
- turtlebot3_slam
- turtlebot3_teleop

turtlebot3_simulations: Contiene paquetes que en su mayoría ofrecen ejemplos de simulación y ejecución de aplicaciones conocidas para un mayor entendimiento del

funcionamiento ROS - Gazebo tomando como agente el TurtleBot 3, los mismos que se mencionan a continuación:

- turtlebot3_fake
- turtlebot3_gazebo
- turtlebot3_simulations

El metapaquete *turtlebot3* instalado contiene los tres modelos de robot móvil desarrollados para el ambiente ROS – Gazebo, los cuales son Waffle, Waffle Pi y Burger. Para evitar exportar el modelo cada vez que se requiera ejecutar un archivo, se recomienda agregar el modelo de robot a la fuente *bashrc*, de esta manera para próximas simulaciones se tomará el modelo de robot móvil elegido, para este caso se ha elegido el modelo Burger.

Se procede a abrir el archivo *bashrc* en el editor del equipo desde el terminal.

```
$ gedit ~/.bashrc
```

Se añade el modelo del robot deseado al final del archivo por medio de la siguiente terminación *export TURTLEBOT3_MODEL=burger,* tal como se muestra en la Figura 2.2.

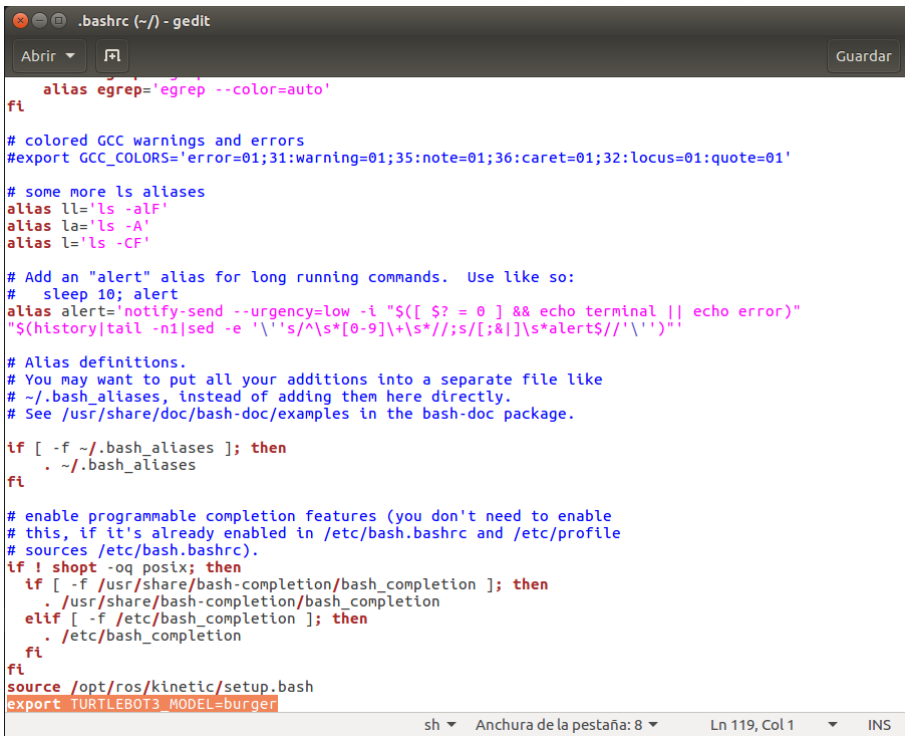
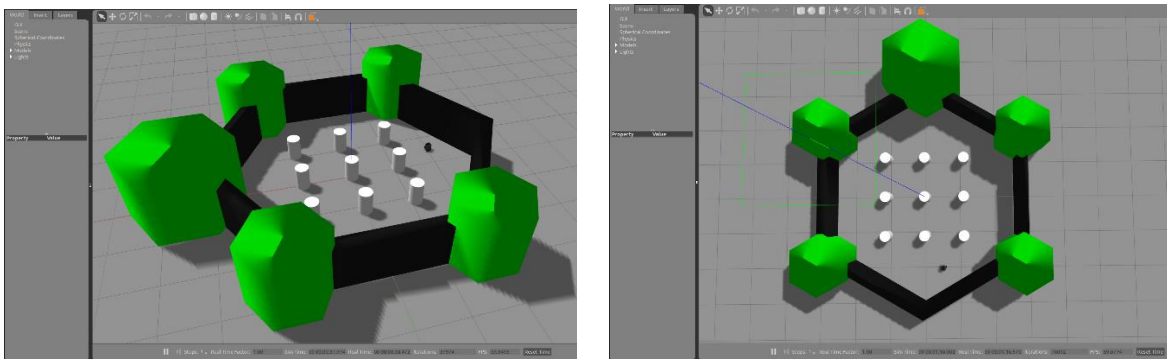


Figura 2.4 Edición de archivo bashrc. [Fuente propia]

Se ejecuta a través de un archivo de lanzamiento la simulación de uno de los ejemplos propios del robot móvil para así garantizar su correcta descarga y funcionamiento por medio del comando mostrado a continuación, mismo que se visualiza en la Figura 2.3.

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



(a) (b)

Figura 2.5 (a), (b) Simulación del TurtleBot 3. [Fuente propia]

Al tener ya todos los paquetes necesarios instalados en el equipo para la simulación del control cooperativo de un sistema multi-agente, conformado a través de las plataformas robóticas Quadrotor y TurtleBot 3, se prosigue al tratamiento de estos, comenzando por la creación de los distintos escenarios de simulación.

2.2 ESCENARIOS DE SIMULACIÓN

2.2.1 CREACIÓN DE MUNDOS EN GAZEBO 3D

A continuación, se procede a la creación de dos mundos diferentes que se van a utilizar para el desarrollo del control cooperativo de un sistema multi-agente para tareas complementarias, uno de ellos tendrá la temática de una casa como ambiente común, y otro un ambiente de laberinto.

Building Editor

Gazebo tiene una herramienta propia de edición donde se puede crear y diseñar un mundo desde cero, o a su vez modificar algún escenario ya precreado. Para esto, a través de la barra de herramientas en Gazebo se debe seleccionar *Edit >> Building Editor* para ingresar al editor de mundos. Una vez que se ingrese al editor se presentará una interfaz como la mostrada en la Figura 2.6.

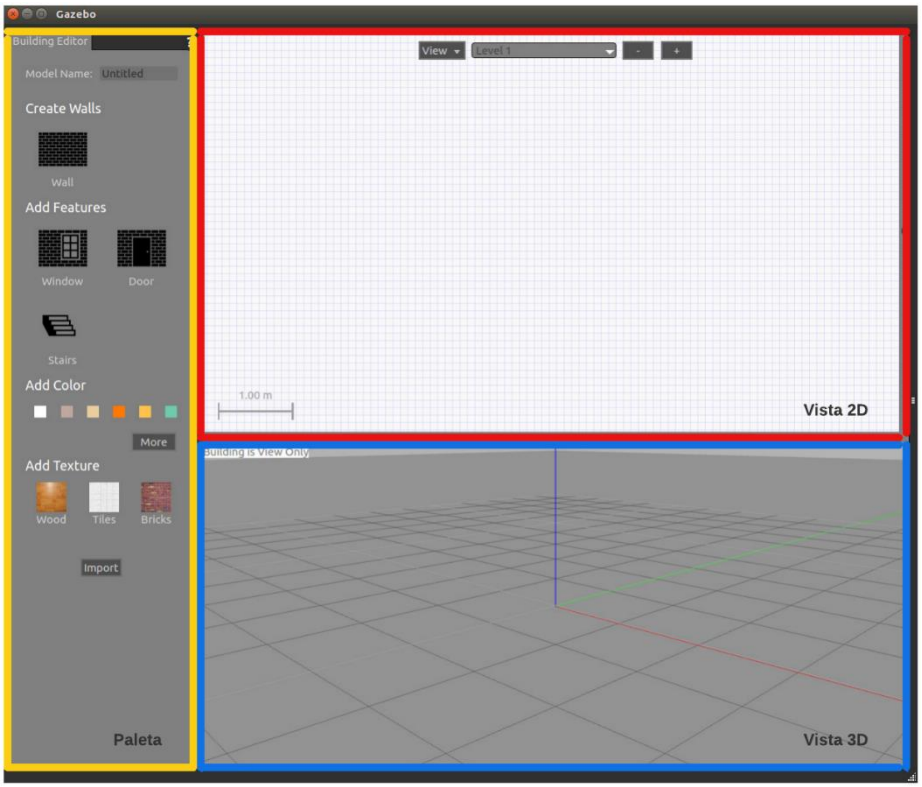


Figura 2.6 Building Editor. [Fuente propia]

La interfaz gráfica de edición se divide en tres secciones principales: Paleta, Vista 2D, Vista 3D. La sección de Paleta contiene opciones y materiales para la edición del mundo, como son puertas, paredes, ventanas, escaleras, edición de color y texturas. La sección Vista 2D ubicada en la parte superior del editor sirve para el dibujado de paredes y todos los materiales contenidos en la sección de Paleta. En la sección de Vista 3D se puede cambiar el color y texturas de las paredes, además de una visualización previa de cómo va quedando el mundo o escenario creado.

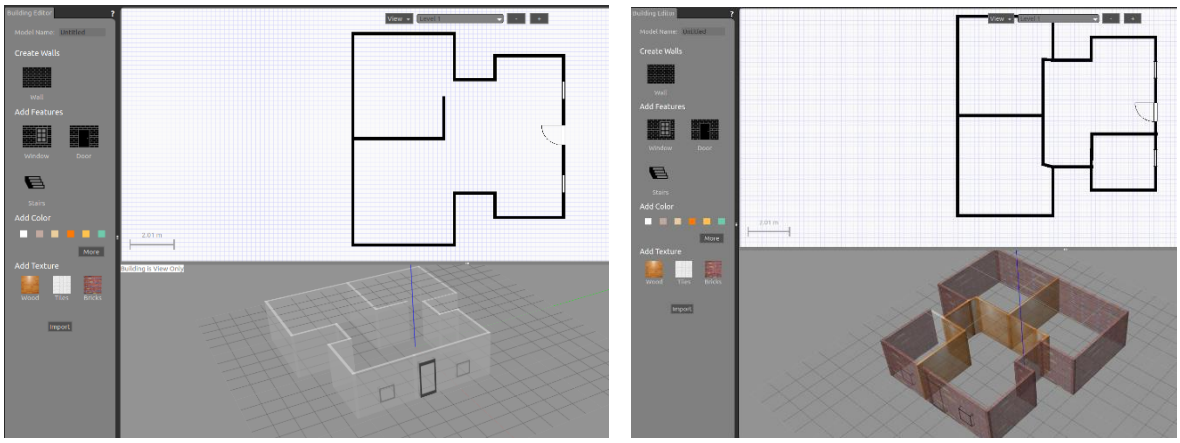
Como se mencionó anteriormente, se empieza creando dos mundos diferentes a través de Building Editor en Gazebo, uno tiene la temática de una casa común de una sola planta, mientras que el otro tiene un enfoque de un laberinto. Los distintos escenarios tienen la finalidad de probar el SLAM en conjunto con la Navegación Autónoma en dos ambientes distintos para variar y analizar cómo se desarrolla y ejecuta el control cooperativo del sistema multi-agente. La casa y el laberinto se los ha denominado HOUSE y LAB respectivamente, mismos que son detallados a continuación.

Escenario HOUSE

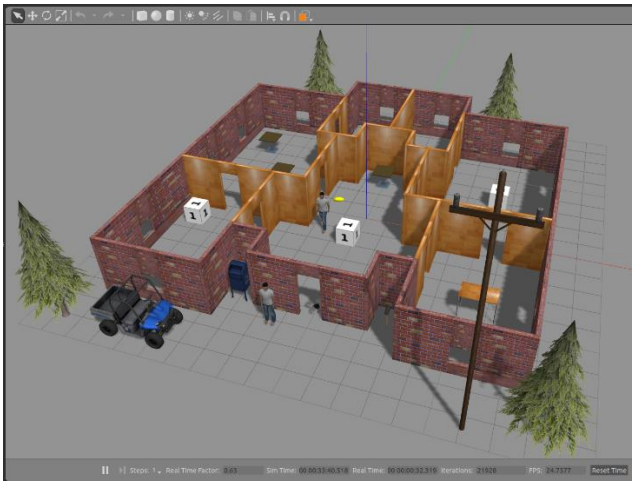
La elaboración del escenario con enfoque en una vivienda de una sola planta se visualiza en la Figura 2.7. Cabe recalcar que el editor permite diseñar el escenario a escala, es decir

se tiene medición en unidades en metros para las dimensiones y paredes del mismo, de esta manera se consigue simetría y orden.

Se diseña una casa de 6 habitaciones alrededor de una sala principal o lobby, con un total de 7 puertas para el ingreso a cada área individual y 10 ventanas, todo esto a través de la sección de la Paleta, y las visiones 2D y 3D que proporciona la edición de mundos visto en la Figura 2.7 (a), (b).



(a) (b)



(c)

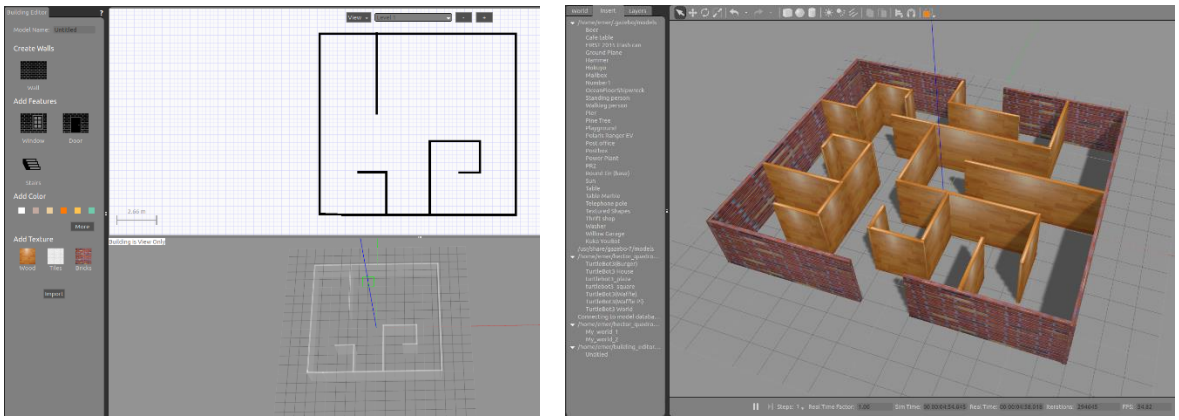
Figura 2.7 (a) Creación y diseño de paredes, puertas y ventanas. (b) Cambio de texturas. (c) Escenario HOUSE finalizado. [Fuente propia]

Una vez que se diseñó y recreó la casa o vivienda de una sola planta, se debe guardar el mundo creado en una ubicación dentro de la carpeta o directorio principal, se recomienda que sea en una ubicación corta y de fácil acceso. Al finalizar mediante la pestaña *Insert* en la barra de herramientas de Gazebo se despliega una lista de elementos extras para una

mejor estructura y visualización de la vivienda, por lo que a su vez se busca probar las tareas complementarias mediante obstáculos, arboles, autos, personas, etc. El escenario finalizado se muestra en la Figura 2.7 (c), siendo este uno de los mundos para la realización de las tareas complementarias.

