Control de seguimiento de dos trayectorias para un robot diferencial usando el Sistema Operativo de Robótica ROS

Alexander Cedeño Rincon
Jeison Ferney Quevedo Arteaga

Universidad Antonio Nariño
Facultad de Ingeniería Mecánica, Electrónica y Biomédica
Neiva, Colombia
2020

Control de seguimiento de dos trayectorias para un robot diferencial usando el Sistema Operativo de Robótica ROS

Alexander Cedeño Rincón
Jeison Ferney Quevedo Arteaga

Proyecto de grado presentado como requisito parcial para optar al título de:

Ingeniero Electrónico

Director (a):

Ingeniero Julián David Pareja Garzón

Codirector (a):

Ingeniero Julián René Chaúx Cedeño

Línea de Investigación:

Mecatrónica y Robótica

Grupo de Investigación:

Percepción y robótica (GEPRO)

Universidad Antonio Nariño

Facultad de Ingeniería Mecánica, Electrónica y Biomédica

Neiva, Colombia

2020

A mi madre Irma Inés Rincón, mi padre Fernando Cedeño y mis hermanas Leidy Lorena Cedeño y Luisa Fernanda Cedeño, porque sin ellos nada de esto estaría hecho. Por último, a Steven Solano por devolverme la fe en mí mismo y enseñarme que a pesar de las adversidades, con esfuerzo y dedicación todo se puede lograr. Muchas gracias a ustedes.

Alexander Cedeño Rincón

A mi madre Ruby Arteaga, mi hermano Cristhian Murcia Arteaga y mi abuela Emma Vargas, por su incondicional apoyo y es por ustedes que logre esto.

Jeison Ferney Quevedo Arteaga

Agradecimientos

En primer lugar, gracias a Dios por darnos salud y gracia. Agradecer a nuestros padres por su arduo trabajo y voluntad de apoyo en todo el proceso de formación que comenzó hace mucho tiempo.

En segundo lugar, darle las gracias a nuestro codirector del proyecto el Ingeniero Julián René Chaúx Cedeño por la dedicación y su valioso apoyo brindado a este trabajo, por escuchar nuestras sugerencias y confiar en el talento de nosotros.

Por último, pero no menos importante, agradecer a nuestro director de trabajo de grado, el ingeniero Julián David Pareja por su invaluable apoyo profesional y el tiempo dedicado a ser guía y consejero para lograr culminar nuestro proyecto de grado.

Resumen

El sistema operativo ROS se ha consolidado a nivel internacional como una de las principales herramientas para la programación de robots, tanto en la academia como en la industria de la robótica. Actualmente, en el programa de ingeniería electrónica de la Universidad Antonio Nariño sede Neiva se emplea Matlab y diferentes herramientas hardware y software de código abierto para apoyar el proceso de aprendizaje de los estudiantes en el curso de robótica. Sin embargo, no se cuenta con un entorno virtual por completo en el que se pueda incluir un robot 3D empleando el Sistema Operativo de Robótica ROS, por lo que este proyecto propone desarrollar el control de seguimiento de dos trayectorias en un robot diferencial empleando ROS, no sin antes diseñar el robot mencionado en el software tipo CAD/CAM SolidWorks y el entorno 3D en el simulador Gazebo.

Palabras clave: Sistema Operativo de Robótica ROS, Control de seguimiento de trayectorias, robot móvil diferencial, Entorno virtual, Gazebo.

Abstract

The ROS operating system has become internationally established as one of the main tools for robot programming, both in academia and in the robotics industry. Currently, in the program of electronic engineering of the University Antonio Nariño Neiva headquarters uses Matlab and different hardware tools and open source software to support the learning process of students in the course of robotics. However, there is not a complete virtual environment in which a 3D robot can be included using the ROS Robotics Operating System, so this project proposes to develop the tracking control of two trajectories in a differential robot using ROS, not before designing the robot mentioned in CAD/CAM SolidWorks software and the 3D environment in the Gazebo simulator.

Keywords: Robotics Operating System, Path Tracking Control, Differential Mobile Robot, Virtual Environment, Gazebo.

Contenido

Lista de figuras

Pág.

Lista de tablas

Pág.

.

