UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INGENIERÍA, INDUSTRIA Y CONSTRUCCIÓN

CARRERA DE INGENIERÍA ELÉCTRICA

PRUEBAS DE GENERACIÓN DE MAPAS Y LOCALIZACIÓN SIMULTÁNEA (SLAM) EN UN ROBOT MÓVIL PARA INTERIORES

TRABAJO DE TITULACIÓN O PROYECTO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELÉCTRICO

AUTOR: CARLOS SANTIAGO GUAMAN QUIZHPI

DIRECTOR: ING. CARLOS ALBERTO FLORES VASQUEZ Msc.

CUENCA – ECUADOR

2020

DECLARACIÓN

Yo, Carlos Santiago Guaman Quizhpi, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento; y eximo expresamente a la Universidad Católica de Cuenca y a sus representantes legales de posibles reclamos o acciones legales.

La Universidad Católica de Cuenca puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y la normatividad institucional vigente.

Carlos Santiago Guaman Quizhpi

CERTIFICADOS

Certifico que el presente trabajo fue desarrollado por Carlos Santiago Guaman Quizhpi, bajo mi supervisión.

Ing. Carlos Flores Vásquez MsC.

DIRECTOR

DEDICATORIA

A Dios, por guiarme siempre por el camino del éxito, por darme vida y sabiduría para alcanzar mis metas como persona y como profesional, gracias por permitirme culminar mis estudios.

Mi tesis lo dedico de manera especial, con mucho amor y cariño a mis queridos padres: Rumaldo e Isabel, gracias por ser mi fuente de inspiración, gracias por apoyarme económica y moralmente, gracias por tanto esfuerzo y sacrificio echo por mí, a pesar de las duras y difíciles circunstancias vividas durante mi periodo de estudio por fin he logrado cumplir con tu sueño mi querido padre.

A mi adorable familia: mi esposa Aida, mis hijos Lenin y Evelyn, gracias por confiar en mis capacidades y poder conseguir este título para nuestro futuro, gracias por estar siempre a mi lado en todo momento, a pesar de muchas dificultades ustedes, siempre han sido el motor de mi vida.

Con mucho amor dedico también a mi hijo Kevin Guaman, a pesar de la distancia que nos encontramos tú, siempre has estado acompañándome en mis pensamientos durante todo el periodo de estudio.

De manera especial agradezco a mis queridos padres Rumaldo Guaman e Isabel Quizhpi, a mis hermanos María, Luis, Rosa, Daisy, Ángel y a todos mis amigos que a lo largo de mi carrera estuvieron ahí siempre apoyándome en todo momento.

Con la inmensa satisfacción del deber cumplido, estoy infinitamente agradecido con Dios, por darme vida y salud y sobre todo por permitirme cumplir mi sueño anhelado que con gran esfuerzo lo he conseguido el objetivo propuesto, culminar mi con éxito mi carrera de pregrado, mi Ingeniería.

Gracias a todos.

AGRADECIMIENTO

Agradezco infinitamente a todas las personas quienes me apoyaron para concluir con éxito este proyecto, mi agradecimiento especial a la Carrera de Ingeniería Eléctrica de la Unidad Académica de Ingeniería, Industria y Construcción de la Universidad Católica de Cuenca, por facilitarme los espacios físico para desarrollar las pruebas de mi investigación, especialmente también agradezco al Grupo de Investigación en Radiación Visible y Prototipado (GIRVyP), quienes con su proyecto “Robots meseros con enfoque en Robótica Social” facilitaron los equipos necesarios para el desarrollo del proyecto, mi reconocimiento especial al Ing. Carlos Flores Msc, quien fue el director de mi proyecto y me supo motivar pese a muchas dificultades encontradas, gracias por el apoyo brindado durante todo el tiempo que duró el proyecto.

RESUMEN

La siguiente investigación plantea el desarrollo de pruebas respecto a la localización y generación de mapas de forma simultánea, también conocida como SLAM.

Para abordar esta problemática se la dividió en dos partes: En la primera se analiza los programas, paquetes y algoritmos (Software) necesarios para la implementación de SLAM. En la segunda los requisitos tecnológicos (hardware), para trabajar con una plataforma mínima suficiente para realizar las pruebas de localización y mapeo simultáneos.

Respecto a la primera parte se determinó trabajar con el sistema operativo UBUNTU 16.04, ROS Kinetic (Robot Operating System) como entorno de trabajo además de programas y paquetes relacionados como: OPENNI 2, RPLIDAR, GAZEBO para la simulación y HECTOR SLAM para localización y mapeo simultáneos.

El hardware determinado para las pruebas del sistema consistió en un Robot Móvil Turtlebot 2 (Kobuki), un sensor tipo lidar RPLIDAR A3, cámara Orbbec Astra, ordenador Intel NUC i5 para el manejo del robot, sensores y periféricos, pantalla táctil de 13 pulgadas GECHIC 1303i. Adicional a esto un ordenador Toshiba Intel Core I5 con 6Gb RAM, que cumple con los requisitos para trabajar con Ubuntu y ROS.

En la parte de simulación se trabajó con el conjunto de herramientas de ROS, realizamos varias actividades de puesta a punto en nuestra PC como: instalación y configuración de UBUNTU, ROS y paquetes relacionados. El diseño de los espacios de simulación empezó con la importación del plano de la Unidad Académica de Ingeniería, Industria y Construcción de la Universidad Católica de Cuenca, realizado en CAD y exportado en formato JPG al editor GAZEBO para crear el nuevo mundo virtual, luego se inserta el mundo creado junto con el robot TurtleBot 2.

La parte real del robot, se ha conseguido realizando la conexión entre la PC del robot TurtleBot 2 y la PC personal, aquí ambos deben estar dentro de la misma red inalámbrica, con el fin de obtener una comunicación bidireccional entre los dos PC, dando lugar a ejecutar los comandos de ROS en forma remota.

De las pruebas realizadas con el robot TurtleBot 2 en simulación y real, se obtuvieron mapas creados mediante SLAM alcanzando el principal objetivo planteado.

Palabras clave: SLAM, ROBÓTICA MÓVIL, LIDAR, UBUNTU, ROS.

ABSTRACT

The following research proposes the development of tests regarding the location and generation of maps simultaneously, also known as SLAM.

To address this problem, it was divided into two parts: in the first, the programs, packages, and algorithms (Software) necessary for the implementation of SLAM are analyzed. In the second, the technological requirements (hardware), to work with a minimum platform sufficient to conduct the simultaneous location and mapping tests.

Concerning the first part, it was determined to work with the UBUNTU 16.04 operating system, ROS Kinetic (Robot Operating System) as a working environment in addition to related programs and packages, such as OPENNI 2, RPLIDAR, GAZEBO for simulation, and HECTOR SLAM for localization and simultaneous mapping.

The hardware determined for the system tests consisted of a Turtlebot 2 Mobile Robot (Kobuki), an RPLIDAR A3 lidar-like sensor, an Orbbec Astra camera, Intel NUC i5 computer for handling the robot, sensors, and peripherals, 13-inch GECHIC touch screen. 1303i. Additionally, a Toshiba Intel Core I5 computer with 6Gb RAM, which meets the requirements to work with Ubuntu and ROS.

In the simulation part, we worked with the ROS toolset, we carried out several setup activities on our PC, such as installation and configuration of UBUNTU, ROS, and related packages. The design of the simulation spaces began with the import of the plan of the Academic Unit of Engineering, Industry, and Construction of the Catholic University of Cuenca, made in CAD and exported in JPG format to the GAZEBO editor to create the new virtual world, then inserts the world created together with the TurtleBot 2 robot.

The real part of the robot has been achieved by making the connection between the TurtleBot 2 robot PC and the personal PC, here both must be within the same wireless network, to obtain a bidirectional communication between the two PCs, resulting in to run ROS commands remotely.

From the tests carried out with the TurtleBot 2 robot in simulation and real, maps created using SLAM were obtained, reaching the main objective set.

KEYWORDS: SLAM, MOBILE ROBOTICS, LIDAR, UBUNTU, ROS.

ÍNDICE DE CONTENIDOS

LISTA DE FIGURAS

FIGURA .....PÁG.

LISTA DE TABLAS

TABLA.....PÁG.

LISTA DE ANEXOS

ANEXOS..... PÁG.

CAPÍTULO 1

1. INTRODUCCIÓN

En la actualidad los estudios y trabajos de investigación referentes a la robótica han sido empleados en la exploración espacial, marítima, entornos aéreos, entre otros y eso ha permitido potenciar cada vez más los conocimientos de los investigadores sobre navegación.

Existe varios trabajos de investigación sobre robótica realizados a lo largo de este tiempo, eso nos ha brindado las pautas necesarias para realizar el presente proyecto que está directamente relacionado con la navegación autónoma de vehículos y robots aplicando la técnica probabilística SLAM, esta técnica genera mapas y localiza simultáneamente la posición del robot durante la movilización de un punto a otro. La comunidad científica ROS ha planteado muchas herramientas sobre el tema y es por eso que nosotros hemos trabajado en la aplicación. En este proyecto se trabajó con el robot móvil TurtleBot 2, es un modelo de robot móvil construido para emplearse en el campo de la investigación y de la educación. La estructura del robot dispone de una plataforma robótica móvil equipada con sensores 2D y 3D, por lo que se logró que el robot pueda alcanzar una navegación autónoma en interiores.

Para demostrar la efectividad de la técnica probabilística de SLAM, que es la más usada en la resolución del problema de SLAM, aplicada en el robot TurtleBot 2, se consideró emplear un sensor RPLIDAR A3 y una cámara 3D marca Orbbec Astra. Con el objetivo de lograr una efectividad considerable de la técnica, se realizó varias pruebas de ejecución y medición.

Para justificar lo expuesto primero partimos de la determinación de las características y especificaciones del hardware y software para el equipo de cómputo, luego en nuestra computadora se realizó la descarga e instalación de; el sistema operativo GNU/LINUX con distribución UBUNTU Xenial 16.04, el conjunto de herramientas ROS y los algoritmos necesarios para SLAM. Se logró administrar correctamente al robot, se identificó correctamente todos los diferentes paquetes que se encuentran dentro de la comunidad ROS. También logramos trabajar con archivos que se encuentran dentro de la plataforma Github, esto nos sirve para verificar los algoritmos que se encuentran alojados como proyectos de desarrollo colaborativo.

Las primeras pruebas realizadas son; simulaciones del robot en el computador, para el efecto iniciamos generando los planos de la Unidad Académica en AutoCAD, luego importamos desde ROS Gazebo, para poder construir un mundo idéntico al real de la Universidad, ejecutando un sinnúmero de comando logramos mover al robot en el entorno, mientras que de navegación física

del robot se realizó en los corredores interiores de la Unidad Académica de Ingeniería, Industria y Construcción de la Universidad Católica de Cuenca.

Con el fin de garantizar la eficacia en el funcionamiento del robot TurtleBot 2 y el computador administrador, verificamos sus conexiones y el correcto funcionamiento del sistema operativo robótico.

Es necesario saber que para que un robot pueda navegar en un entorno se requiere de tres tareas que son: El modelado, la localización y la planificación de su recorrido.

El modelado se logra con la extracción de información que brindan los sensores, con esta información se puede saber cómo es el ambiente del entorno y se plantearía la siguiente pregunta "¿Cómo es el entorno que nos rodea?".

La localización es saber cuál es la posición del robot de acuerdo al mapa, dividiéndose en una localización inicial del robot respecto así mismo, es decir el eje de coordenadas se ubica en un extremo de la base del robot y una localización global que se refiere a que no hay una posición de partida del robot. La pregunta que debe responder el robot es "¿Dónde estoy?".

La planificación de su recorrido tiene que ver con el grado de eficacia que se orienta el robot para que llegue a su meta planteada durante toda su trayectoria de camino. La pregunta seria "¿Cómo puedo llegar a un lugar determinado?". [1]

El documento, recoge todas las actividades realizadas como son; diseños de planos, simulaciones y pruebas realizadas en la ejecución del proyecto de investigación, todo está contemplado y organizado en cuatro capítulos que se expone a continuación.

Capítulo 1: Muestra la problemática y alcance del proyecto, Introducción de lo desarrollado, descripción de los objetivos a efectuar, y la descripción del estado del arte, donde se indica la fundamentación y justificación teórica de lo realizado.

Capítulo 2: Describe la metodología empleada en el estudio, desarrollo de lo Implementado, detalle paso a paso de las pruebas realizadas, y funcionamiento.

Capítulo 3: Indica las pruebas realizadas, análisis de resultados de la problemática y alcances obtenidos,

Capítulo 4: Muestra las conclusiones y recomendaciones finales obtenidas a lo largo de todo el proyecto, posibles modificaciones y futuras aplicaciones mediante esta investigación.

1.1. Antecedentes

SLAM (Mapeo y localización simultánea) es un método que permite resolver los problemas que surgen al colocar un robot móvil en un determinado entorno y posición desconocida. El robot móvil o vehículo autónomo, debe ser capaz de construir de manera incremental un mapa con características de su entorno y mediante dicho mapa pueda determinar su propia localización.

Basándose en este contexto el proyecto abarca, el armado, puesta a punto de un robot móvil y la aplicación del sistema operativo robótico (ROS) hasta los algoritmos más utilizados para implementar SLAM.

El objetivo principal de este proyecto es realizar pruebas con el robot móvil para ello se utilizará el Framework (entorno de trabajo) ROS y algoritmos de la comunidad ROS para implementar SLAM, obteniendo al final un análisis de los resultados con los sensores (Cámara 3D y LIDAR) para estos algoritmos y que permita tener una visión más clara acerca de la eficacia de los mismos. Este trabajo obtendrá métricas y comparativas de los métodos para la aplicación de SLAM con el robot TurtleBot 2.

1.2. Objetivos

1.2.1. Objetivo general

Probar la eficacia del SLAM en el robot TurtleBot 2 considerando un sensor RPLIDAR A3 y una cámara 3D marca Orbbec Astra.

1.2.2. Objetivos específicos.

- Implementar el sistema operativo robótico ROS para la administración del robot.
- Identificar a los diferentes algoritmos de implementación de SLAM expuestos en la comunidad ROS.
- Realizar pruebas de ejecución y mediciones con cada uno de los sensores (Lidar RPLIDAR A3 y cámara 3D Orbbec Astra).

1.3. Problema

Desde hace muchos años la comunidad científica ha venido estudiando los problemas del Mapeo y localización simultánea (SLAM). Sin embargo, uno de los problemas es la complejidad que tiene que ver con el aspecto computacional de las soluciones planteadas al SLAM.

El modo en que el robot perciba su entorno, la cantidad de información disponible, así como las técnicas empleadas en su procesamiento, interpretación y combinación, determinarán los recursos computacionales necesarios para la construcción del mapa.

Todos estos recursos no son ilimitados menos aún si es que el objetivo es regirse a los recursos disponibles de un computador Intel Core I5 de octava generación. Aunque se podría decir que el problema principal sería saber la eficacia o idoneidad de los algoritmos utilizados y la posibilidad de obtener soluciones cuya implementación sea posible en tiempo real, sabiendo que cualquier solución al problema del SLAM siempre se tendrá la necesidad de trabajar con grandes volúmenes de información, la misma que puede estar contaminada de ruido que de una u otra manera la mayoría de las veces no son sino aproximaciones a la realidad. Por tanto, las soluciones más exitosas hasta el momento están basadas en técnicas probabilísticas.

1.4. Alcance

Basados en la utilización de las técnicas más utilizadas como es la probabilística y que han obtenido mejores resultados a la hora de abordar el problema del SLAM, en este trabajo se realizará pruebas con los sensores LIDAR y una cámara 3D Orbbec Astra. La resolución de LIDAR permitirá decidir directamente si el robot puede construir un mapa de forma rápida y precisa, mientras que la cámara permitirá el reconocimiento de formas y profundidad mediante la captura de imágenes; todo esto se abordará con los diferentes algoritmos existentes en la comunidad científica ROS.

1.5. Justificación

En la actualidad la tecnología ha permitido avanzar significativamente en el área de la robótica, tanto así que se han realizado muchos estudios o trabajos de investigación relacionados a la misma, como es el caso del SLAM para el mapeo y la localización simultánea en vehículos autónomos.

Los vehículos autónomos son robots necesarios para solucionar los problemas planteados por la comunidad científica, para ello este tipo de vehículos han sido utilizados en la investigación espacial, marítima y en entornos aéreos, como es el caso por ejemplo del robot “Oportunity” de la NASA que es utilizado para realizar estudios del terreno en un entorno marciano.

Existen plataformas que permiten al usuario poder programar y controlar los movimientos del robot, este es el caso del sistema operativo robótico ROS, ampliamente utilizado por universidades e investigadores de todo el mundo. Estas universidades son tanto nacionales como extranjeras que hacen uso de este sistema operativo como es el caso nacional por ejemplo de la Universidad Técnica del Norte en la cual implementaron una plataforma robótica móvil para que tanto los estudiantes como docentes puedan realizar experimentos. [1]

La Universidad de Antofagasta de Chile es otro claro ejemplo del uso de este sistema no solo para realizar pruebas experimentales sino más bien para un uso más científico orientado a la tele-operación en un observatorio astronómico. [2]

Ros fue desarrollado inicialmente por el laboratorio de Inteligencia Artificial de Standford posteriormente también fue desarrollado en cooperación conjunta por varias instituciones con el objetivo de poder operar a múltiples robots, por tal motivo su característica principal es que es un framework que tiene su propia comunidad científica con acceso a los repositorios y que están disponibles al público.[3]

Las pruebas se pueden realizar de manera online, esto quiere decir que la información es procesada en el mismo robot mientras éste navega en su entorno. Para ello se dispone actualmente del equipamiento necesario tanto de hardware como de software; el Grupo de Investigación en Radiación Visible y Prototipito (GIRVyP) de la Universidad Católica de Cuenca permitirá el acceso al robot TurtleBot 2 para las pruebas y por otra parte el sistema operativo robótico ROS al igual que otros paquetes informáticos necesarios se encuentran disponibles para el acceso al público (Open Source); también se dispone del hardware computacional que puede ser utilizado para la operatividad tanto del robot como del ROS.

1.6. Fundamentación teórica

1.6.1. Estado del arte.

Actualmente existen muchos tipos de robots: androides, móviles, médicos, etc. Este proyecto se centra en aplicación del robot móvil TurtleBot 2 para SLAM. Este robot es de código abierto y bajo costo, es utilizado en el campo educativo e investigativo. El equipamiento de este robot se

encuentra conformado por: una base de robot Kobuki, sensor Orbbec Astra (3D), sensor RPLIDAR A3 y un computador Intel NUC I5.

La navegación autónoma de cualquier robot es uno de los principales problemas que presentan los mismos, esto debido a su complejidad y dinamismo ya que depende de factores como son: ambientales, interacción, personas o cualquier otro factor de cambio dentro de su entorno. Sin embargo, existen trabajos de investigación realizada por la comunidad científica en donde brindan aportes acerca del uso de los robots para la navegación autónoma utilizando un TurtleBot 2.

Un proyecto interesante realizado por Vivet [4], en el que se refiere a la programación de comportamientos de un robot autónomo, el cuál utilizan el robot TurtleBot 2 programándolo para que realice comportamientos autónomos, es decir que dentro de su entorno global sea capaz de desplazarse hacia distintos lugares dados por el usuario y a su vez pueda evadir objetos que interrumpieran su trayectoria, además de esto tiene la capacidad de poder encontrar una estación de carga al momento de detectar que su batería este en nivel bajo. Para lograr los objetivos de este proyecto utilizaron el sistema operativo robótico ROS, el robot TurtleBot 2 y dos computadoras; la una (máster) permite controlar los procesos del robot y la otra (estación de trabajo) para dar órdenes al máster. Los resultados obtenidos en este proyecto es que el robot pudo alcanzar cualquier coordenada dada dentro de su mapa global, también logró evadir los obstáculos dinámicos que se le fueron presentando.

Desde el punto de vista o enfoque probabilístico con técnicas de estimación y filtros bayesianos, el trabajo de grado realizado por Aldas [5], se determina los elementos de hardware a utilizar para realizar pruebas acerca del SLAM de acuerdo a este enfoque. El sensor de detección de luz y rango (LIDAR) es el que se destacó. Para la evaluación del funcionamiento de la plataforma robótica móvil establecieron un protocolo de pruebas que constituía en un test individual de los componentes y un test integral, sobre el cual se ejecutaron los paquetes del SLAM en el sistema operativo robótico ROS en diferentes ambientes controlados; se obtuvieron resultados satisfactorios.

RPLIDAR es un sensor con una tecnología basada en triangulación láser que permite obtener de la mejor manera la distancia entre el sensor y un objeto, esto gracias a la diferencia entre el tiempo de salida y el tiempo de regreso del haz de luz reflejado. Por lo tanto este tiene la capacidad de generar nubes de puntos de información en alta resolución lo que permite realizar representaciones tanto bidireccionales como tridimensionales de los diferentes objetos del entorno Ponce [6]. En el trabajo del autor anteriormente citado, se utiliza un sensor RPLIDIAR A1

similar al A3 (omnidireccional de 360 grados) los cuáles difieren en su rango de distancia y frecuencia de escaneo (A1: 6 metros 5,5hz, A3: 10-25 metros 10hz). Éste se lo utilizó como escáner laser 2D para realizar un barrido con un haz de luz de 360 grados para producir una nube de puntos en 2D y mediante el uso de una plataforma giratoria acoplada en el sensor para realizar la reconstrucción en 3D. Los datos son procesados en MATLAB 2017b para el proceso de reconstrucción basándose en el uso de criterios matemáticos. Con la implementación de herramientas el usuario mediante una interfaz gráfica puede manipular la nube de puntos para la visualización de la misma. Los resultados obtenidos son nubes de puntos filtradas que son las representaciones en 3D de los entornos tanto en interiores como en exteriores.

En la resolución del problema del SLAM los sensores y los algoritmos cumplen una función primordial para la obtención de los datos y su correcta interpretación en la generación de los mapas, es por eso que la comunidad científica internacional han desarrollado una serie de algoritmos para dar soluciones al SLAM como es el caso del trabajo de Monroy [7], en donde desarrolla un algoritmo que proporciona mapas de ambientes simulados a partir de datos obtenidos por sensores; los resultados obtenidos mostraron que el algoritmo logró darle autonomía al robot generando mapas y mostrando la ubicación del mismo.

Otro caso a destacar en cuanto al desarrollo de algoritmos SLAM es el que se utiliza imágenes omnidireccionales para poder estimar la posición y orientación de un robot móvil y al mismo tiempo crea un mapa de su entorno método. El utiliza un sistema catadióptrico montado en el robot el mismo que está formado por una cámara que apunta hacia un espejo convexo y que proporciona información mediante imágenes omnidireccionales con un campo de visión de 360 grados. El método funciona de la siguiente manera:

- El robot calcula su posición y orientación creando un nuevo nodo en el mapa.
- Luego detecta cierres de bucle entre el nuevo nodo y los nodos del mapa.
- Finalmente, el mapa se optimiza utilizando un algoritmo de optimización y los cierres de bucle detectados.

Los resultados se analizaron creando mapas con optimización y sin optimización mostrando la eficacia del método. Berenguer [8]

Si se habla de robótica se puede ver claramente que el sistema operativo robótico (ROS) es ampliamente utilizado debido a muchas de sus características que brinda a los investigadores ya que en su comunidad se aborda el problema del SLAM mediante los diferentes algoritmos de código abierto, partiendo de ahí existe una investigación realizada por Ibragimov & Afanasyev [9] en donde realizan una serie de pruebas con los algoritmos de SLAM sobre una plataforma

robótica móvil utilizando los sensores LIDAR, cámara monocular, ZED cámara y Kinect los cuáles brindan la información para que los diferentes algoritmos puedan determinar la velocidad y precisión de los mismos. Hay que destacar que en las pruebas realizadas el sensor LIDAR es el que más se destacó por su precisión y velocidad.

1.6.2. Plataforma Robótica TurtleBot.

El TurtleBot es un kit de Robot personal que tiene un bajo costo en el mercado robótico y tiene una particularidad principal, que posee un software de código abierto. Este robot es considerado el más popular por el mismo hecho de su bajo costo y que es de código abierto, por tal razón es muy utilizado tanto para la educación como para la investigación, ofreciendo una plataforma ideal para desarrollo de cualquier tipo de prácticas para el aprendizaje. TurtleBot 2 fue credo por Willow Garage, Melonee Wise y Tutty Foote en noviembre de 2010. [10]

Fig. 1: Robot TurtleBot 2

Fuente: Open Source Robotics Foundation [10]

Este es capaz de ejecutar algoritmos SLAM (mapeo y localización simultánea) para construir mapas que pueda servir al robot en un ambiente específico. Además de estas características el robot puede ser controlado de forma remota mediante un computador portátil, Tablet o cualquier teléfono inteligente que esté basado en Android, incluso tiene la capacidad de manipulación de

objetos al agregar el accesorio de brazo robótico propio de la plataforma. Los usuarios pueden descargar el SDK de la wiki de ROS

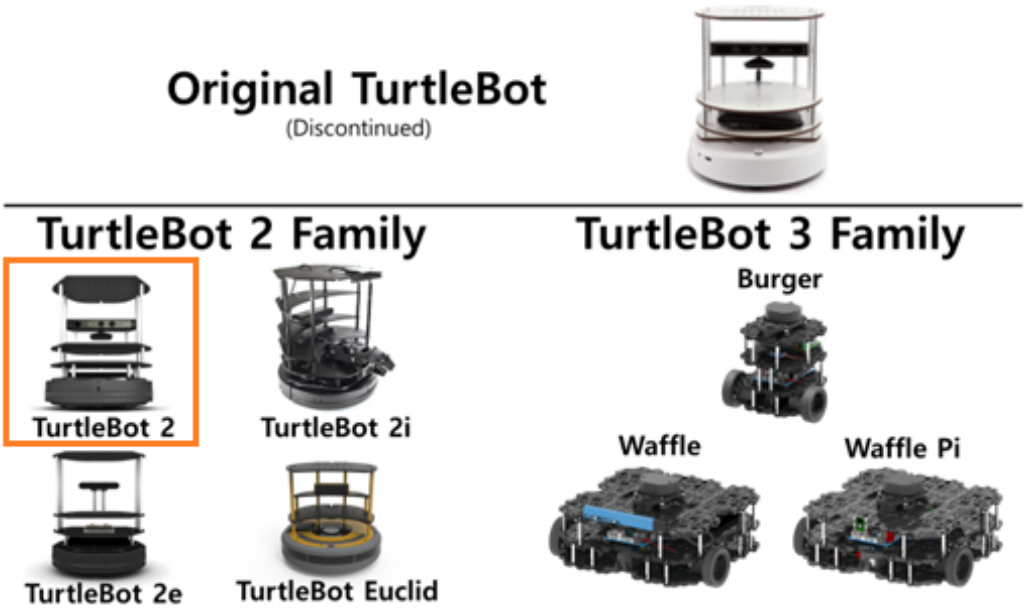


Fig. 2: Familia de Modelos del TurtleBot

Fuente: Open Source Robotics Foundation [10]

A continuación, se muestra una tabla con cada uno de los modelos existentes y sus principales características.

Tabla 1: Partes del Robot TurtleBot 2

ROBOT	BASE	BATERÍA	SENSOR	COMPUTADOR	KIT
TurtleBot 2	Yujin Kobuki	2.200 mAh	Kinect	Portátil Asus 1215N con un procesador de doble núcleo, cargador rápido, base de carga	Une todo y agrega sensores futuros

Fuente: Open Source Robotics Foundation [10]

El equipamiento del robot TurtleBot 2 que fue utilizado para el desarrollo de este trabajo consta de las siguientes partes:

- Cámara Orbbec Astra
- Controlador NUC-i5
- Estación de carga
- Cable US NEMA 1-15

- Batería adicional 4S2P (4400 mAh)
- Disco extra NUC 500 GB HDD
- RPLIDAR A3 (25m) laser scanner

1.6.3. Sistema Operativo Robótico (ROS)

El sistema operativo robótico ROS fue desarrollado originalmente en 2007 bajo el nombre de switchyard por el laboratorio de inteligencia artificial de Stanford, a partir del año 2008 el desarrollo continúa principalmente en el instituto de investigación robótico Willow Garage que cuenta con más de veinte instituciones colaborando en un modelo de desarrollo federado.

ROS es una plataforma de desarrollo open source para sistemas robóticos. Esta plataforma no es exactamente un sistema operativo sino más bien un framework que posee un set de herramientas, las mismas que proveen la funcionalidad de un sistema operativo. Dentro de estas herramientas están una serie de servicios y librerías que facilitan de manera simple la creación de aplicaciones para robots.