Escenario LABERINTO

El siguiente escenario se basa en un laberinto, esto con la finalidad de probar con un ambiente distinto a lo que es una vivienda o casa detallado en anterioridad, buscando así ampliar las opciones de funcionamiento para las que está enfocado el control cooperativo del grupo de robots heterogéneos y analizar su respuesta ante varios escenarios.



(a) (b)

Figura 2.8 (a) Creación y diseño de paredes y puertas. (b) Escenario LABERINTO finalizado. [Fuente propia]

En la Figura 2.8 (a) y Figura 2.8 (b) se muestra el proceso para la creación del escenario de simulación, que permitirá probar la tarea de SLAM para la creación del mapa bidimensional por parte del Quadrotor y la respuesta del robot móvil TurtleBot 3 ante la Navegación Autónoma debido a la elección de diferentes puntos destino en un ambiente algo más complicado de navegar. De igual manera se procede diseñando el laberinto añadiendo paredes, para seguir con el cambio de textura y color.

2.2.2 ARRANQUE DEL MUNDO

Al momento de haber creado los escenarios a utilizar durante el desarrollo del proyecto es necesario ejemplificar y explicar el código en lenguaje XML que es usado para el lanzamiento de cada uno de los mundos. El código mostrado se basa en el ejemplo simulado a través del archivo de lanzamiento *turtlebot3_empty_world.launch*. A continuación se ejecuta dentro de un archivo de lanzamiento de extensión tipo .launch la

apertura y ejecución de Gazebo tomando como ambiente de simulación el escenario HOUSE creado anteriormente.

```
<!-- Start Gazebo with my world -->
<include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/HOUSE_WORLD.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>
```

Se explica de manera detallada el código para el lanzamiento de un mundo en Gazebo.

`<include file="$(find gazebo_ros)/launch/empty_world.launch">:` Incluye el archivo de lanzamiento llamado *empty_world.launch* en la ubicación *gazebo_ros/launch* mismo que ejecuta el nodo Gazebo y en el cual se deben variar parámetros y argumentos de simulación.

`<arg name="world_name" value="$(find turtlebot3_gazebo) /worlds/HOUSE_WORLD.world"/>:` Modifica el argumento del archivo fuente del mundo a ejecutar, en este caso se toma como opción el mundo de la casa creado con anterioridad y guardado en la ubicación preferida. En el caso que se desee elegir el segundo escenario de simulacion creado, se debe modificar por *<arg name="world_name" value="$(find turtlebot3_gazebo) /worlds/LAB_WORLD.world"/>.*

`<arg name="paused" value="false"/>:` Inicia la simulación en marcha, en este caso se coloca un valor de `"false"` argumentando que no está pausada.

`<arg name="use_sim_time" value="true"/>:` Solicitud de los nodos de ROS para adquirir el tiempo publicado en Gazebo sobre el tópico o tema */clock.*

`<arg name="gui" value="true"/>:` Inicia la ventana o interfaz de usuario de Gazebo.

`<arg name="headless" value="false"/>:` Deshabilita el estado de grabación en Gazebo.

`<arg name="debug" value="false"/>:` Inicio del servidor de Gazebo (gzserver) en modo depuración.

Una vez que se tiene claro la simulación de un mundo en Gazebo, tomando como referencia los mapas creados y ejemplificados en esta sección, se prosigue a la ejecución de las tareas complementarias del sistema multi-agente heterogéneo.

2.3 SLAM

El Mapeo Simultaneo y Localización más conocido como SLAM es la primera tarea complementaria por realizar, la misma que será ejecutada por la plataforma robótica aérea Quadrotor. Se ejemplifica a continuación todo el proceso para su simulación, desde la aparición del robot, nodos que intervienen, modificación de código en lenguaje XML para variar su comportamiento, entre otros parámetros y características.

2.3.1 APARICIÓN DEL QUADROTOR

El código de simulación del Quadrotor se lo diseña a partir de uno de los archivos de lanzamiento precreado en ROS - Gazebo *spawn_quadrotor.launch* mismo que ejecuta el agente robótico en coordenadas predeterminadas tomando en cuenta sus argumentos y controladores a emplear.

```
<!-- Spawn simulated quadrotor uav -->
<include file="$(find
hector_quadrotor_gazebo)/launch/spawn_quadrotor.launch" >
  <arg name="model" value="$(find hector_quadrotor_description)
/urdf/quadrotor_hokuyo_utm30lx.gazebo.xacro"/>
  <arg name="controllers" value="
      controller/attitude
      controller/velocity
      controller/position
      "/>
  <arg name="x" value="0.0"/>
  <arg name="y" value="-6.0"/>
  <arg name="z" value="0.3"/>
</include>
```

El código presentado tiene el siguiente detalle.

`<include file="$(find hector_quadrotor_gazebo)/launch/spawn_quadrotor.launch" >:` Incluye el

archivo de lanzamiento `SPAWN.launch` mismo que ejecuta la aparición del Quadrotor basado en un archivo de lanzamiento ya probado en ROS - Gazebo.

`<arg name="model" value="$(find hector_quadrotor_description) /urdf/quadrotor_hokuyo_utm30lx.gazebo.xacro"/>`: Toma el modelo URDF del Quadrotor desde uno de los paquetes instalados con anterioridad.

`<arg name="controllers" value="controller/attitude controller/velocity controller/position"/>`: Modifica el argumento de los controladores del Quadrotor, superponiendo sus controladores de altitud, velocidad y posición.

`<arg name="x" value="0.0"/>`: Modifica el argumento de las coordenadas de posición para la aparición del agente robótico aéreo, se puede cambiar el argumento tanto para el eje x, y y z.

Un ejemplo de la simulación del agente aéreo se puede evidenciar en la Figura 2.1 donde el archivo de lanzamiento *quadrotor_empty_world.launch* efectuado en el terminal ejecuta Gazebo y la aparición del robot aéreo por medio de los parámetros explicados en el presente apartado.

```
$ roslaunch hector_quadrotor_gazebo quadrotor_empty_world.launch
$ rosrun rqt_graph rqt_graph
```

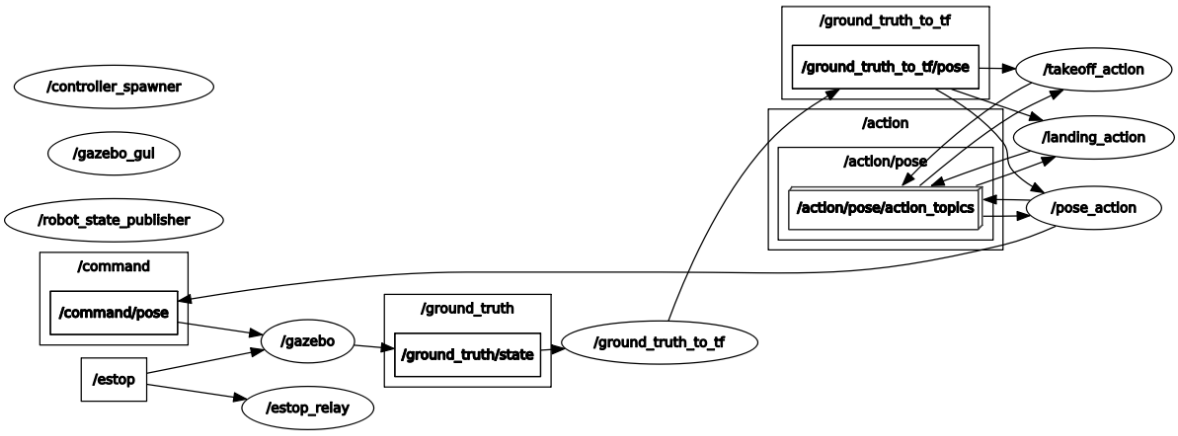


Figura 2.9 Red de grafos para la aparición del Quadrotor. [Fuente propia]

Por medio del comando *rqt_graph* se inicializa la GUI o interfaz gráfica de usuario vista en la Figura 2.9, en la cual se muestra la arquitectura y red de grafos para un mejor entendimiento del funcionamiento del archivo de lanzamiento en aplicación. A

continuación, se redacta el detalle de cada uno de los nodos ejecutados representados por un círculo, siendo estos un eje principal para la comunicación y desarrollo en ROS.

controller_spawner: Nodo que permite funciones como la carga, descarga, inicio y paro de los controladores de posición, velocidad y altitud, los mismos que tienen archivos con extensión .yaml.

gazebo y gazebo_gui: Nodos necesarios para la comunicación en Gazebo e inicialización de la interfaz gráfica respectivamente.

robot_state_publisher: Publica el estado del robot en *tf*, donde *tf* es un tipo de dato *geometry_msgs/PoseStamped* el cual contiene la referencia de tiempo, coordenadas, posición y orientación del robot. Este tipo de dato realiza una variación de las referencias en el tiempo para facilitar la comunicación con otros componentes en ROS, es una manera fácil de acceso para conocer la posición del Quadrotor.

estop_relay: Nodo funcional responsable de la transmisión de datos.

ground_truth_to_tf: Nodo encargado de traducir la posición y orientación del robot a tipo de dato *tf.* Posee un argumento booleano llamado *use_ground_truth_for_tf (*por defecto está definido como *true*), el cual transmite el dato de referencia del robot desde el inicio de la simulación.

takeoff_action, landing_action y pose_action: Nodos que simulan el despegue, aterrizaje y movimiento respectivamente de la plataforma robótica aérea hacia una determinada posición.

Una vez se ha explicado los nodos que intervienen y están activos para la aparición de Quadrotor, se continua con el desarrollo y explicación de la estructura de código en XML para la primera tarea complementaria del control cooperativo usando SLAM.

2.3.2 MAPEO BIDIMENSIONAL

La finalidad del mapeo bidimensional es recrear un tipo ‘croquis’ que sirve para el tratamiento de información sucesiva para la segunda tarea complementaria de Navegación Autónoma. Por ende, es de vital importancia realizar la tarea de SLAM para obtener un mapa con buena referencia, resolución y guía por parte de la plataforma robótica aérea Quadrotor. A continuación, se explica los paquetes y nodos que intervienen para dicha aplicación a partir del código empleado, el cual ejecuta el entorno ROS – Gazebo y RViz como medio de simulación.

```
<!-- Start SLAM system -->
```

```
<include
file="$(find hector_mapping)/launch/mapping_default.launch">
  <arg name="odom_frame" value="world"/>
</include>
```

Se incluye el paquete *hector_mapping*, el cual ofrece un nodo específicamente para la tarea de SLAM que se basa en el sensor LIDAR sin odometría, su ventaja es que utiliza pocos recursos informáticos. Dicho nodo usa las transformaciones tf para los datos de escaneo por lo que no es necesario datos de odometría.

```
<!-- Start GeoTIFF mapper -->
<include
file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
  <arg name="trajectory_publish_rate" value="4"/>
</include>
```

Se incluye el paquete *hector_geotiff,* el cual ofrece un nodo para el almacenamiento de mapas y trayectorias de plataformas robóticas como archivos tipo *geotiff* junto con información georreferenciada.

```
<!-- Start rviz visualization with preset config -->
<node pkg="rviz" type="rviz" name="rviz" args="-d $(find
hector_quadrotor_demo)/rviz_cfg/rviz.rviz"/>
```

Se inicializa el nodo *rviz* que ejecuta la herramienta de simulación RViz, misma que ayuda al mapeo de la tarea de SLAM. A su vez toma el archivo de extensión .rviz del ejemplo de SLAM *indoor_slam.launch.* para un ambiente cerrado efectuado desde el archivo de lanzamiento mencionado.

A continuación, se prosigue con la metodología para la simulación de la Navegación Autónoma, siendo esta la siguiente tarea complementaria para lograr el control cooperativo entre los robots heterogéneos.

2.4 NAVEGACIÓN AUTÓNOMA

La Navegación Autónoma es la segunda tarea complementaria por realizar, la misma que será ejecutada por la plataforma robótica terrestre TurtleBot 3. Se ejemplifica a continuación todo el proceso para su simulación, desde la aparición del robot, nodos que intervienen, modificación de código en lenguaje XML para variar su comportamiento y argumentos, entre otros parámetros y características.

2.4.1 APARICIÓN DEL TURTLEBOT 3

El código de simulación del TurtleBot 3 se lo diseña en base a uno de los archivos de lanzamiento precreado en ROS - Gazebo *turtlebot3_empty_world.launch* mismo que ejecuta el agente robótico en coordenadas predeterminadas tomando en cuenta sus argumentos y modelo a emplear. El código en XML se presenta y detalla a continuación.

```
<!-- Arguments -->
<arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model
type [burger, waffle, waffle_pi]"/>
```

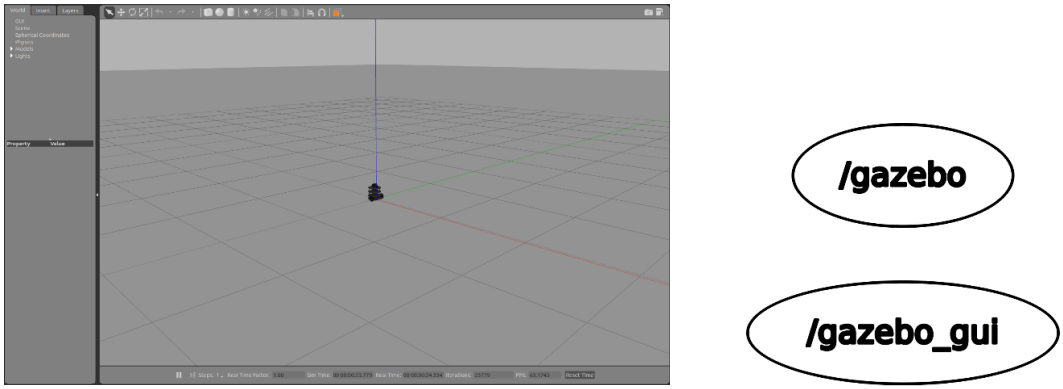
Los argumentos son las variaciones o características que se le da a cada archivo de lanzamiento o nodo para su ejecución, se los coloca en el principio del código para una vez que se los necesite solamente se los direccione, así se evita el spam y reduce la cantidad de código utilizado. En este caso se tiene como argumentos a *model* que es la definición del modelo de robot a utilizar.

```
<!-- Spawn Turtlebot3-->
<param name="robot_description" command="$(find xacro)/xacro --
inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg
model).urdf.xacro" />
<node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"
args="-urdf -model turtlebot3_$(arg model)
    -x 0.0
    -y -6.0
    -z 0.0
    -param robot_description" />
```

El nodo *gazebo_ros* usa el parámetro *robot_description* para la simulación del agente robótico TurtleBot 3 a partir del modelo xacro contenido en el metapaquete *turtlebot3* descargado e instalado con anterioridad, además dicho nodo establece sus argumentos como son el modelo de robot y sus coordenadas de inicio.

Un ejemplo de la simulación del agente terrestre se puede evidenciar en la Figura 2.10 (a), donde el archivo de lanzamiento *turtlebot3_empty_world.launch* efectuado en el terminal ejecuta Gazebo y la aparición del TurtleBot 3.