Lista de Símbolos y abreviaturas

Abreviaturas

Abreviatura	Término
ROS	Robot Operating System
URDF	Unified Robotic Description Format
COLLADA	COLLAborative Desing Activity
XML	Extensible Markup Language
SLAM	Simulatenous Localization And Mapeo
AMCL	Adaptive Monte Carlo Localization
SDF	Simulation Description Format
STL	Standard Triangle Language

Introducción

Para comenzar, según (Vargas et al., 2019) la robótica es una herramienta que se puede utilizar para mejorar la capacidad de programación y enriquecer a los estudiantes con habilidades para resolver problemas mediante la construcción de robots.

Por esta razón, en este trabajo de grado se pretende registrar el diseño de un robot de configuración tipo diferencial elaborado en el software tipo CAD/CAM SolidWorks, con el cual se puedan realizar dos trayectorias usando el Sistema Operativo para Robótica, ROS y su simulador 3D Gazebo. Este simulador fue integrado en ROS por un ingeniero de investigación, llamado Jhon Hsu en el año 2009, y desde entonces se ha convertido en una de las principales herramientas utilizadas en la comunidad ROS. En este caso, para la elaboración de este trabajo de grado se usa ROS Kinetic y por ende la versión Gazebo 7.x; esto con el fin de permitir que el robot sea capaz de ejecutar la trayectoria indicada e interactúe con el entorno elaborado en el simulador mencionado anteriormente.

Cabe mencionar que este trabajo de grado se enmarca por una parte en una investigación exploratoria asociada a un primer acercamiento en ROS por parte de la Universidad Antonio Nariño, lo que permite que en investigaciones posteriores se tenga más documentación en relación a la implementación del robot. Por otra parte, se enmarca en una investigación aplicada en donde se controla el seguimiento de las dos trayectorias propuestas en el objetivo general, teniendo en cuenta las condiciones del robot simulado en el entorno hecho en Gazebo.

En definitiva, se puede decir que en esta tesis se propone desarrollar el control de seguimiento de dos trayectorias empleando el sistema ROS en un robot diferencial

simulado en el entorno 3D Gazebo, lo que permite apoyar el proceso formativo de los estudiantes de ingeniería electrónica, especialmente en el área de robótica, e incentivarlos a seguir trabajando con esta plataforma y que posteriormente se implemente el robot de manera física.

1. Objetivos

1.1 Objetivo General

Controlar el seguimiento de dos trayectorias de un robot diferencial basado en el Sistema Operativo de Robótica.

1.2 Objetivos Específicos

- Diseñar un robot diferencial considerando sus condiciones cinemáticas.
- Desarrollar un algoritmo empleando el sistema ROS para el control de seguimiento de dos trayectorias en el robot diferencial.
- Verificar el correcto seguimiento de cada una de las trayectorias del robot en el entorno 3D gazebo

2. Marco Referencial

2.1 Estado del arte

En los últimos años, la plataforma ROS ha ganado ventajas internacionales y ha ido consolidando un entorno propio para impulsar la programación de diferentes tipos de robots, por lo que incluso empresas especializadas en robótica están orientando sus procesos de programación de robots en ROS (Cacace & Joseph, 2018).

Empresas como BMW están implementando ROS actualmente, pues la introducción a ROS puede considerarse como una ventaja competitiva en campos relacionados (Pyo et al., 2017). Según (MetroRobots.com, 2020) en un mapa interactivo se presentan las siguientes cifras de uso de la plataforma ROS a nivel mundial, se observa que se emplea en 130 instituciones educativas, 26 en institutos o centros de investigación, y en 90 empresas, siendo las regiones con mayor concentración Europa y Estados Unidos, mientras que Latinoamérica y África se encuentran aún rezagadas, en Colombia en particular la Universidad del valle reporta un caso de uso académico, y una aplicación industrial en una empresa de automatización y control con sede principal en la ciudad de Bogotá.