En cuanto tiene que ver al desarrollo ROS permite el uso de diferentes lenguajes de programación como son: Python, C++, Lisp y Java (fase experimental). Este sistema puede ser ejecutado principalmente en sistemas UNIX como Ubuntu y MAC OS X aunque también se puede encontrar soporte para otros sistemas como Fedora, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows [11]

1.6.3.1. Características principales del Sistema Operativo Robótico (ROS)

Las principales características del ROS son:

Código abierto y libre: ROS es de código libre bajo términos de licencia BSD, se encuentra disponible al público en general, es decir su uso es tanto comercial como para la investigación

Peer to peer: Al tratarse de un sistema distribuido los diferentes procesos se comunican entre sí utilizando una topología peer to peer, lo que permite utilizar un canal nuevo para la comunicación entre dos procesos distintos, evitando de esta manera usar un servidor central para comunicar todos los procesos.

Lenguajes: permite trabajar en diferentes lenguajes de programación como son: Python, C++ y Lisp. Contiene además en fase experimental bibliotecas en Java y Lua.

Herramientas: dispone de una serie de herramientas para realizar tareas como:

- Navegación de ficheros
- Modificación de parámetros de configuración del robot, proceso o driver
- Visualización de la topología de los procesos en ejecución
- Realizar la comunicación entre los diferentes procesos

1.6.4. El problema del SLAM.

La localización y mapeo simultáneo o también localización y modelado simultáneo (SLAM), es una técnica que usan los robots y vehículos autónomos que les permite construir un mapa en un entorno totalmente desconocido en el cual este situado el Robot, permitiendo a su vez estimar su trayectoria al tener que desplazarse en dicho entorno.

Pueden existir escenarios en los que se puede conocer de antemano el medio en donde el robot se trasladará y hay la posibilidad de proporcionar un mapa del sitio. Otro escenario podría ser que el robot deba moverse en un entorno desconocido.

1.6.4.1. Definición del SLAM

Aprender de un mapa y localizar el robot simultáneamente así lo definimos a SLAM, además podemos decir que el problema difícil es, que se necesita de un mapa para la localización y de una buena estimación de la postura del robot para el mapeo. Es importante citar que la estimación de la postura de un robot y el mapa del entorno se pueden dar al mismo tiempo. Dotando de una plataforma lo que hace que el robot emplee autonomía necesaria de tal manera que pueda realizar el proceso de localización y mapeo simultáneamente en cualquier entorno inexplorado. A continuación, definimos por separado lo siguiente:

Localización: dado un mapa deduce la ubicación.

Mapeo: dado el lugar genera un mapa

1.6.4.2. Estudio del SLAM.

Existen escenarios mucho más complejos que los citados anteriormente, puesto que el robot puede encontrarse en un punto donde desconoce tanto el entorno y su ubicación. A partir de ahí el robot deberá generar un mapa y mantener su ubicación dentro de su entorno. Para ello resulta una tarea muy compleja ya que el robot tendrá que valerse de un mapa para poder localizarse de forma precisa y para poder crear un mapa es primordial que el robot esté localizado en forma

precisa. Para ello el SLAM se encarga de estudiar esta problemática y es utilizada por ejemplo para: exploración espacial, catástrofes, robots domésticos y vehículos autónomos.

1.6.4.3. Dificultades del SLAM

En el problema del SLAM existen factores que incrementan la dificultad para poder estimar precisamente la ubicación y el mapa de un entorno específico. Algunos de estos factores son:

- Ruido de los sensores
- Desplazamiento impreciso
- Simetrías del entorno
- Visibilidad parcial
- Entorno dinámico
- Capacidad de cómputo
- La cartografía entre las observaciones los puntos de referencia se desconocen
- Escoger asociaciones de datos equivocadas puede tener consecuencias catastróficas (divergencia)

1.6.4.4. Enfoques del SLAM

Para la solución del SLAM existen dos grandes enfoques que son: bioinspirados y probabilísticos. Estos se diferencian primordialmente en la forma que procesan la información de entrada para poder encontrar la mejor estimación acerca de la ubicación del robot y mapa del entorno.

El enfoque más utilizado actualmente es el probabilístico ya que se ha logrado implementaciones en ambientes sumamente complejos y grandes. A continuación, se detalla el enfoque probabilístico que es el cual se vamos a utilizar para desarrollar el presente proyecto de investigación y comprobación.

1.6.4.5. SLAM probabilístico

Este enfoque se centra en encontrar la distribución de probabilidad de la posición del robot y del mapa del entorno a través del tiempo. En la figura 3 se puede analizar este concepto de estimación como una distribución de probabilidad.

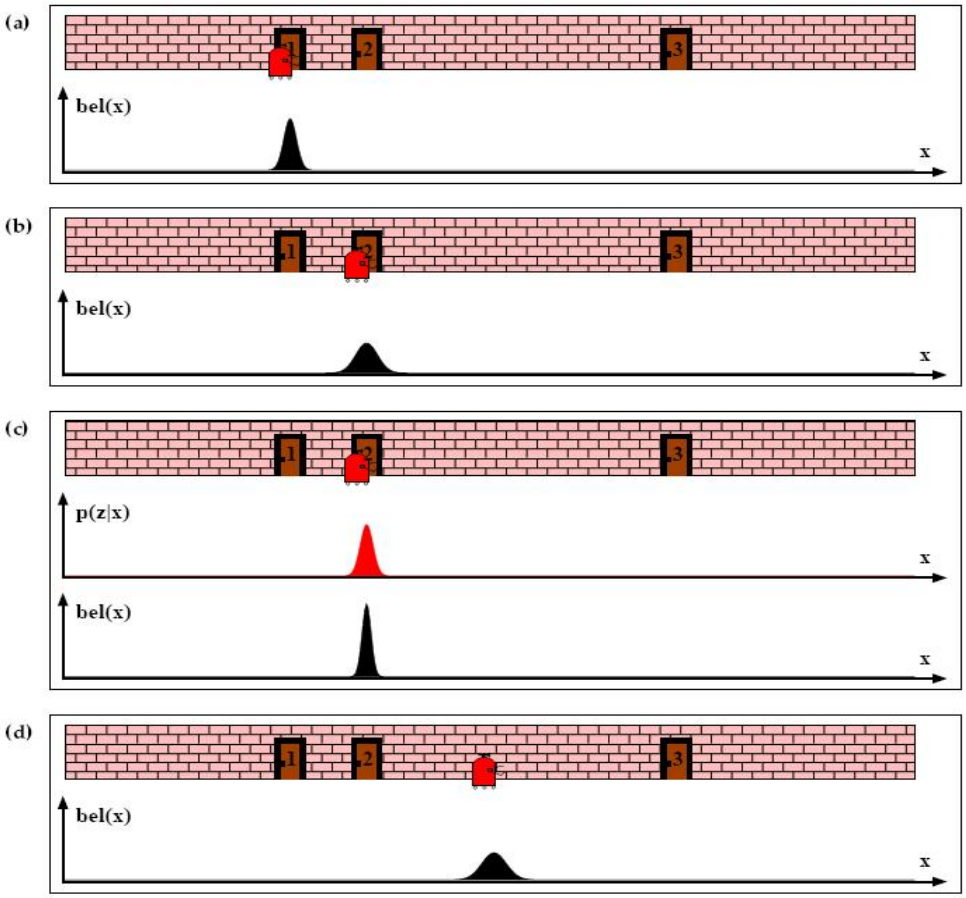


Fig. 3: Estimación de Probabilidad

Fuente: Vázquez A. Ramos F. [12]

Por lo tanto, la estimación de la probabilidad puede formularse con variables estocásticas como:

$$\mathrm{p}(\mathrm{X}_{1:\mathrm{t}} \vee \mathrm{Z}_{1:\mathrm{t}}, \mathrm{U}_{1:\mathrm{t}})$$

En donde:

- $X_{1:t}$: posición a estimar en el instante de tiempo
- $U_{1:t}$: información de movimiento propio
- $Z_{1:t}$: información sensada

El tiempo t se suele tomarlo como un conjunto discreto de instantes, los cuales suelen coincidir con los momentos en el que el robot recibe la información. Los subíndices en dichas variables estocásticas (X_i, Z_i, U_i) corresponden en cambio al instante de tiempo al que corresponden estas variables.

Dentro de la literatura científica la probabilidad puede ser representada por:

$$\mathrm{bel}(\mathrm{X}_{1:\mathrm{t}} \vee \mathrm{Z}_{1:\mathrm{t}}, \mathrm{U}_{1:\mathrm{t}})$$

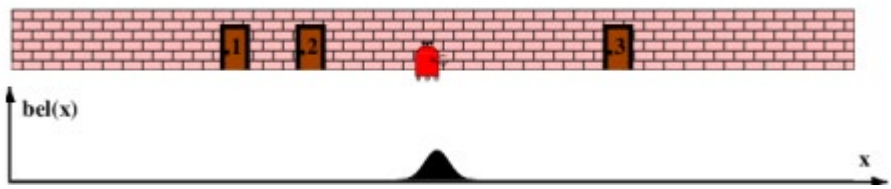


Fig. 4: Agente en Movimiento en un entorno con 3 Objetos Distinguibles

Fuente: Andrade, Federico Tejera, Gonzalo [13]

Dicho de otra manera se forma la ubicación y el mapa como una variable vectorial $X_{1:t}$

Tabla 2: Variables Vectoriales

0	0	0
0.1	0.8	0.1
0	0	0

Fuente: Andrade, Federico Tejera, Gonzalo [13]

Se estima utilizando la información del movimiento propio $U_{1:t}$ y la información sensada $Z_{1:t}$

En definitiva resolver el problema del SLAM implica estimar la distribución de probabilidad para la variable que representa el estado del sistema $X_i$en cada instante del tiempo discreto de la variable $X_{1:t}$

1.6.4.6. Técnicas

Las principales técnicas son las siguientes:

- Filtros de Kalman
- Filtros de Partículas
- Optimización sobre grafos

1.6.4.7. Filtros de Kalman

Los filtros de Kalman son un conjunto de ecuaciones matemáticas las cuales proveen una solución recursiva óptima mediante el método de mínimos cuadrados. Este filtro propone

actualizar el estado del sistema a medida que se obtiene información de odometría y censado por lo tanto se convierte en una solución al problema online SLAM. [13].

Para la creación del mapa de un entorno y que simultáneamente se use para localizar el robot se usa el filtro de Kialman extendido (FKE), que es una técnica de linealización de modelos dinámicos no lineales en donde se aplica el filtro de Kialman.

1.6.4.8. Filtros de partículas

Particularmente el filtro de partículas (PF) consiste en un método aplicado a sistemas no lineales que me permiten resolver problemas de estimación mediante una serie de estimaciones puntuales que involucran el ruido no Gaussiano o multimodal, todo en el modelo de movimiento, es conocido también como el método secuencial de muestreo de Monte-Carlo, basado en la aproximación de la función de densidad de probabilidad (PDF). Representa la densidad de probabilidades y utiliza puntos másicos (partículas) que son los estados posibles del proceso distribuidos en su espacio de estados. De esta manera este filtro brinda una manera simple y efectiva para modelar procesos estocásticos con funciones de distribución, probabilidad y modelos de propagación arbitrarios. [14]

1.6.4.9. Optimización sobre grafos

El SLAM de grafos o conocido también como graphSLAM, a diferencia de los algoritmos de filtros resuelve el problema utilizando un algoritmo basado en marcas. Este algoritmo modela el problema del SLAM como un grafo en donde las posiciones del robot X_i y marcas m^i son representadas por nodos. [15]

1.6.5. Sensores

Los sensores hoy en día son un equipamiento muy importante para los robots ya que facilitan o ayudan conjuntamente con los diferentes algoritmos para elaboración de los mapas y localización del robot. Estos sensores existen para todo tipo de aplicaciones y sus precios obviamente varían de acuerdo a su exactitud (a mayor precisión mayor costo). A continuación, se detalla los sensores a utilizarse en el desarrollo del presente trabajo.

1.6.5.1. Lidar

El RPLIDAR A3 es un sensor cuyo sistema permite una solución de scanner láser 2D de 360 grados dentro de un rango de 25 metros, los datos de nubes de puntos 2D que produce se puede usar para el mapeo y localización modelando un entorno. La frecuencia de escaneo del RPLIDAR A3 puede alcanzar 15 Hz (ajustable entre 10 Hz-20 Hz), funciona en sentido horario omnidireccional de 360 grados para su entorno.

En definitiva, el RPLIDAR A3 es básicamente un sistema de medición por triangulación láser que puede trabajar tanto en ambientes interiores como exteriores (sin luz solar).[16]

Fig. 5: RPLIDAR A3

Fuente: ROS components [17]

Tabla 3: Especificaciones de RPLIDAR A3

Nº Modelo	RPLIDAR A3 (25m)
Dimensiones	72,50 mm x 41 mm x 76 mm
Peso	190 g
Resolución angular	0.225° o 0.36°
Rango	25 metros
Distancia	8 - 25 m
Frecuencia de escaneo	15 Hz (ajustable entre 5 Hz-20 Hz)

Fuente: ROS components [17]

RPLIDAR emite señales láser infrarrojas moduladas las mismas que son reflejadas por el objeto a detectar. La señal de retorno es muestreada por el sistema de adquisición de visión. El sistema DSP (Procesamiento Digital de Señales) que está incorporado en RPLIDAR procesa los datos de muestreo, el valor de distancia de salida, el valor de ángulo entre RPLIDAR y el objeto mediante la interfaz de comunicación. El sistema de escáner de alta velocidad se monta sobre

un eje giratorio con un sistema de codificación angular (encoder) incorporado para realizar la exploración de 360 grados del entorno del sensor.

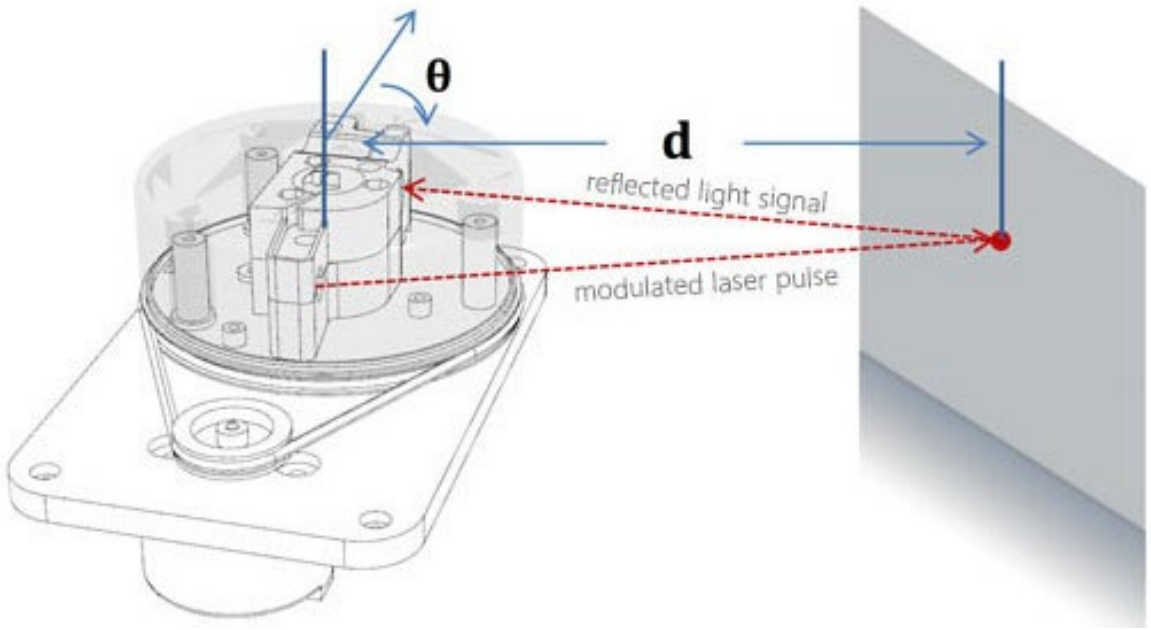


Fig. 6: Modo de Operación del Sensor RPLIDAR

Fuente: Robotics, Second [18]

1.6.6. Paquetes ROS para el SLAM

En el problema del SLAM antes de realizar la navegación primero se debe tener un mapa que permita conocer al Robot su entorno actual. En la comunidad ROS se puede encontrar varios paquetes para realizar SLAM, ya sea mediante visión por cámaras o usando sensores de rango como radares y láseres. A continuación, se detallan cada uno de ellos.

1.6.6.1. Visual SLAM

Vslam es un código de investigación experimental que no tiene soporte continuamente y que se debe utilizar bajo su propia responsabilidad [19]. En la figura 7 se observa el código en acción.

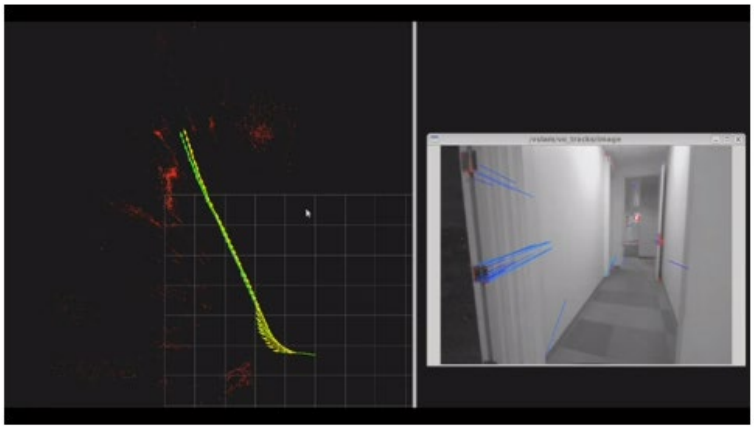


Fig. 7: *Funcionamiento* del Vslam

Fuente: Open Source Robotics Foundation [19]

1.6.6.2. Gmapping

Este paquete Gmapping contiene un soporte de ROS para el paquete externo OpenSlam Gmapping. Este paquete proporciona localización simultánea y mapeado basándose en la información del láser 2D montado en el robot, lo cual permite crear un mapa de cuadricula de ocupación 2D (similar a un plano de planta de un edificio) a partir de los datos que provienen del láser y la información de posición recogida por el robot móvil. [20]

Gmapping es un archivador de partículas Rao-Blackwellized altamente eficiente para aprender mapas de cuadrículas a partir de datos de rango por láser.[21]

Fig. 8: Mapa Generado con Gmapping del Campus Freiburg

Fuente: Burgard;, Giorgio Grisetti; Cyrill Stachniss; Wolfram [21]

1.6.6.3. Karto

La organización internacional independiente de investigación y desarrollo tecnológico llamada SRI internacional y que inicialmente formaba parte de la Universidad de Standford describe a karto 2.0 como el sistema SLAM de más alto rendimiento y con todas las funciones disponibles. Vendido comercialmente, karto es un proyecto de desarrollo avanzado dentro del centro de IA del SRI internacional, sus versiones comerciales y de código abierto ofrecen una solución integral que incorpora algoritmos avanzados para la generación de mapas, navegación autónoma y exploración.[22]

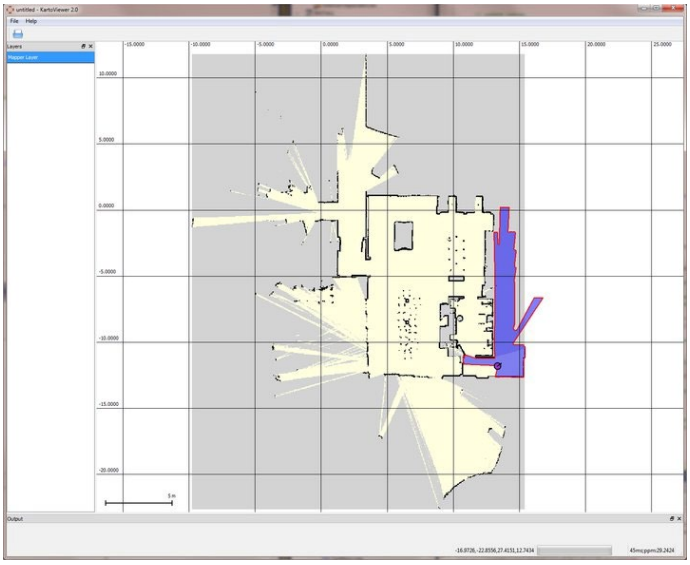


Fig. 9: Interfaz de Usuario para Karto 2.0

Fuente: Karto Robotics [23]

1.6.6.4. Hector SLAM

Hector_slam fue desarrollado por el Team Hector, este equipo comenzó en el 2009 como un esfuerzo interdisciplinario de investigadores de los departamentos de Ciencias de la computación e ingeniería mecánica de TU Darmstadt dentro del programa de doctorado GRK 1362. Hector Slam contiene paquetes ROS que permiten la realización del SLAM en entornos no estructurados como los que se encuentran en los escenarios de Búsqueda y Rescate Urbano (USAR) de la competencia RoboCup Rescue. [24]

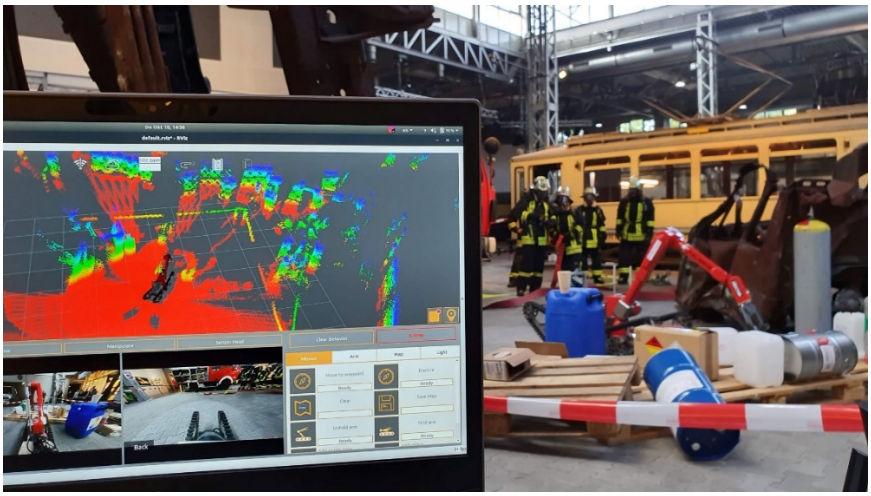


Fig. 10: Robot de Rescate Equipado con Capacidades Autónomas Inteligentes para Primeros Auxilios en Incidentes con Productos Peligrosos

Fuente: Team Hector [25].

El paquete de Héctor (SLAM) se utiliza precisamente para que los robots puedan navegar de forma autónoma, el método de odometría me permite reposicionar el robot en recorridos repetitivos, y mediante el sensor de detección y rango de luz (LIDAR) es posible posicionar al robot en cualquier ambiente escogido para las pruebas. También puede seguir los puntos de ruta y evitar obstáculos. Con el fin de obtener resultados experimentales del robot TurtleBot 2 tanto de manera real física como en simulación haremos uso del hardware y software necesarios para trabajar en el proyecto. [24]

CAPÍTULO 2

2. METODOLOGÍA E IMPLEMENTACIÓN

Con el fin de cumplir con los objetivos propuestos, la segunda parte de nuestra investigación está relacionado directamente con la metodología aplicada al proyecto y la implementación de todos los paquetes de ROS para poner en marcha al robot TurtleBot 2 empleando SLAM, de esa manera dejamos todo listo para desarrollar la parte práctica del proyecto y exponer resultados.

2.1. Metodología

Para la consecución de los objetivos planteados, iniciamos determinando las características y especificaciones del hardware, el equipamiento del equipo de cómputo, la determinación del hardware usado para el robot, periféricos y todo lo demás. Para el efecto nos basamos en el siguiente artículo. [26]

Una vez determinado el hardware procedemos con la instalación del sistema operativo (GNU/LINUX) y todo el conjunto de herramientas (ROS), cabe señalar que la implementación del sistema operativo, se realizó bajo la distribución (UBUNTU).

Los diferentes paquetes y algoritmos para SLAM, encontramos dentro de la página web de la comunidad ROS, están disponibles para descargar y luego instalar correctamente en nuestro computador. Resaltamos que para la solución de problemas de SLAM se emplea la técnica probabilística, que es la más utilizada y para verificar los algoritmos se descarga los códigos fuente que se encuentran en la plataforma llamada Github, es en donde se encuentran alojados como proyectos de desarrollo colaborativo.

Las locaciones involucradas en este trabajo pertenecen a la Universidad Católica de Cuenca de forma específica la Unidad Académica de Ingeniería, Industria y Construcción, en sus pasillos, aulas y oficinas se realizaron las respectivas pruebas de navegación con el robot real, posterior a las simulaciones utilizando los mismos espacios, pero de manera virtual en el simulador.

2.2. Implementación (funcionamiento)

Iniciamos definiendo al sistema operativo implementado, su distribución y sus herramientas utilizados en el proyecto, detallamos todos los pasos y procesos de instalación de cada uno de ellos, presentamos las especificaciones y configuraciones, finalmente exponemos todos los inconvenientes encontrados en el proceso, con sus debidas correcciones.

2.2.1. Definición del Sistema Operativo y su distribución

GNU/LINUX: Linux o GNU/LINUX es un sistema operativo de software libre, que se presenta en distribuciones. Una distribución consta del sistema operativo, más el programa de instalación y una selección de aplicaciones.

UBUNTU Xenial 16.04 LTS: Dentro de la gran variedad de distribuciones existentes, nosotros utilizamos Ubuntu que es una distribución de Linux, Xenia 16.04 LTS (Long term support) es una versión de (soporte a largo plazo) que instalamos para nuestro proyecto.

ROS kinetic: Según sus siglas se define como Sistema Operativo Robótico que no es más que un conjunto de herramientas que trabaja sobre el sistema operativo GNU/LINUX, además es una versión exclusiva y compatible para trabajar con Ubuntu 16.04.

Distribución = Núcleo de Linux + Programa de instalación + Aplicaciones

Fig. 11: Esquema del sistema operativo Linux Ubuntu más Ros Kinetic

Fuente: Creación propia

2.2.2. Requisitos mínimos para instalar Linux Ubuntu en nuestro computador

Según las especificaciones técnicas recomiendas por el sistema para que el ordenador ejecute correctamente Ubuntu 16.04, estos son los requisitos mínimos que debe cumplir:

- Procesador Dual Core.
- 2GB de memoria RAM.
- 16GB de disco duro.
- Acceso a internet.
- Pendrive de 2 GB o más que usaremos como medio de instalación.

2.2.3. Características y especificaciones del computador de trabajo

Tabla 4: Características de la computadora Toshiba

Descripción	Detalle
Marca	Toshiba
Modelo	C45-ASP4311FL
Procesador	Intel Core i5-3230M 2.6GHz (3.2GHz c/TB)
Memoria RAM	6 GB DDR3
Disco Duro	750GB 5400RPM
Pantalla	LED 14.0" HD
Tarjeta de video	Chipset Mobile Intel® Series 8 Express
Peso	2.1 Kg
Batería	6 celdas - ion de litio
Conectividad	Gigabit Ethernet (RJ-45), Wi-Fi 802.11b/g/n
Multimedia	Cámara web HD integrada Entrada para auriculares estéreo/micrófono Altavoces SRS Premium Sound HD®
Otros	DVD ± RW, HDMI, USB 3.0, Bluetooth 4.0, Network (RJ-45), Lector de tarjetas de memoria, Audio in/out, Windows 8

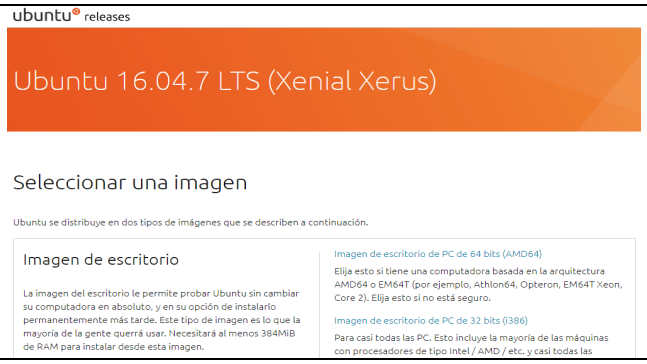
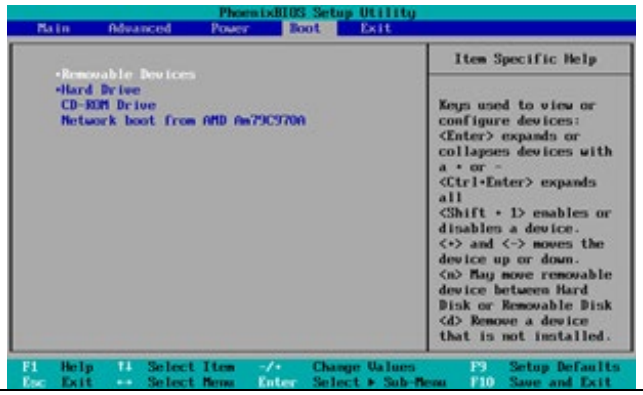
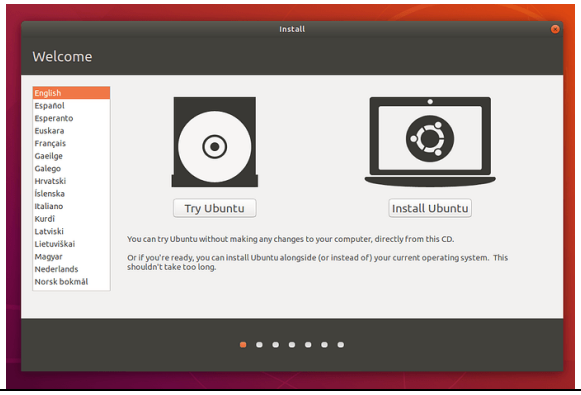
Fuente: Creación propia

2.2.4. Instalación del Sistema Operativo GNU/LINUX con su distribución UBUNTU Xenial 16.04 LTS

Se ha determinado esta versión de Ubuntu porque es un sistema más compacto y se acoplan adecuadamente a los distintos paquetes que se instaló, y que son indispensables para mover al robot TurtleBot 2.