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
$ rosrun rqt_graph rqt_graph
```



(a) (b)

Figura 2.10 (a) Simulación del TurtleBot 3. (b) Red de grafos para la aparición del TurtleBot 3. [Fuente propia]

Al ejecutar la interfaz GUI por medio del comando *rqt_graph* se visualiza en la Figura 2.10 (b) los nodos activos durante la simulación. Solamente se tiene el nodo *gazebo_gui* y *gazebo*, los cuales ejecutan la comunicación en Gazebo e inicializa la interfaz gráfica respectivamente. La razón por la cual no existe un nodo asociado al robot TurtleBot 3 pese a que se logra visualizar en la simulación es debido a que el nodo */gazebo* usa el parámetro *robot_description* para importar el modelo del agente robótico, mas no le da una acción. Como se explicó en la Sección 1.4.3.1 una manera de entender que es un nodo es relacionarlo con una acción, por ende, al no darle ninguna acción al robot terrestre no debería aparecer ningún nodo asociado.

2.4.2 NAVEGACIÓN AUTÓNOMA

El objetivo de la Navegación Autónoma es lograr que la plataforma robótica terrestre TurtleBot 3 llegue a su punto destino por sus propias capacidades sin necesidad de un manejo manual, más que la designación de un punto final. Dicha tarea complementaria usa como referencia el mapa bidimensional creado por la primera tarea complementaria de SLAM, por ello es necesario que se tenga una secuencia durante la ejecución de dichas acciones. A continuación, se explica los paquetes y nodos que intervienen para la Navegación Autónoma a partir del código empleado, el cual ejecuta el entorno ROS – Gazebo y RViz como medio de simulación.

```
<!-- Arguments -->
<arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model
type [burger, waffle, waffle_pi]"/>
<arg name="map_file" default="$(find
turtlebot3_navigation)/maps/map.yaml"/>
```

Se tiene como argumentos a *model* que es la definición del modelo de robot a utilizar, y a *map_file* el cual busca el archivo del mapa bidimensional con extensión .yaml creado anteriormente por el Quadrotor con la tarea complementaria de SLAM.

```
<!-- Turtlebot3 -->
<include file="$(find
turtlebot3_bringup)/launch/turtlebot3_remote.launch">
  <arg name="model" value="$(arg model)" />
</include>
```

Se incluye el archivo de lanzamiento *turtlebot3_remote.launch* mismo que contiene el nodo *robot_state_publisher* junto con el modelo de robot para su aparición en el mundo de Gazebo, el modelo lo direcciona con el argumento especificado al principio del código.

```
<!-- Map server -->
<node pkg="map_server" name="map_server" type="map_server"
args="$(arg map_file)"/>
```

El nodo *map_server* es capaz de leer un mapa con extension .yaml desde una ruta de almacenamiento del equipo para ofrecerlo como un servicio de ROS. A su vez también es capaz de guardar mapas generados por el SLAM.

```
<!-- AMCL -->
<include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>
```

Se incluye el archivo de lanzamiento *amcl.launch* el cual ofrece un sistema estadístico para la localización de un agente robótico a través de un filtro de partículas en un mapa previamente conocido. Dicho archivo de lanzamiento proporciona un nodo de su mismo nombre, encargado de escanear un mapa bidimensional con láser y transformarlo a mensajes para así generar datos de posición estimados.

```
<!-- move_base -->
<include file="$(find
turtlebot3_navigation)/launch/move_base.launch">
  <arg name="model" value="$(arg model)" />
  <arg name="move_forward_only" value="false"/>
</include>
```

El archivo de lanzamiento *move_base.launch* es el eje encargado de realizar la aplicación de la Navegación Autónoma, dicho archivo contiene el paquete *move_base*, mismo que por medio de un nodo del mismo nombre, realiza la acción de intentar llegar a un punto

destino conocido. El nodo *move_base* es el principal elemento dentro de la pila de navegación, en la Figura 2.11 se explica a detalle el comportamiento de este nodo.

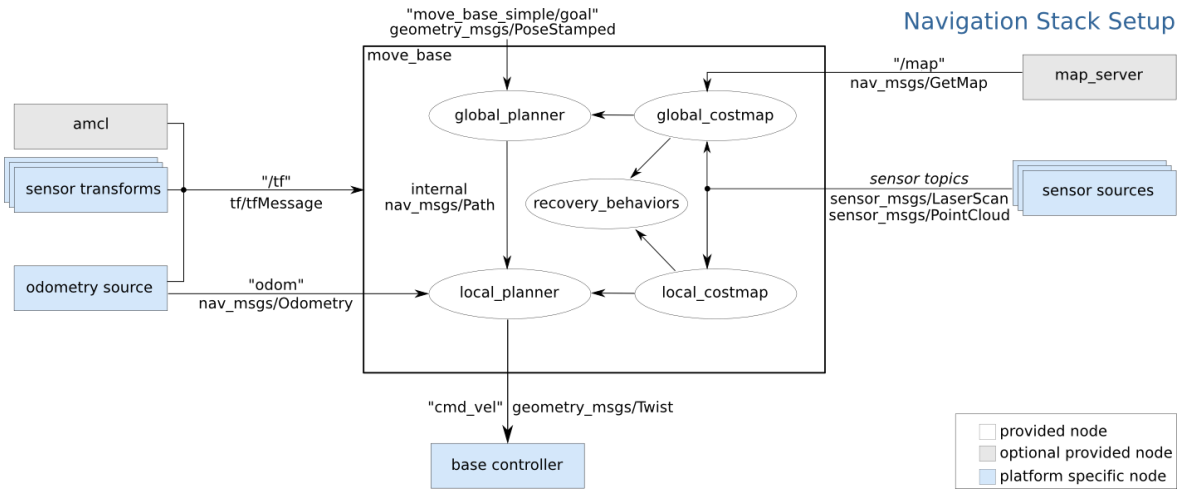


Figura 2.11 Funcionamiento del nodo *move_base* [12]

La pila de navegación (Navegation Stack Setup) obtiene información del mapa bidimensional a través de uno de los nodos opcionales proporcionales *map_server* (nodos grises), de igual manera los datos de sensores y odometría son obtenidos por los nodos específicos del agente robótico en aplicación (nodos azules) encargados de recopilar esta información La información de sensores es transformada a mensajes a través del nodo *amcl* para su tratamiento y fácil adquisición de datos. Para realizar la tarea de navegación el nodo *move_base* se relaciona con un planificador global y otro local, mismos que se vinculan con dos mapas de costos para cada uno de los nodos respectivamente (nodos blancos).

La finalidad del nodo *move_base* en un agente robótico es intentar que el robot llegue a su punto destino partiendo de un mapa previamente conocido, las características de su comportamiento se enlistan a continuación.

- El nodo *move_base* es capaz de recuperar e intentar mover a la plataforma robótica en caso de atasco para seguir con su trayectoria.
- Los objetos que se encuentren fuera del mapa bidimensional previamente conocido y tomado como referencia se eliminan o a su vez el agente no los toma en cuenta.
- En caso de que el robot haya agotado sus opciones de desatasco, como la rotación de lugar para despejar espacio, el agente abortará la ejecución de nodo y enviará un mensaje por el terminal comunicando lo sucedido.

```
<!-- rviz -->
<group if="true">
  <node pkg="rviz" type="rviz" name="rviz" required="true"
        args="-d $(find
turtlebot3_navigation)/rviz/turtlebot3_navigation.rviz"/>
</group>
```

El nodo *rviz* inicializa la herramienta de simulación RViz, misma que ayuda al desarrollo y visualización de la Navegación Autónoma. A su vez toma una configuración ya probada por ROS – Gazebo, específicamente de un ejemplo de navegación para el agente robótico TurtleBot 3, denominado *turtlebot3_navigation.launch.*

Una vez se tiene claro la estructura de código y herramientas necesarias para la simulación tanto de la tarea complementaria de SLAM como de la Navegación Autónoma, se realiza a continuación la integración de las dos tareas complementarias.

2.5 INTEGRACIÓN DEL SISTEMA MULTI-AGENTE

La integración consiste en realizar una programación conjunta de las dos tareas complementarias por medio de dos archivos de lanzamiento para una ejecución reducida y eficiente, manteniendo así una secuencia, orden y fácil entendimiento por parte del desarrollador y espectador.

2.5.1 LANZAMIENTO DEL MAPEO BIDIMENSIONAL Y LOCALIZACIÓN (SLAM)

El archivo de lanzamiento denominado *MAIN_SLAM.launch* es el eje de integración para la ejecución de la tarea primera tarea cooperativa realizada por el robot aéreo Quadrotor, la cual es el Mapeo Bidimensional y Localización. La estructura del archivo de lanzamiento mencionado se muestra en la Figura 2.12.

En la estructura presentada se muestra la secuencia de la ejecución de los diferentes archivos de lanzamiento y nodos presentados en la Sección 2.3, comenzando con el inicio o arranque de Gazebo y terminando con la ejecución y configuración de RViz. De esta manera se logra juntar en un mismo archivo de lanzamiento principal todos los archivos de lanzamiento secundarios y nodos modificados que contienen lo necesario para la aplicación de SLAM efectuada por el UAV.

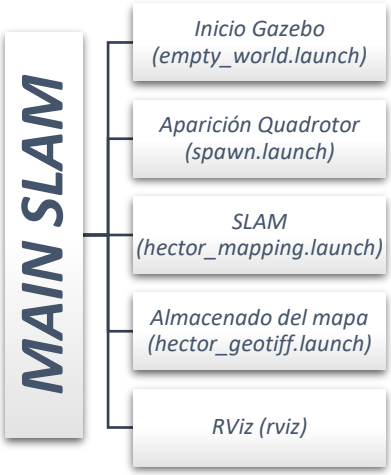


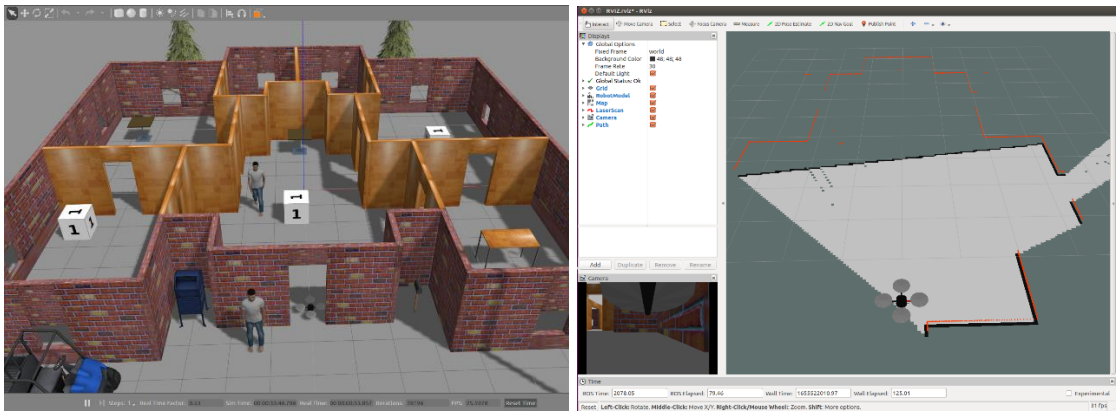
Figura 2.12 Estructura de archivo de lanzamiento MAIN SLAM [Fuente Propia]

Desde el terminal se corre el archivo tipo launch denominado *MAIN_SLAM.launch*, para ello se usa la denominación *roslaunch* seguido de la ubicación del archivo de lanzamiento y después su nombre en específico, tal como se muestra a continuación.

```
$ roslaunch hector_quadrotor_demo MAIN_SLAM.launch
```

Se recomienda una ubicación corta y fácil de acceder. El código completo del archivo mencionado se encuentra en el Anexo II. Archivo de Lanzamiento MAIN SLAM. Por otro lado, el código del archivo de lanzamiento de SLAM para el escenario referente a un laberinto se encuentra en el Anexo IV. Archivo de Lanzamiento MAIN SLAM 2.

Se abrirán dos ambientes de simulación simultáneamente, Gazebo y RViz, tal como se muestra en la Figura 2.12 (a) y Figura 2.12 (b) respectivamente.



(a) (b)

Figura 2.12 (a) Simulación del Quadrotor en escenario HOUSE en Gazebo. (b) Simulación del Quadrotor en RViz para la tarea de SLAM. [Fuente propia]

Después de correr la simulación referente a la tarea de SLAM tanto en Gazebo como RViz, se procede a la creación del mapa bidimensional, por lo que primero se debe activar los motores del Quadrotor para su control y manipulación de trayectoria a través de un servicio de ROS.

```
$ rosservice call /enable_motors "enable: true"
```

El comando presentado llama al servicio */enable_motors* para la activación de los motores de las hélices. Una vez que se tenga activo dicho servicio se ingresa al control de las hélices del Quadrotor de manera manual por medio del teclado. El siguiente comando corre el teleoperador genérico para robots en ROS, permitiendo maniobrar el UAV con una configuración de teclado ya establecida.

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Figura 2.12 Configuración de teclado para operación del Quadrotor mostrada en el terminal. [Fuente propia]

En la Figura 2.12 se determina la configuración de teclado para el control del Quadrotor, el cual posee cuatro hélices a ser controladas y que le dan su dirección y trayectoria. Dichas hélices se activan y mantiene una inclinación por medio de las teclas: U, I, O, J, K, L, M, <, >. Mientras para que elevar y aterrizar la plataforma robótica aérea se usa las teclas T y B respectivamente.

De manera manual se mapea el escenario en análisis, es decir, se va formando un tipo “croquis” o mapa bidimensional tal como se muestra en la Figura 2.13. La configuración en RViz dada por el archivo *rviz.rviz* tomada del ejemplo de *indor_slam.launch* ayuda

significativamente al control y orientación del Quadrotor dentro de la herramienta de simulación.

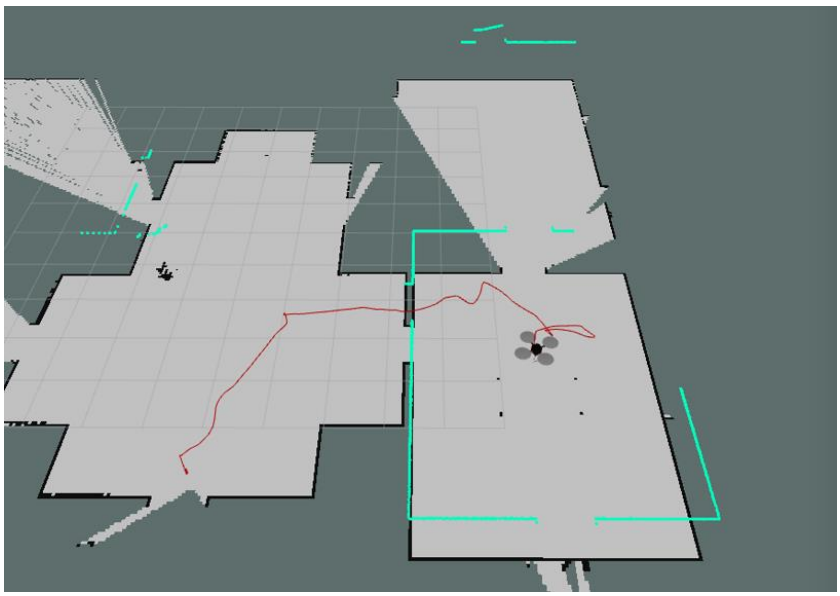


Figura 2.13 Mapeo manual de escenario HOUSE. [Fuente propia]

Teniendo la simulación activa, para lograr visualizar la representación de la red de grafos en ROS de la tarea complementaria de SLAM se ejecuta la interfaz gráfica o GUI a través del siguiente comando.