En el ámbito de la investigación y la educación, entre los diferentes proyectos desarrollados con ROS, destaca el proyecto propuesto por (Plaza Cano, 2019) en el que se implementa el control de trayectoria en el robot Lego para interceptar objetos en movimiento que siguen curvas de Bézier. Por otro lado, en (Karhumaa et al., 2015), ROS se utiliza para realizar el control de navegación en el robot móvil Pioneer 3-AT, según los autores para el desarrollo del proyecto fue necesario actualizar los paquetes de software disponibles para hacer que ROS fuera compatible con los robots Pioneer de Michigan Tech. Otro de los proyectos destacados en la implementación de ROS es el de (Besseghieur et al., 2018) donde el propósito fue desarrollar un marco basado en ROS que permita a los investigadores

implementar sus algoritmos de control de seguimiento de trayectorias en robots móviles que admiten ROS. También se propone la integración de un robot comercial con un controlador desarrollado por ROS, que tiene como objetivo reducir el tiempo de implementación del proyecto (Araújo et al., 2014). Otra de las aplicaciones interesantes de ROS ha sido en el campo de la agricultura, donde se ha empleado para el desarrollo de robots agrícolas y forestales (Barth et al., 2014).

Adicional a los proyectos que ya se han mencionado, actualmente hay varios robots que usan ROS. En el sitio oficial de ROS (robots.ros.org, 2020) se muestran algunos de los robots que lo usan, como por ejemplo el robot aéreo Gapter o el Trébol COEX, el robot terrestre Turtleblot o Pepper; también robots manipuladores (como Fanuc o ABB) y robots marinos (como Clearphat Heron USV). Además, se pueden apreciar sus principales características y se pueden encontrar fácilmente por categoría o nombre del robot.

Finalmente, en la Universidad Antonio Nariño aún no se ha desarrollado ningún proyecto relacionado con ROS, lo que indica que el proyecto de grado será el primero en implementarse.

2.2 Justificación

Recientemente, especialmente en los campos de la educación y la investigación, diferentes entornos de programación han llamado la atención para impulsar el desarrollo de proyectos relacionados con la robótica (Gawryszewski et al., 2017). Sin embargo, los programas implementados en estos entornos no siempre se pueden migrar fácilmente de un programa a otro, e incluso para programas muy complejos es posible que sea necesario volver a desarrollar todo el proyecto (Araújo et al., 2014).

Por ello, en 2007 un grupo de investigadores de la Universidad de Stanford concibió el proyecto ROS (del inglés, Robot Operating System) como una plataforma para el desarrollo de aplicaciones en robótica que permitiera desarrollar programas versátiles capaces de funcionar con diferentes robots sin necesidad de implementar cambios sustanciales en el código. Como plataforma de código abierto, ROS tiene una extensa biblioteca de paquetes de software profesionales que pueden promover la reutilización de código y tiene una

comunidad de desarrollo y soporte internacional compuesta por la industria y el mundo académico (Quigley et al., 2015). Otras de sus características principales son: es modular, tiene una comunidad de desarrolladores activa, permite la interoperabilidad entre plataformas, facilita la implementación de computación distribuida Y puede utilizar el simulador 3D proporcionado (Gazebo) para la simulación (Pyo et al., 2017). Gazebo es un simulador dinámico 3D que considera la interacción entre el robot simulado y el entorno complejo, lo que permite visualizar y analizar el comportamiento dinámico del robot (Rivera et al., 2019)(Gazebo, 2014).

Por lo anterior, se puede afirmar que la versatilidad de la plataforma ROS, así como su amplia difusión y adopción en ambientes académicos e industriales la hace una herramienta de interés para estudiantes y profesionales de la ingeniería relacionados con la robótica, porque permite utilizar y simular diferentes tipos de robots en el desarrollo de aplicaciones en un entorno de trabajo colaborativo e interoperable a escala internacional.

2.3 Marco teórico

2.3.1 Definición de robots móviles

Un robot móvil es un elemento autónomo que no se mueve sobre rieles, sino que usa dispositivos de locomoción como ruedas, patas, cadenas, entre otros. Por lo general, funcionan con baterías porque no pueden conectarse mediante ningún cable. Como elementos autónomos necesitan sensores como el GPS para guiarse a sí mismos. También puede tener rutas predefinidas, esto quiere decir que son completamente autónomos o tienen una determinada ruta de desarrollo, y necesitan sensores para evitar colisiones y caídas. Según (Gil, 2017) este sistema de sensorización está compuesto por sensores extereoceptivos (externos) y propioceptivos (Internos), los cuales aportan al robot información sobre su entorno y su estado respectivamente.