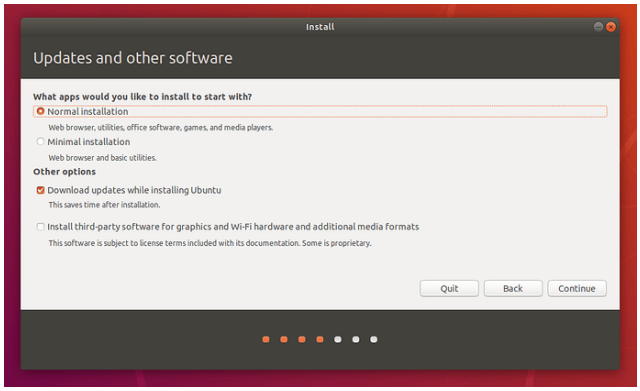
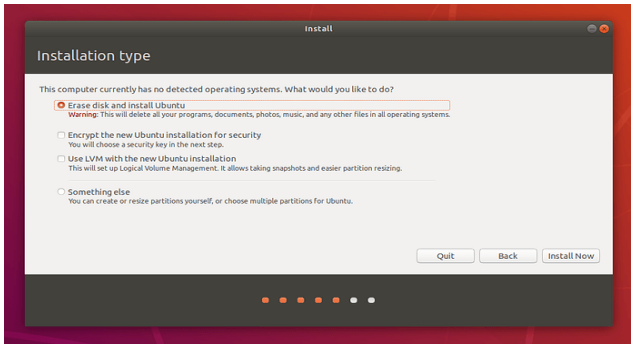
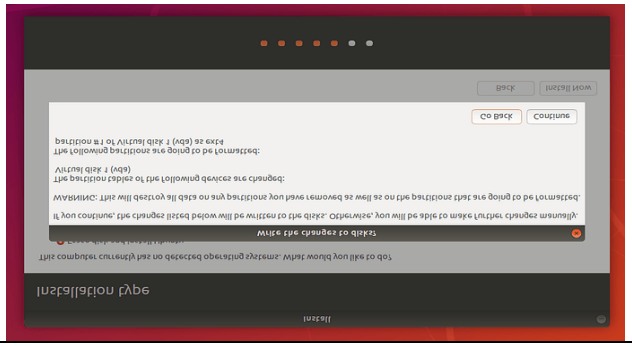
Una vez descargado el software y conocido los requerimientos del sistema, con el fin de dinamizar el proceso de instalación se presenta los siguientes paso y recomendaciones expuestas en la tabla 5:

Tabla 5: Pasos para instalación de Ubuntu 16.04 LTS

Paso 0	**Requisitos preliminares** • Conectar la computadora portátil a un suministro de energía. • Disponer de almenos 5GB de espacio de almacenamiento libre. • Tener acceso a una unidad flash USB que contenga Ubuntu 16.04.	
Paso 1	**Descargar el programa en formato ISO** • Descargar Ubuntu de la página oficial recomendada. https://ubuntu.com/16.04.7/	
Paso 2	**Configuración de la BIOS** Al momento de arrancar nuestro ordenador, pulsando repetidamente la tecla F12 para configurar el arranque de la BIOS, elegimos como primera opción que arranque desde una USB, proceso que me lleva a instalar Ubuntu 16.04 en nuestro disco duro del computador.	
Paso 3	**Instalación desde una unidad flash USB** Ya configurado la computadora arranca automáticamente desde la USB. Debe estar insertada la unidad flash USB antes de encender la computadora o hay que reiniciarla. En este paso de instalación tenemos la opción de elegir un idioma.	

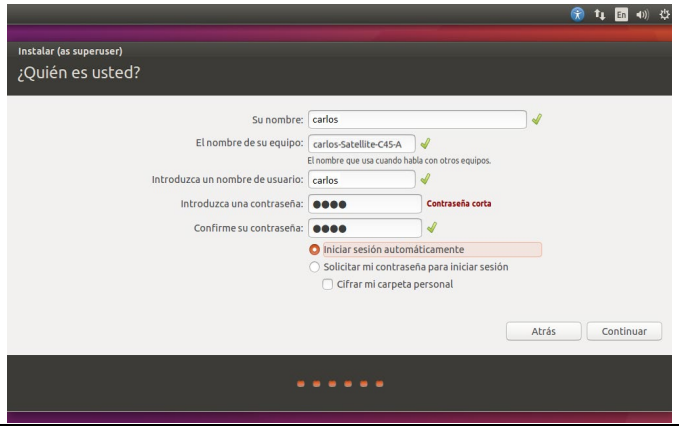
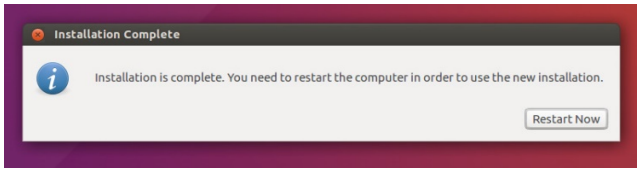
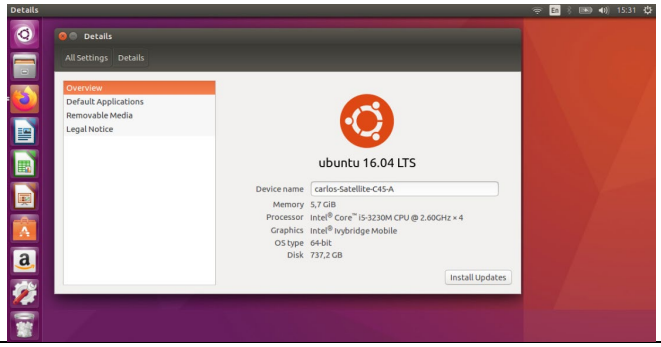
Fuente: Creación propia

Tabla 6: Pasos para instalación de Ubuntu 16.04 LTS, continuación I

Paso 4	**Preparando para instalar Ubuntu** • De las dos opciones de instalación normal o mínima, escogemos las dos opciones, de esa manera se actualiza automáticamente durante la instalación. • Se recomienda mantenerse conectado a Internet para que pueda obtener las últimas actualizaciones mientras instala Ubuntu.	
Paso 5	**Asignación de espacio en el disco** Elegimos la opción donde y como queremos instalar Ubuntu • Si junto con otro sistema operativo • Eliminando el sistema operativo existente y reemplazarlo con Ubuntu *(para nuestro proyecto escogemos esta opción)* o, • Si es un usuario avanzado.	
Paso 6	**Inicio de la instalación** Una vez configurado el lugar de almacenamiento, presionamos en el botón “Instalar ahora”.	
Paso 7	**Selección de ubicación** Se recomienda que la computadora esté conectada a Internet, para que pueda detectar automáticamente su ubicación para instalar correctamente.	

Fuente: Creación propia

Tabla 7: Pasos para Instalación de Ubuntu 16.04 LTS, continuación II

Paso 8	**Detalles de inicio de sesión** La pantalla muestra la opción que nos solicita poner nuestro nombre de usuario y contraseña, así cada vez que arranque nuestra PC o para ejecutar códigos dentro de un terminal nos pedirá esta contraseña	
Paso 9	**Segundo plano de instalación** Aquí vemos como se completa en segundo plano la instalación Depende de la velocidad de nuestro computador y la conexión de red que tengamos.	
Paso 10	**Proceso de instalación finalizada** Si todo el proceso de instalación se realizó correctamente, aparecerá una pequeña ventana pidiendo que reinicie el computador. Recomendamos retirar la unidad flash USB.	
Paso 11	**Características de Ubuntu** Culminado con la instalación abrimos una ventana q nos indica las características de Ubuntu.	

Fuente: Creación propia

2.2.5. Instalación y configuración de ROS-Kinetic, GAZEBO y TURTLEBOT dentro de Ubuntu Xenial 16.04

Una vez instalado GNU/LINUX Ubuntu Xenial 16.04 en nuestro ordenador, podremos a instalar ROS Kinetic dentro de Ubuntu, teniendo en cuenta que los paquetes de Ubuntu están construidos para las siguientes distribuciones y arquitecturas mostradas en la tabla 8, en nuestro caso utilizamos la distribución Xenial.

Tabla 8: Distribuciones y arquitecturas de ROS

Distribución	Arquitectura		
Distro	amd64	i386	armhf
Astuto	√	√	
Xenial	√	√	√

Fuente: Open Source Robotics Foundation [19]

Se recomienda realizar la instalación de los paquetes en el siguiente orden: ROS-kinetic, Gazebo y TurtleBot.

2.2.5.1. Instalación de ROS-Kinetic

Teniendo en cuenta que ROS Kinetic admite la distribución Xenial Ubuntu 16.04, a continuación, detallamos todo el proceso de instalación y configuración de los principales paquetes de ROS. En la página (wiki.ros.org) encontramos información relacionado a los diferentes paquetes utilizados en este proyecto. En la figura 12 se indica el modelo de una terminal nueva, en donde escribiremos y ejecutaremos los distintos comandos.

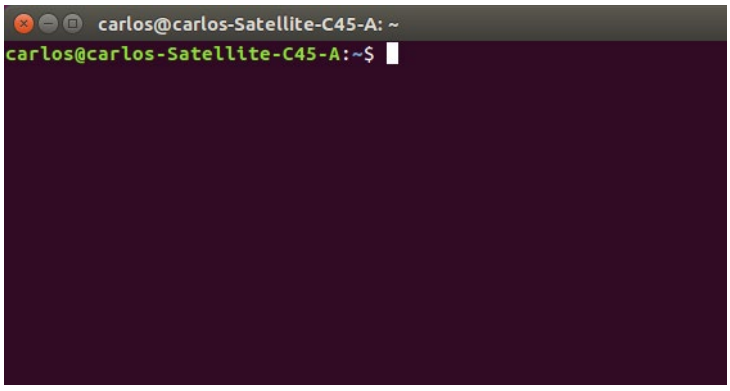
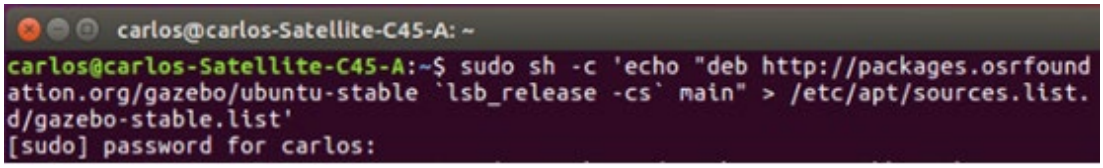
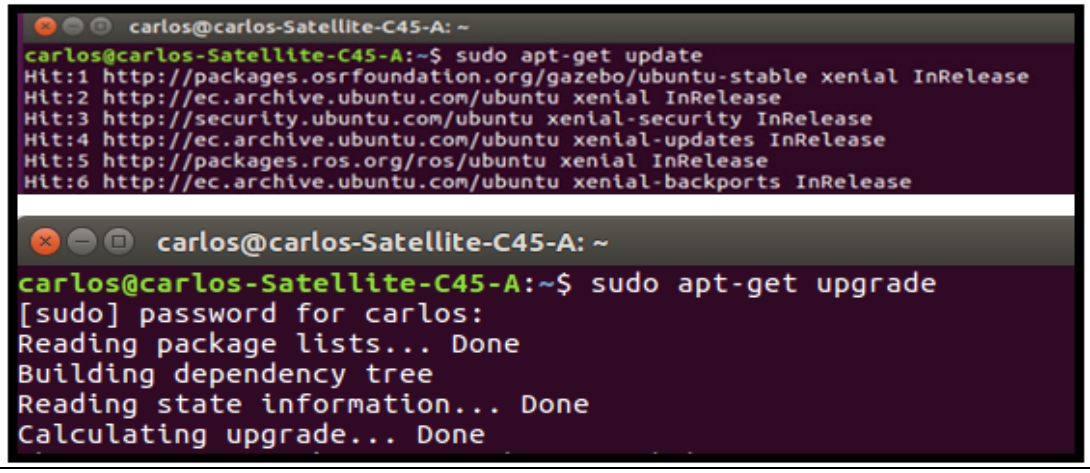


Fig. 12: Terminal nueva con nombre de usuario

Fuentes: Creación propia

Tabla 9: Pasos para Instalar Ros Kinetic

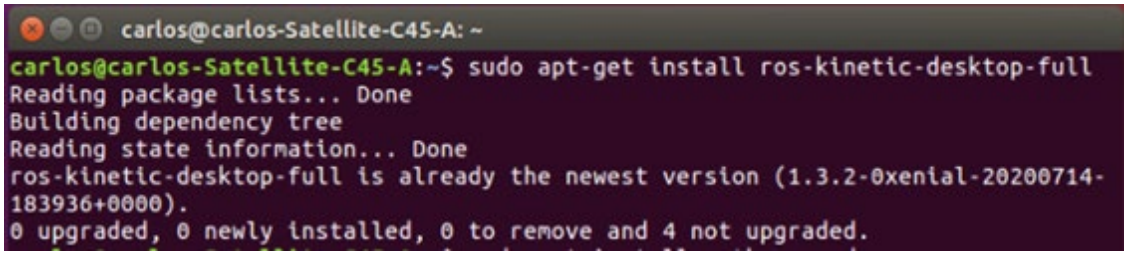
<table>
<tr><td rowspan="4">Paso 1</td><td>Configuración del repositorio de ROS</td></tr>
<tr><td>Primero configuramos los repositorios de Ubuntu y los ficheros de sources.list, para que el ordenador acepte el software packages.ros.org. En el terminal escribimos el siguiente comando.</td></tr>
<tr><td><pre>>> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'</pre></td></tr>
<tr><td> 
</td></tr>
<tr><td colspan="2">Resaltamos que cada vez que ejecutemos los comando de instalación de cualquier paquete, el sistema por seguridad nos pedirá que escribamos la contraseña de usuario.</td></tr>
<tr><td rowspan="4">Paso 2</td><td>Configuración del KEYS</td></tr>
<tr><td>Este comando se utiliza para pedir acceso al servidor, por eso es importante introducir correctamente las llaves o keys, escribimos la siguiente línea en el terminal.</td></tr>
<tr><td><pre>>> sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654</pre></td></tr>
<tr><td> 
</td></tr>
<tr><td rowspan="4">Paso 3</td><td>Actualización de paquetes</td></tr>
<tr><td>Ejecutando las siguientes líneas de comando nos aseguramos que se actualicen correctamente.</td></tr>
<tr><td><pre>>> sudo apt-get update
>> sudo apt-get upgrade</pre></td></tr>
<tr><td> 
</td></tr>
</table>

Fuente: Creación propia

Tabla 10: Pasos para Instalar Ros Kinetic, Continuación I

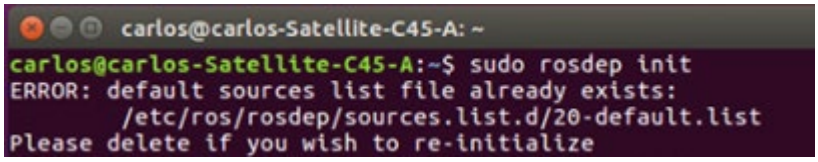
<table>
<tr><td rowspan="3">Paso 4</td><td>Instalación versión completa de ROS</td></tr>
<tr><td>De las tres opciones de instalación existentes: 1) versión completa, 2) de escritorio y 3) para máquinas remotas, escogemos la primera opción para nuestro proyecto.
Las ventajas que presenta esta opción es que incluye el núcleo de ROS, tiene herramientas de visualización rqt y rviz, dispone de bibliotecas genéricas de robots, tiene simuladores 2D / 3D y puede realizar navegación y percepción 2D / 3D.</td></tr>
<tr><td>

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

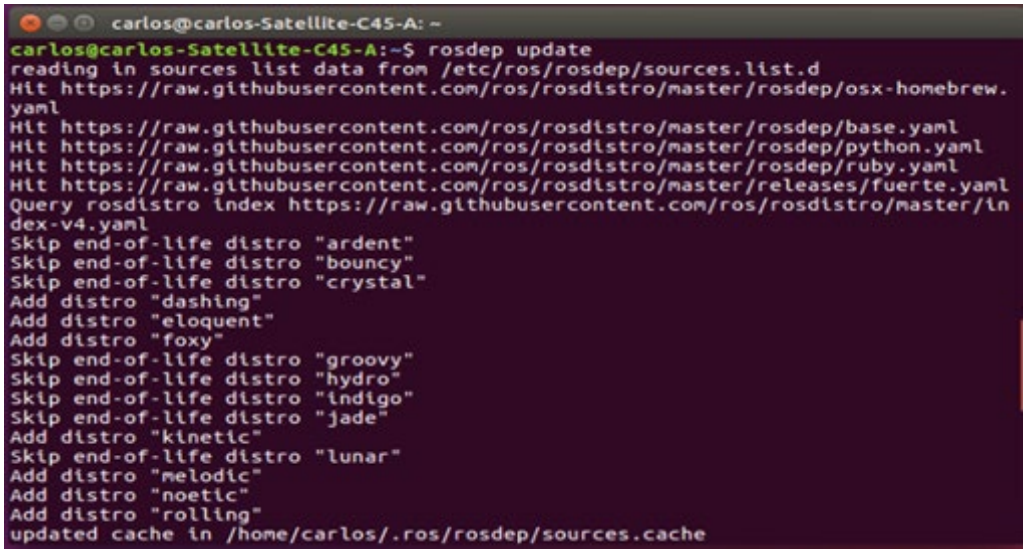


</td></tr>
<tr><td rowspan="4">Paso 5</td><td>Inicialización de rosdep</td></tr>
<tr><td>Revisa las listas de los paquetes para las distintas distribuciones de ROS. Con la ejecución de los siguientes comandos uno a la vez.</td></tr>
<tr><td>

```
>> sudo rosdep init
>> rosdep update
```



</td></tr>
<tr><td>Antes trabajar con ROS, inicializamos rosdep que permite instalar fácilmente las dependencias para cualquier código que compilemos y algunos componentes del núcleo de ROS.</td></tr>
<tr><td>



</td></tr>
</table>

Fuente: Creación propia

Tabla 11: Pasos para Instalar Ros Kinetic, Continuación II

Paso 6	**Configuración del entorno de trabajo para Ubuntu**
	Primero configuramos nuestro entorno de trabajo de ROS, y así sea visible y accesible desde cualquier terminal y no tengamos que estar ejecutando el comando cada vez que se abra un nuevo terminal. Para eso ejecutamos los siguientes comandos, uno a la vez.
	`>> echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc` `>> source ~/.bashrc`
	carlos@carlos-Satellite-C45-A: ~ carlos@carlos-Satellite-C45-A:~$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc carlos@carlos-Satellite-C45-A:~$ source ~/.bashrc
Paso 7	**Dependencias para construir paquetes (rosinstall)**
	Para compilar los paquetes ROS e instalar las herramientas y otras dependencias, ejecutamos el siguiente comando.
	`>> sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential`
	carlos@carlos-Satellite-C45-A: ~/catkin_ws carlos@carlos-Satellite-C45-A:~$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential Reading package lists... Done Building dependency tree Reading state information... Done build-essential is already the newest version (12.1ubuntu2). python-rosdep is already the newest version (0.19.0-1). python-rosinstall is already the newest version (0.7.8-1). python-rosinstall-generator is already the newest version (0.1.22-1). python-wstool is already the newest version (0.1.17-1). 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

Fuente: Creación propia

2.2.5.2. Corrección de ERROR de inicialización de ROS

Encontramos un error al momento de ejecutar el comando para inicializar ROS, el error no permite que ROS se inicialice correctamente, por lo tanto, primero resolvemos el error que se muestra en la figura 13.

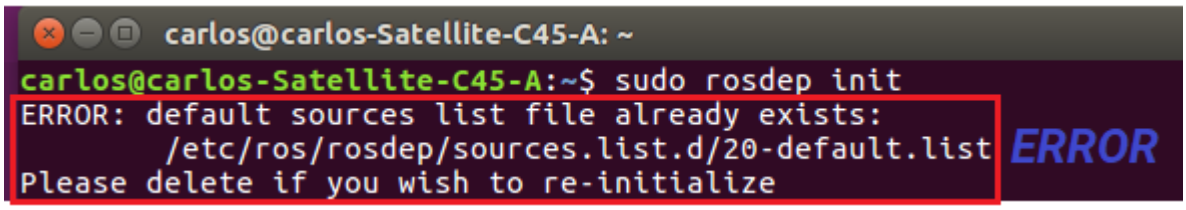


Fig. 13: Error de Inicialización de ROS

Fuente: Creación propia

ERROR: Me dice que el archivo de la lista de fuentes por defecto ya existe y que si queremos reiniciar debemos de eliminar del directorio.

```
$$ /etc/ros/rosdep/sources.list.d/20-default.list
```

Para corregir el error, recurrimos a la página de la comunidad de (wiki ROS), donde encontramos varias opciones de solución de muchos investigadores que solventaron y expusieron dentro de la comunidad.

CORRECCIÓN: Para eliminar el archivo de la fuente, ejecutando el siguiente comando.

```
>> sudo rm -rf /etc/ros/rosdep/sources.list.d/*
```

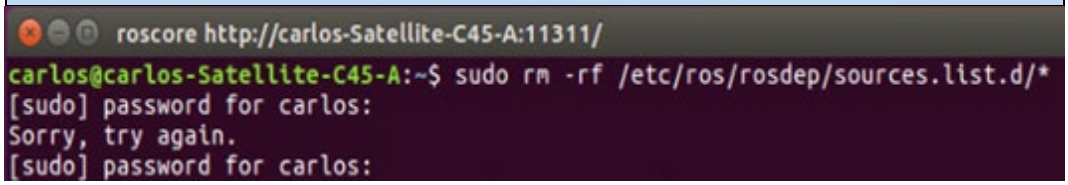


Fig. 14: Corrección de error de Inicialización de ROS

Fuente: Creación propia

Para el efecto ejecutamos nuevamente el comando *init*, con el fin de recuperar las fuentes predeterminadas, ahora podemos ver en la figura 15 que ya no, nos muestra ningún error.

```
>> sudo rosdep init
```

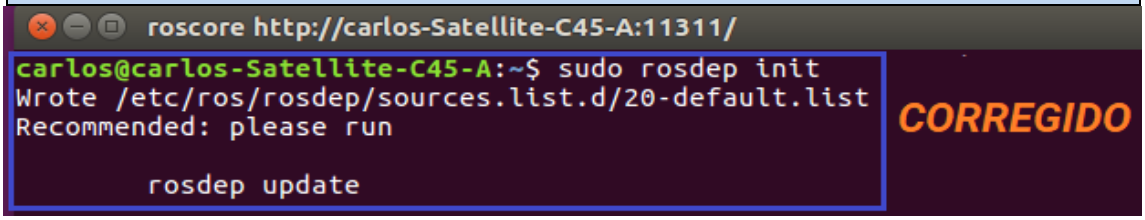


Fig. 15: Inicialización de ROS correctamente

Fuente: Creación propia

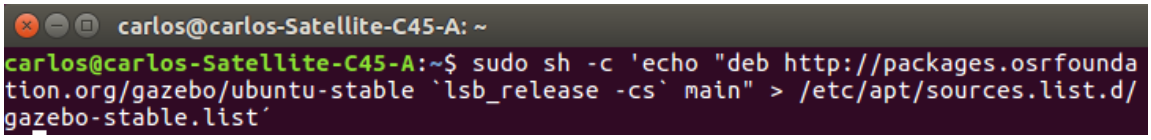
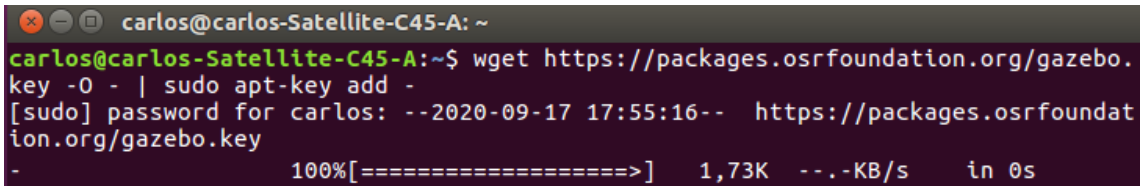
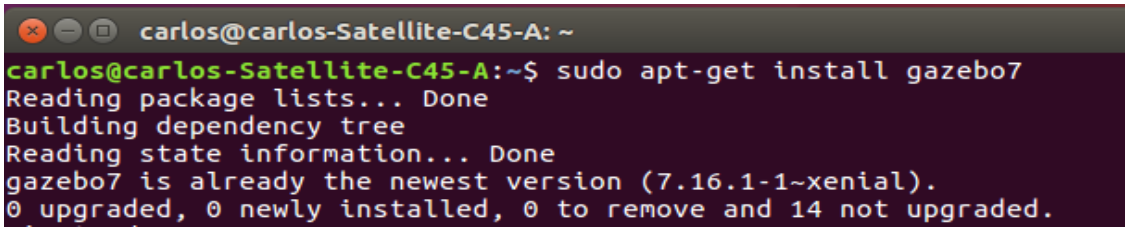
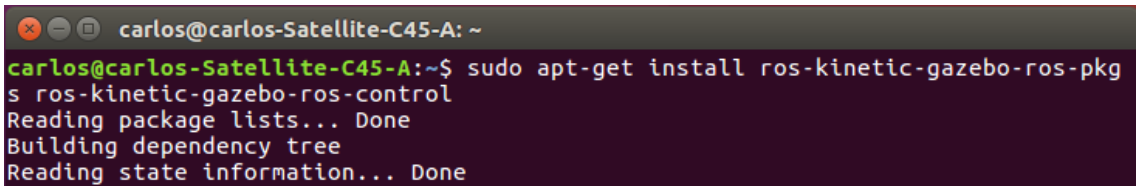
2.2.5.3. Instalación de Gazebo

Considero que el paquete Gazebo, es el más importante de ROS, ya que mediante este paquete visualizamos las simulaciones del robot TurtleBot 2. La plataforma de simulación Gazebo es un software de código abierto que permite simular gracias a la compatibilidad que existe con ROS y es mantenido por Open Robotics. Existe dos métodos de instalación.

Opción 1: Instalación predeterminada, se ejecuta una sola línea de comando

Opción 2: Instalación alternativa paso a paso, es la que optamos con el fin de ir configurando

Tabla 12: Pasos para instalar Gazebo 7

Paso 1	**Configuración de la computadora**
	Configuramos nuestra computadora, con el fin de que el computador acepte el software de packages.osrfoundation.org.
	`>> sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list´`
	
Paso 2	**Configuración de claves**
	Este comando permite configurar las claves de acceso.
	`>> wget https://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -`
	
Paso 3	**Instalación de Gazebo.**
	Antes de ejecutar el comando de instalación primero actualizo la base de datos de Debian. Ejecutamos el comando de instalación Gazebo versión 7.
	`>> sudo apt-get update` `>> sudo apt-get install gazebo7`
	
Paso 4	**Instalación de gazebo_ros_pkgs**
	Este paquete está disponible para ROS-Kinetic, sin parches y errores.
	`>> sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control`
	

Fuente: Creación propia

Una vez completado la instalación del paquete completo Gazebo, y sin ningún error de instalación, procedemos a visualizar por primera vez la ventana de GUI Gazebo. Ejecutamos el siguiente comando en una terminal nueva.

```
>> gazebo
```

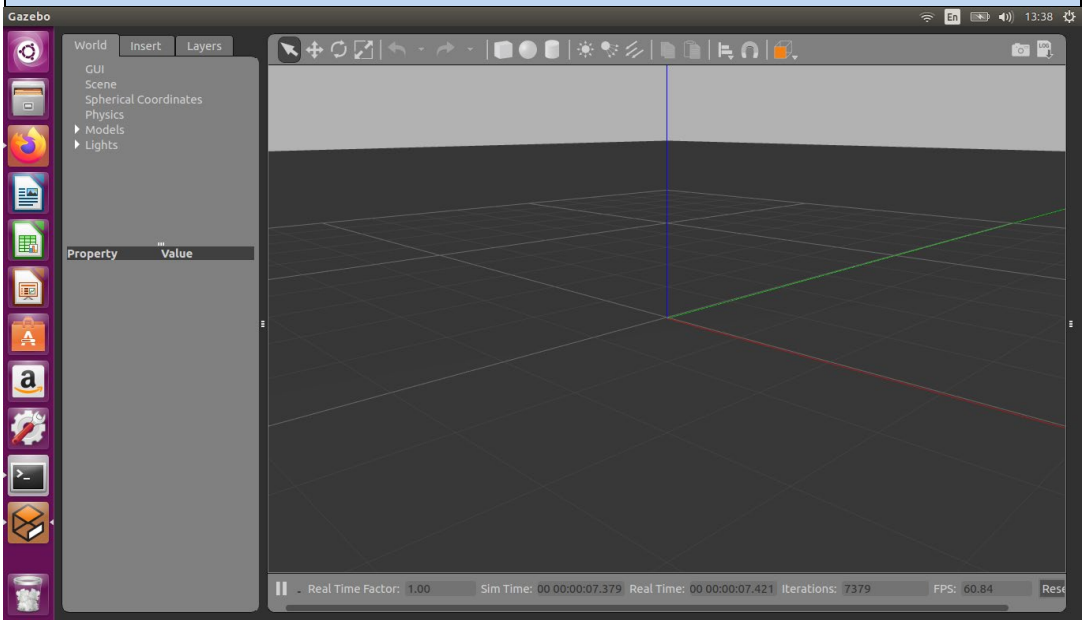


Fig. 16: Ventana GUI Gazebo

Fuente: Creación propia

2.2.5.4. Comando para clonar repositorios de Github

Es necesario instalar el paquete de Github, con el fin de clonar cualquier repositorio anterior o actualizado que se encuentre en la plataforma Github. Ejecutamos los siguientes comandos.

```
>> sudo apt-get install git
>> sudo apt-get install -y libgazebo7-dev
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ sudo apt-get install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
carlos@carlos-Satellite-C45-A:~$ sudo apt-get install -y libgazebo7-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Fig. 17: Comando para clonar repositorios de Github,

Fuente: Creación propia

2.2.5.5. Instalación de paquetes TurtleBot 2

Dentro de la plataforma TurtleBot, encontramos una lista de paquetes disponibles para trabajar con ROS, ejecutando cada comando se instalarán los paquetes expuestos.