```
$ rosrun rqt_graph rqt_graph
```

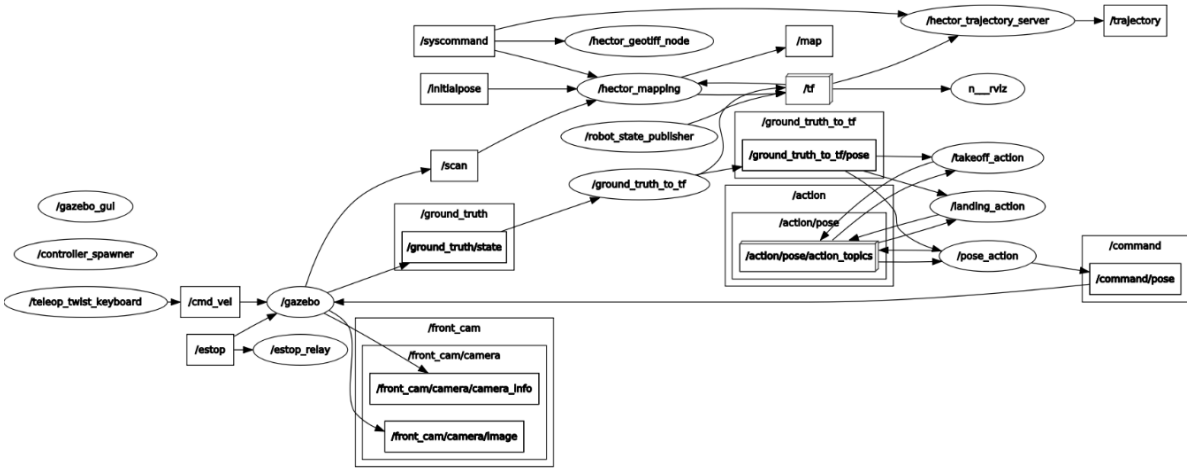


Figura 2.13 Red de grafos para la tarea de SLAM. [Fuente propia]

Por medio de la Figura 2.13 se puede tener un mejor entendimiento de cómo se ejecuta la interconexión entre los nodos y tópicos para el funcionamiento de la aplicación activa

representada por la tarea de SLAM. En seguida se explica cada uno de los nodos que realizan la tarea complementaria en análisis, algunas de ellos son propios de la aparición de la plataforma aérea Quadrotor, mismos que ya están detallados en la Sección 2.3.1.

teleop_twist_keyboard: Ofrece la teleoperación o control de robots twist por medio del teclado, en este caso del Quadrotor. Este nodo se comunica con Gazebo por medio del topico */cmd_vel* para lograr comandar el agente robótico, la configuración de teclado para el control manual se muestra en la Figura 2.12.

hector_mapping: Nodo que necesariamente debe recopilar datos del sensor laser desde el tópico */scan* proveniente de Gazebo, del mismo modo se encarga principalmente del mapeo simultaneo y localización o SLAM por parte de la plataforma robótica aérea a través del recibimiento de datos transformados por *tf.*

hector_geotiff_node: básicamente este nodo genera el mapa bidimensional con datos de georreferencia para su próximo almacenamiento.

hector_trajectory_server: Este nodo almacena la trayectoria del robot aéreo, misma información que está basada en *tf* y la pone a disposición a través de un servicio o tema.

n_rviz: Nodo que inicializa la herramienta de simulación RViz con la configuración preestablecida por el archivo de extensión .rviz proporcionado y detallado en codigo.

Al finalizar la tarea complementaria de SLAM para el control cooperativo de un sistema multi-agente heterogéneo por parte del robot Quadrotor, es necesario almacenar el mapa bidimensional para usarlo como referencia para la siguiente acción complementaria. El guardado del plano se lo realiza a través del nodo *map_server* ejecutando el siguiente comando y a su vez especificando la dirección en el equipo donde se desea guardar el mapa.

```
$ rosrun map_server map_server ~f ~/HOUSE_1
```

2.5.2 LANZAMIENTO DE LA NAVEGACIÓN AUTÓNOMA

El archivo de lanzamiento denominado *MAIN_AUTO.launch* es el eje de integración para la ejecución de la segunda tarea cooperativa realizada por el agente móvil TurtleBot 3, la cual es la Navegación Autónoma. La estructura de archivo de lanzamiento mencionado tiene la siguiente ejemplificación mostrada en la Figura 2.13.

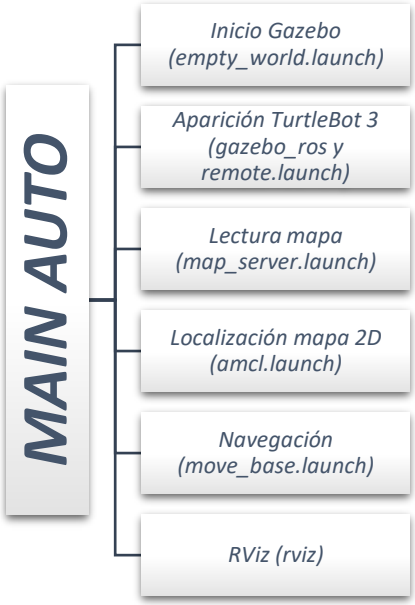


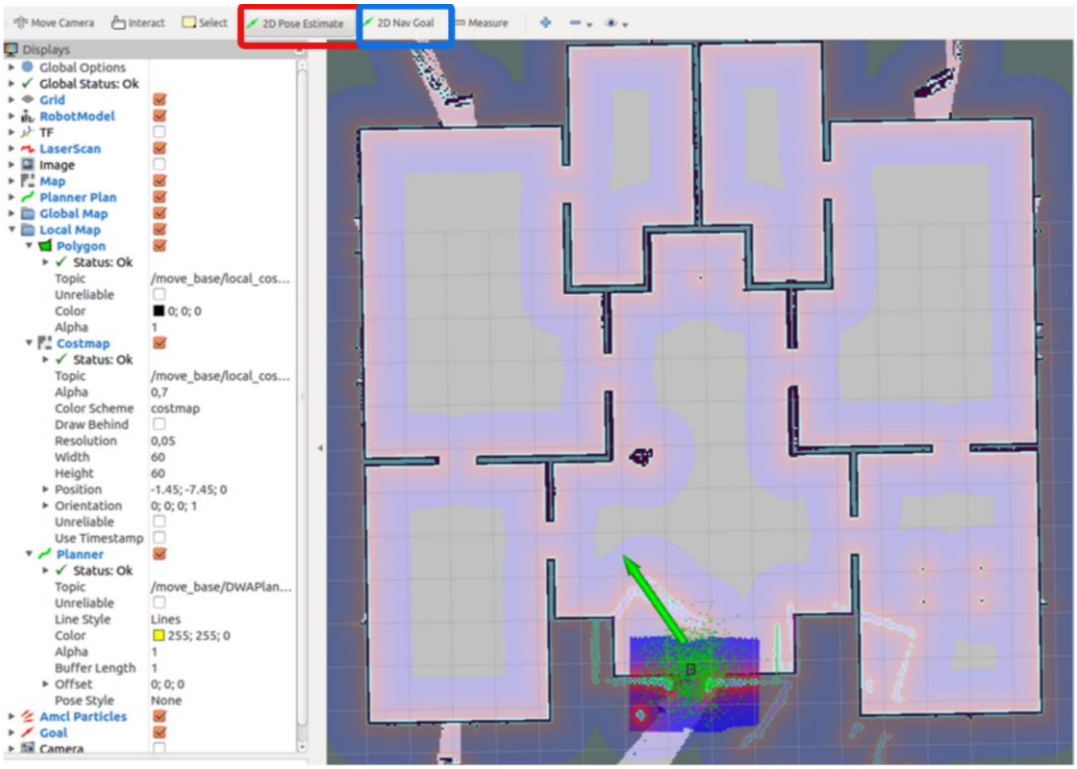
Figura 2.13 Estructura de archivo de lanzamiento MAIN AUTO [Fuente propia]

Se muestra la secuencia de la ejecución de los diferentes archivos de lanzamiento y nodos presentados en la Sección 2.4, comenzando con el inicio o arranque de Gazebo y terminando con la arranque y configuración de RViz. De esta manera se logra juntar en un mismo archivo de lanzamiento principal todos los archivos de lanzamiento secundarios y nodos modificados que contienen lo necesario para la aplicación de la Navegación Autónoma efectuada por el UGV.

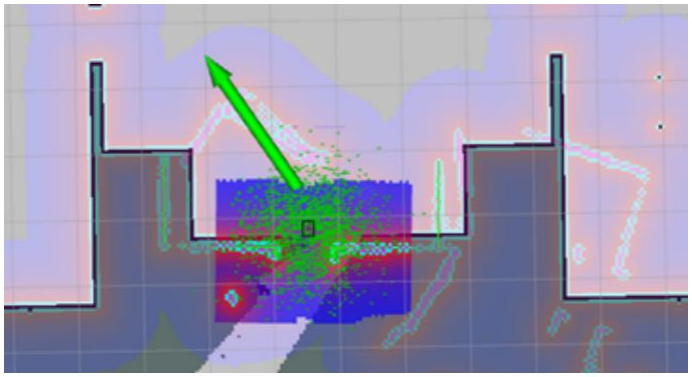
Desde el terminal se corre el archivo tipo launch denominado *MAIN_AUTO.launch* especificando la fuente del archivo .yaml del mapa que se usara de referencia para su ejecución. Se recomienda una ubicación corta y fácil de acceder. El código completo del archivo mencionado se encuentra en el Anexo III Archivo de Lanzamiento MAIN AUTO.

El código del archivo de lanzamiento de Navegación Autónoma para el escenario referente a un laberinto se encuentra en el Anexo V. Archivo de Lanzamiento MAIN AUTO 2.

```
$ roslaunch turtlebot3_navigation MAIN_AUTO.launch map_file:$HOME/HOUSE_1.y
aml
```



(a)



(b)

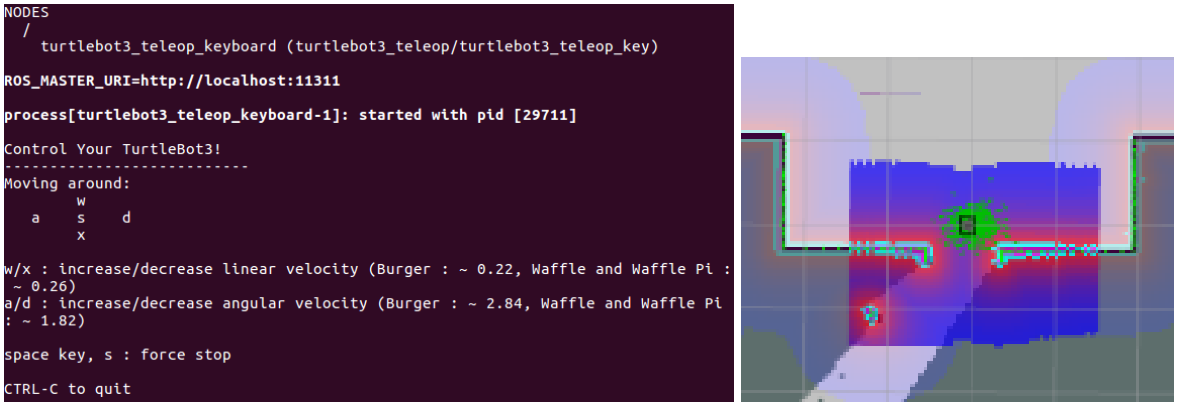
Figura 2.14 (a) Simulación del Quadrotor en RViz para la tarea de Navegación Autónoma. (b) Acercamiento simulación del Quadrotor en RViz. [Fuente propia]

En la Figura 2.14 (b) se puede determinar que los sensores laser del robot móvil denotados con color verde fosforescente no coinciden con el mapa bidimensional tomado como referencia, para ello es necesario realizar el encaje del mapa con los sensores del robot. Se coloca la posición estimada del agente mediante el botón *2D Pose Estimate* señalado con rojo en la Figura 2.14 (a) ubicado en la barra de herramientas de RViz, así se posiciona con la flecha color verde la dirección que se asume del TurtleBot 3. Si todavía los sensores

del robot móvil no se alinean a las paredes del plano existente se realiza el encaje manual de los mismos por medio del siguiente comando.

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

De esta manera se inicializa el nodo de teleoperación para el control del TurtleBot 3 por medio del teclado. En la Figura 2.15 (a) se muestra la configuración y teclas correspondientes para el movimiento del robot móvil, donde su movimiento se realiza por medio de las teclas A, W, D, X y paro a través de la tecla S.



(a) (b)

Figura 2.15 (a) Configuración de teclado para operación del TurtleBot 3 a través del terminal. (b) Encaje de los sensores laser del robot con el mapa de referencia. [Fuente propia]

Una vez se logre ubicar de manera correcta los sensores laser de robot con el mapa de referencia tal como se muestra en la Figura 2.15 (b), es necesario deshabilitar el nodo de teleoperación aplicando Ctrl+C en el terminal, esto para evitar conflictos con el nodo de Navegación que ejecutara la tarea de la Navegación Autónoma.

El próximo paso es ubicar puntos destino utilizando la herramienta de simulación RViz para probar la navegación del agente justamente por medio del botón *2D Nav Goal* (señalado con azul en la Figura 2.14 (b)) ubicado de igual manera en la barra de herramientas.

Para lograr visualizar la red de grafos en ROS de la tarea complementaria de Navegación Autónoma completa se ejecuta la interfaz gráfica o GUI a través del siguiente comando.