2.3.1.1 Configuración de robots móviles

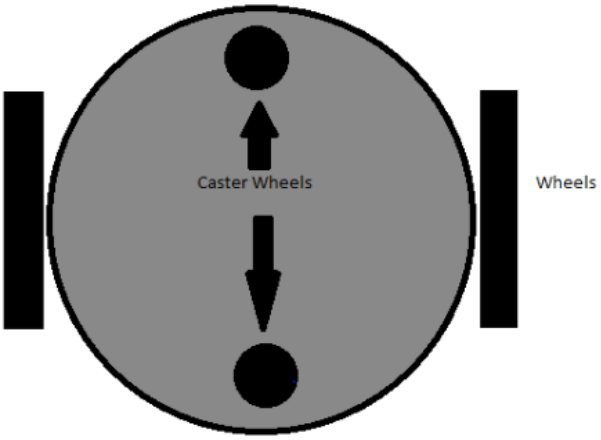
Los robots móviles se clasifican según su configuración cinemática, por lo general divididos en Diferencial, Skid Steer, Síncrona, Omnidireccional, Triciclo y Ackerman. Suelen

depender del planteamiento del proyecto a realizar ya sea de aplicación, operabilidad, estabilidad o controlabilidad.

2.3.1.1.1 Configuración Diferencial

Un robot móvil con configuración diferencial puede moverse en línea recta, seguir una curva y girar sobre sí mismo; esto se logra mediante las dos ruedas laterales del robot montadas en un eje común. La dirección está determinada por la diferencia de velocidades entre las dos ruedas. Para mayor estabilidad, se pueden usar una o más ruedas en el robot. En la Figura 2-1 se muestra un sistema de transmisión diferencial típico:

Figura 2-1: Modelo de configuración diferencial.



Fuente: Adaptado de Learning Robotics Using Python (Joseph, 2015).

2.3.1.1.2 Configuración tipo triciclo

Como sugiere el nombre, el robot móvil tipo triciclo opera de la misma forma que un triciclo, es decir, consta de tres ruedas, una rueda delantera y dos ruedas traseras. Estas últimas, funcionan sin ningún tipo de tracción, mientras que la rueda delantera se usa tanto para el direccionamiento como para la tracción.

2.3.1.1.3 Configuración Ackerman

Esta configuración es una de las configuraciones más utilizadas para vehículos convencionales de cuatro ruedas. La tracción de esta configuración está en las ruedas traseras, mientras que la dirección del robot está en las dos ruedas delanteras. Su maniobrabilidad es limitada porque para el vehículo, su propia orientación requiere de un radio de giro mínimo y un posicionamiento preciso. Además, requiere varios movimientos

hacia adelante y hacia atrás para lograr ubicarse. La estabilidad es mejor, porque sus dos ruedas delanteras le permiten girar sin volcar.

2.3.1.1.4 Configuración Skid Steer

Esta configuración es similar a la configuración diferencial, excepto que sus ruedas laterales ya no son solo dos. El movimiento del robot se logra combinando la velocidad de cada rueda del lado izquierdo y derecho. Es importante decir que cada rueda es guiada por un motor de tracción y las ruedas del mismo lado giran a velocidades iguales.

2.3.1.1.5 Configuración Síncrona

Los robots móviles con este tipo de configuración, como su nombre lo indica, la dirección y rotación de las ruedas deben estar sincronizadas, es decir, todos los neumáticos o llantas deben tener la misma dirección y la misma rotación. En esta configuración, solo hay tres ruedas posicionadas en forma de triángulo debajo de una base circular. Estas ruedas están unidas y controladas por uno o dos motores independientes.

2.3.1.1.6 Configuración Omnidireccional

Esta configuración permite una mayor movilidad en el entorno gracias a que cuenta con tres grados de libertad, esto significa que puede moverse en cualquier dirección sin la necesidad de reorientarse, lo que les da una ventaja frente a las otras configuraciones, pues la movilidad de otros tipos de configuraciones es limitada.

2.3.2 Sistema Operativo de Robótica ROS

Como primer acercamiento a ROS es importante saber lo básico que lo conforma. Robot Operating System (ROS) es un marco flexible para escribir software de robot. Es una colección de herramientas (como el simulador 3D Gazebo) y bibliotecas diseñadas para simplificar la tarea de crear comportamientos robóticos complejos y robustos en varias plataformas robóticas. Según (ROS.org, 2010), ROS se creó desde cero para fomentar el desarrollo de software de robótica colaborativa, lo que significa que ROS esta especialmente diseñado para que diferentes grupos de desarrolladores puedan colaborar y puedan basarse en los trabajos de otras personas.