```
>> sudo apt-get install ros-kinetic-turtlebot
>> sudo apt-get install ros-kinetic-turtlebot-apps
>> sudo apt-get install ros-kinetic-turtlebot-interactions
>> sudo apt-get install ros-kinetic-turtlebot-simulator
>> sudo apt-get install ros-kinetic-kobuki-ftdi
>> sudo apt-get install ros-kinetic-ar-track-alvar-msgs
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ sudo apt-get install ros-kinetic-turtlebot ros-
kinetic-turtlebot-apps ros-kinetic-turtlebot-interactions ros-kinetic-turtlebot-
simulator ros-kinetic-kobuki-ftdi ros-kinetic-ar-track-alvar-msgs
[sudo] password for carlos:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-kinetic-ar-track-alvar-msgs is already the newest version (0.7.1-0xenial-201
91214-055016+0000).
ros-kinetic-kobuki-ftdi is already the newest version (0.7.8-1xenial-20191214-00
2148+0000).
ros-kinetic-turtlebot is already the newest version (2.4.2-0xenial-20200310-1734
13+0000).
ros-kinetic-turtlebot-apps is already the newest version (2.3.7-0xenial-20200427
-195117+0000).
ros-kinetic-turtlebot-interactions is already the newest version (2.3.1-0xenial-
20200406-145123+0000).
ros-kinetic-turtlebot-simulator is already the newest version (2.2.3-0xenial-202
00709-182330+0000).
```

Fig. 18: Pasos para instalar paquetes de Turtlebot 2

Fuente: Creación propia

2.2.6. Configuración del entorno de trabajo ROS.

Para configurar correctamente el entorno de trabajo ROS, ejecutamos el siguiente comando, de esa manera aseguramos que las variables de entorno como ROS_ROOT y ROS_PACKAGE_PATH estén bien establecidos.

```
>> printenv | grep ROS
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ printenv | grep ROS
ROS_ROOT=/opt/ros/kinetic/share/ros
ROS_PACKAGE_PATH=/opt/ros/kinetic/share
ROS_MASTER_URI=http://localhost:11311
ROS_PYTHON_VERSION=2
ROS_VERSION=1
ROSLISP_PACKAGE_DIRECTORIES=
ROS_DISTRO=kinetic
ROS_ETC_DIR=/opt/ros/kinetic/etc/ros
```

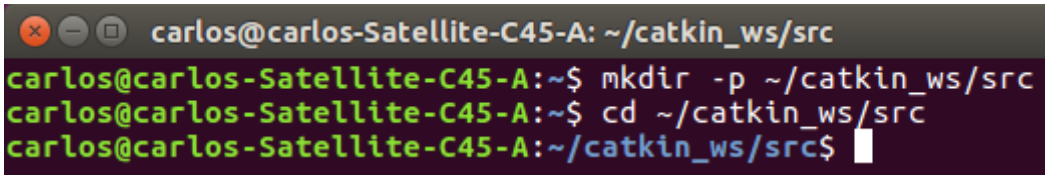
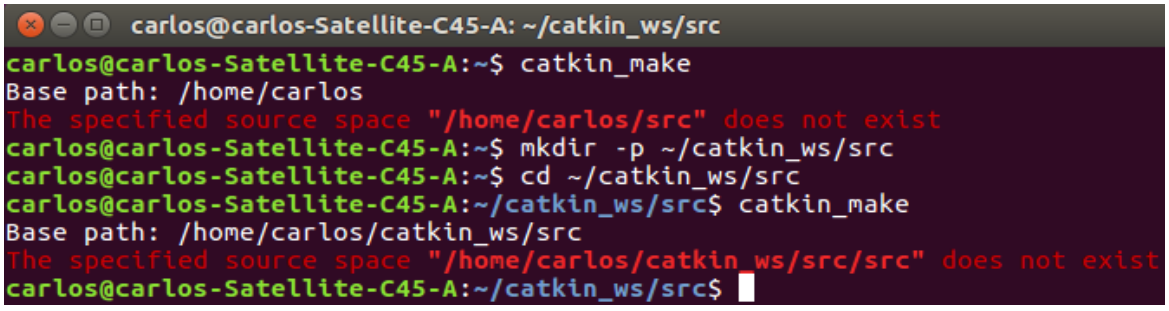
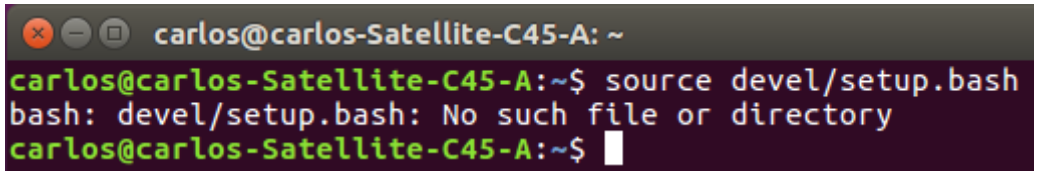
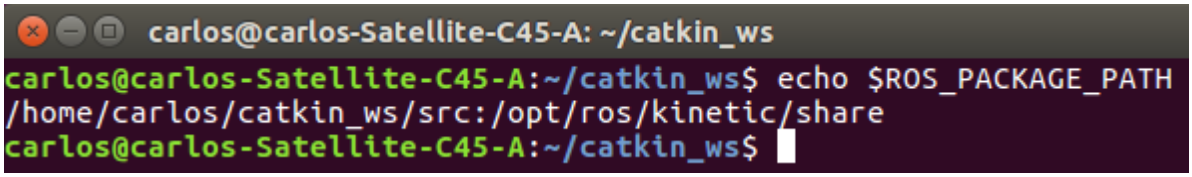
Fig. 19: Configuración del entorno de trabajo ROS,

Fuente: Creación propia

2.2.6.1. Espacio de trabajo ROS

Para realizar las simulaciones es necesario crear un espacio de trabajo ROS, por eso primero creamos un directorio *workspace* que le llamamos *catkin*, dentro de este directorio creamos una sub carpeta *src*, con el comando *cd* podemos navegar dentro de este directorio, verificamos que el entorno de trabajo y los paquetes estén instalados correctamente y finalmente nos asegurarnos que nuestro entorno haya sido configurado correctamente. Ejecutando en el orden que se muestra en la tabla 13, cada uno de los comandos nos aseguramos que nuestro espacio de trabajo esté listo para trabajar dentro de él.

Tabla 13: Espacio de trabajo catkin_ws

Creamos nuevo directorio catkin_ws y src y ubicación dentro del directorio

Ver fichero en el directorio

Consulta de la documentación general de catkin

Verificación del espacio de trabajo que este correctamente superpuesto por script de configuración


Fuente: Creación propia

Si verificamos nuestro entorno de trabajo catkin_ws, dentro de él, nos mostrara los sub directorios indicando que to ha creado correctamente.

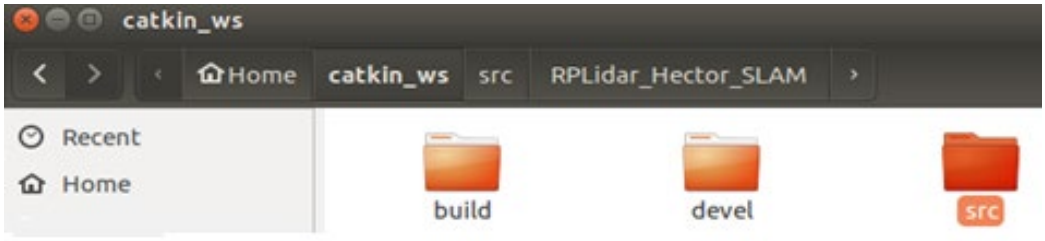


Fig. 20: Directorio de trabajo bild, devel y src

Fuente: Creación propia

2.2.7. Verificación del ROS master

Ejecutando el siguiente comando en un terminal nuevo, comprobamos que ROS se haya instalado correctamente, si todo está bien me debe aparecer una imagen como la figura 21.

```
>> roscore
```

```
roscore http://carlos-Satellite-C45-A:11311/
carlos@carlos-Satellite-C45-A:~$ roscore
... logging to /home/carlos/.ros/log/f9a7c7ee-e36e-11ea-be30-a4db3039ed4b/roslau
nch-carlos-Satellite-C45-A-23250.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://carlos-Satellite-C45-A:36227/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [23260]
ROS_MASTER_URI=http://carlos-Satellite-C45-A:11311/

setting /run_id to f9a7c7ee-e36e-11ea-be30-a4db3039ed4b
process[rosout-1]: started with pid [23273]
started core service [/rosout]
```

Fig. 21: Ejecución de ROSmaster roscore

Fuente: Creación propia

2.2.8. Instalación de Hector SLAM

Usando el administrador de paquetes, en un terminal nuevo ejecutamos el siguiente comando e instalamos el paquete de hector_slam.

```
>> sudo apt-get install ros-kinetic-hector-slam
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ sudo apt-get install ros-kinect-hector-slam
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Fig. 22: Instalación del paquete Hector SLAM

Fuente: Creación propia

2.2.9. Configuración de Hector SLAM para TurtleBot 2

El paquete hector_slam no es más que un metapaquete, que contiene un conjunto de paquetes dentro de él, en la figura 23 se indica la estructura de archivos del paquete hector_slam.

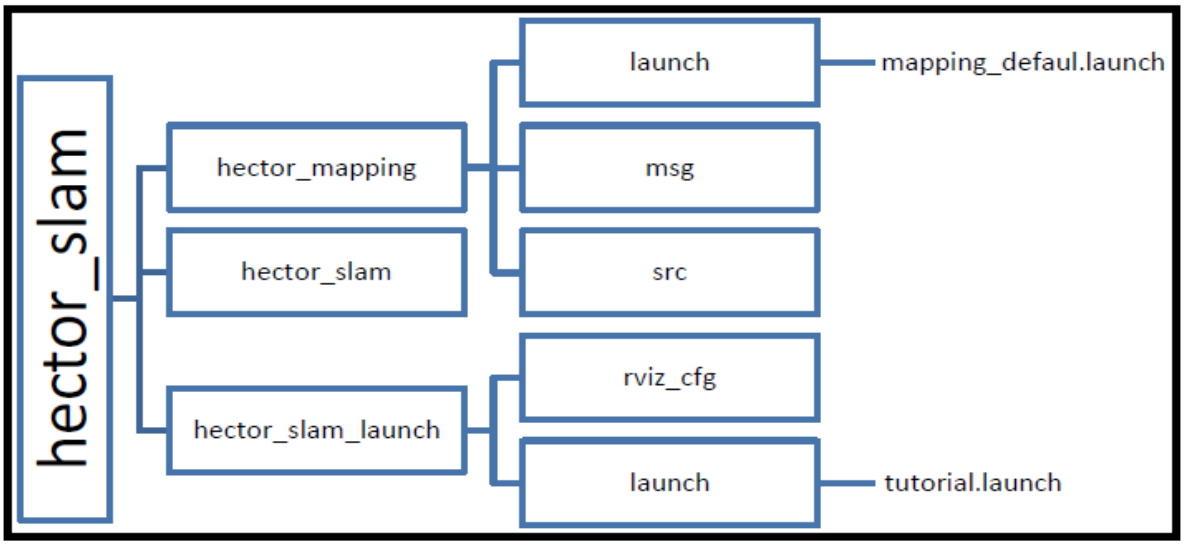


Fig. 23: Estructura de archivos del paquete hector_slam,

Fuente: *Eddison Ivan Aldas. [5]*

2.2.9.1. Visión general de Hector SLAM

Los fotogramas de la figura 24, muestra una vista 2D simplificada de un robot, cuando viaja de un punto a otro, el movimiento y balanceo generan un cabeceo de la plataforma. [19]

map, odom, base_link: la relación entre las tres es conocidos como marco de coordenadas.

base_footprint: nos indica la posición y orientación del robot, representa una pose 2D.

base_stabilized: nos da información sobre la altura del robot. [19]

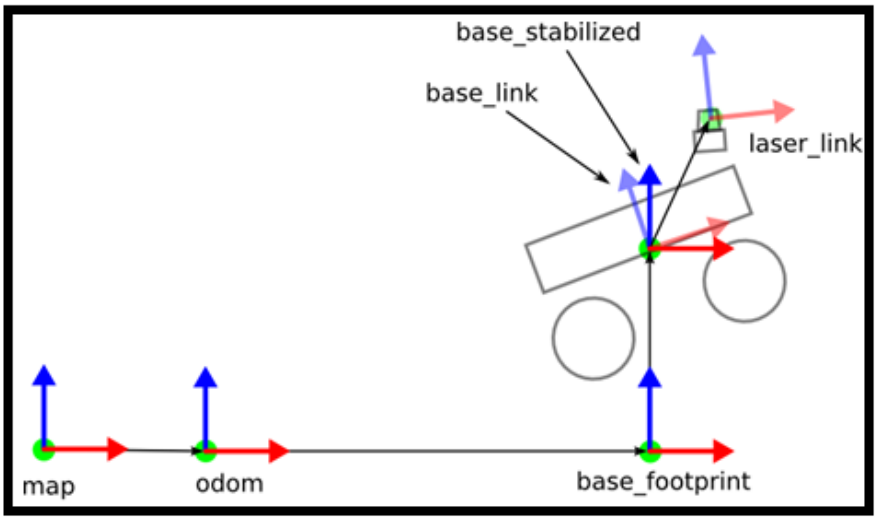


Fig. 24: Fotograma vista 2D de un Robot

Fuente: *Open Source Robotics Fundation, Hector_slam. [19]*

En la figura 25, se muestra la modificación del archivo l*auch* para realizar SLAM. La explicación de la modificación del archivo *launch*, se da en la línea del parámetro *use_sim_time seteado*.

*tutorial.launch (~/catkin_ws/src/RPLidar_Hector_SLAM/hector_slam/hector_slam_launch/launch) - gedit

Open Save

```
<?xml version="1.0"?>

<launch>

  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>

  <param name="/use_sim_time" value="false"/>

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

  <include file="$(find hector_mapping)/launch/mapping_default.launch"/>

  <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  </include>

</launch>
```

Fig. 25: Archivo tutorial.launch

Fuente: Creación propia

true, está escrito por defecto y nos indica que el ros master usa el tiempo de simulación en lugar del tiempo real proporcionado por algún otro proceso.

false, se debe establecer esté parámetro para trabajar con datos del láser para que construya el mapa el robot, en tiempo real. [5]

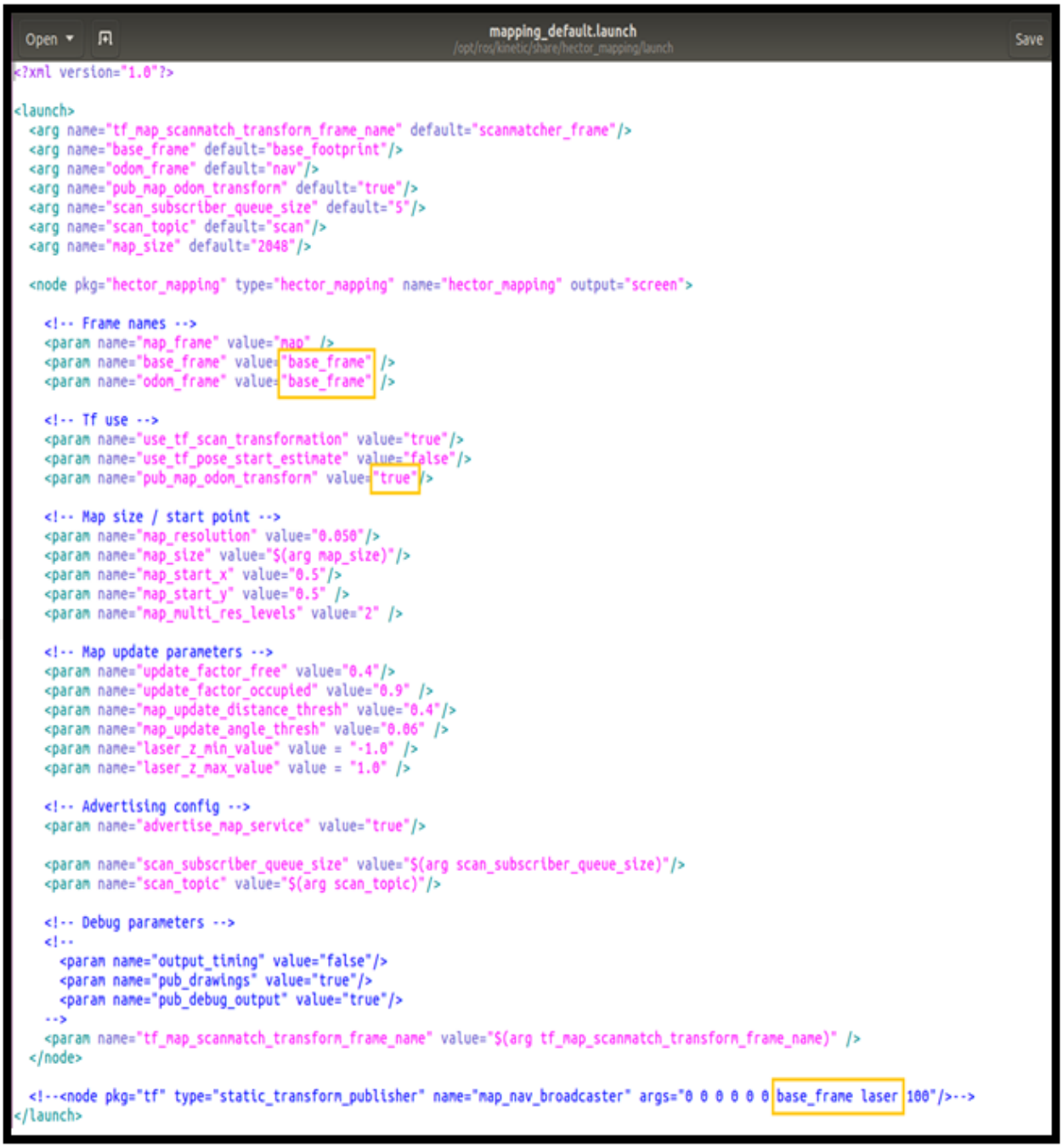


Fig. 26: Archivo mapping_default.launch,

Fuente: Creación propia

La explicación a los cambios realizados en algunas líneas del archivo *mapping_default.launch* se debe a que el archivo *RasPi_mBot* no está dotado de odometría, los cambios realizados los podemos ver en la figura 26.

base_frame dentro del paquete *hector_slam* se genera transformaciones entre el mapa y el *base_frame*, por lo que se necesita hacer transformaciones entre los archivos *world_frame* y el *map_frame*, también el *base_frame* y el *laser_frame.* Hay un nodo que publica estas transformaciones estáticas. debido a que el archivo *RasPi_mBot* no está dotado de odometría se realiza los cambios en el archivo *mapping_default.launch*.

2.2.10. Lista de comandos utilizados en el proyecto

A continuación, se indica algunos comandos empleados en el presente proyecto.

Tabla 14:Comando más utilizados en el proyecto

1. Actualizar el listado de paquetes disponibles:
`>> sudo apt-get update`
2. Comprobar que todo ha ido bien tras la utilización de apt-get update:
>> sudo apt-get check
3. Instalar programas deseados:
`>> sudo apt-get install (nombre de paquete)`
4. Reinstalar un programa:
`>> sudo apt-get -reinstal install (nombre de paquete)`
5. Actualizar solo los paquetes ya instalados que no necesitan, como dependencia, la instalación o desinstalación de otros paquetes:
`>> sudo apt-get upgrate`
6. Actualizar todos los paquetes del sistema, instalando o desinstalando los paquetes que sean necesarios para resolver las dependencias que puedan generar la actualización de algún paquete:
`>> sudo apt-get dist-upgrate`
7. Desinstalar un paquete:
`>> sudo apt-get remove (nombre de paquete)`
8. Desinstalar un paquete y eliminar los archivos de configuración:
`>> sudo apt-get remove --purge (nombre de paquete)`

Fuente: Creación propia

CAPÍTULO 3

3. PRUEBAS Y ANÁLISIS DE RESULTADOS

3.1. Pruebas

3.1.1. Simulación del TurtleBot 2

Lo más importante de trabajar en el simulador es, no necesariamente disponer del robot físico ya que todas las aplicaciones empleadas en el simulador lo podemos transportar al robot físico. A continuación, presentamos algunas ventajas de trabajar con un simulador de robótico.

3.1.1.1. Ventajas de usar el simulador de TurtleBot 2

- Se puede simular sin disponer de un robot físico.
- Ahorro de tiempo y dinero.
- Se puede probar programas nuevos sin consumo y sin riesgo de dañar el hardware.
- Se puede ejecutar diferentes tipos de robots a la vez, en un modelo de forma paralela.
- No se tiene que esperar que el robot recargue la batería.
- Se puede probar el mismo sistema una y otra vez con diferentes entradas utilizando un software de simulación.
- Puede elegir cualquier entorno, cualquier robot y cualquier sensor.
- Se puede probar diferentes ideas bajo situaciones exactas, en la vida real, es difícil. [27]

3.1.1.2. Comprobación del software

Una vez instalado los paquetes de ROS Kinetic y TurtleBot, donde desarrollamos la simulación del robot TurtleBot 2, ejecutamos el siguiente comando para verificar la versión de Ubuntu.

```
>> lsb_release -a
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.6 LTS
Release:        16.04
Codename:       xenial
```

Fig. 27:Versión de Ubuntu

Fuente: Creación propia

3.1.1.3. Actuación de Gazebo con un robot TurtleBot 2

De acuerdo a las necesidades que tengamos, Gazebo nos permite simular desde sistemas simples hasta muy complejo, entre algunas de las acciones que podemos ordenar al robot, por ejemplo: que levante, agarre o empuje cualquier objeto o para que realice cualquier otra actividad. Ejecutando el siguiente comando en una nueva terminal, se abre una ventana llamado mundo Gazebo, que muestra dentro del GUI de Gazebo al robot TurtleBot y algunos objetos por defecto.

```
>> roslaunch turtlebot_gazebo turtlebot_world.launch
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ roslaunch turtlebot_gazebo turtlebot_world.launch
... logging to /home/carlos/.ros/log/e06e0b40-d29c-11ea-b931-a4db3039ed4b/roslaunc
h-carlos-Satellite-C45-A-3797.log
```

Fig. 28: Lanzamiento de Gazebo world_launch

Fuente: Creación propia

Reportamos un error e indicamos en la figura 29.

```
IOError: [Errno 13] Permission denied: '/home/carlos/.ros/rosdep/sources.cache/i
ndex'
[spawn_turtlebot_model-3] process has died [pid 10172, exit code 1, cmd /opt/ros
/kinetic/lib/gazebo_ros/spawn_model -unpause -urdf -param robot_description -mod
el mobile_base __name:=spawn_turtlebot_model __log:=/home/carlos/.ros/log/ce8fef
f4-d825-11ea-b8f9-a4db3039ed4b/spawn_turtlebot_model-3.log].
log file: /home/carlos/.ros/log/ce8feff4-d825-11ea-b8f9-a4db3039ed4b/spawn_turtl
ebot_model-3*.log
```

Fig. 29: Error en lanzamiento de Gazebo

Fuente: Creación propia

Sin embargo, se abre el GUI de Gazebo, pero no se visualiza al robot y en la ventana del RVIZ me indica el error en las líneas de Grid y RoboyModel.