```
$ rosrun rqt_graph rqt_graph
```

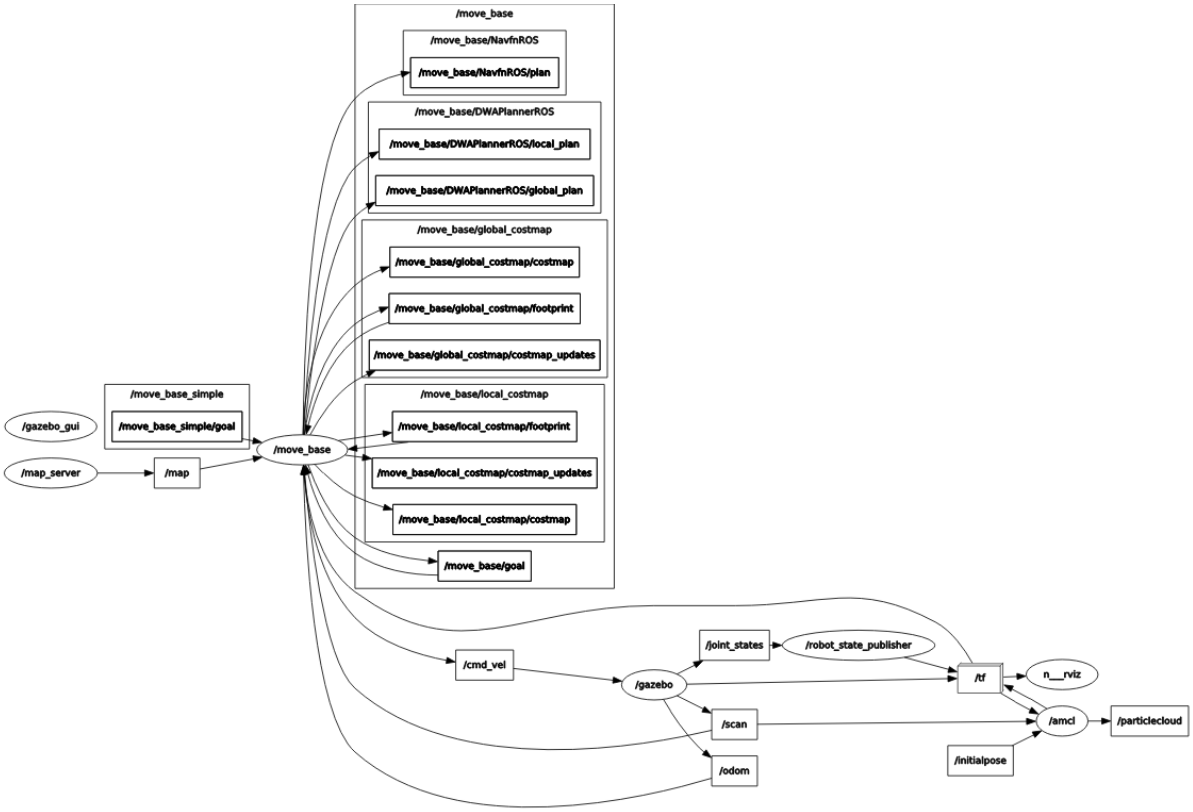


Figura 2.15 Red de grafos para la tarea de Navegación Autónoma. [Fuente propia]

A través de la Figura 2.15 se puede tener un mejor entendimiento del desarrollo de interconexión entre los nodos y tópicos para el funcionamiento de la tarea complementaria de la Navegación Autónoma. A continuación, se explica cada uno de los nodos que realizan la tarea complementaria en aplicación, algunos de ellos son propios de la aparición del robot móvil, mismos que ya están detallados en la Sección 2.4.1.

map_server: La función de este nodo es proporcionar el mapa bidimensional creado por la tarea de SLAM, dicho mapa está en un formato con extensión .yml, conteniendo características de imagen, resolución, origen, entre otros.

move_base: Nodo encargado plenamente de la Navegación Autónoma por parte de robot móvil TurtleBot 3, su finalidad es intentar alcanzar un punto destino conocido navegando a través del mundo evadiendo cualquier obstáculo presente. Su comportamiento se detalla en la sección 2.4.2.

robot_state_publisher: Este nodo presenta el modelo 3D del robot por medio de la interpretación de un árbol cinemático, además de datos de posición a través del tema

join_states. A su vez utiliza el URDF del robot definido por el parámetro *robot_description*, los resultados obtenidos se publican por medio del tipo de dato *tf.*

amcl: Nodo que lee el plano 2D del escenario en aplicación, escanea el mismo por láser, modifica mensajes y proporciona estimaciones de posición de robot usando un filtro de partículas.

n_rviz: Nodo que inicializa la herramienta de simulación RViz con una configuración preestablecida que ayuda para un mejor control y visualización de la plataforma robótica aérea.

Después de haber realizado toda la integración del sistema multi-agente heterogéneo para el control cooperativo por parte de un robot aéreo y otro móvil, se prosigue con los resultados, conclusiones y recomendaciones que se tuvieron durante el desarrollo de proyecto.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 PRUEBAS Y RESULTADOS

A continuación, se presentan los resultados obtenidos durante el desarrollo del presente proyecto aplicando su respectiva metodología. Se muestran las correspondientes gráficas obtenidas bajo los dos escenarios creados HOUSE y LABERINTO, para el control cooperativo y resolución de tareas complementarias de un sistema multi-agente usando robots de diferentes características, una plataforma robótica aérea Quadrotor y otra móvil TurtleBot 3 que realizan las tareas de SLAM y Navegación Autónoma respectivamente.

3.1.1 PRUEBAS Y RESULTADOS BAJO ESCENARIO HOUSE

3.1.1.1 Creación de mapa bidimensional a través de SLAM

En la Figura 3.1 se presenta la simulación de la primera tarea complementaria de SLAM desarrollada por el agente robótico aéreo Quadrotor, su ejecución se da a través de dos ambientes de simulacion RViz y Gazebo, donde el primero ayuda a la visualizacion y mapeo simultáneo del entorno en aplicación y el otro es la herramienta de simulación 3D. Todo esto por medio del archivo de lanzamiento *MAIN_SLAM.launch.*

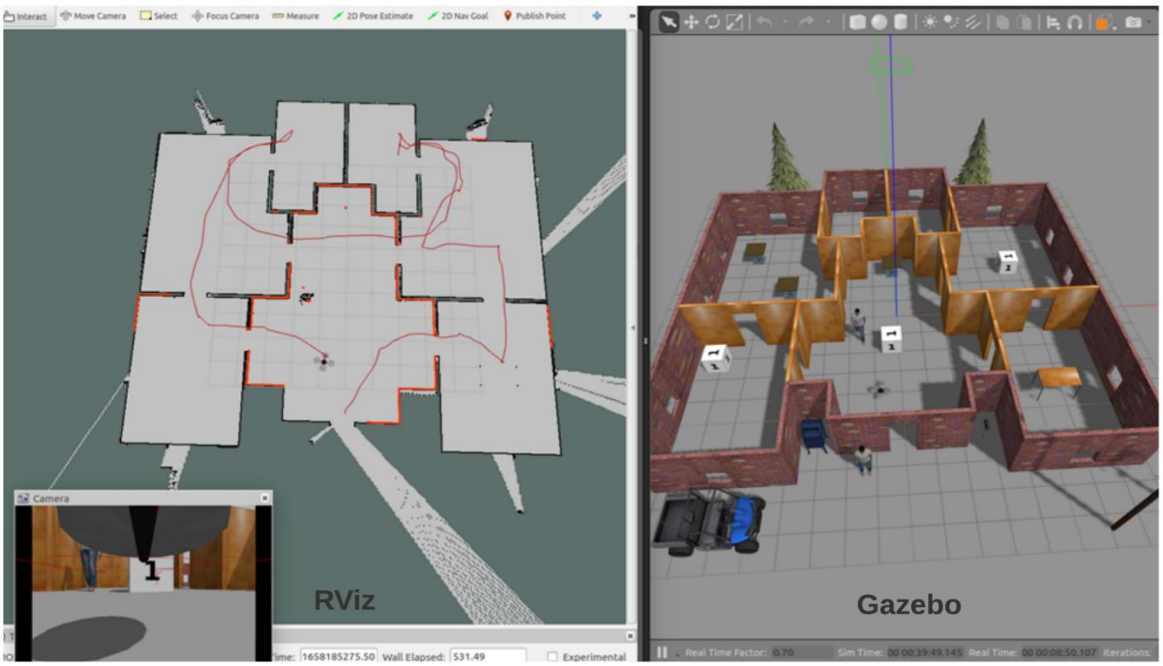


Figura 3.1 Simulación de la tarea de SLAM bajo escenario HOUSE. [Fuente propia]

La configuración prestablecida en RViz permite el trazo de la trayectoria de la plataforma robótica aérea para reconocer el camino seguido por el mismo. Además, se han tomado algunos parámetros referentes al tiempo de simulación para analizar el rendimiento de la aplicación, los cuales se presentan en la Tabla 3.1. Cabe recalcar que los valores de tiempo tomados dependerán de factores como las características propias del equipo donde se esté corriendo la aplicación, mismas que se encuentran especificadas en la Tabla 2.1, y también dependen del usuario quien opere el robot aéreo Quadrotor al instante de realizar el mapeo bidimensional.

Tabla 3.1 Tiempos tomados durante simulación de SLAM

Parámetro	Tiempo aproximado de duración
Arranque	4 [seg]
Mapeo	240 [seg]
Guardado de mapa 2D	Instantáneo

El control manual del Quadrotor se lo hace por medio del teleoperador genérico de teclado y con ayuda de la cámara del robot. El tiempo de mapeo depende plenamente de la habilidad del operador y velocidad de vuelo del Quadrotor ya establecida al instante de realizar el mapeo bidimensional y moverse a través del escenario HOUSE. En este caso se da un valor aproximado de 240 segundos o 4 minutos, tiempo guía para un total mapeo del escenario.

(a) (b)

Figura 3.2 (a) Error de mapeo en escenario HOUSE. (b) Mapa final creado por el Quadrotor a través de SLAM. [Fuente propia]

Existe la posibilidad de que en algún instante la altura de vuelo del Quadrotor sobrepase las paredes del escenario o sus sensores no reconozcan toda la infraestructura del mapa. Uno de estos inconvenientes se muestra en la Figura 3.2 (a), donde la altura de vuelo del agente aéreo no permite el reconocimiento del espacio de las puertas del escenario HOUSE. Para corregir este error se vuelve realizar el mapeo bidimensional desde el principio.

El resultado final del plano creado se evidencia en la Figura 3.2 (b), mismo que tiene una cercanía casi exacta en cuanto al escenario representado en Gazebo, excepto por algunos detalles de resolución en los bordes del mapa. Después de guardarlo en una carpeta de preferencia, se continua con la siguiente tarea complementaria que es la Navegación Autónoma.

Los nodos presentes durante el funcionamiento de la aplicación se los puede enlistar por medio del siguiente comando, que es propio de ROS.

```
$ emer@emer:~$ rosnode list
/controller_spawner
/estop_relay
/gazebo
/gazebo_gui
/ground_truth_to_tf
/hector_geotiff_node
/hector_mapping
/hector_trajectory_server
/landing_action
/pose_action
/robot_state_publisher
/rosout
/rviz
/takeoff_action
```

3.1.1.2 Pruebas de tarea complementaria de Navegación Autónoma

En la Figura 3.3 se muestra la simulación de la segunda tarea complementaria referente a la Navegación Autónoma efectuada por la plataforma móvil TurtleBot 3, su simulación se da a traves de dos ambientes, la herramienta de ROS RViz y Gazebo.

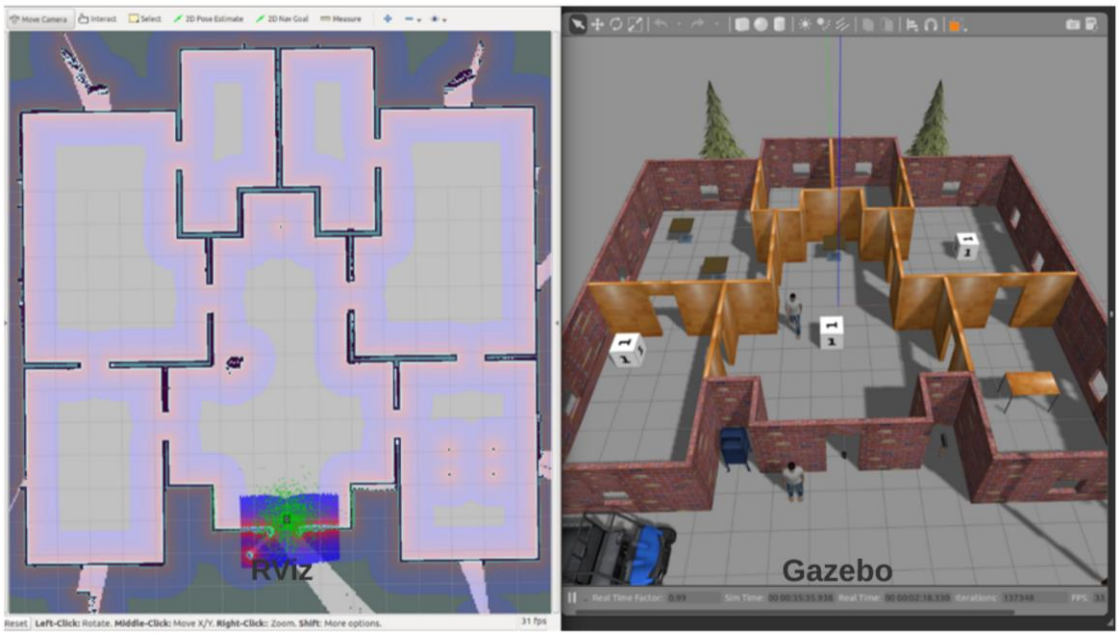
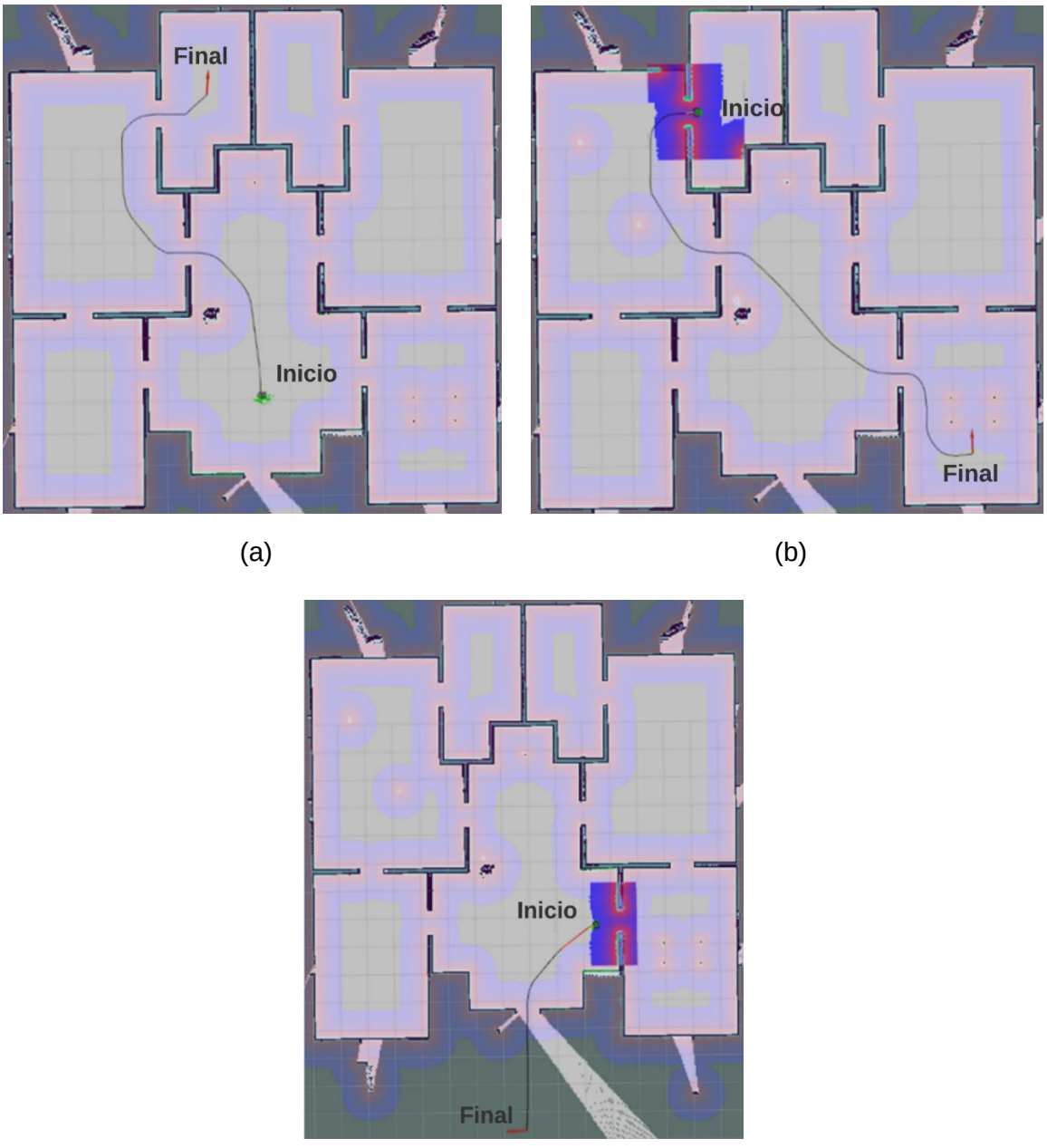


Figura 3.3 Simulación de la tarea de Navegación Autónoma bajo escenario HOUSE. [Fuente propia]

Por medio del archivo de lanzamiento *MAIN_AUTO.launch* se puede ejecutar tanto el ambiente de simulación RViz como Gazebo, de esta manera se tiene una mejor calidad de visión para el desarrollo de la Navegación Autónoma.

Adicional se realizan tres pruebas para el análisis del comportamiento e intento de llegada de la plataforma móvil TurtleBot 3 a su punto destino, las mismas que se presentan a continuación.



(a) (b)

(c)

Figura 3.4 (a) Prueba sobre primer punto destino bajo escenario HOUSE. (b) Prueba sobre segundo punto destino bajo escenario HOUSE. (c) Prueba sobre tercer punto destino bajo escenario HOUSE. [Fuente propia]

Al realizar tres pruebas sobre el escenario en análisis se busca garantizar que el robot móvil logre llegar a su punto destino, las mismas que se muestran en la Figura 3.4 y las observaciones y tiempos aproximados obtenidos se detallan en la Tabla 3.2 presentados a continuación.

Tabla 3.2 Análisis de pruebas en Navegación Autónoma bajo escenario HOUSE.

Prueba	Coordenada de inicio (X ; Y)	Coordenada final (X ; Y)	Tiempo de trayecto aproximado	Observaciones
(a)	(-1,5 ; -4,8)	(-3,2 ; 4,05)	60 [seg]	Objetivo logrado
(b)	(-3,2 ; 4,05)	(4,1 ; -6)	85 [seg]	Objetivo logrado
(c)	(4,1 ; -6)	(-1,7 ; -13,25)	65 [seg]	Objetivo logrado pese a no tener referencia del plano

En la Figura 3.4 (a) se elige un punto destino sobre la esquina superior izquierda del mapa, logrando su objetivo sin ningún tipo de inconveniente y evadiendo todo tipo de obstáculo se le presentase. En la Figura 3.4 (b) se prueba la Navegación Autónoma en la esquina inferior derecha del escenario HOUSE, de igual manera el robot logra llegar a su punto destino conocido. Se realiza una última prueba sobre un punto fuera del plano proporcionado tal como se presenta en la Figura 3.4 (c), los resultados fueron satisfactorios ya que, pese a no tener una referencia conocida del mapa, el TurtleBot 3 navega fuera del mismo llegando a su punto destino.

De igual manera se puede enlistar los nodos en funcionamiento para la Navegación Autónoma en ROS para conocer la información durante el desarrollo de la aplicación.