Para entender un poco mejor, ROS tiene un sistema de archivo que cubre principalmente los recursos ROS en el disco, como paquetes, datos de paquetes, tipos de mensajes y

tipos de servicios. ROS utiliza paquetes para organizar sus programas. En otras palabras, se puede pensar en un paquete como todos los archivos contenidos en un programa ROS en particular. Todos sus archivos cpp, archivos python, archivos de configuración, archivos de compilación, archivos de lanzamiento y archivos de parámetros. Todos esos archivos en el paquete están organizados con una carpeta de lanzamiento o carpeta Launch; esta carpeta contiene archivos de lanzamiento. También una carpeta src la cual contiene archivos fuente (cpp, python). Por último, una lista CMakeLists.txt para compilación y un archivo .xml que da la información del paquete y dependencias del mismo.

Por otro lado, ROS se basa en una arquitectura gráfica que puede procesar datos como un todo o en conjunto. Se basa en una arquitectura cliente-servidor; el cliente envía una solicitud a otro programa (servidor) y este responde. Los principales elementos de esta arquitectura son los nodos, los mensajes y los topics o temas.

Los nodos son básicamente programas escritos en ROS, usan dependencias de C ++ (roscpp) o Python (rospy) para realizar tareas. Debe quedar claro que ROS está diseñado para ser modular, por lo que varios nodos pueden realizar diferentes tareas. Por ejemplo, en un sistema de control de robot, generalmente se incluyen nodos personales para cada función, para controlar los láseres un nodo diferente al nodo que controla los motores de las llantas, de igual forma para posicionamiento del robot y la planificación de rutas se usa un nodo diferente. Estos nodos se comunican entre sí mediante la publicación de mensajes sobre temas. Un mensaje es una estructura de datos simple, que comprende campos escritos y pueden ser de tipo entero, punto flotante, booleano, etc. Por otro lado, un tema es como una tubería. Los nodos utilizan temas para publicar información en otros nodos para que puedan comunicarse. Según la página oficial de ROS-Nodos (ROS.org, 2020) el uso de nodos en ROS proporciona muchos beneficios para todo el sistema como por ejemplo la tolerancia adicional a fallas debido al aislamiento de las fallas en nodos individuales.

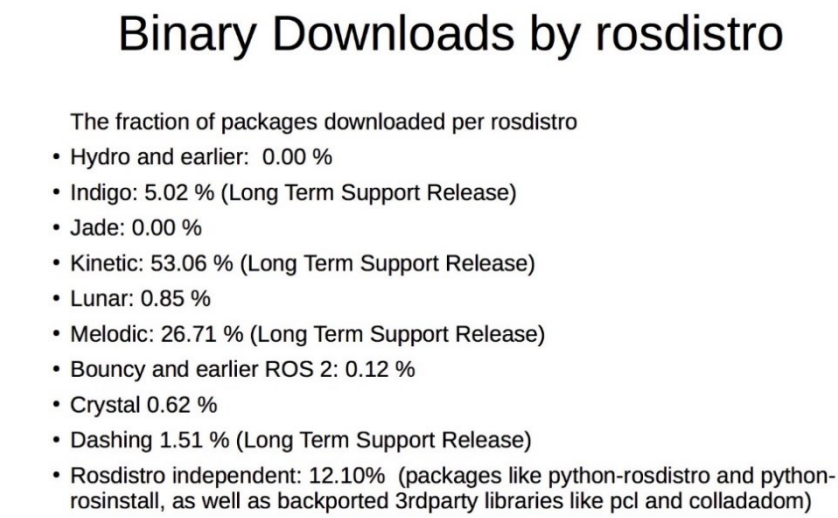
2.3.2.1 Distribución ROS KINETIC

Una distribución ROS es una colección de paquetes de software ROS. En este trabajo de grado se utiliza la distribución ROS Kinetic, específicamente es el décimo lanzamiento de distribución de ROS.