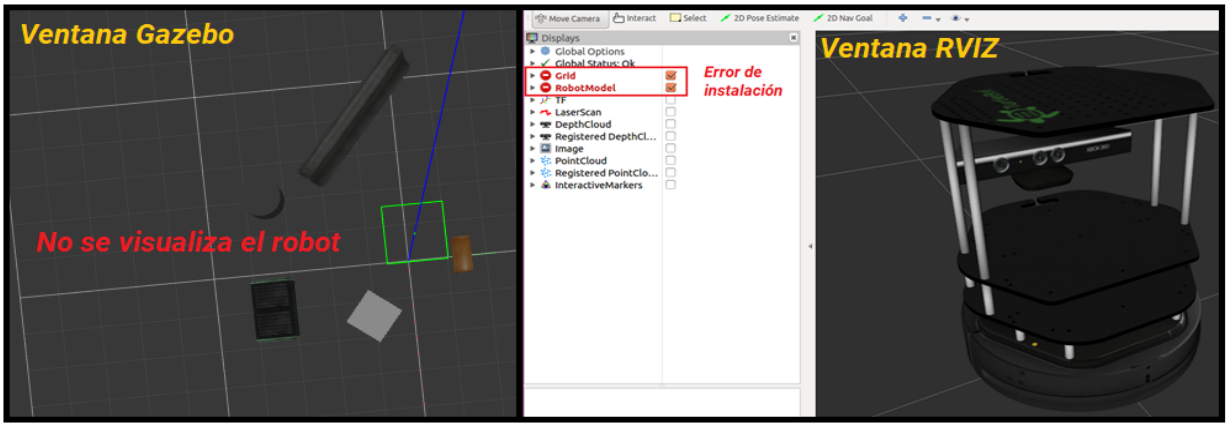


Fig. 30: Ventana Gazebo y RVIZ con error

Fuente: Creación propia

3.1.1.4. Corrección del error de lanzamiento de Gazebo y RVIZ

El error de inicialización de ros *init*, es un error que repercutía en el lanzamiento de Gazebo y RVIZ, una vez corregido el error en la instalación de ROS, volvemos a ejecutar nuevamente los comandos de lanzamiento y tendremos el resultado de la figura 31.

```
>> roslaunch turtlebot_gazebo turtlebot_world.launch
```

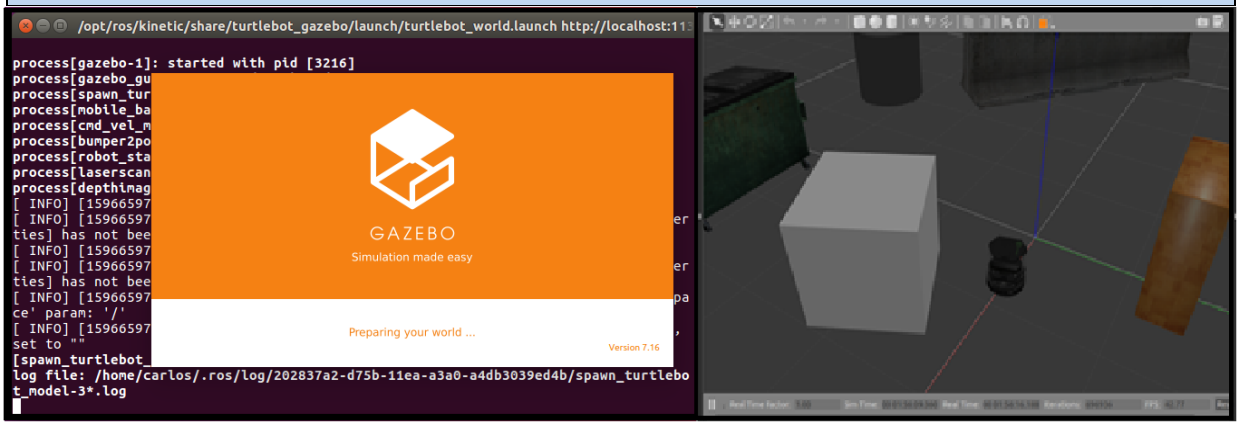


Fig. 31: Lanzamiento Corregidor de GUI Gazebo

Fuente: Creación propia

3.1.1.5. Actuación del visualizador Rviz

Rviz es una herramienta de visualización en 3D para ROS, este entorno visualiza y permite ver lo que el robot está viendo, pensando y haciendo. Una parte importante en el desarrollo del simulador es la visualización y el registro de la información del sensor del robot. Ejecutamos el comando rviz paralelamente a Gazebo en una nueva terminal.

```
>> roslaunch turtlebot_rviz_launchers view_robot.launch
```

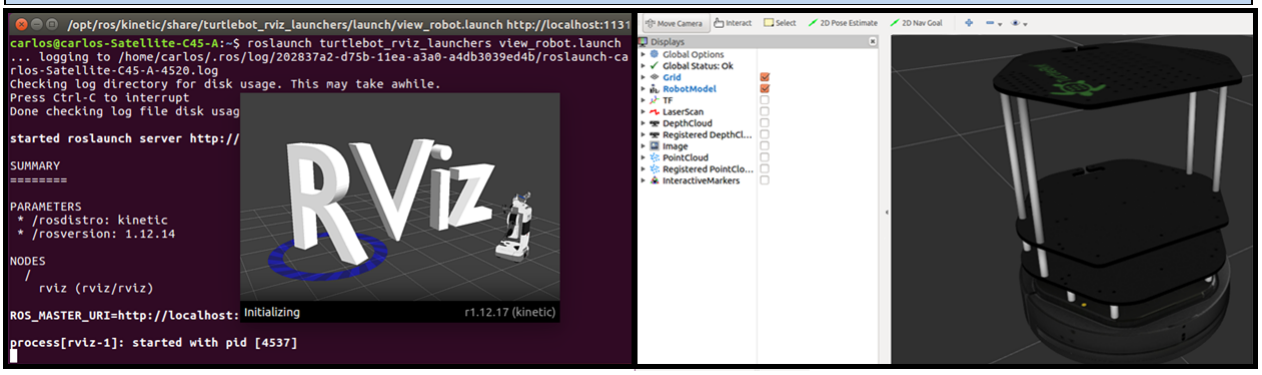


Fig. 32: Lanzamiento Corregidor de RVIZ

Fuente: Creación propia

3.1.1.6. Prueba de cámara Kinect en el simulador

Las tres partes de Kinect funcionan siempre juntas con el fin de que el robot TurtleBot 2 vea el mundo en 3D, rastrea y detecta los objetos:

- Una cámara RGB
- Un sensor de profundidad
- Micrófono multi-array

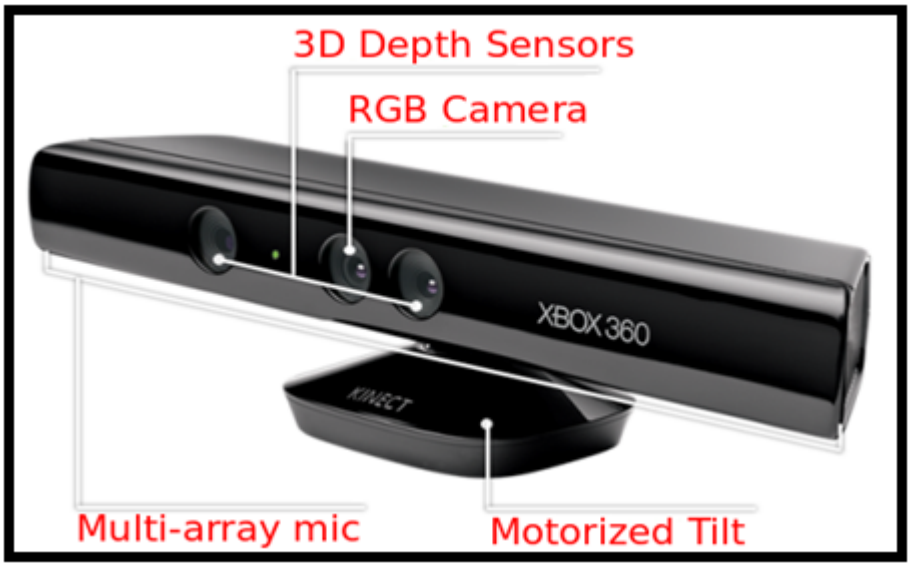


Fig. 33:Cámara Kinect

Fuente: Silliman Markw. [27]

Para probar el sensor Kinect dentro del entorno del simulador se requiere primeramente tener instalados todos los paquetes *openni*, ejecutando los comandos mencionados.

```
>> sudo apt-get install ros-kinetic-openni-*
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ sudo apt-get install ros-kinetic-openni-*
[sudo] password for carlos:
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Fig. 34: Instalación de Kinect

Fuente: Creación propia

Ejecutando el primer comando verificamos que el sensor 3D de TurtleBot este activo. Dependiendo que sensor 3D activemos, ya sea Kinect o *asus_xtion_pro* u otro, debemos establecer otro valor *bashrc*.

```
>> echo $TURTLEBOT_3D_SENSOR
>> echo "export TURTLEBOT_3D_SENSOR=kinect" >> .bashrc
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ echo $TURTLEBOT_3D_SENSOR
kinect
carlos@carlos-Satellite-C45-A:~$ echo "export TURTLEBOT_3D_SENSOR=kinect" >> .ba
shrc
```

Fig. 35: Verificación del sensor Kinect

Fuente: Creación propia

3.1.1.7. Teleoperdor del robot TurtleBot 2

La teleoperación permite controlar movimientos del robot TurtleBot 2 de forma manual, de esa manera controlamos que el robot se mueva hacia adelante, atrás y girar además podemos aumentar o disminuir la velocidad. Una vez que se esté ejecutado Gazebo y Rviz, abrimos una nueva terminal y ejecutamos el comando.

```
>> roslaunch turtlebot_teleop keyboard_teleop.launch
```

```
/opt/ros/kinetic/share/turtlebot_teleop/launch/keyboard_teleop.launch http://localhos
 /
   turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

ROS_MASTER_URI=http://localhost:11311

process[turtlebot_teleop_keyboard-1]: started with pid [11913]

Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
```

Movimiento en ejes

Aumenta y disminue la velocidad

Velocidad lineal actual

Velocidad angular actual

Fig. 36: Teleopración del robot Turtlebot 2

Fuente: Creación propia

Para controlar al robot TurtleBot en la terminal Rviz existe varias formas de hacerlo: teclado, joystick, QT teleop y marcadores interactivos. Nosotros utilizamos los marcadores interactivos, pero primero instalamos los paquetes ejecutando el comando.

```
>> sudo apt-get install ros-kinetic-turtlebot-interactive-markers
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ sudo apt-get install ros-knetic-turtlebot-inter
active-markers
[sudo] password for carlos:
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package ros-knetic-turtlebot-interactive-markers
```

Fig. 37: Marcadores interactivos del Teleoperador

Fuente: Creación propia

Paralelo a las ventanas de Gazebo y Rviz, ejecutamos el siguiente comando.

```
>> roslaunch turtlebot_interactive_markers interactive_markers.launch
```

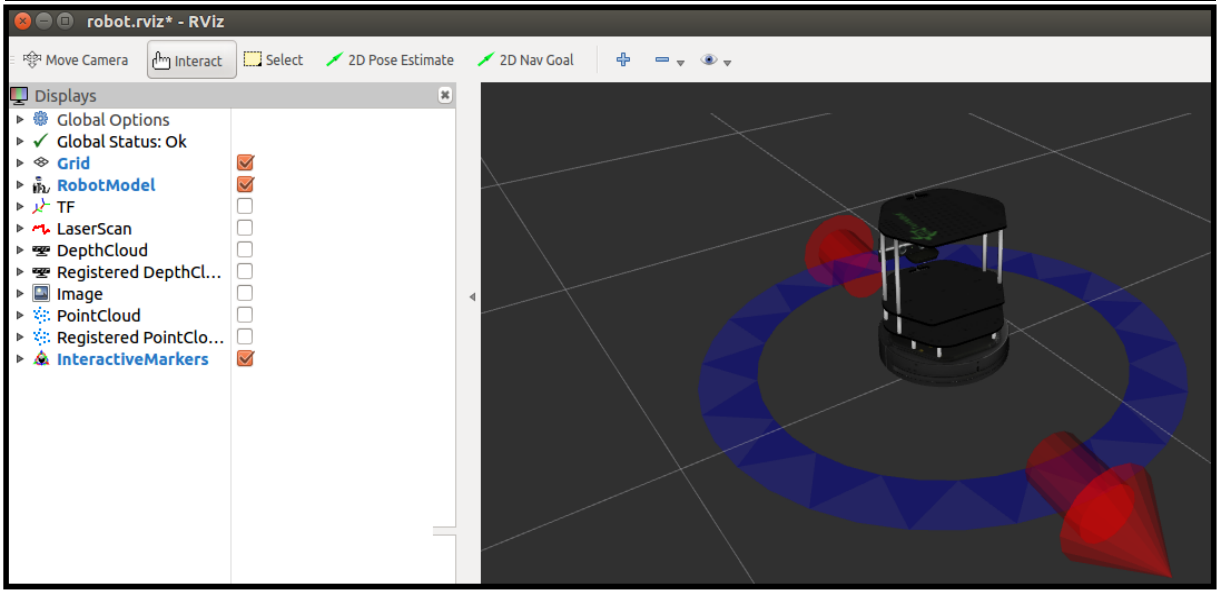


Fig. 38: Marcadores del Teleoperador

Fuente: Creación propia

3.1.1.8. Edición del mundo Gazebo en el simulador

En el presente apartado detallaremos todo el proceso a seguir para crear un nuevo ambiente de simulación o conocido también como creación de un mundo nuevo, indicamos también como elegir el nuevo ambiente de simulación y finalmente desarrollamos todo el proceso de edición del mundo exterior de trabajo.

Especificación del mundo Gazebo para la simulación

En esta sección nos referirnos a los comandos más importantes al momento de ejecutar Gazebo, de esta manera especificaremos que mundo utilizaremos para la simulación. La explicación parte del comando que ejecutamos para abrir Gazebo.

Tabla 15: Explicación del comando roslaunch

>> roslaunch turtlebot_gazebo turtlebot_world.launch	
Comandos	**Descripción**
>> roslaunch	Es una herramienta muy importante que sirve para gestionar el inicio y la parada de los procesos de ROS. además me permite lanzar fácilmente múltiples nodos ROS
>> turtlebot_gazebo	Es un nombre de paquete
>> turtlebot_world.launch	Es un archivo de inicio
>> world_file	Especifica que mundo queremos usar
/opt/ros/kinetic/share/turtlebot_gazebo/worlds	Encuentra archivos del mundo existentes en esta carpeta
>> corridor.world	Ejecuta

Fuente: Creación propia

Resaltamos que, al ejecutar un *roslaunch* inicia automáticamente *roscore*. Y para actualizar y ver en que variable de entorno se encuentra el mundo predeterminado, ejecutamos el siguiente comando y tendrá efecto todas las terminales nuevas.

```
>> echo "export TURTLEBOT_GAZEBO_WORLD_FILE=<full path to the world file>" >> .bashrc
```

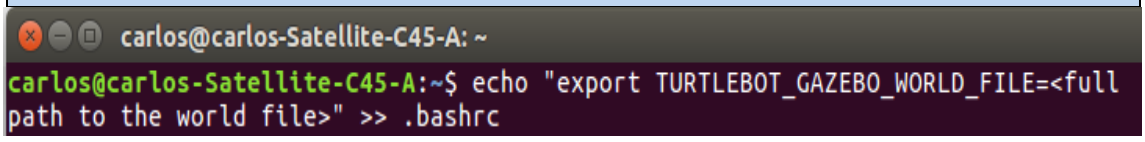


Fig. 39: Actualización del archivo world

Fuente: Creación propia

Lanzamiento de un mundo vacío

Primero creamos una nueva carpeta para guardar nuestra edición nuevo mundo, ejecutamos el siguiente código.

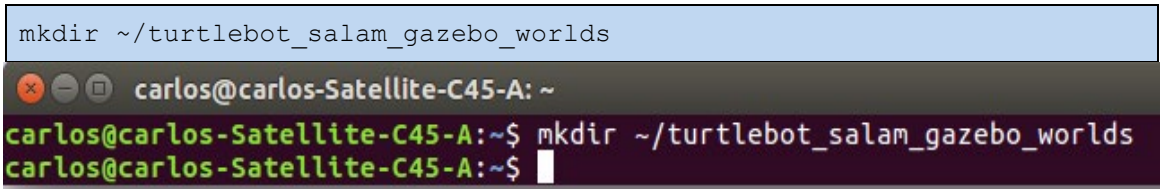


Fig. 40: Nuevo entorno de trabajo mkdir

Fuente: Creación propia

Edición de un mundo nuevo

El ambiente que creamos para realizar las simulaciones, se trata del edificio de la Unidad Académica de Industria y Construcción de la UCACUE. Las pruebas del robot tanto en la simulación como real están se realizaron en el mismo ambiente.

Una vez ejecutado el comando Gazebo, nos dirigimos a la barra de menús, dentro de *Edit* esta *Building Editor*, escogemos esa opción y se abre el editor de entorno donde trabajamos.

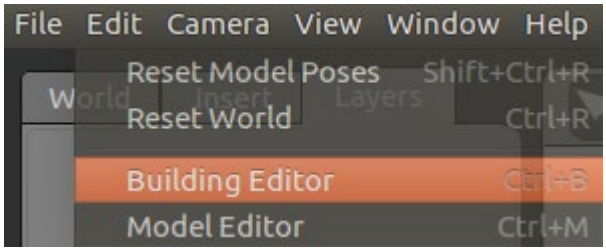


Fig. 41: Crear un edificio nuevo,

Fuente: Creación propia

La interfaz gráfica está compuesta de 3 áreas principales para poder editar un ambiente,

Área 1: es conocida como paleta y es en donde están las herramientas y materiales para construir el entorno, desde ahí tenemos la opción de importar un plano para dibujar sobre el o editar directamente.

Área 2: La Vista 2D, es donde dibujaremos paredes, ventanas, puertas y escaleras, desde aquí también podemos cambiar medias, colores e incluir el número de niveles que deseemos construir.

Área 3: La Vista 3D, es donde podemos ver una vista previa en 3D de cómo queda nuestro diseño del entorno.

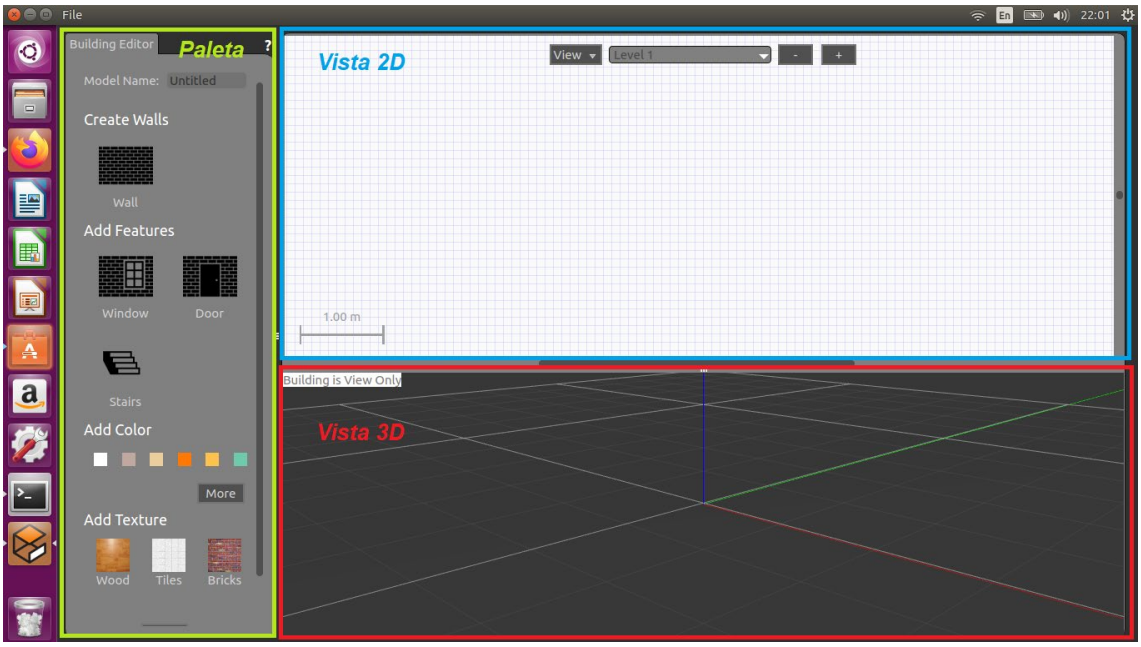


Fig. 42: Área de trabajo Gazebo

Fuente: Creación propia

Importación del plano a nuestro entorno Gazebo

Una vez realizado el plano en Autocad, procedemos a guardar el dibujo como imagen en formato JPG, tal y como se indica en la figura 43.

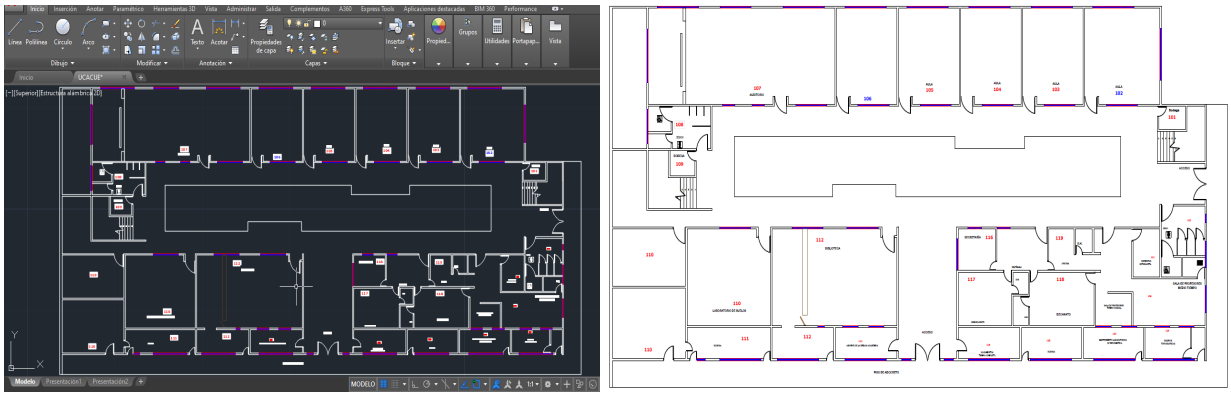


Fig. 43: Plano en Autocad y formato JPG de la Unidad Académica de la Ucacue

Fuente: Creación propia

Una vez importado ponemos las medidas reales del plano en metros, con el fin de construir el entorno a escala real y con una excelente resolución.

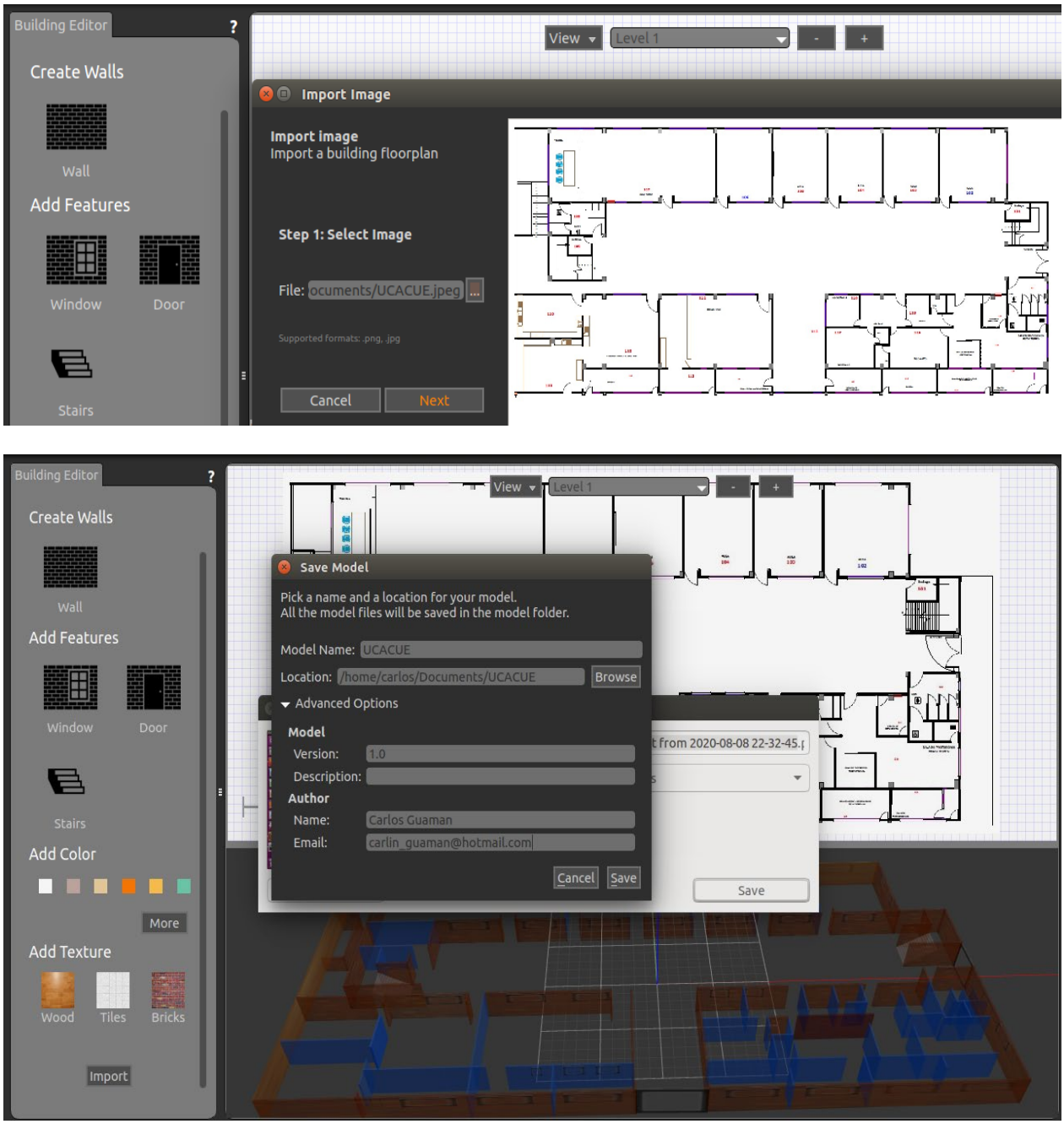


Fig. 44: Proceso de construcción del entorno

Fuente: Creación propia

Lo siguiente es insertado paredes, ventanas, puertas, niveles colores sobre el plano importado, finalmente concluimos guardando, tenemos que darle un nombre a nuestro nuevo entorno de trabajo, porque una vez que salgamos del editor del entorno ya no se puede volver a seguir editando.

En la siguiente figura 45, presentamos el entorno final terminado.

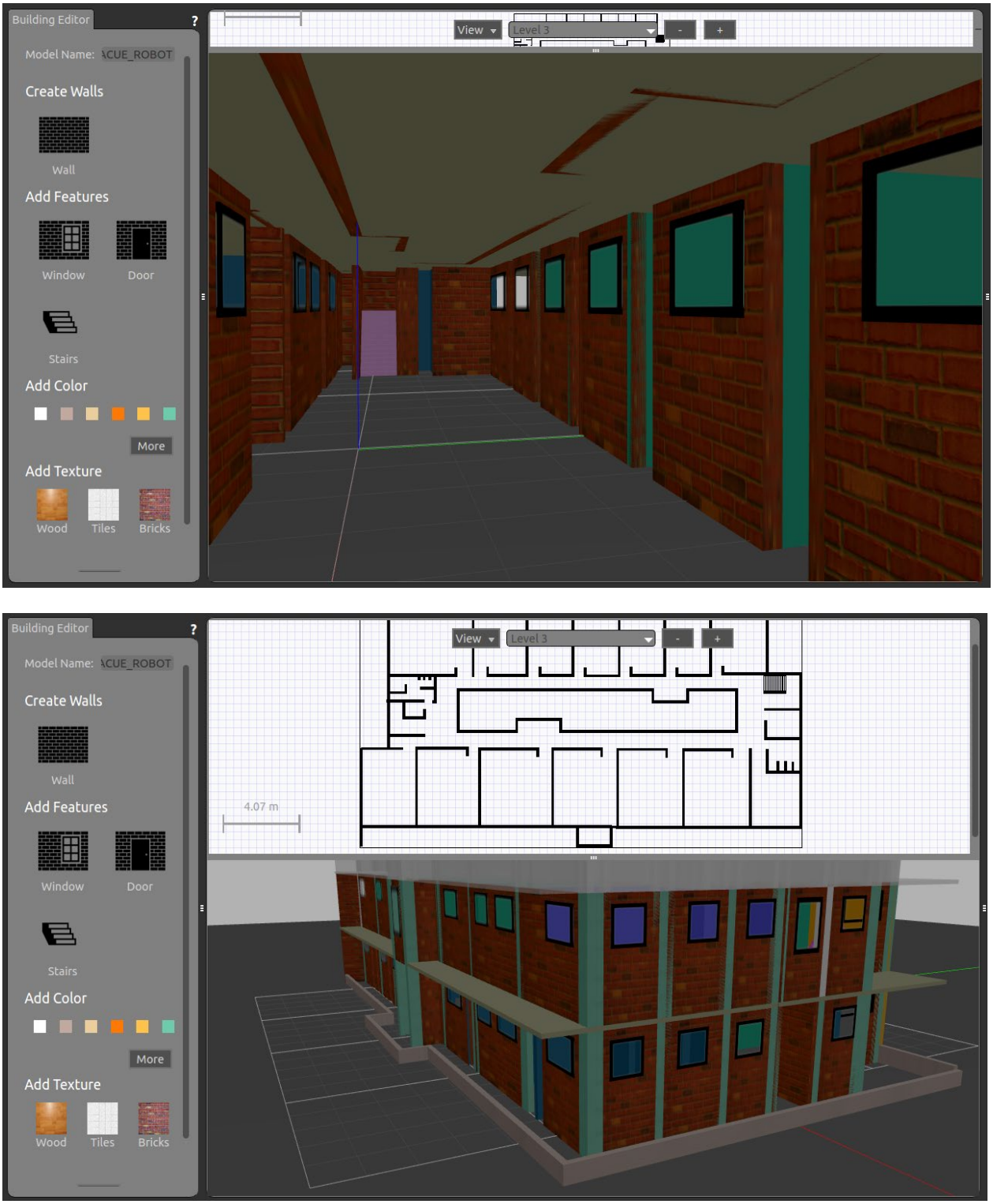


Fig. 45: Entorno Ucacue

Fuente: Creación propia

3.1.1.9. Edición del entorno

Una vez dibujado el edificio de la Unidad Académica de Industria y Construcción de la UCACUE, lanzamos en una nueva terminal el nuevo mundo Gazebo, elegimos la opción *insert*, e insertamos el entorno dibujado, dentro de esta ventana insertamos algunos obstáculos para que el robot en su trayectoria de dirigirse de un logar a otro pueda generar el mapa y esquivarlo, finalmente nos dirigimos a la barra de menús de Gazebo seleccionando la opción *File* y *Save World As* guardamos.

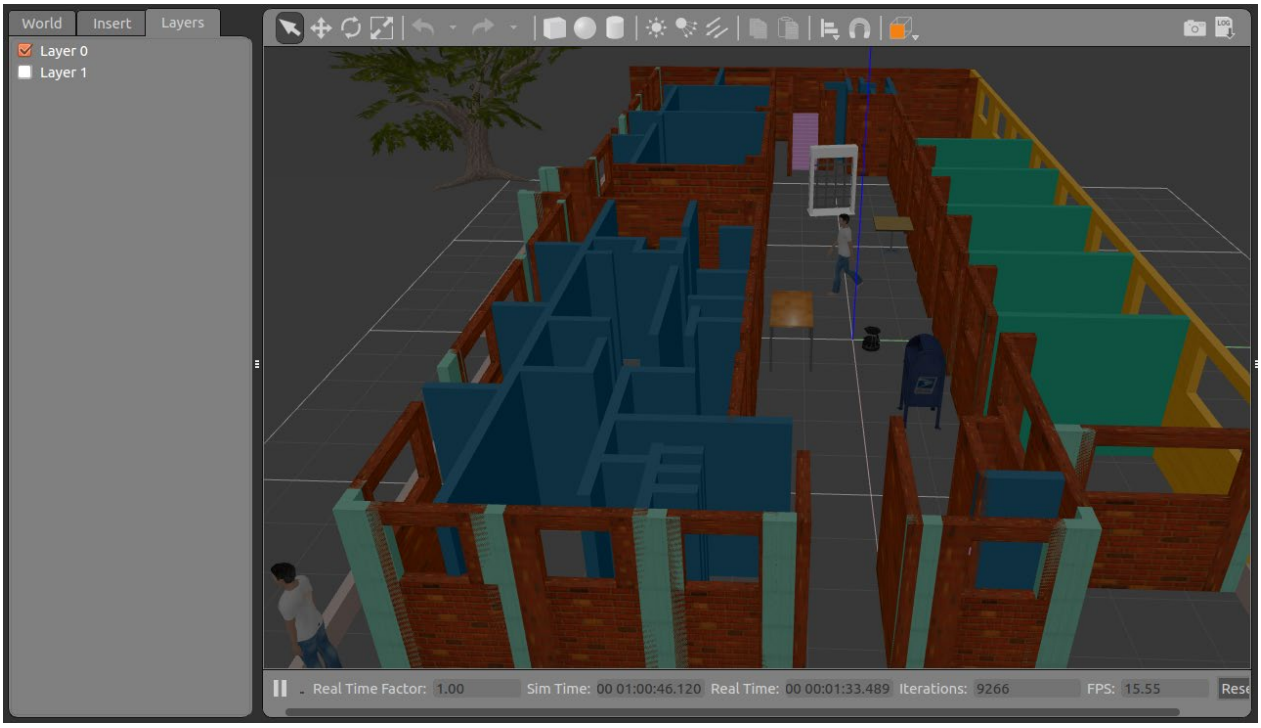


Fig. 46: Entorno Gazebo World.As

Fuente: Creación propia

Finalmente seleccionamos la carpeta en donde guaramos el entorno nuevo, cerramos la ventana de gazebo y también el terminal.

El proceso para grabar es ir a la barra de menús y seleccionamos File con el cursor bajamos y seleccionamos la opción *Save Word As*, automáticamente se abre una ventana *Open Image*, donde seleccionamos la carpeta en donde queremos guardar, le ponemos un nombre a nuestro archivo y finalmente presionamos *Save*.

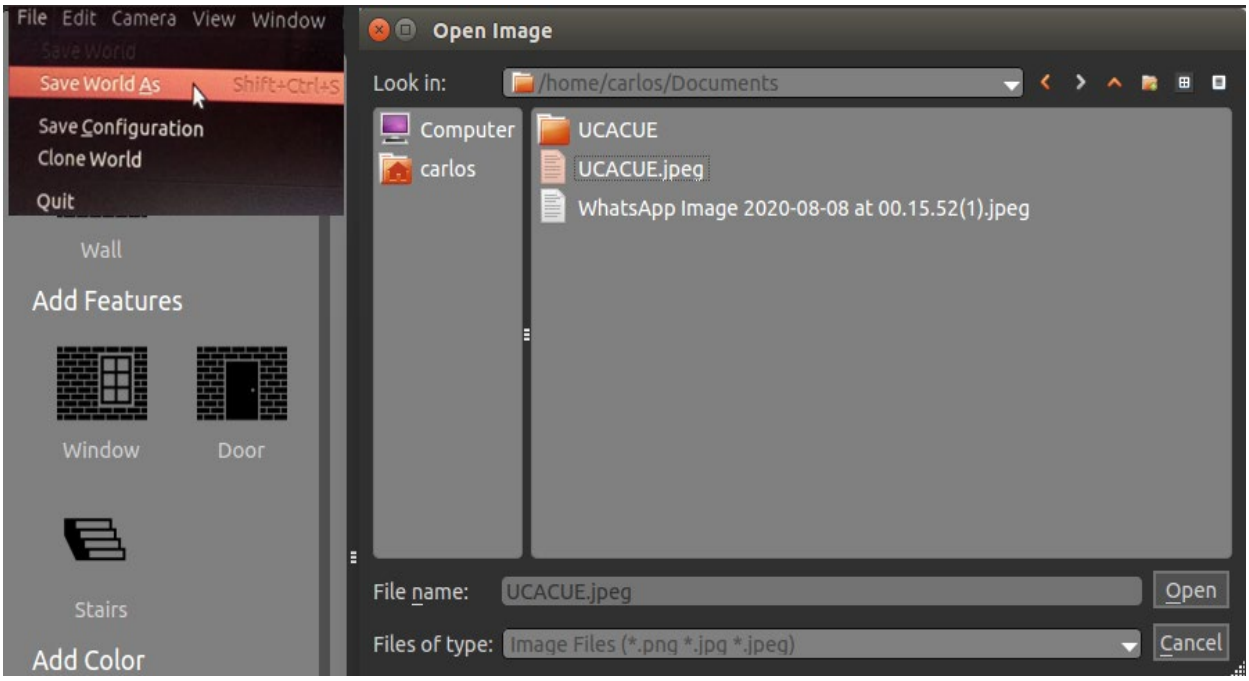


Fig. 47: Guardando el entorno creado

Fuente: Creación propia

En un nuevo terminal ejecutamos el comando descrito, ahora cada vez que trabajemos con este archivo lo invocamos tal y como se indica eso mostrara la ruta completa del archivo y veremos al robot TurtleBot 2 dentro del entorno.