```
$ emer@emer:~$ rosnode list
 /amcl
/gazebo
/gazebo_gui
/map_server
/move_base
/robot_state_publisher
/rosout
/rviz
```

3.1.2 PRUEBAS Y RESULTADOS BAJO ESCENARIO LABERINTO

3.1.2.1 Creación de mapa bidimensional a través de SLAM

A través de la Figura 3.4 se muestra la simulación y ejecucion de la primera tarea complementaria referente al SLAM desarrollada por la plataforma aérea Quadrotor bajo el escenario Laberinto, su desarrollo se da por medio de dos ambientes de simulacion RViz y Gazebo, donde el primero muestra la visualizacion del mapeo y el otro es la herramienta de simulación 3D usada para tener una mejor nocion del escenario. Todo esto por medio del archivo de lanzamiento *MAIN_AUTO.launch.*

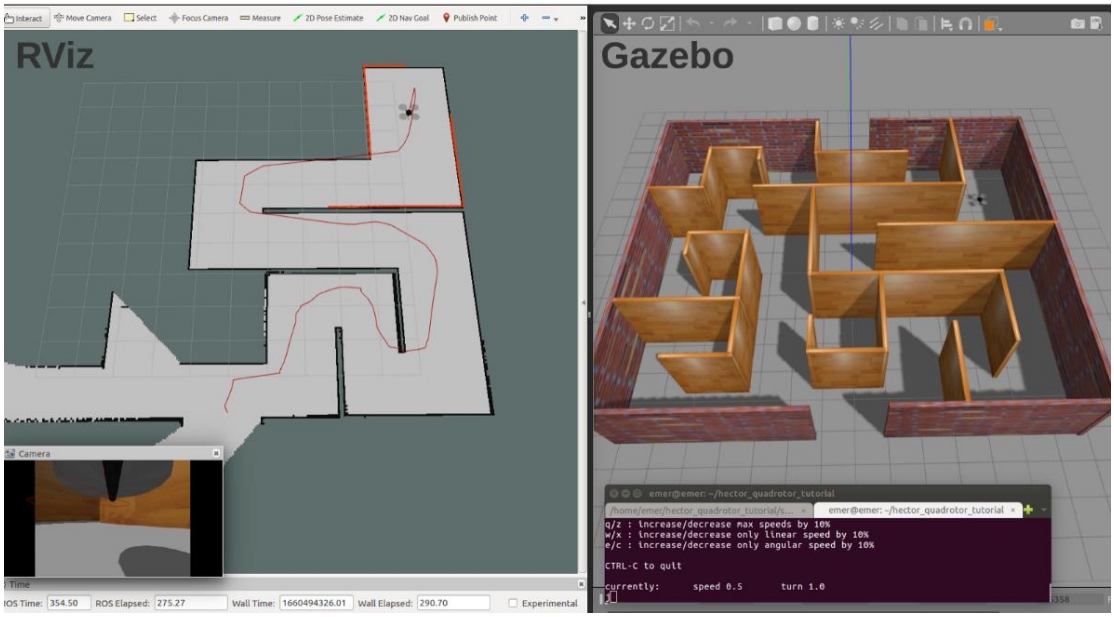


Figura 3.4 Simulación de la tarea de SLAM bajo escenario LABERINTO. [Fuente propia]

RViz permite visualizar la trayectoria del robot aereo para reconocer el camino seguido por el mismo. De igual manera se toman datos de tiempo de simulación para analizar el rendimiento de la aplicación, mismos que se presentan en la Tabla 3.3.

Tabla 3.3 Tiempos tomados durante simulación de SLAM.

Parámetro	Tiempo aproximado de duración
Arranque	3 [seg]
Mapeo	360 [seg]
Guardado de mapa 2D	Instantáneo

Como ya se enfatizo, el vuelo manual del Quadrotor se lo hace por medio del teleoperador genérico de teclado y con ayuda de la cámara del robot. También el tiempo de mapeo depende del control del usuario al instante de realizar el mapeo bidimensional y moverse

a través del LABERINTO. En este caso para el mapeo completo del escenario se tiene un tiempo aproximado de 360 segundos o 6 minutos, algo mayor en comparacion del anterior escenario HOUSE puesto a prueba, ya que existe más trayecto por recorrer para su total reconocimiento.

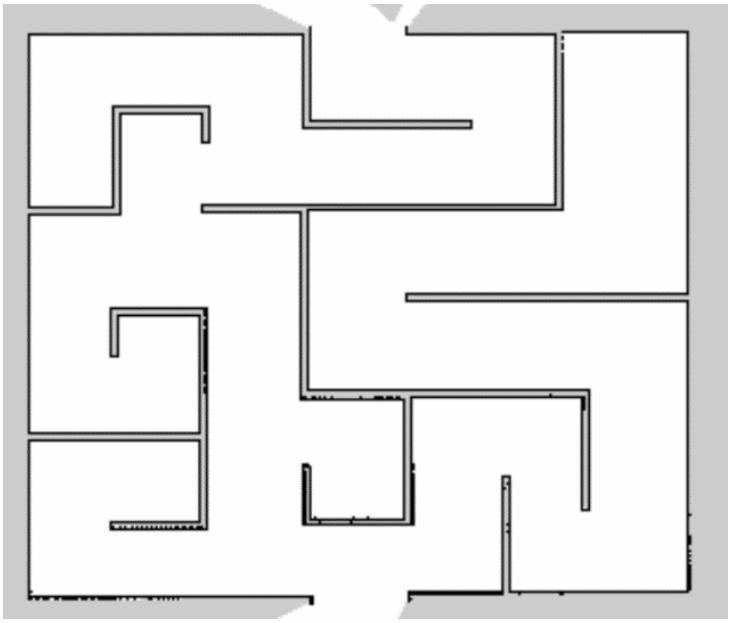


Figura 3.5 Mapa final creado por el Quadrotor a través de SLAM. [Fuente propia]

El resultado final del mapa creado se presenta en la Figura 3.5. Se tiene un buen resultado debido a la semejanza entre el plano y el escenario en simulación. Una vez que se cuenta con el mapa final, se continua con la siguiente tarea complementaria de Navegación Autónoma.

3.1.2.2 Pruebas de tarea complementaria de Navegación Autónoma

En la presentacion de la Figura 3.6 se muestra la simulación de la segunda tarea complementaria de Navegación Autónoma ejecutada por el robot móvil TurtleBot 3, su simulación se da a traves de dos ambientes, los cuales son RViz y Gazebo.

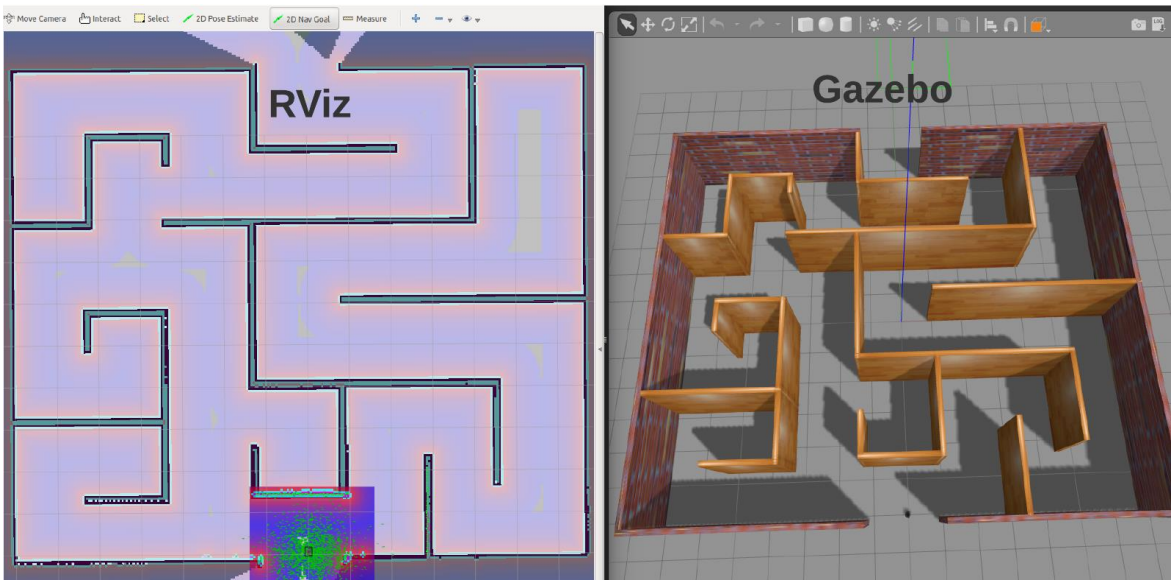
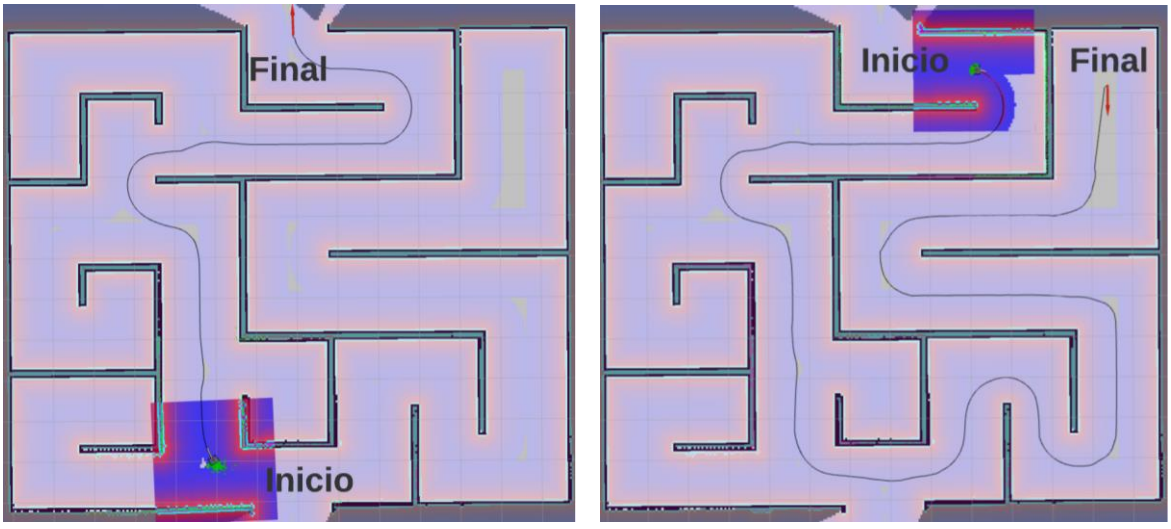


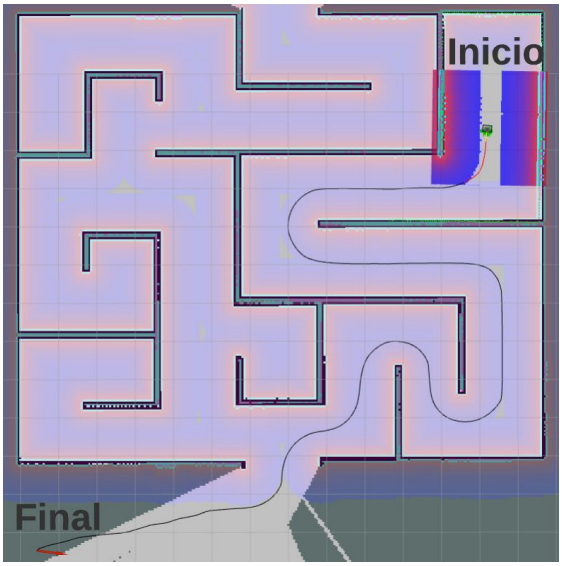
Figura 3.6 Simulación de la tarea de Navegación Autónoma bajo escenario LABERINTO. [Fuente propia]

A traves de le ejecución del archivo de lanzamiento *MAIN_AUTO.launch* inicia el ambiente de simulación ROS con su herramienta de visualización RViz y Gazebo, de esta manera se tiene una mejor calidad de visión para el desarrollo de la tarea complementaria.

De igual manera como en el escenario HOUSE, se realizan tres pruebas para el análisis del comportamiento e intento de llegada de la plataforma móvil TurtleBot 3 a su punto destino bajo el escenario LABERINTO, las mismas que se muestran a continuación.



(a) (b)



(c)

Figura 3.7 (a) Prueba sobre primer punto destino bajo escenario LABERINTO. (b) Prueba sobre segundo punto destino bajo escenario LABERINTO. (c) Prueba sobre tercer punto destino bajo escenario LABERINTO. [Fuente propia]

La idea de efectuar tres pruebas sobre el escenario en análisis es buscar garantizar que el robot móvil logre llegar a su punto destino, pese a estar en un escenario algo menos común a una vivienda como en las primeras pruebas realizadas. Los tiempos aproximados obtenidos se detallan en la Tabla 3.4.

Tabla 3.4 Análisis de pruebas en Navegación Autónoma bajo escenario LABERINTO.

Prueba	Coordenada de inicio (X ; Y)	Coordenada final (X ; Y)	Tiempo de trayecto aproximado	Observaciones
(a)	(-1,45 ; -7,45)	(-1,5 ; 3,95)	100 [seg]	Objetivo logrado
(b)	(-1,5 ; 3,95)	(3,95 ; 2,7)	220 [seg]	Objetivo logrado
(c)	(3,95 ; 2,7)	(-7,9 ; -10)	155 [seg]	Objetivo logrado pese a no conocer mapa

La primera prueba realizada se evidencia en la Figura 3.7 (a), donde se escoge el punto destino a la salida del laberinto, el robot móvil logra moverse a través del escenario de simulación llegando al final sin ningún inconveniente. La segunda prueba mostrada en la Figura 3.7 (b) tiene como inicio las coordenadas finales de la primera prueba, y se elige un

punto en la esquina superior derecha del plano cuyas coordenadas se encuentran en la Tabla 3.4. De igual manera se logra el punto destino con una demora mucho mayor pero entendible debido al largo trayecto por recorrer por parte del TurtleBot 3. La última prueba realizada evidenciada en la Figura 3.7 (c), es en un punto fuera del mapa bidimensional en la parte inferior de mismo, con la finalidad de verificar la funcionalidad de la Navegación Autónoma. Pese a no tener una referencia en el plano, la plataforma robótica terrestre llega a su objetivo sin ningún problema, denotando que en caso de no conocer todo el mapa creado por el SLAM, el robot móvil intenta y logra llegar a su punto destino.

Los nodos en funcionamiento tanto para la tarea de SLAM y la tarea de Navegación Autónoma son los mismos para el escenario HOUSE como para el escenario LABERINTO, ya que únicamente cambia el ambiente en el que se desarrolla la aplicación.

Se ha elaborado un manual de usuario para la ejecución de la integración de las tareas complementarias para el sistema cooperativo, mismo que se encuentra en el Anexo VI. En dicho documento se encuentra un resumen para la obtención de los resultados a partir de proyecto realizado. Ademas, en el Anexo VII se encuentran los links de los videos demostrativos del presente trabajo de integración curricular, donde se demuestra la funcionalidad del Control Cooperativo de un Sistema Multi-agente Heterogéneo para la Resolución de Tareas Complementarias.

3.2 CONCLUSIONES

La investigación y recolección de información acerca del framework de desarrollo ROS en su versión Kinetic, se usó para la elaboración de un Manual de Instalación presentado en el Anexo I. Esto para que pueda ser empleado como referencia de futuros proyectos y así facilitar y prevenir problemas durante la instalación del software y los posibles errores que se podrían presentar.

Se logra tener una curva de aprendizaje beneficiosa en cuanto al estudio de las características de un sistema multi-agente heterogéneo compuesto de un UAV Quadrotor y un UGV TurtleBot 3, así como la ejecución de las tareas complementarias de SLAM y Navegación Autónoma de manera independiente para su integración en ROS.

La creación de dos diferentes escenarios en Gazebo a través de su editor de mundos Building Editor es intuitivo, ya que el editor ofrece mediciones que facilitan el diseño de la infraestructura de la escena, así como opciones de estilización por medio de sus distintas vistas.

La simulación de las tareas complementarias de SLAM y Navegación Autónoma por parte del Quadrotor y TurtleBot 3 respectivamente en el entorno de simulación ROS – Gazebo y Rviz, se ejecutan a través de dos archivos de lanzamiento principales tipo .launch. Los cuales contienen paquetes y la configuración necesaria para garantizar la implementación de las acciones complementarias.

Durante las pruebas realizadas se comprueba la obtención del mapa bidimensional tanto en el escenario HOUSE como LABERINTO a través de la tarea de SLAM realizada por el Quadrotor, teniendo un plano de referencia parecido al escenario en aplicación, mismo que es usado para la tarea de Navegación Autónoma realizada por el TurtleBot 3.

Los tiempos de trayectoria para la creación del mapa través de SLAM dependen plenamente de la habilidad y control del operador, mientras que la duración para alcanzar el destino final en la Navegación Autónoma efectuada por el TurtleBot 3 en los distintos escenarios dependen de las características propias del robot móvil.

Al momento de elegir puntos finales de prueba fuera del mapa de referencia en la tarea de Navegación Autónoma, el agente robótico TurtleBot 3 no tiene mayor problema en alcanzar su objetivo, evadiendo obstáculos no tomados en cuenta en el mapa creado. Por lo tanto, se concluye que en el caso de no conocer una sección en el mapa bidimensional como referencia, el robot móvil intenta y logra llegar a su destino. También el plano sirve para que el operador ubique los puntos finales en el mismo.

3.3 RECOMENDACIONES

Se recomienda seguir y estudiar todas las facilidades y tutoriales que ofrece ROS por medio de su comunidad y soporte (http://wiki.ros.org/ROS/Tutorials), esto mejora el entendimiento de dicho framework de desarrollo, así como la ejemplificación de aplicaciones ya creadas en el mismo para empaparse de información y conocer la libertad que proporciona el entorno de simulación ROS – Gazebo y RViz para aplicaciones robóticas.

Se aconseja la utilización del complemento GUI contenido en el paquete *rqt_graph* que presenta una red de grafos, así se logra comprender de mejor manera la arquitectura en ROS y su interconexión entre nodos y tópicos en funcionamiento.

Durante la creación del mapa bidimensional realizado por el Quadrotor en la tarea de SLAM, se complica el control y manejo del UAV a través del teclado, por lo que se recomienda implementar los controladores de la consola de Xbox para la teleoperación del agente robótico aéreo. Esto facilitaría el control de vuelo del Quadrotor y reduciría los tiempos de mapeo.

Al lograr el objetivo de simulación del sistema multi-agente heterogéneo para la resolución de tareas complementarias, se abre la opción de implementar el presente proyecto de manera física, donde se busque cuidar el bienestar humano a través del reconocimiento y pruebas en escenarios desconocidos, por medio de la ejecución de las acciones complementarias y efectuadas por las plataformas robóticas heterogéneas.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] J. Jiménez, M. Vallejo, J. Ochoa, "Metodología para el Análisis y Diseño de Sistemas Multi-Agente Robóticos: MAD-Smart", *Revista Avances en Sistemas e Informática*, vol. 4, núm. 2, pp. 61-69, septiembre 2007.

[2] G. Acosta, "AMBIENTE MULTI-AGENTE ROBÓTICO PARA LA NAVEGACIÓN COLABORATIVA EN ESCENARIOS ESTRUCTURADOS", Trabajo de fin de máster, Fac. de Minas, Universidad Nacional de Colombia, Melledín, 2010.

[3] A. Soriano, "INTEGRACIÓN DE SISTEMAS MULTI-AGENTE EN PLATAFORMAS EMBEBIDAS HETEROGÉNEAS CON RECURSOS LIMITADOS PARA TAREAS DE LOCALIZACIÓN Y COORDINACIÓN EN DETECCIÓN Y EVASIÓN DE COLISIONES EN ROBÓTICA MÓVIL", Tesis doctoral, Dpto. de Ingeniería de Sistemas y Automática, Universidad Pólitecnica de Valencia, Valencia, 2017.

[4] C. Mamani, "IMPLEMENTACIÓN DE UN ROBOT DE ASISTENCIA PARA GUIADO DE PERSONAS INVIDENTES BASADO EN PLATAFORMAS DE SOFTWARE LIBRE PARA AMBIENTES CERRADOS - INDOOR", Tesis de grado, Fac. de Ingeniería de Producción y Servicios, Universidad Nacional de San Agustín de Arequipa, Arequipa, 2020.

[5] D. Schermuk, "Diseno e Implementación de un Controlador para la Orientación de un QuadRotor", Tesis de grado, Fac. de Ingeniería Electrónica, Universidad de Buenos Aires, Buenos Aires, 2012.

[6] F. Pastor, "Técnicas de guiado, SLAM y visión artificial para robots móviles (Guide, SLAM and computer vision techniques for mobile robots)", Tesis de grado, Escuela Técnica Superior De Ingenieros Industriales Y De Telecomunicación, Universidad de Cantabria, Santander, Cantabria, 2014.

[7] S. Carrasco, "Navegación de Robots Móviles en entorno Matlab-ROS", Tesis de grado, Fac. Ingeniería en Electrónica y Automática Industrial, Universidad de Alcalá, Alcalá de Henares, Madrid, 2019.

[8] G. del Holmo, "Simulación en Gazebo de robots móviles para tareas de transporte y manipulación", Tesis de grado, Escuela de Ingenierías y Arquitectura, Universidad Zaragoza, Aragón, Zaragoza, 2019.

[9] M. Ibañez, " Control de vuelo para seguimiento de trayectorias de un cuadricóptero simulado mediante ROS/Gazebo", Tesis de grado, Escuela de Ingenierías y Arquitectura, Universidad Zaragoza, Aragón, Zaragoza, 2020.

[10] S. Sánchez, " Implementación y experimentación de herramientas para el diseño y desarrollo de comportamientos en robots tipo Turtlebot", Tesis de grado, Escuela Politécnica Superior, Universidad Autónoma Metropolitana, Ciudad de Méexico, 2019.

[11] C. Fairchild y Dr. T. L. Harman, *ROS Robotics By Example*, 2nd ed. Birmingham: Packt, 2017. Accessed: Dec.

[12] “move_base” http://wiki.ros.org/move_base (Accessed Jul 11, 2022).

5 ANEXOS

Los Anexos del presente Trabajo de Integración Curricular se enlistan a continuación:

ANEXO I. Manual de Instalación de ROS

ANEXO II. Archivo de Lanzamiento MAIN SLAM

ANEXO III. Archivo de Lanzamiento MAIN AUTO

ANEXO IV. Archivo de Lanzamiento MAIN SLAM 2

ANEXO V. Archivo de Lanzamiento MAIN AUTO 2

ANEXO VI. Manual de Usuario

ANEXO VII. Links de Videos Demostrativos

ANEXO I. Manual de Instalación de ROS

Configuración previa

Se debe tener en consideración que ROS Kinetic solamente es compatible con las versiones Wily de Ubuntu 15.10, Xenial de Ubuntu 16.04 y para los paquetes debian solamente con la versión Jessie de Debian 8. Para la aplicación en estudio, se instalará y usará la versión Xenial (Ubuntu 16.04).

El primer paso para la instalación de ROS Kinetic es configurar la computadora para la aceptación de software de packages.ros.org con el siguiente comando.

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

A continuación, se configuran las llaves del software.