Es importante aclarar la razón por la cual se eligió esta distribución y no otra, pues si se observa en (Foote, 2019), aquí están los datos extraídos del último informe ROS Metrics. Como se puede ver, ROS Kinetic es, con mucho, la versión más descargada (más del 50%). Esto se debe a que, aunque no es la última, es la versión más estable en cuanto a uso y paquetes disponibles.

A continuación, en la Figura 2-2 se ilustran los datos mencionados:

Figura 2-2: Informe ROS Metrics



Fuente: Adaptado de Community Metrics Report Tully ((Foote, 2019)

2.3.2.2 Exportador de SolidWorks a URDF

Para contextualizar, SolidWorks es un software tipo CAD/CAM utilizado para el diseño y elaboración de piezas mecánicas, ensamblajes, diseño de robots, etc. En este caso, se utiliza para el diseño de las piezas del robot (chasis, ruedas, base) y el ensamblaje de estas, para crear el robot móvil diferencial que se quiere diseñar. Posterior a la elaboración del diseño se debe exportar a URDF (Formato de Descripción de Robot Unificado) para representar el modelo del robot. El exportador de SolidWorks a URDF (SW2URDF) es un complemento de SolidWorks que puede exportar fácilmente piezas y ensamblajes de software a archivos URDF. Este, crea un paquete similar a ROS que contiene directorios (archivos URDF) para mallas, texturas y robots. En la página web (wiki.ros.org, 2020)

puede encontrar el paso a paso de como instalar el complemento en tal caso de que no lo tenga.

2.3.2.3 Simulador Gazebo

De manera breve, Gazebo es un entorno de simulación 3D que proporciona la función de simular de manera precisa y eficiente el número de robots en entornos complejos de interior y exterior. El simulador comúnmente es utilizado para probar algoritmos de robótica, diseñar robots y realizar pruebas de regresión; esto gracias a que una de sus características principales es que cuenta con interfaces programáticas y gráficas. Además, cuenta con una rica biblioteca de modelos y entornos de robots, una amplia variedad de sensores y múltiples motores de física como ODEE y BULLET. Estos últimos son útiles para simular vehículos, objetos en entornos de realidad virtual y para usar herramientas de creación 3D.

Con respecto a sus versiones, actualmente la última versión de Gazebo es la 11, según lo escrito en la página oficial de Gazebo, esta versión fue seleccionada como la versión oficial y seguirá siendo válida durante el periodo de vida útil de ROS. Sin embargo, para el desarrollo de este trabajo de grado se usa la distribución ROS KINETIC, es decir que se usa la versión de Gazebo 7.x.

.

3. Marco Metodológico

3.1 Recolección de información

La recopilación de información se llevó a cabo consultando recursos como Google académico, repositorios de universidades y bibliotecas en línea para encontrar los documentos, artículos y libros que aporten al desarrollo de la investigación; además el uso de los recursos que brinda la página oficial de ROS como tutoriales, guías en el manejo de esta plataforma, manejo de herramientas como el simulador Gazebo, etc. Esto permitió aclarar el diseño del robot y el entorno en donde se va a interactuar con el modelo del robot.

3.2 Instalación ROS KINETIC

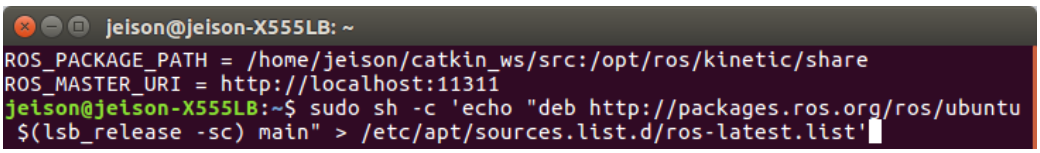
Para este trabajo de grado se usó la distribución Ros Kinetic instalada en un Ubuntu 16.04 LTS. Es importante tener en cuenta que ROS Kinetic solo es compatible con la versión de Ubuntu 15.10 y Ubuntu 16.04.

Para la instalación se debe seguir los siguientes pasos:

- Configurar el sources.list

Lo primero que se debe hacer es configurar su computadora para poder descargar paquetes de tipo packages.ros.org. Esto se realiza ejecutando el comando de la Figura 3-1 en la terminal de Linux.