```
>> roslaunch turtlebot_gazebo turtlebot_world.launch
world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/boturtle.world
```

```
/opt/ros/kinetic/share/turtlebot_gazebo/launch/turtlebot_world.launch http://localhost
carlos@carlos-Satellite-C45-A:~$ roslaunch turtlebot_gazebo turtlebot_world.laun
ch world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/EntorUcacue.world
... logging to /home/carlos/.ros/log/a5ab52a6-fa97-11ea-a872-a4db3039ed4b/roslau
nch-carlos-Satellite-C45-A-4399.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: Traditional processing is deprecated. Switch to --inorder processing!
To check for compatibility of your document, use option --check-order.
For more infos, see http://wiki.ros.org/xacro#Processing_Order
xacro.py is deprecated; please use xacro instead
started roslaunch server http://carlos-Satellite-C45-A:39017/
```

Fig. 48: Actuación del entorno roboturtle.worl

Fuente: Creación propia

3.1.1.10. Escribir un guion para TurtleBot 2

Para ordenar que el robot TurtleBot 2 realice una instrucción, realizamos mediante la escritura de un guion o programa, dentro de un script de Python logramos crear y modificar las instrucciones del robot. En la tabla explicamos el procedimiento a seguir y los comandos a ejecutar para escribir el guion de nuestro robot.

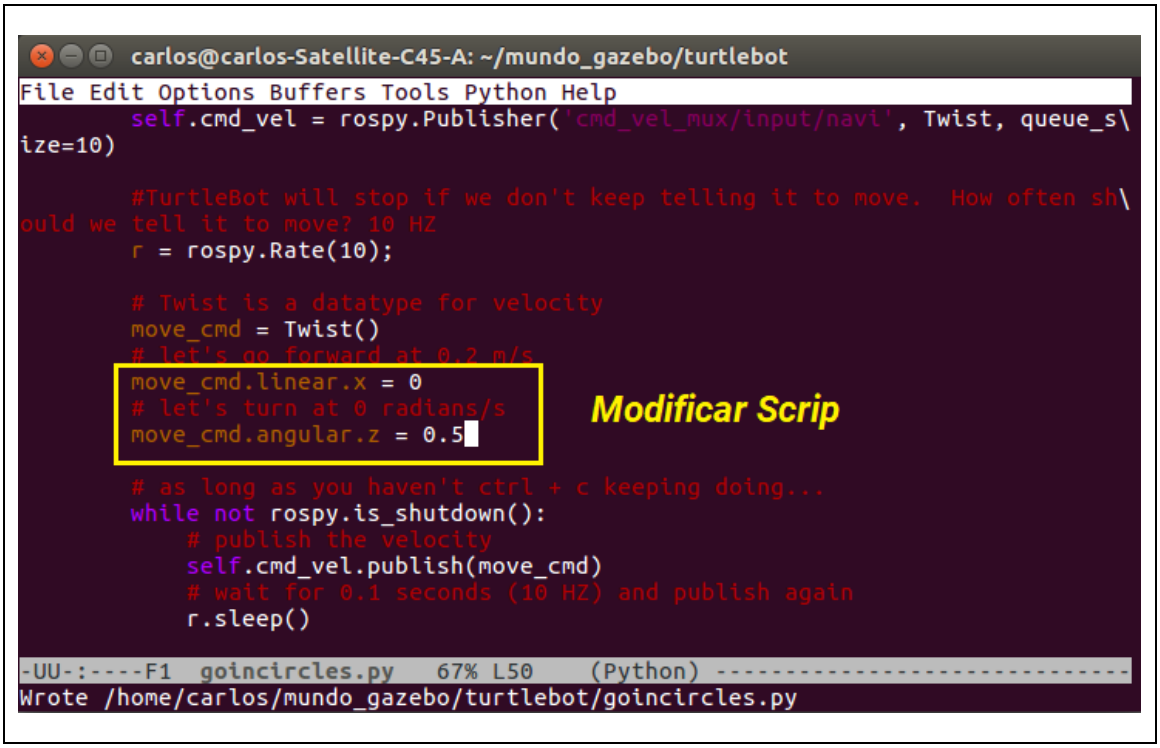
Tabla 16: Pasos para ejecutar scrip Paython

Comando	Descripción
`>> mkdir ~/mundo_gazebo`	Crear un directorio nuevo
`>> cd ~/mundo_gazebo/`	Cambio de directorio a mundo_gazebo
`>> git clone https://github.com/markwsilliman/turtlebot/`	Descargar fuente de plataforma github y crear una cuenta para poder ejecutar.
`>> cd turtlebot`	Cambio de directorio dentro de mundo_gazebo a trtlebot

`>> python goforward.py`
`>> python goincircles.py`
`>> python draw_a_square.py`
	TurtleBot 2 avanza en línea recta
	TurtleBot 2 gira sobre su eje
	TurtleBot 2 dibuja un cuadrado

`>> cp goforward.py goincircles.py`	Crea una copia
`>> emacs -nw goincircles.py`	Abre una ventana de scrip

```
carlos@carlos-Satellite-C45-A: ~/mundo_gazebo/turtlebot
carlos@carlos-Satellite-C45-A:~$ mkdir ~/mundo_gazebo
carlos@carlos-Satellite-C45-A:~$ cd ~/mundo_gazebo/
carlos@carlos-Satellite-C45-A:~/mundo_gazebo$ git clone https://github.com/markw
silliman/turtlebot/
Cloning into 'turtlebot'...
remote: Enumerating objects: 102, done.
remote: Total 102 (delta 0), reused 0 (delta 0), pack-reused 102
Receiving objects: 100% (102/102), 26.82 KiB | 0 bytes/s, done.
Resolving deltas: 100% (52/52), done.
Checking connectivity... done.
carlos@carlos-Satellite-C45-A:~/mundo_gazebo$ cd turtlebot
carlos@carlos-Satellite-C45-A:~/mundo_gazebo/turtlebot$ python goforward.py
[INFO] [1600553981.068572, 0.000000]: To stop TurtleBot CTRL + C
^C[INFO] [1600553985.230825, 5536.000000]: Stop TurtleBot
[INFO] [1600553986.341190, 5537.000000]: GoForward node terminated.
carlos@carlos-Satellite-C45-A:~/mundo_gazebo/turtlebot$ cp goforward.py goincirc
les.py
carlos@carlos-Satellite-C45-A:~/mundo_gazebo/turtlebot$ emacs -nw goincircles.py
```



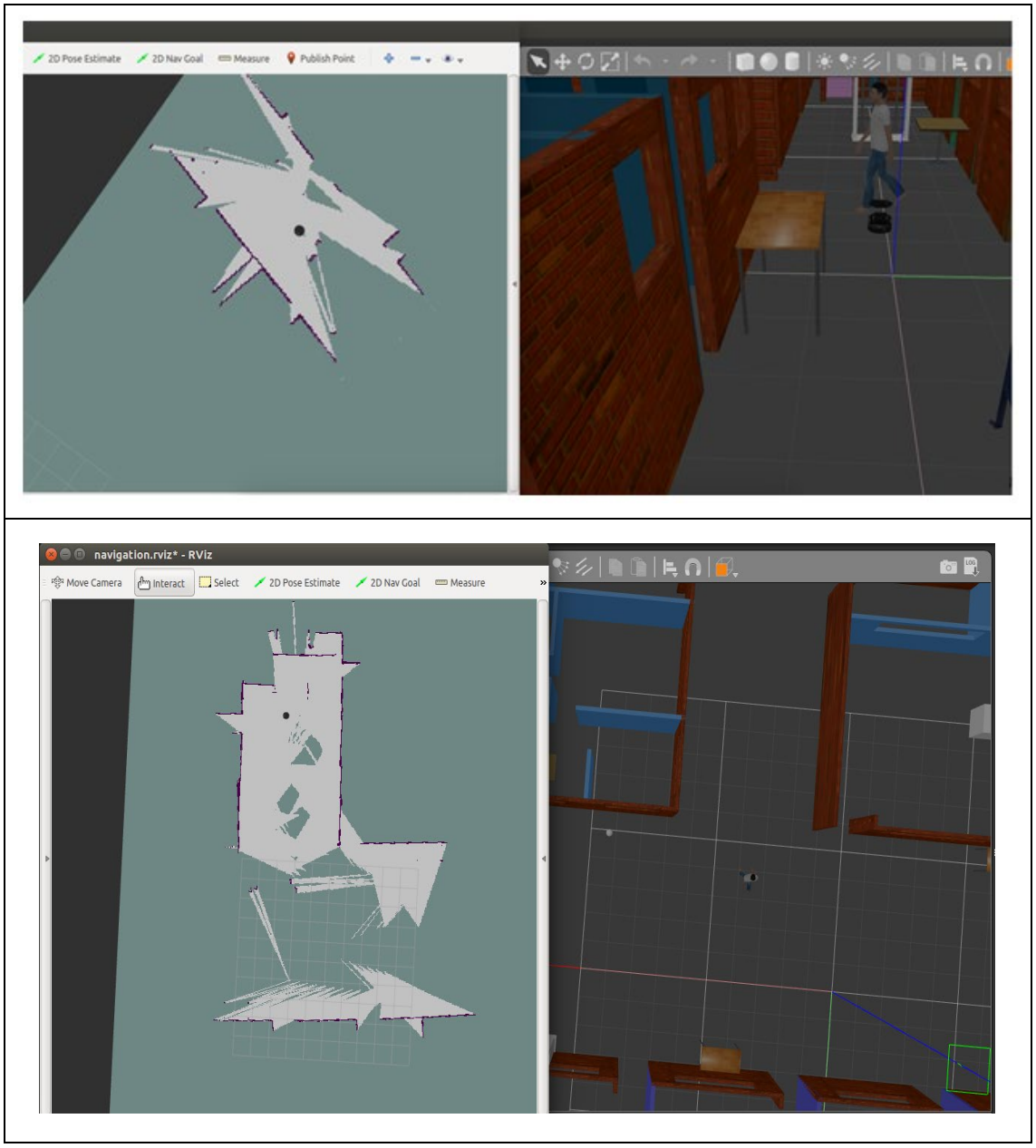
Fuente: Creación propia

3.1.1.11. Instituyendo un mapa con TurtleBot 2

Aquí probamos si el robot es capaz de construir un mapa dentro de las instalaciones de la facultad de Ingenierías de la Ucacue, y luego en base a ese mapa el robot pudo navegar de forma autónoma, aplicando la condición de memoria que le permite al robot recordar el entorno por donde se movió. Estos son los pasos.

Tabla 17:Pasos para construir un mapa con TurtleBot 2

1) Directorio nuevo	`>> mkdir ~/mapa_turtlebot`
2) Mundo Gazebo	`>> roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/carlos/turtlebot_robot_gazebo_ worlds/EntorUcacue.world`
3) Construcción del mapa	`>> roslaunch turtlebot_gazebo gmapping_demo.launch`
4) Visualizador RVIZ	`>> roslaunch turtlebot_rviz_launchers view_navigation.launch`
5) Teleoperador	`>> roslaunch turtlebot_teleop keyboard_teleop.launch`
6) Guardar mapa	`>> rosrun map_server map_saver -f /home/carlos/simulacion_turtlebot/robotsimul`



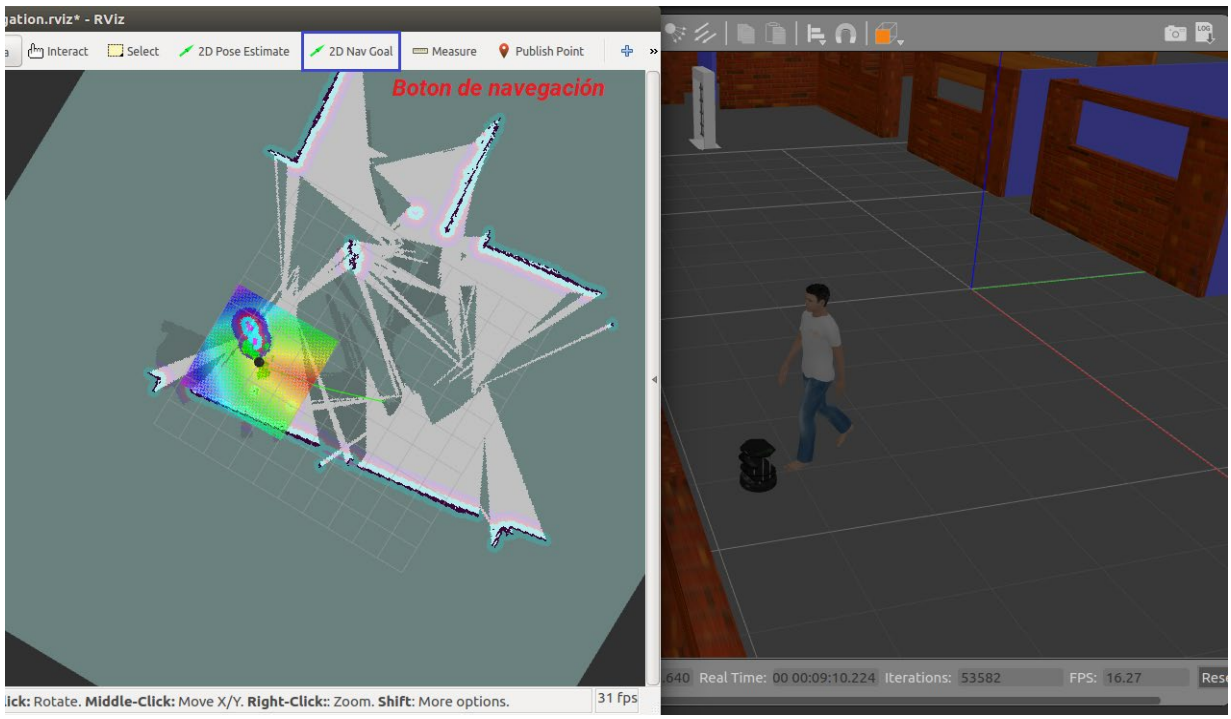
Fuente: Creación propia

3.1.1.12. Navegación Autónoma del TurtleBot 2

Partiendo del mapa conocido realizamos las pruebas de navegación del robot para comprobar si el robot puede navegar de un lugar a otro con la ayuda de la instrucción del botón *2D Naw Goal*. Estos son los pasos a seguir.

Tabla 18: Navegación Autónoma del robot

Descripción	Comando
Mundo Gazebo	`roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/EntorUcacue.world`
Demostración de navegación	`>> roslaunch turtlebot_gazebo amcl_demo.launch map_file:=/home/carlos/simulacion_turtlebot/simulrobot.yaml`
Visualizador RVIZ	`>> roslaunch turtlebot_rviz_launchers view_navigation.launch`



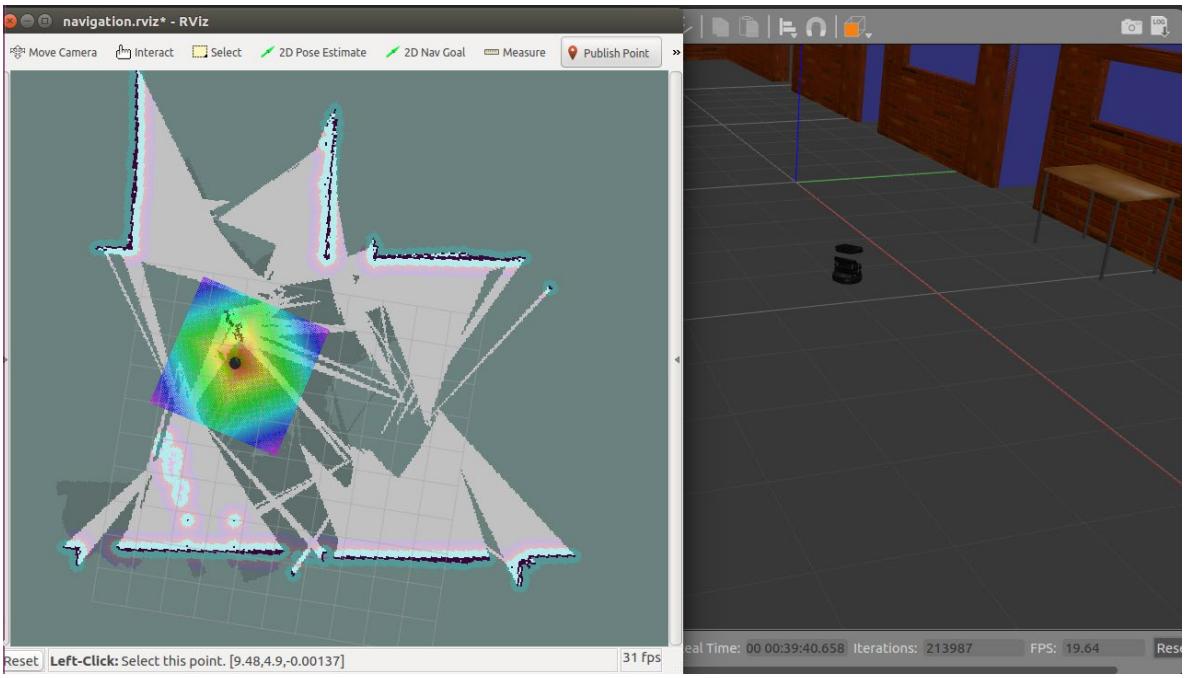
Fuente: Creación propia

3.1.1.13. Avanzando y evitando obstáculos usando un código

Para trabajar dentro de este apartado hacemos uso del repositorio Github, como ya lo tenemos clonado únicamente hacemos uso de su repositorio, resultó un poco más complicado porque el comando que se manejó aquí es nuevo y desconocido, logramos dar instrucciones al robot diciendo que realice; avanzar una cierta distancia independientemente del camino que se escoja. A continuación, siguiendo el orden de la tabla 19, logramos mover al robot evitando cualquier obstaculo.

Tabla 19: Movimiento del robot evitando obstáculos

Descripción	Comando
Mundo Gazebo	`roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/EntorUcacue.world`
Demostración de navegación	`>> roslaunch turtlebot_gazebo amcl_demo.launch map_file:=/home/carlos/simulacion_turtlebot/simulrobot.yaml`
Visualizador RVIZ	`>> roslaunch turtlebot_rviz_launchers view_navigation.launch`
Ejecutar el scrip	`>> python ~/hmundo/turtlebot/goforward_and_avoid_obstacle.py`



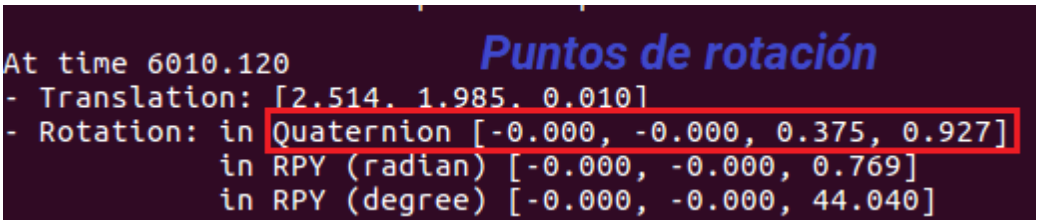
Fuente: Creación propia

3.1.1.14. Ir a una ubicación específica en un mapa usando un código

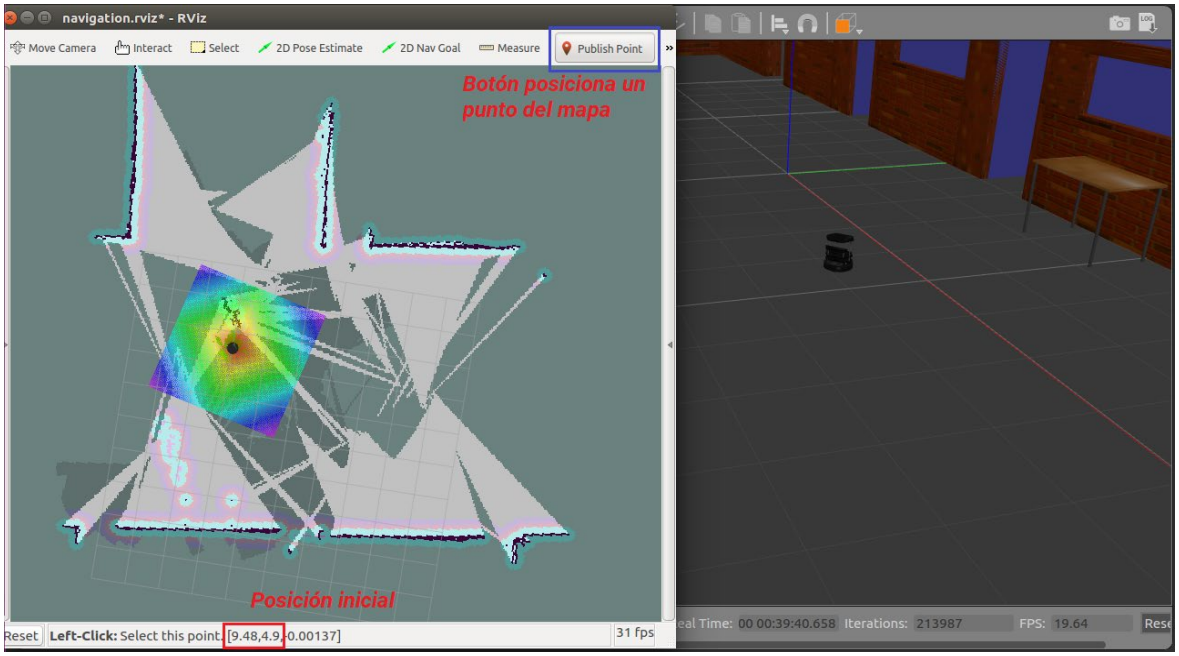
De igual manera que el apartado anterior, aquí empleamos los archivos del repositorio de Github para mover al robot de un punto a otro punto específico con tan solo enviar un comando dentro del mapa generado. En el orden que se indica en la tabla 20, ejecutamos cada uno de los comandos.

Tabla 20. Traslado del robot de un punto a otro

Descripción	Comando
Mundo Gazebo	`roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/EntorUcacue.world`
Demostración de navegación	`>> roslaunch turtlebot_gazebo amcl_demo.launch map_file:=/home/carlos/mapa_turtlebot/simulrobot.world`
Visualizador RVIZ	`>> roslaunch turtlebot_rviz_launchers view_navigation.launch`
Rotación del robot	`>> rosrun tf tf_echo /map /base_link`

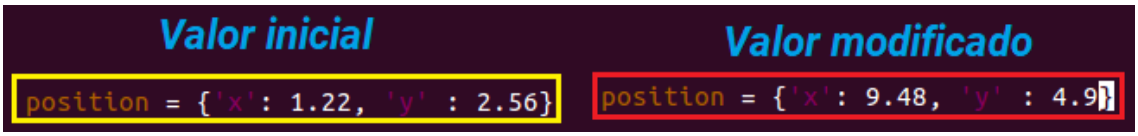
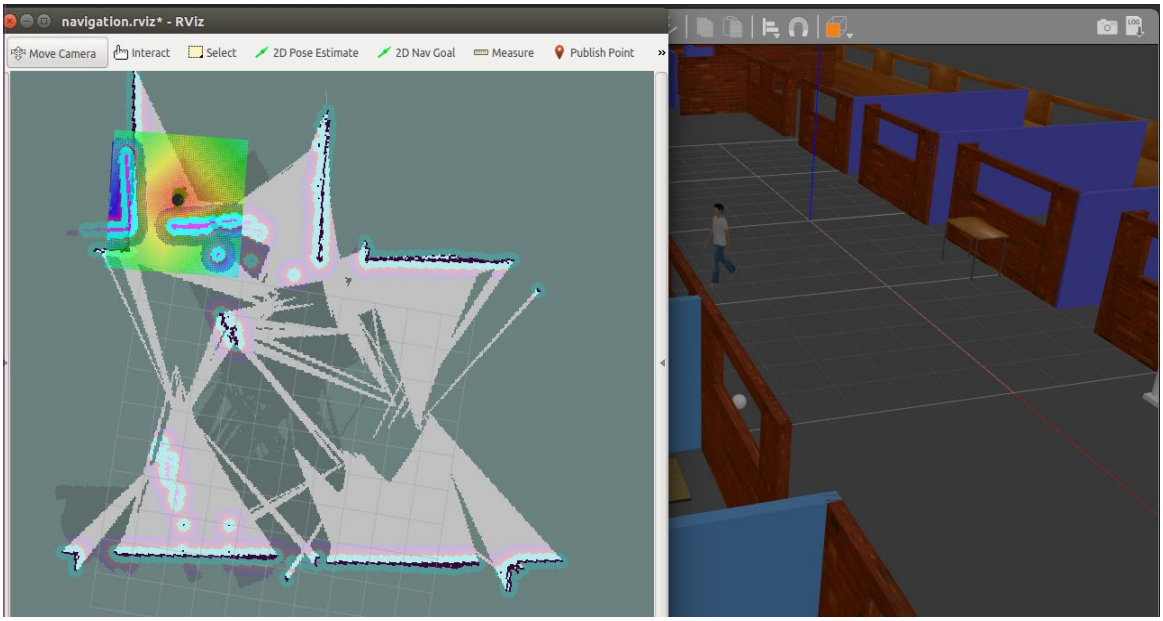


Dentro de la ventana del *Rviz*, seleccionamos *Publish pointy*, para elegir un punto en el mapa, indicando hacia donde queremos que se mueva el robot TurtleBot 2. Anotamos los dos números que se muestran en la esquina inferior izquierda de la ventana de *Rviz*, tal y como se indica en la siguiente figura.



Cambiamos de directorio	`>> cd ~/mundo_gazebo/turtlebot`

Ejecutamos los comando y editamos en el scrip	`>> go_to_specific_point_on_map.py`
	
	`>> emacs -nw go_to_specific_point_on_map.py`

Modificamos los valores antiguos por los nuevos:	`>> position = {'x': 1.22, 'y': 2.56}` `>> position = {'x': 9.48, 'y': 4.9}`
	
Guardar cambios y salir	`Ctrl+X; Ctrl+S y Ctrl+X; Ctrl+C`
Ejecutamos el scrip	`>> python go_to_specific_point_on_map.py`
	

Fuente: Creación propia

3.1.1.15. Tomando una foto

Como se mencionó anteriormente en el capítulo 2, una de las características que tiene el robot TurtleBot 2, tanto en el simulador como en real dispone de una cámara que es un visor de imágenes ROS, y gracias a eso logramos visualizar y fotografiar el entorno en donde se está moviendo el robot. En la tabla 21, se observa los pasos que debemos seguir para conseguir que el robot tome una foto del entorno.

Tabla 21: Fotografía por el robot

Descripción	Comando
Inicie el mundo Gazebo	`roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/EntorUcacue.world`
Cambiamos de directorio	`>> cd ~/mundo_gazebo`
Ejecutamos image_view	`>> rosrun image_view image:= /camera/rgb/image_raw`
Para tomar una foto, lo hacemos con el mouse dando clic derecho	
Abrir la imagen cualquiera de los 2	`>> ristretto pasillo_ucacue.npg` `>> eog pasillo_ucacue.png`
Para tomar una foto lo podemos hacer también utilizando un guion. Aquí necesitamos de archivo del repositorio Github.	
Ejecutamos el scrip	`>> python take_photo.py`
Cambiar nombre de imagen y comprobar	`>> python take_photo.py _image_title:=pasillo_ucacue.png` `>> rosparam list | grep image_title`
	

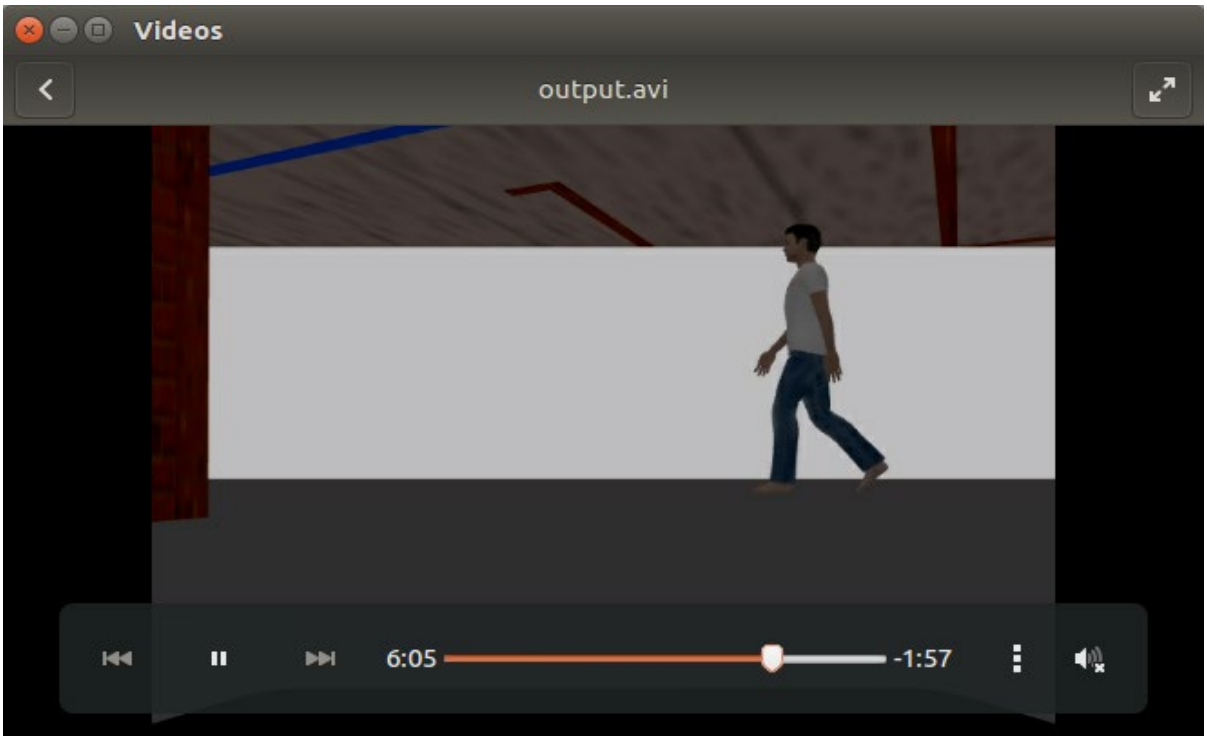
Fuente: Creación propia

3.1.1.16. Grabar un video

Haciendo uso de la cámara que tiene el robot TurtleBot 2 en el simulador, grabamos un video del entorno de los pasillos por donde se mueve y el archivo lo almacenamos en una carpeta.

Tabla 22: Grabar un video con el robot

Descripción	Comando
Mundo Gazebo	`roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/carlos/turtlebot_robot_gazebo_worlds/EntorUcacue.world`
Ejecute el teleoperador	`>> roslaunch turtlebot_teleop keyboard_teleop.launch`
Ejecutamos el comando image_view	`>> rosrun image_view image:=/camera/rgb/image_raw`
Cambiamos de directorio	`>> cd ~/mundo_gazebo/turtlebot/`
Ejecutamos el comando grabar	`>> rosrun image_view video_recorder image:=/camera/rgb/image_raw`
Cambiar nombre de imagen y comprobar	`>> python take_photo.py _image_title:="pasillo.png"` `>> rosparam list | grep image_title`
Presionando Ctrl+C finalizamos la grabación del video	
Reproducir el video	`>> vlc output.avi`



Fuente: Creación propia

3.2. Resultados

3.2.1. Resultados del Robot TurtleBot 2 en simulador

Una vez instalado y modificado el archivo l*aunch* de *hector_slam*, el fin es obtener navegación autónoma del robot, con fines prácticos descargamos el archivo bag de la plataforma (wiki ros), que se trata de un sistema de mapeo portátil, con nombre RoboCup 2011 Rescue Arena. El paquete *hector slam* genera la transformación entre el mapa y el marco base, sin embargo, necesitamos transformaciones entre el mundo y el marco del mapa y el marco base y el marco del láser.