```
$ sudo apt install curl # if you haven't already installed curl

$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sud
o apt-key add -
```

Instalación de ROS Kinetic

Como primer paso durante la instalación, se debe asegurar que los paquetes Debian estén actualizados.

```
$ sudo apt-get update
```

A continuación, se recomienda realizar la instalación completa de escritorio, la cual abarca y contiene herramientas como ROS, rqt, RViz, bibliotecas genéricas de plataformas robóticas, visualización y percepción 2D y 3D.

```
$ sudo apt-get install ros-kinetic-desktop-full
```

Configuración del entorno

Con la siguiente instrucción las variables de entorno ROS se agregan automáticamente a la cuenta bash cada vez que se inicializa un nuevo shell.

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

```
$ source ~/.bashrc
```

Dependencias de ROS

Las dependencias son necesarias para crear y administrar espacios de trabajo propios en ROS. Un ejemplo claro es la herramienta rossinstall, la cual por medio de una línea de comando se puede descargar árboles fuente para paquetes ROS.

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

Rosdep

Es una herramienta que permite al sistema instalar las dependencias para la fuente de compilación de manera más versátil y a su vez ejecuta algunos componentes esenciales en ROS. Para su instalación e inicialización es necesario ejecutar los siguientes comandos en orden.

```
$ sudo apt install python-rosdep
$ sudo rosdep init
$ rosdep update
```

Creación de un espacio de trabajo

Como primero se crea una carpeta principal que contendrá los paquetes principales de ejecución.

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

El ultimo comando ejecutado *catkin_make* construye la carpeta o directorio principal, mismo que efectúa herramientas prácticas para el desarrollo de aplicaciones en espacios de trabajo tipo catkin. Esta ejecución creara un archivo CMakeLists.txt en la carpeta ‘src’; además se tendrán las carpetas ‘build’ y ‘devel’, donde en la segunda mencionada se encontrarán archivos tipo setup.*sh.

Se toma la fuente de archivo setup.*sh.

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

Se asegura que el espacio de trabajo este superpuesto por el script de configuración verificando que la variable de entorno ROS_PACHAKE_PATH este incluida en la carpeta principal.

```
$ echo $ROS_PACKAGE_PATH
```

Resultado esperado.

```
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

Requisitos previos

Se recomienda instalar el paquete de ros-tutorials para la exploración de sus herramientas. En el comando apt-get install ros-<distro>-ros-tutorials, se puede remplazar <distro> por la distribución en uso.

```
$ sudo apt-get install ros-kinetic-ros-tutorials
```

El paquete rqt contiene herramientas como rqt_graph la cual crea una red de grados donde se puede tener una mejor visualización de los nodos, tópicos, mensajes, entre otros elementos que intervienen en la ejecución de aplicaciones ROS. Su instalación viene a través de los siguientes comandos desde el terminal.

```
$ sudo apt-get install ros-kinetic-rqt
$ sudo apt-get install ros-kinetic-rqt-common-plugins
```

De igual manera se puede remplazar en el apartado donde se menciona la distribución de ROS por la distribución en uso, tal cual el ejemplo anterior.

ANEXO II. Archivo de Lanzamiento MAIN SLAM

```
<launch>

  <!-- Start Gazebo with my world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/HOUSE_WORLD.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>

  <!-- Spawn simulated quadrotor uav -->
  <include file="$(find
hector_quadrotor_gazebo)/launch/spawn_quadrotor.launch" >
    <arg name="model" value="$(find
hector_quadrotor_description)/urdf/quadrotor_hokuyo_utm30lx.gazebo
.xacro"/>
    <arg name="controllers" value="
        controller/attitude
        controller/velocity
        controller/position
        "/>
    <arg name="x" value="0.0"/>
    <arg name="y" value="-6.0"/>
    <arg name="z" value="0.3"/>
  </include>

  <!-- Start SLAM system -->
  <include
      file="$(find hector_mapping)/launch/mapping_default.launch">
    <arg name="odom_frame" value="world"/>
  </include>

  <!-- Start GeoTIFF mapper -->
  <include
      file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_publish_rate" value="4"/>
  </include>

  <!-- Start rviz visualization with preset config -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
hector_quadrotor_demo)/rviz_cfg/rviz.rviz"/>

</launch>
```

ANEXO III. Archivo de Lanzamiento MAIN AUTO

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model
type [burger, waffle, waffle_pi]"/>
  <arg name="map_file" default="$(find
turtlebot3_navigation)/maps/map.yaml"/>

  <!-- Star Gazebo with my world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/HOUSE_WORLD.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>

  <!-- Spawn Turtlebot3-->
  <param name="robot_description" command="$(find xacro)/xacro --
inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg
model).urdf.xacro" />
  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"
args="-urdf -model turtlebot3_$(arg model)
    -x 0.0
    -y -6.0
    -z 0.0
    -param robot_description" />

  <!-- Turtlebot3 -->
  <include file="$(find
turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- Map server -->
  <node pkg="map_server" name="map_server" type="map_server"
args="$(arg map_file)"/>

  <!-- AMCL -->
  <include file="$(find
turtlebot3_navigation)/launch/amcl.launch"/>

  <!-- move_base -->
  <include file="$(find
turtlebot3_navigation)/launch/move_base.launch">
    <arg name="model" value="$(arg model)" />
    <arg name="move_forward_only" value="false"/>
  </include>
```

```
  <!-- rviz -->
  <group if="true">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find
turtlebot3_navigation)/rviz/turtlebot3_navigation.rviz"/>
  </group>

</launch>
```

ANEXO IV. Archivo de Lanzamiento MAIN SLAM 2

```
<launch>

  <!-- Start Gazebo with my world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/LAB_WORLD.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>

  <!-- Spawn simulated quadrotor uav -->
  <include file="$(find
hector_quadrotor_gazebo)/launch/spawn_quadrotor.launch" >
    <arg name="model" value="$(find
hector_quadrotor_description)/urdf/quadrotor_hokuyo_utm30lx.gazebo
.xacro"/>
    <arg name="controllers" value="
        controller/attitude
        controller/velocity
        controller/position
        "/>
    <arg name="x" value="0.0"/>
    <arg name="y" value="-6.0"/>
    <arg name="z" value="0.3"/>
  </include>

  <!-- Start SLAM system -->
  <include
     file="$(find hector_mapping)/launch/mapping_default.launch">
    <arg name="odom_frame" value="world"/>
  </include>

  <!-- Start GeoTIFF mapper -->
  <include
     file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_publish_rate" value="4"/>
  </include>

  <!-- Start rviz visualization with preset config -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
hector_quadrotor_demo)/rviz_cfg/RVIZ.rviz"/>

</launch>
```

ANEXO V. Archivo de Lanzamiento MAIN AUTO 2

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model
type [burger, waffle, waffle_pi]"/>
  <arg name="map_file" default="$(find
turtlebot3_navigation)/maps/map.yaml"/>

  <!-- Star Gazebo with my world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/LAB_WORLD.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>

  <!-- Spawn Turtlebot3-->
  <param name="robot_description" command="$(find xacro)/xacro --
inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg
model).urdf.xacro" />
  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"
args="-urdf -model turtlebot3_$(arg model)
    -x 0.0
    -y -6.0
    -z 0.0
    -param robot_description" />

  <!-- Turtlebot3 -->
  <include file="$(find
turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- Map server -->
  <node pkg="map_server" name="map_server" type="map_server"
args="$(arg map_file)"/>

  <!-- AMCL -->
  <include file="$(find
turtlebot3_navigation)/launch/amcl.launch"/>

  <!-- move_base -->
  <include file="$(find
turtlebot3_navigation)/launch/move_base.launch">
    <arg name="model" value="$(arg model)" />
    <arg name="move_forward_only" value="false"/>
  </include>
```

```
  <!-- rviz -->
  <group if="true">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find
turtlebot3_navigation)/rviz/turtlebot3_navigation.rviz"/>
  </group>

</launch>
```

ANEXO VI. Manual de Usuario

El presente manual de usuario detalla de una manera breve la secuencialidad de los comandos para la ejecución del control cooperativo de un sistema multi-agente heterogéneo para la resolución de tareas complementarias.

Los comandos usados se basan en la metodología de presente proyecto de integración curricular, siguiendo las rutas de almacenamiento de los diferentes archivos usados durante su desarrollo y tomando en cuenta como opción los dos escenarios de simulación creados, HOUSE y LABERINTO.

Como primer paso se debe ingresar al directorio principal donde se tiene descargado todos los paquetes necesarios y tomar su fuente como referencia.

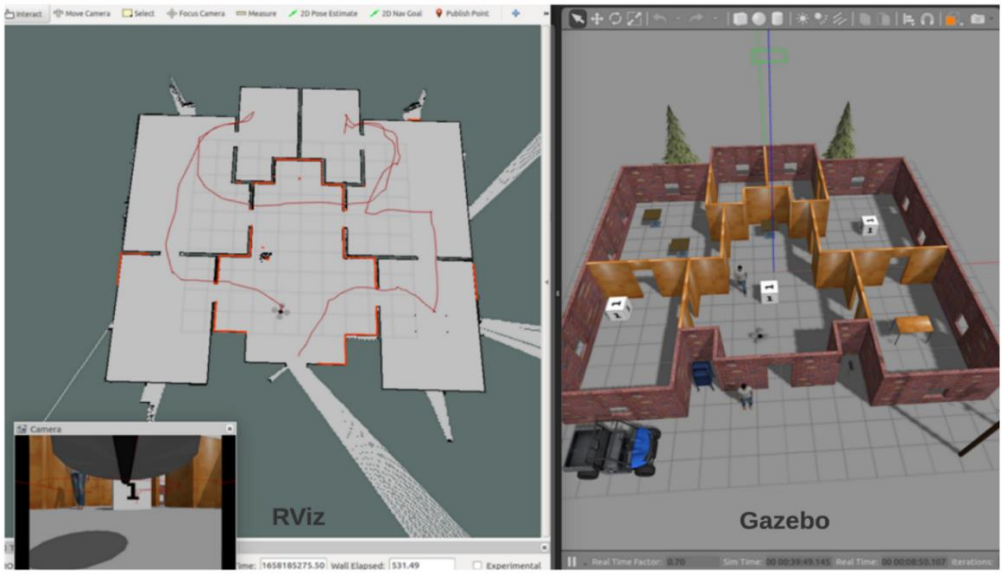
```
$ cd ~/hector_quadrotor_tutorial
$ source devel/setup.bash
```

Se procede a la ejecución de la primera tarea complementaria de SLAM bajo el escenario HOUSE.

```
$ roslaunch hector_quadrotor_demo MAIN_SLAM.launch
```

En el caso que se desee trabajar bajo el escenario LABERINTO se ejecuta el siguiente comando.

```
$ roslaunch hector_quadrotor_demo MAIN_SLAM2.launch
```



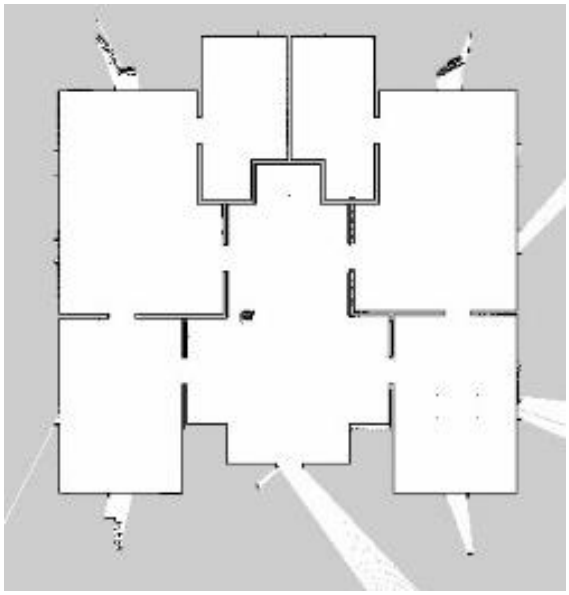
A continuación, se activan los motores del Quadrotor y el nodo de teleoperador genérico para el control de robot aéreo y mapear el escenario de manera manual.

```
$ rosservice call /enable_motors "enable: true"
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

El operador realiza el mapeo manual hasta obtener un mapa bidimensional completo para la Navegación Autónoma. Se prosigue a guardar el plano en un directorio de preferencia en el equipo por medio del siguiente comando.

```
$ rosrun map_server map_server ~f ~/HOUSE_1
```

Se guardara dos tipos de archivos de extensión .png y .yaml con el nombre colocado a disposición. El archivo .png muestra el mapa guardado, el cual se presenta en la siguiente imagen.

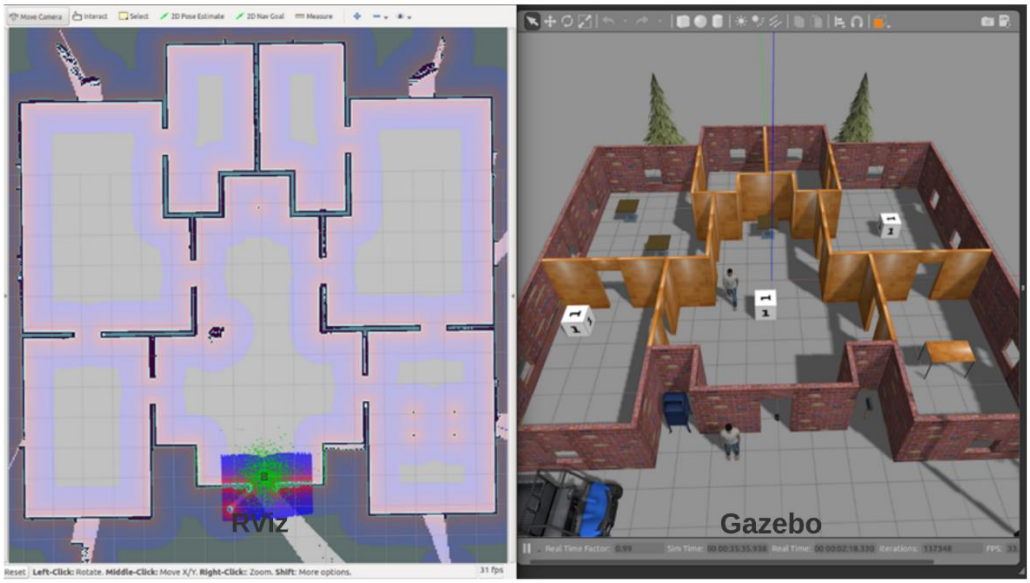


Siguiente se cierra la tarea referente al SLAM insertando Ctrl + C en el terminal. Próximamente se ejecuta la segunda tarea que es la Navegación Autónoma efectuando su archivo de lanzamiento, y tomando como referencia el plano guardado con anterioridad.

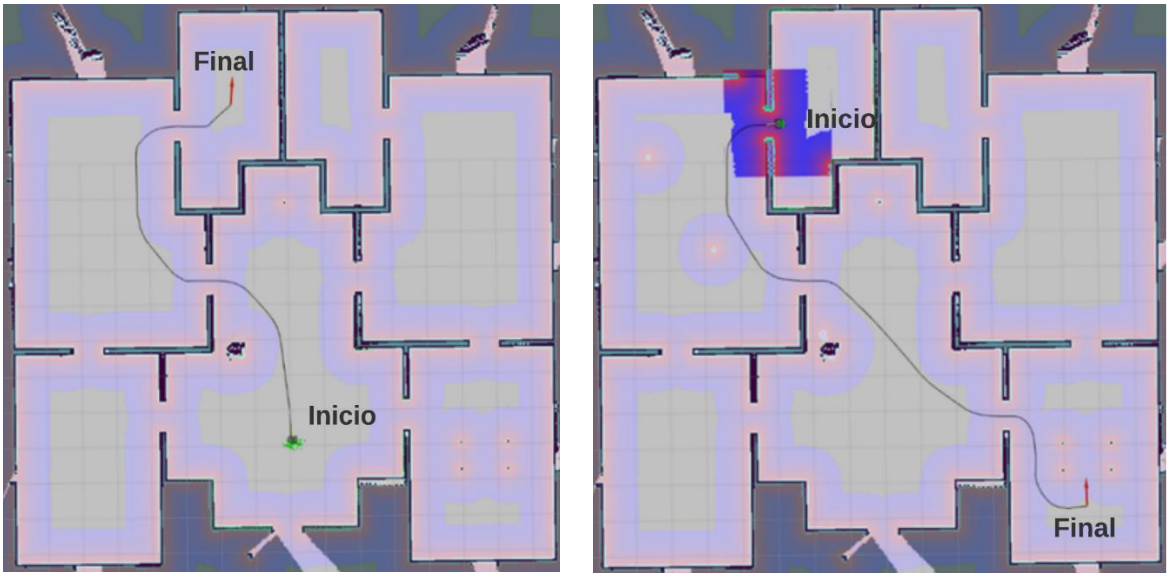
```
$ roslaunch turtlebot3_navigation MAIN_AUTO.launch map_file:$HOME/HOUSE_1.y
aml
```

Para el caso que se este trabajando bajo el escenario LABERINTO se ejecuta el siguiente comando. Cabe recalcar que para esta opción se debería ya tener así mismo un mapa bidimensional de referencia para la Navegación Autónoma previamente creado por la tarea de SLAM y guardado con un nombre por afinidad bajo el mismo escenario de LABERINTO.

```
$ roslaunch turtlebot3_navigation MAIN_AUTO2.launch map_file:$HOME/LAB_1.yaml
```



Una vez se ejecuten los dos ambientes de simulación tanto RViz como Gazebo se procede a colocar puntos destinos en el mapa referencial por medio del botón *2D Nav Goal* en RViz para la verificación de la Navegación Autónoma por parte del robot móvil TurtleBot 3.



ANEXO VII. Links Videos Demostrativos

El siguiente link contiene los videos demostrativos del presente proyecto bajo el escenario HOUSE y bajo el escenario LABERINTO, así como los links de los archivos de simulación. También se encuentra un archivo Readme.txt en el que se describe la funcionalidad de la aplicación.

https://epnecuador-my.sharepoint.com/:f:/g/personal/emerson_aldas_epn_edu_ec/EvQyXp_-8R9Nts59e9OuDdIB6nFeESv0YjFoIThvfcRnKQ?e=ue9COL