```
>> wget https://storage.googleapis.com/google-code-archive-
downloads/v2/code.google.com/tu-darmstadt-ros-
pkg/Team_Hector_MappingBox_RoboCup_2011_Rescue_Arena.bag
```

Para iniciar el mapeo con *hector_slam* debemos de ejecutar los siguientes comandos en el orden que se indica y en diferentes terminales.

Terminal 1

Primero, ejecuta el *roscore,* conocido también con rosmaster.

```
>> roscore
```

```
roscore http://carlos-Satellite-C45-A:11311/
carlos@carlos-Satellite-C45-A:~$ roscore
... logging to /home/carlos/.ros/log/f9a7c7ee-e36e-11ea-be30-a4db3039ed4b/roslau
nch-carlos-Satellite-C45-A-23250.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://carlos-Satellite-C45-A:36227/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [23260]
ROS_MASTER_URI=http://carlos-Satellite-C45-A:11311/

setting /run_id to f9a7c7ee-e36e-11ea-be30-a4db3039ed4b
process[rosout-1]: started with pid [23273]
started core service [/rosout]
```

Fig. 49: Inicialización del rosmaster

Fuente: Creación propia

Terminal 2

Aquí ejecutamos el comando que tiene relación con el tiempo simulado *Tutorial.bag*

```
>> rosbag play Lecture3SLAM_Tutorial.bag -clock
```

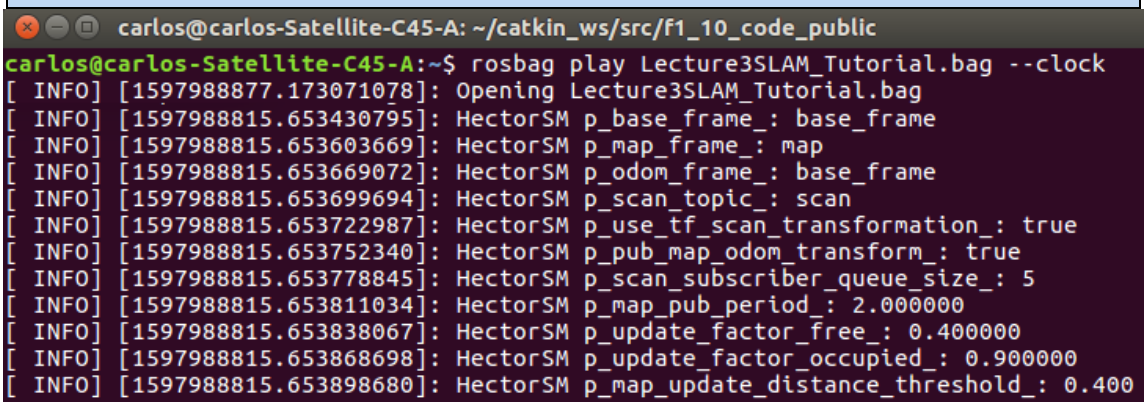


Fig. 50: Ejecución del mapa tutorial

Fuente: Creación propia

Terminal 3

Este archivo de lanzamiento inicia el nodo *hector_mapping* así como los nodos *hector_trajectory_server* y *hector_geotiff* necesarios para generar mapas de *geotiff*. Ejecutando el siguiente comando vemos en la ventana Rviz y en tiempo real, como el robot se mueve siguiendo el mapa guardado.

```
>> roslaunch hector_slam_launch tutorial.launch
```

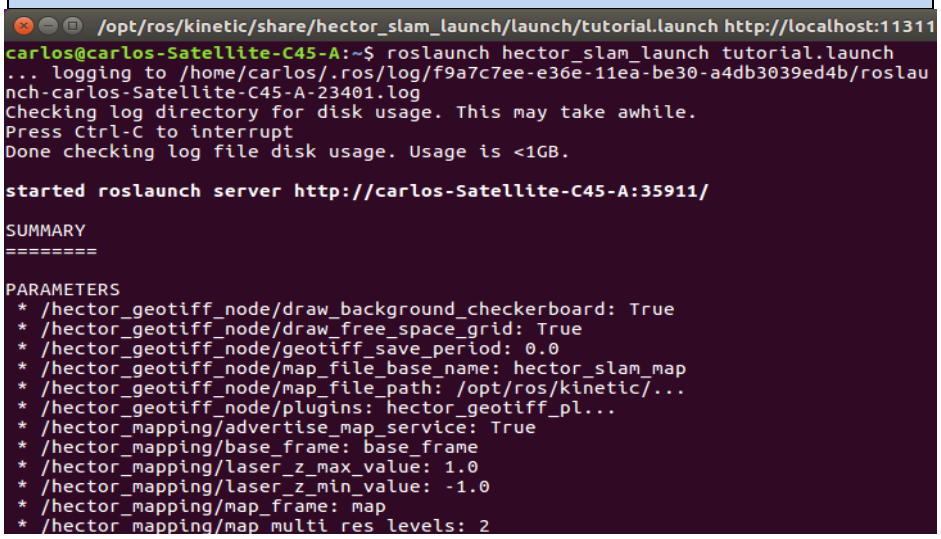


Fig. 51: Ejecución del comando Hector_Slam

Fuente: Creación propia

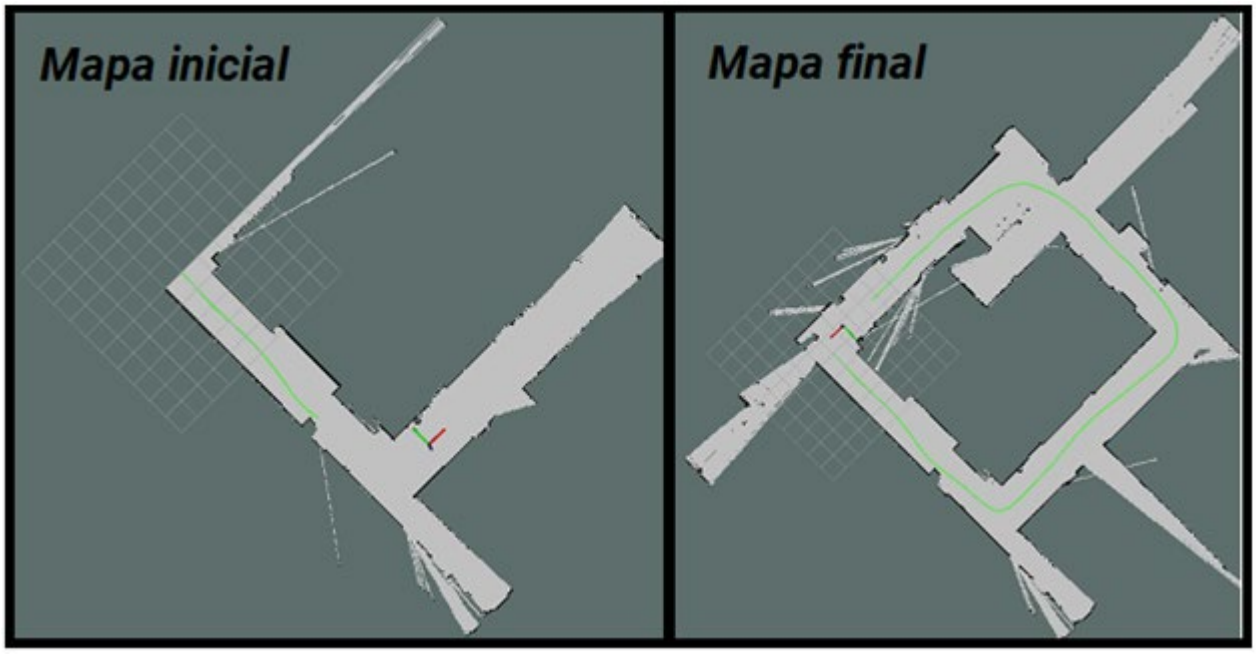


Fig. 52: Generación del mapa por el robot con SLAM

Fuente: Creación propia

Terminal 4

El comando que se muestra a continuación guarda el proceso de mapeo del archivo getiff estos archivos se almacenan dentro de la carpeta hector_slam / hector:geotiff / maps. Debería encontrar un archivo *.tif* que contiene los datos de la imagen y un archivo *.tfw* que contiene la información de georreferencia.

```
>> rostopic pub syscommand std_msgs / String "savegeotiff"
>> rosrun map_server map_saver -f mymap
```

Comando rqt_graph

El comando *rqt_graph* es un complemento GUI que sirve para visualizar los gráficos de cálculo de ROS. El comando es capaz de crear gráficos dinámicos y así poder saber lo que sucede en el sistema. Hay versiones de ros que vienen incluidos el paquete caso contrario hay que instalar independientemente.

Ejecutando el comando *rqt_dep* averiguamos todos los paquetes que dependen de *rqt_graph*.

```
>> rqt_graph
```

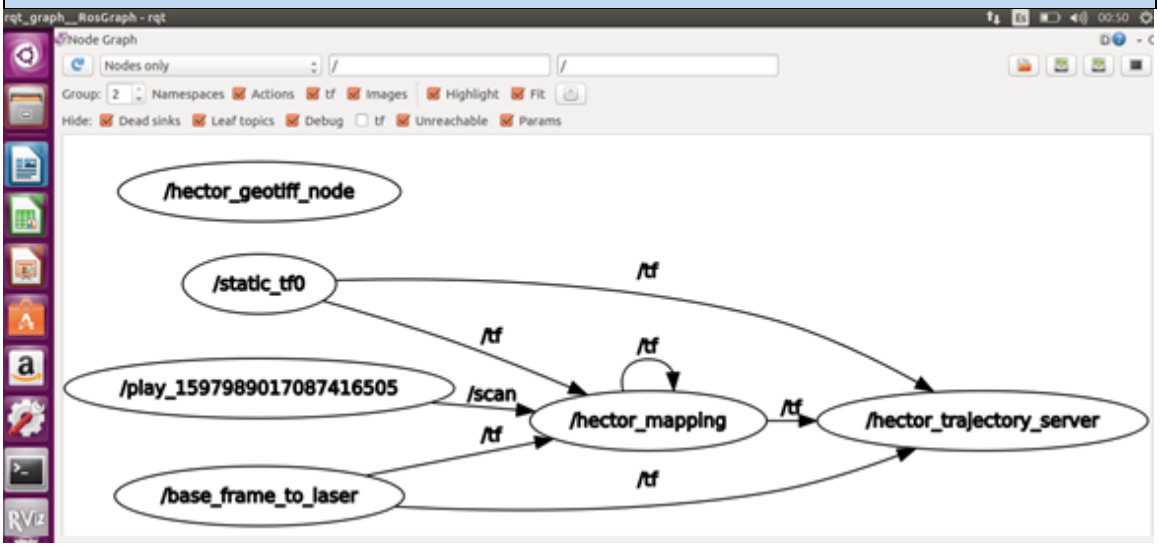


Fig. 53: Distribución de nodos para hector_slam simulador

Fuente: Creación propia

3.2.2. Resultados del Robot TurtleBot 2 físico

Para llegar a los resultados finales sobre la generación de mapas empleando el robot físico y aplicando SLAM, primero realizamos las respectivas configuraciones entre el robot físico y el computador personal utilizado para controlar al robot.

- Para ejecutar *hector slam* trabajamos dentro del directorio: *cd catkin_ws/*
- Para *setear* y lograr que el robot no genere fallas ejecutamos el siguiente comando: *source devel/setup.bash*
- Para poner en marcha al robot recomendamos seguir y respetar siempre es siguiente orden:

Terminal 1: *roscore*

Terminal 2: *roslaunch rplidar_ros rplidar_a3.launch*

Terminal 3: *roslaunch hector_slam_launch tutorial.launch*

Configuración de la red entre el robot y computador personal

Para la configuración de la red lo realizamos mediante el tipo de comunicación SSH (*Secure Shell)*, que no es más que un protocolo de conexión entre dos sistemas.

El objetivo es lograr que la máquina maestra y la maquina esclavo utilicen el mismo *rosmaster*.

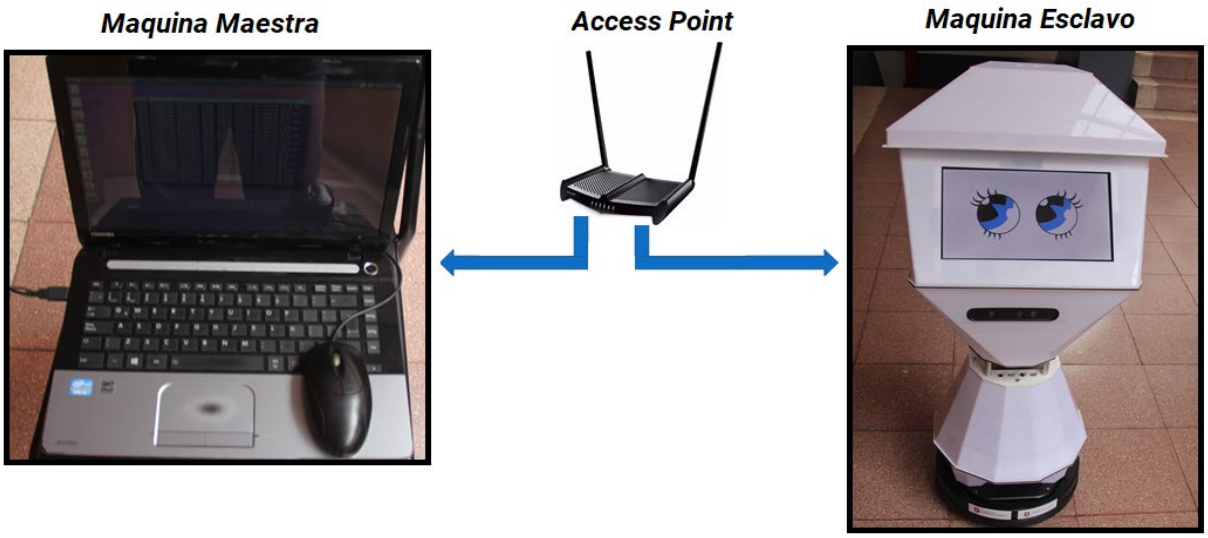


Fig. 54: Configuración de la red

Fuente: Creación propia

Obtencion de direcciones IP

La red de comunicación en ROS es bidireccional entre los diferentes nodos. Para lograr esa comunicación primero se tiene que conocer las direcciones IP, tanto del nodo maestro, como las IP del nodo esclavo, ejecutando el comando ifconfig obtenemos la IP maestro.

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ ifconfig
enp3s0    Link encap:Ethernet  HWaddr 08:9e:01:e5:86:07
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:19

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:28786 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28786 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2150265 (2.1 MB)  TX bytes:2150265 (2.1 MB)

wlp2s0    Link encap:Ethernet  HWaddr a4:db:30:39:ed:4b
          inet addr:192.168.0.101  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::d6cc:af3a:902e:f35/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:68 errors:0 dropped:0 overruns:0 frame:0
          TX packets:231 errors:0 dropped:0 overruns:0 carrier:0
```

Fig. 55: Obtención de IP de la maquina maestro

Fuente: Creación propia

Instalación del openssh y prueba tu conexión

Primero, instalamos el servidor *ssh* en la máquina maestro, en la máquina del robot ya está instalado y configurado.

Tabla 23: Requisitos para configuración de la red

Detalle	Comando
Comando de instalación del servidor	`>> sudo apt-get install openssh-server`
Configuración de conexión entre el robot TurtleBot 2 y el ordenados maestro	`>> ssh carlos@192.168.0`

Fuente: Creación propia

Configuración de la maquina maestro y esclavo para usar el mismo ROS Master

En un terminal nuevo ejecutamos el comando *gedit .bashrc*, y se abre el archivo *bash*, en la parte final incluimos unas líneas dentro del archivo y escribimos la dirección IP de maquina esclavo guardamos y luego probamos.

```
>> gedit .bashrc
```

```
Open                                                                    Save
# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/
[;&|]\s*alert$//'\'')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
source /opt/ros/kinetic/setup.bash
export TURTLEBOT_3D_SENSOR=kinect

source /opt/ros/kinetic/setup.bash
#The IP address for the Master node
export ROS_MASTER_URI=http://192.168.0.104:11311
#The IP address for your device/host IP address
export ROS_HOSTNAME=192.168.0.101
```

Modoficacion del bash

Fig. 56: Modificación del archivo bash

Fuente: Creación propia

Pruebas de configuración

Ejecutando el comando que se muestra a continuación observamos la lista de dos tópicos que nos indica que están correctamente enlazados entre la maquina maestro y esclavo.

```
>> rostopic list
```

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ rostopic list
/rosout
/rosout_agg
carlos@carlos-Satellite-C45-A:~$
```

Fig. 57: Tópicos de enlace entre maestro y esclavo

Fuente: Creación propia

Una vez que ya esté funcionando todo ejecutamos el comando *rostopic list* y vemos la lista de todos los tópicos.

```
carlos@carlos-Satellite-C45-A: ~
carlos@carlos-Satellite-C45-A:~$ rostopic list
/diagnostics
/diagnostics_agg
/diagnostics_toplevel_state
/joint_states
/mobile_base/commands/controller_info
/mobile_base/commands/digital_output
/mobile_base/commands/external_power
/mobile_base/commands/led1
/mobile_base/commands/led2
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/sound
/mobile_base/commands/velocity
/mobile_base/controller_info
/mobile_base/debug/raw_control_command
/mobile_base/debug/raw_data_command
/mobile_base/debug/raw_data_stream
/mobile_base/events/bumper
/mobile_base/events/button
/mobile_base/events/cliff
/mobile_base/events/digital_input
/mobile_base/events/power_system
/mobile_base/events/robot_state
/mobile_base/events/wheel_drop
/mobile_base/sensors/core
/mobile_base/sensors/dock_ir
/mobile_base/sensors/imu_data
/mobile_base/sensors/imu_data_raw
/mobile_base/version_info
/mobile_base_nodelet_manager/bond
/odom
/rosout
/rosout_agg
/tf
```

Fig. 58: Lista de tópicos

Fuente: Creación propia

Finalmente ejecutamos el comando *kobki_keyop*, a la par con los comandos de la maquina esclavo, para que se enlacen entre ellos, y desde la maquina maestro podremos administrar todos los movimientos del robot, por ejemplo, ordenamos que el robot avance hacia adelante, hacia atrás, giros, etc. También podemos regular la velocidad del robot.

```
>> roslaunch kobuki_keyop keyop.launch
```

```
carlos@carlos-Satellite-C45-A:~$ roslaunch kobuki_keyop keyop.launch
... logging to /home/carlos/.ros/log/ba0bc1bc-f43f-11ea-a24d-645d86cffb7f/roslaunch-carlos-Satellite-C45-A-4461.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.101:42537/

SUMMARY
========

PARAMETERS
 * /keyop/angular_vel_max: 6.6
 * /keyop/angular_vel_step: 0.33
 * /keyop/linear_vel_max: 1.5
 * /keyop/linear_vel_step: 0.05
 * /keyop/wait_for_connection_: True
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
  /
    keyop (kobuki_keyop/keyop)

ROS_MASTER_URI=http://192.168.0.104:11311

process[keyop-1]: started with pid [4470]
[ INFO] [1599837702.098731422]: KeyOpCore : using linear  vel step [0.05].
[ INFO] [1599837702.099002735]: KeyOpCore : using linear  vel max  [1.5].
[ INFO] [1599837702.099190730]: KeyOpCore : using angular vel step [0.33].
[ INFO] [1599837702.099288502]: KeyOpCore : using angular vel max  [6.6].
[ WARN] [1599837702.156802719]: KeyOp: could not connect, trying again after 500ms...
[ INFO] [1599837702.657210351]: KeyOp: connected.
Reading from keyboard
---------------------------
Forward/back arrows : linear velocity incr/decr.
Right/left arrows : angular velocity incr/decr.
Spacebar : reset linear/angular velocities.
d : disable motors.
e : enable motors.
q : quit.
[ INFO] [1599837707.539819100]: KeyOp: angular velocity incremented [0|0.33]
[ INFO] [1599837710.183048255]: KeyOp: angular velocity decremented [0|0]
```

Fig. 59: Ejecutor del rosmaster que controla los movimientos del robot

Fuente: Creación propia

Una vez comprobado que la red esté correctamente configurada, procedemos a realizar las respectivas pruebas del robot TurtleBot 2, sobre los pasillos de la primera planta de la Unidad Académica de la Universidad, logrando hacer mover al robot de un punto a otro aplicando SLAM, el resultado de la generación de mapas lo pudimos observar en la pantalla táctil del robot.

En la figura 60, observamos los movimientos y como gracias al sensor Lidar logra generar el mapa del entorno.

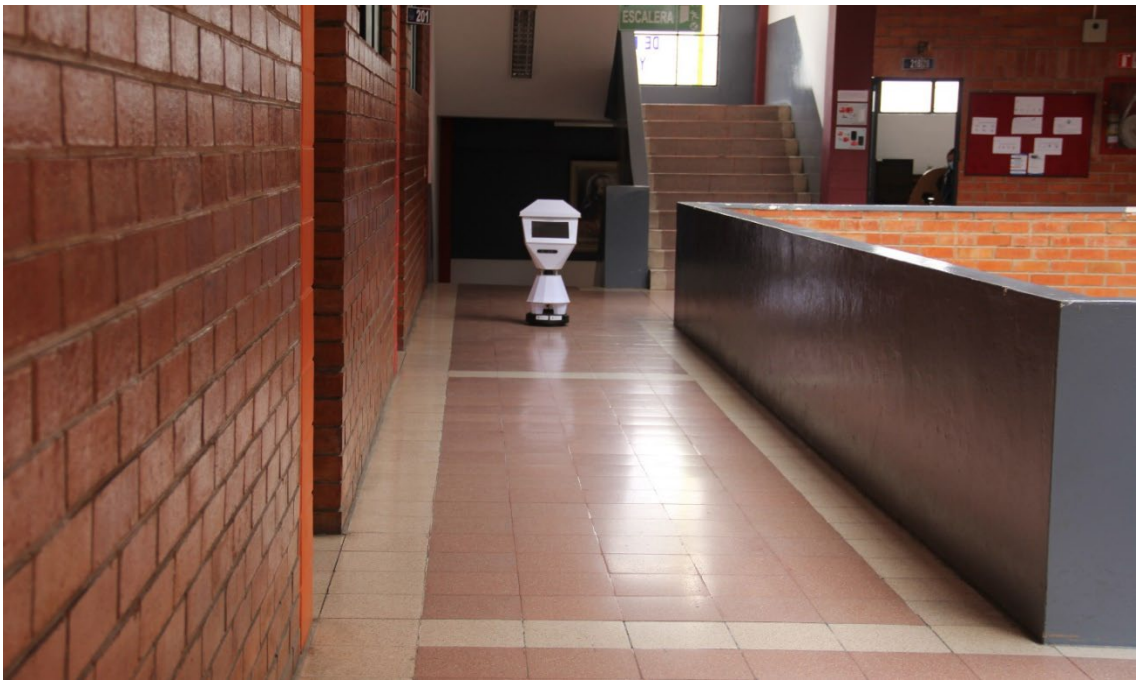


Fig. 60: Entorno de prueba y movilización del robot TurtleBot 2

Fuente: Creación propia

Fig. 61: Robot TurtleBot 2

Fuente: Creación propia

En la figura 62, indicamos el mapa generado por el robot TurtleBot 2, empleando SLAM, durante las pruebas realizadas en el pasillo de la Unidad Académica de la Universidad, estos son los resultados.

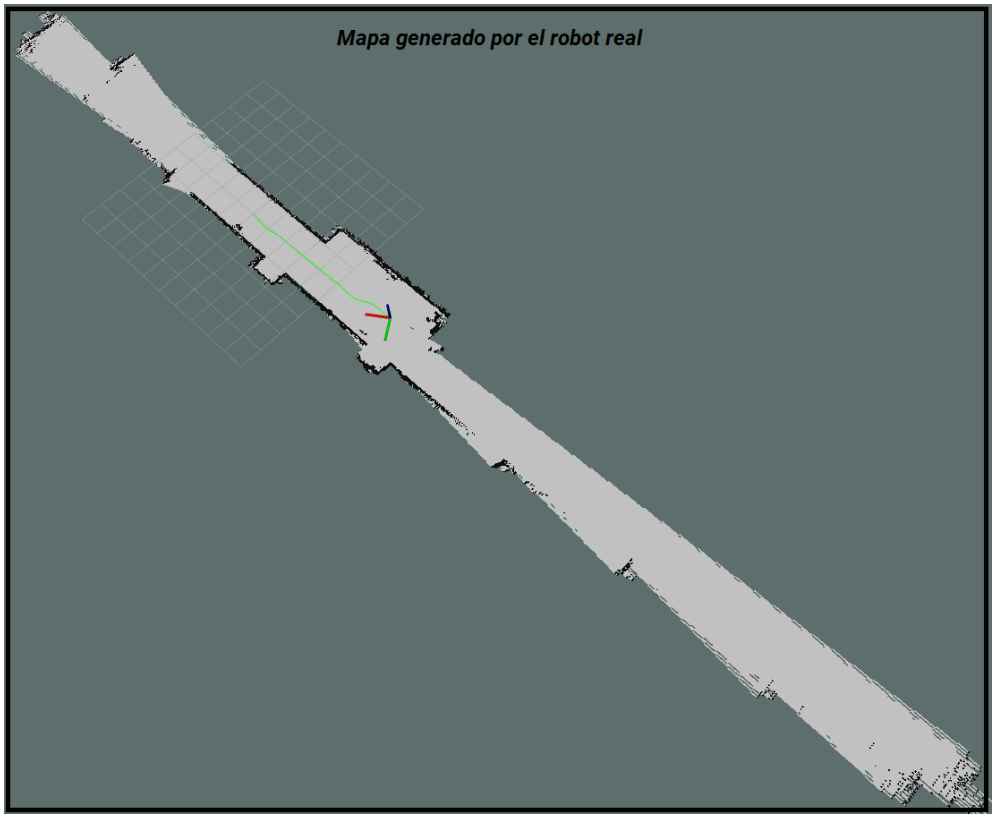


Fig. 62: Mapa generado por el robot TurtleBot 2 usando SLAM

Fuente: Creación propia

Finalmente, en la figura 63, indicamos la configuración de los nodos de Hector Slam.

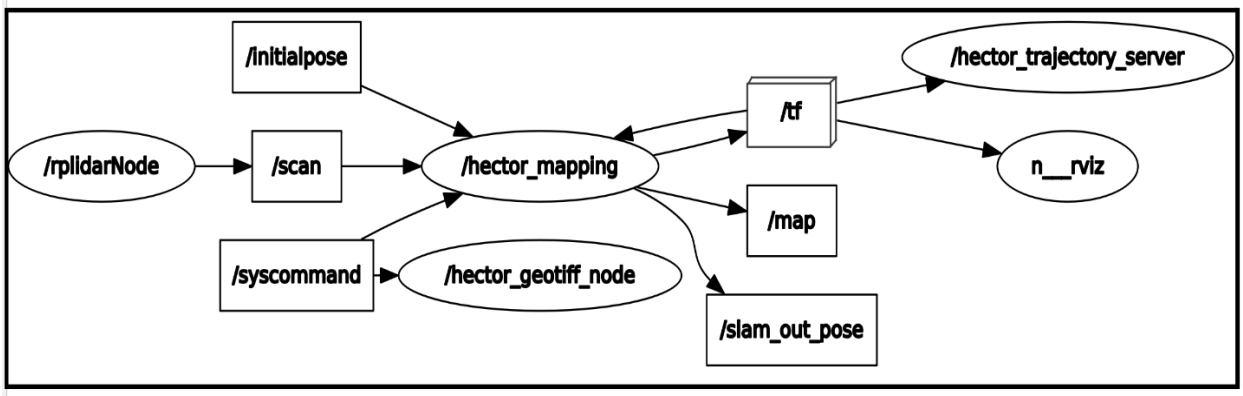


Fig. 63: Distribución de nodos de Hector Slam

Fuente: Creación propia

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

Una vez finalizado el proyecto concluimos lo siguiente:

- Con éxito logramos realizar las pruebas con el robot TurtleBot 2, aplicando SLAM y empleando el sensor RPLIDAR A3 (real), y la cámara 3D marca Orbbec Astra (equivalente de simulación), logrando obtener como resultado final, hacer que el robot se mueva (real) de forma autónoma (simulación) y pueda generar los mapas.
- Para la consecución del proyecto fue indispensable implementar el sistema operativo robótico ROS, o conocido también como el conjunto de herramientas que trabaja sobre la distribución Linux/Ubuntu, toda la instalación lo realizamos sin ningún problema, el inconveniente que tuvimos fue al momento de inicializar el comando ROS *init*, en un terminal, ventajosamente se logró corregir el error, gracias a las ayudas que se encuentran en la plataforma de (wiki ros), todo esto se realizó con el fin administrar correctamente al robot y que no nos genere ningún problemas.
- Dentro de la plataforma de la comunidad ROS encontramos un sinnúmero de algoritmos de implementación de SLAM, una vez realizado previamente la investigación para nuestros intereses del proyecto, seleccionamos los algoritmos conformes a nuestras necesidades tal es el caso de (Hector SLAM).
- Durante las pruebas realizadas al robot real, con éxito se logró obtener los resultados de generación de mapas del entorno gracias a las ventajas que tiene el sensor Lidar RPLIDAR A3, gracias a su capacidad de censar 360 grados y observar el entorno en el que se está moviendo hasta una distancia de 25 metros. Con éxito también se logró capturar imágenes y video con la cámara del robot, de esa manera comprobamos de lo que el robot está observando (simulador).
- Resaltamos el cumplimiento de los objetivos planteados, pese a diversas dificultades encontradas a lo largo que duro el proyecto, la circunstancia de la pandemia impidió experimentar en forma real durante varios meses. Con total certeza podemos decir que capacitarnos en las funciones básicas del sistema operativo o herramientas ROS, Linux-Ubuntu y todos los paquetes empleados, ha sido una experiencia de mucho aprendizaje

ya que se ha logrado entender las funciones que cumplen cada comando al ejecutarlo en cada sección de pruebas.

- Durante las pruebas realizadas en el simulador encontramos dificultades como ejecución errónea de comandos, instalación de paquetes innecesarios y errores de configuración de archivos. Todo esto convirtiéndose en retos cumplidos porque se logró solucionar a cada error encontrado.

4.2. Recomendaciones

Recomendaciones al sistema:

- A la carrera de Ingeniería Eléctrica de la Universidad Católica de Cuenca recomendaría dar mayor énfasis a este tipo de proyectos que son sumamente importantes dentro del desarrollo tecnológico actual.
- Usar la versión Ubuntu 16.04 LTS junto con ROS Kinetic adicional a los mencionados para este tipo de proyectos utilizar Gazebo7 son programas gratuitos y compatibles entre sí.
- No utilizar las versiones más actuales de ROS y Ubuntu, debido a que algunos paquetes necesarios para este proyecto de investigación todavía no están soportados en las nuevas versiones.

Recomendaciones y Aplicaciones futuras

- El presente documento es punto de partida, para las diversas y futuras investigaciones, considerando que el software probado aquí se debe seguir desarrollando. El software implementado y probado en este proyecto podría servir en aplicaciones futuras como entrega y transporte de objetos de pequeño volumen mediante un robot móvil.
- Las características actuales del robot permiten emplearlo dentro de la Unidad Académica, ayudaría a trasladar objetos como carpetas, documentos y hasta café entre distintas oficinas y dependencias. La utilidad aumenta si bajo las circunstancias actuales es tan relevante el distanciamiento social.
- El campo de aplicaciones de estos sistemas de navegación autónoma en vehículos es amplio, entre otras las compañías Google y Tesla están probando sistemas basados en SLAM para parqueo de un vehículo y navegación en la ciudad. Básicamente son ayudas electrónicas para el manejo de un vehículo, si implementamos en un vehículo real el radar (LIDAR) le va decir donde hay personas u obstáculos. El sensor láser (LIDAR) en nuestro prototipo tiene un alcance de 25 metros, pero en el mercado existen sensores con mayor alcance 100 y 250 metros, lo cual permite al conductor precautelar su seguridad, mediante una pantalla que muestra un mapa incluyendo objetos y peatones (obstáculos) por donde estoy circulando. También se podría aplicar como asistente de colisión, cuando el vehículo se acerca demasiado a otro vehículo de adelante, automáticamente el vehículo reduciría la velocidad y me ayudaría a frenar.

- En el sector público se podría emplear como un robot asistente de hospitales, en esta época de pandemia, por ejemplo, el robot podría entregar medicamentos e insumos evitando contacto entre personal administrativo, logístico, con médicos y pacientes. Además, realizar desinfección de quirófanos, pasillos y cuartos con solo agregar un módulo de UV u Ozono, con esto se lograría evitar el contacto de las personas con radiación y el gas.
- En el sector privados se puede emplear en restaurantes, hoteles, fabricas, etc. El robot podría llevar toallas, sabanas, comida o cualquier encomienda.
- Este sistema de navegación (SLAM) de vehículos autónomos también puede ser empleado en misiones de búsqueda y rescate, en una catástrofe natural y cuando las circunstancias imposibilitan que el ser humano puede acceder al lugar del siniestro, es ahí donde puede actuar el robot asistente, claro ejemplo del robot que monitorio el siniestro de los trabajadores atrapados en un túnel de una mina en Chile.
- Dentro de la empresa privada se podría desarrollar un robot de asistencia domestico que sería un electrodoméstico más en nuestro hogar, este sería un proyecto futuro de emprendimiento, teniendo como socio estratégico a empresas como Indurama o Fibroacero, que son fábricas locales de electrodomésticos, se podría gestionar un convenio para que ellos fabriquen el prototipo del robot y nosotros desarrollar el software, patentado por el investigador y la Universidad.
- Finalmente mencionamos una de las motivaciones para realizar este tipo de investigaciones, si bien es cierto en otros países la tecnología está más avanzada que en el nuestro, por tal motivo somos consumidores del software que ellos desarrollen pagando plataformas y licencias. Nuestro principal reto debe ser, convertirnos en generadores de nuestras propias plataformas, debemos ser desarrolladores de software y así no ser dependientes tecnológicos de otros, si queremos utilizar algo de ellos tenemos que pagar, porque no dejar ese dinero en productores locales.

REFERENCIAS BIBLIOGRÁFICAS

[1] E. Fitriatun, “Plataforma de Robótica Móvil a través del Sistema Operativo de Robots (ROS),” vol. 53, no. 9, pp. 1689–1699, 2019.

[2] A. Villalobos Rivera and J. Gallardo Arancibia, “Diseño e implementación de un observatorio robótico teleoperado basado en Robot Operating System,” *Ingeniare. Rev. Chil. Ing.*, vol. 26, pp. 12–19, 2018.

[3] C. C. Open Robotics, “ROS.org | History,” 23-May-2017. [Online]. Available: https://www.ros.org/history/. [Accessed: 17-Jun-2020].

[4] G. V. Moral, “Programación de comportamientos de un robot autónomo .,” no. Cvc, pp. 1–8, 2016.

[5] E. I. A. Serrano, “Plataforma robótica móvil para experimentos de mapeo y localización simultanea (SLAM) en base a sensores de rango,” vol. 5, p. 146, 2017.

[6] P. Ponce H, “Reconstrucción de Entorno 3D Mediante un sensor LIDAR,” vol. 4, no. 1, pp. 75–84, 2019.

[7] P. Monroy, “Desarrollo de Algoritmo Basado en Inteligencia Artificial y SLAM,” vol. 53, no. 9, pp. 1689–1699, 2013.

[8] Y. B. Fern, R. Garc, and L. Pay, “Uso de Descriptores Hostilísticos para la Localización y Creacion de Mapas.,” 2018.

[9] I. Z. Ibragimov and I. M. Afanasyev, “Comparison of ROS-based visual SLAM methods in homogeneous indoor environment,” *2017 14th Work. Positioning, Navig. Commun. WPNC 2017*, vol. 2018-Janua, no. July 2018, pp. 1–6, 2018.

[10] Open Source Robotics Foundation, “TurtleBot,” 2017. [Online]. Available: https://www.turtlebot.com/. [Accessed: 19-Jun-2020].

[11] A. J. P. Rodríguez, “Desarrollo de Sistema de Navegación para Vehículos Autónomos Terrestres Utilizando ROS,” 2017.

[12] A. S. Vázquez, F. Ramos, and C. De Robótica, “Curso de Robótica Móvil con Arduino y

Android," 2016.

[13] F. Andrade and G. Tejera, "SLAM Estado del Arte," p. 44, 2012.

[14] J. M. M. M. Oscar and G. G. Javier, "SLAM Monocular en Tiempo Real," *Engineering*, 2019.

[15] U. Frese, "Closing a Million-Landmarks Loop."

[16] Shanghai Slamtec, "RPLidar A3 Datasheet," p. 18, 2017.

[17] ROS Components, "RPLIDAR A3 ," 2016. [Online]. Available: https://www.roscomponents.com/es/lidar-escaner-laser/247-rplidar-a2m8.html. [Accessed: 23-Jun-2020].

[18] S. Robotics, "New Sensors," 2017. [Online]. Available: https://secondrobotics.com/2014/06/25/new-sensors/. [Accessed: 23-Jun-2020].

[19] Open Source Robotics Foundation, "Documentation - ROS Wiki," 2020. [Online]. Available: http://wiki.ros.org/. [Accessed: 23-Jun-2020].

[20] J. Rapado García, "Interfaz Gráfica de Usuario para Mapeado de Entornos y Navegación en ROS," p. 70, 2016.

[21] G. G. C. S. W. Burgard;, "22-OPEN SLAM." 2018.

[22] Robotics Business review, "SLAM Open-Source Option in Version 2.0 - Robotics Business Review," 2020. [Online]. Available: https://www.roboticsbusinessreview.com/rbr/karto-slam-solution-from-sri-adds-features-and-open-source-option-in-versio/. [Accessed: 23-Jun-2020].

[23] Karto Robotics, "Publicaciones Facebook," 2020. [Online]. Available: https://www.facebook.com/kartorobotics/photos/a.401983719034/401983724034/?type=3&theater. [Accessed: 23-Jun-2020].

[24] Team Hector, "Team Hector - Team Hector," 2020. [Online]. Available: https://www.teamhector.de/. [Accessed: 23-Jun-2020].

[25] Team Hector Darmstadt, "Team Hector Darmstadt Facebook," 2018. [Online]. Available: https://www.facebook.com/TeamHectorDarmstadt/. [Accessed: 23-Jun-2020].

[26] C. Flores-Vázquez, C. A. Bahon, D. Icaza, and J. C. Cobos-Torres, “Developing a Socially-Aware Robot Assistant for Delivery Tasks,” *Commun. Comput. Inf. Sci.*, vol. 1194 CCIS, pp. 531–545, 2020.

[27] Silliman Marws, “Testing the TurtleBot Simulation,” 01-Feb-2020. [Online]. Available: https://learn.turtlebot.com/2015/02/03/3/. [Accessed: 18-Sep-2020].

GLOSARIO

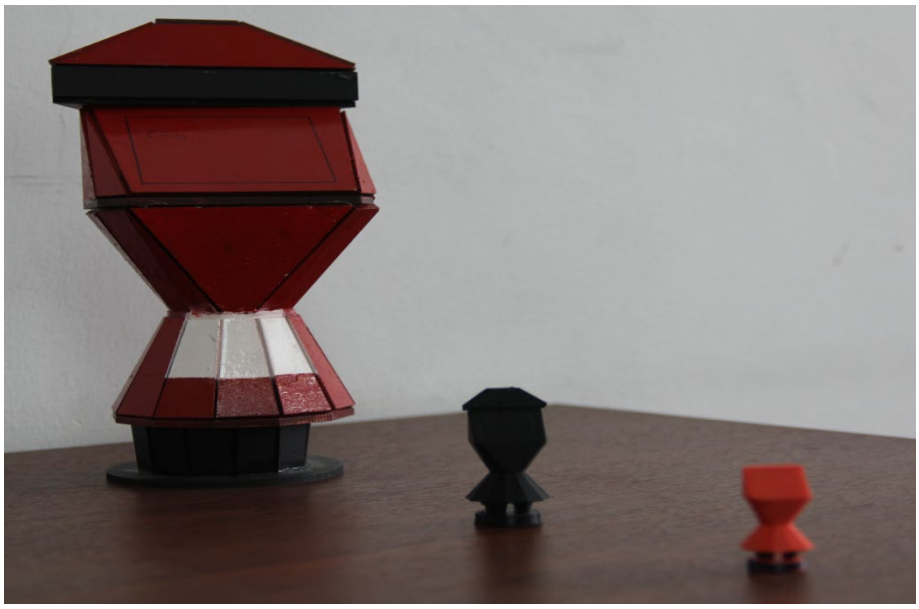
- **SLAM:** (*Simultaneous Localization and Mapping*), Mapeo y Localización Simultánea.
- **ROS:** (*Robot Operative System*), Sistema Operativo Robotico.
- **GNU/LINUX:** Es un Sistema Operativo de software libre.
- **RPLIDAR:** Es un sensor cuyo sistema permite una solución de scanner láser 2D de 360 grados dentro de un rango de 25 metros.
- **UBUNTU:** Es una distribución de Linux.
- **FKE:** Filtro de Kialman Extendido.
- **PDF:** Función de Densidad de Probabilidad.
- **PF:** Filtro de Particulas.
- **RVIZ:** Es una herramienta de visualización en 3D para ROS, que permite visualizar el entorno y ver lo que el robot está viendo, pensando y haciendo.
- **Github:** Es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git.
- **Kinetic:** Es una versión de ROS, para trabajar con Ubuntu.
- **Gazebo:** Es un simulador de entornos 3D que posibilita evaluar el comportamiento de un robot en un mundo virtual.
- **Robot TurtleBot:** Es un kit de robot personal de bajo costo con software de código abierto, es muy utilizado tanto para la educación como para la investigación.
- **Robot Autónomo (RA):** Son sistemas completos que operan eficientemente en entornos complejos sin necesidad de estar constantemente guiados y controlados por operadores humanos.
- **Algoritmo:** Es un conjunto de reglas establecidas en la programación de un sistema de gestión orientadas a la consecución de los objetivos previamente definidos.

- **Kinect:** es una cámara RGB que logra la captura de 30 cuadros por segundo con resolución de 640x480 píxeles de color codificados con 8-bits, el sensor captura colores utilizando un filtro de Bayer.
- **Teleoperador:** Es una herramienta que permite controlar los movimientos del robot, desde un computador.
- **Python:** Es un lenguaje de programación, que puede crear y modificar las instrucciones de un robot.
- **Hector SLAM:** Contiene paquetes de ROS relacionados con la ejecución de SLAM.
- **Ssh:** Es el Protocolo seguro para comunicación entre PC y servidor remoto.

ANEXOS

Los anexos que contiene el trabajo de investigación, son de autoría propia como figuras, tablas y documentación que por su extensión no se ha podido incluir en los capítulos anteriores.

Anexo A: Diseños del prototipo del robot TurtleBot 2 de la carrera de Ingeniería Eléctrica de la Ucacue

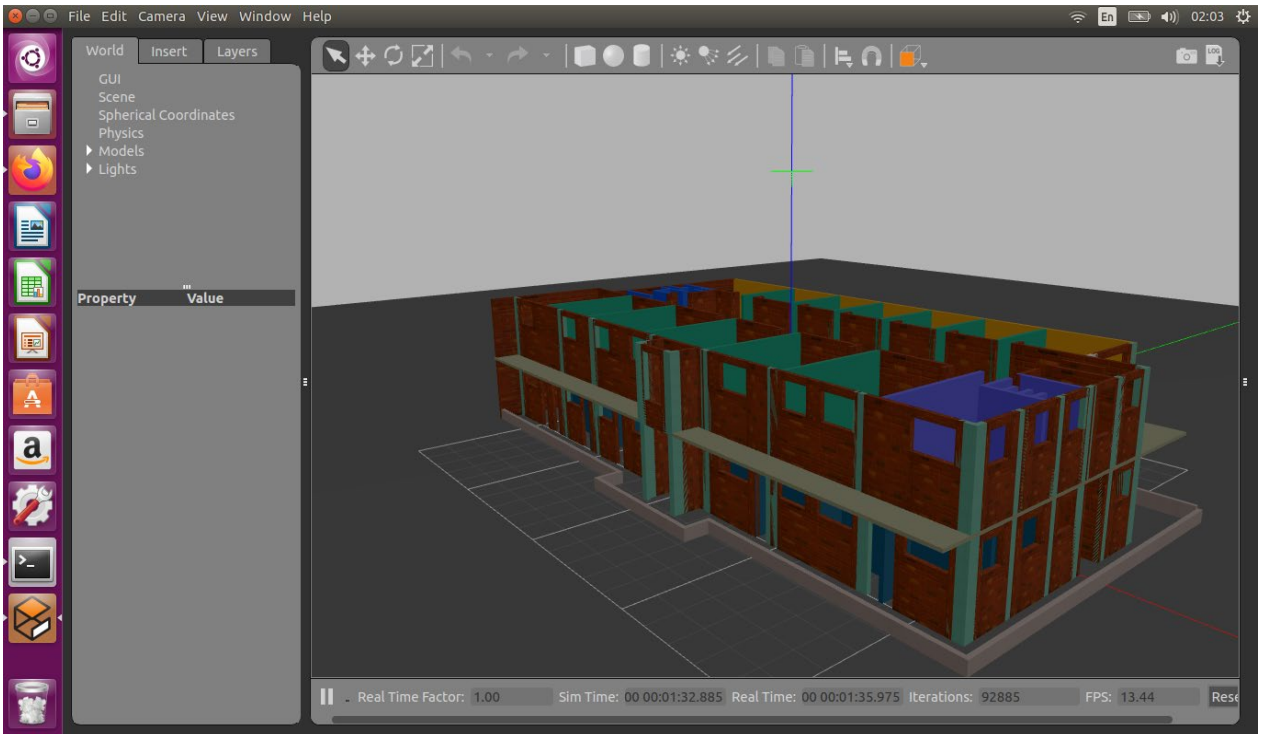


Anexo B: Espacio de pruebas del robot TurtleBot 2

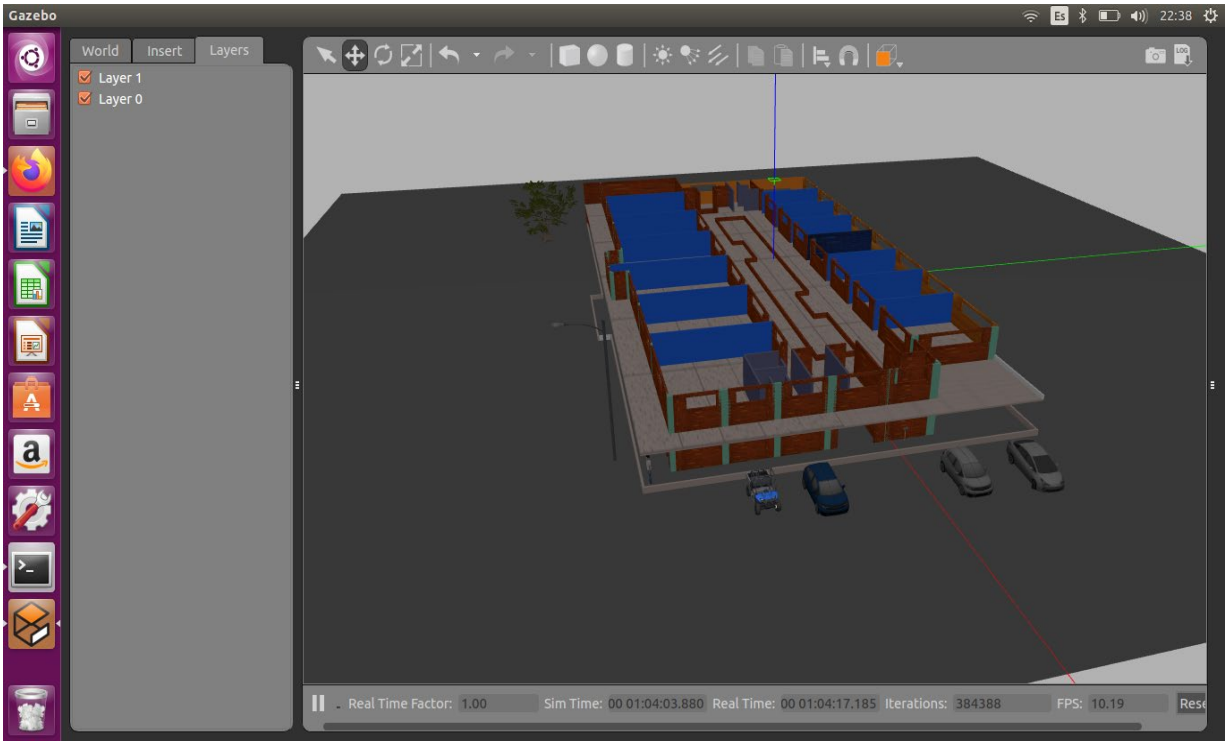
Anexo C: Equipos y materiales utilizados durante las pruebas del robot

Anexo D: Mapa del entorno generado por el robot

Anexo E: Entorno de la Ucacue creado por el simulador Gazebo, para pruebas



Anexo F: Entorno de pasillos internos de la Ucacue creado por el simulador Gazebo, para pruebas del robot



Anexo G: Pruebas del robot TurtleBot 2

AUTORIZACION DE PUBLICACION EN EL REPOSITORIO INSTITUCIONAL

Yo, **Carlos Santiago Guaman Quizhpi** portador de la cédula de ciudadanía Nº 0301901294. En calidad de autor/a y titular de los derechos patrimoniales del trabajo de titulación **"Pruebas de Generación de Mapas y Localización Simultánea (SLAM) en un Robot Móvil para Interiores"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos, Así mismo; autorizo a la Universidad para que realice la publicación de éste trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, 03 de octubre de 2020

F: .....

Carlos Santiago Guman Quizhpi

0301901294