Figura 3-1: Configuración sources.list

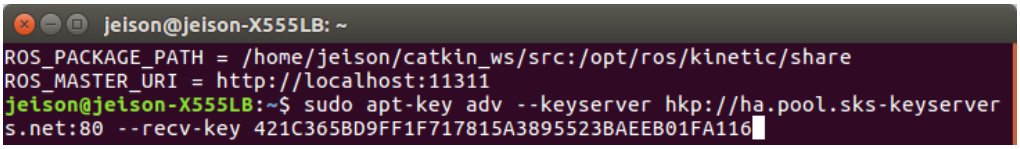


Fuente: Figura hecha por los autores.

- Configurar Keys

Usando el siguiente comando se descarga la clave de keyserver (Figura 3-2).

Figura 3-2: Configuración keys.

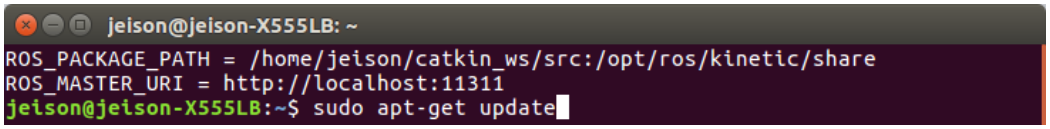


Fuente: Figura hecha por los autores

- Instalación

Los primero que se debe hacer es verificar que el índice del paquete Debian este actualizado (Figura 3-3).

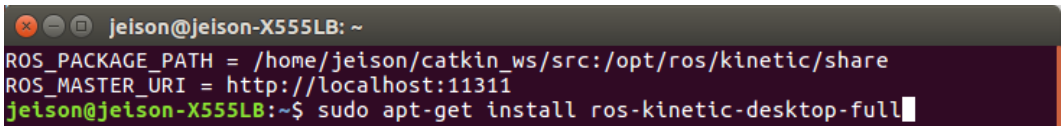
Figura 3-3: Comando para verificar índice paquetes Debian



Fuente: Figura hecha por los autores

Ahora se debe instalar Deskop Full para obtener los paquetes básicos que nos permitirán empezar a trabajar con ROS. Para ello, ejecute el siguiente comando (Figura 3-4).

Figura 3-4: Comando para instalar Deskop Full

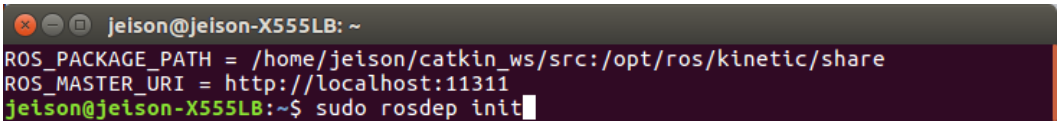


Fuente: Figura hecha por los autores

- Iniciar rosdep

Antes de comenzar a usar ROS se debe inicializar rosdep. Esto permite instalar fácilmente las dependencias del sistema y ejecutar algunos componentes centrales de ROS (Figura 3-5).

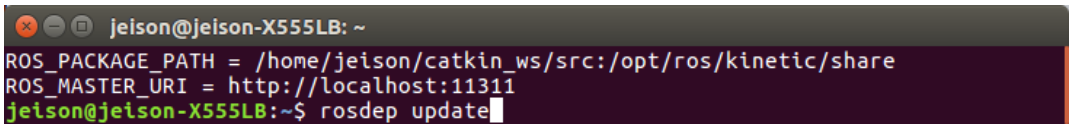
Figura 3-5: Comando para iniciar rosdep



Fuente: Figura hecha por los autores

Para actualizar rosdep (Figura 3-6).

Figura 3-6: Comando para actualizar rosdep.

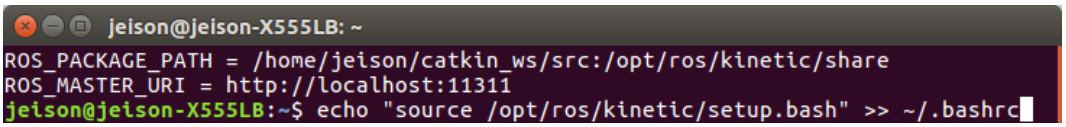


Fuente: Figura hecha por los autores

- Configuración del entorno

Por último, se debe agregar las variables de entorno ROS a la sesión de *.bashrc. Para esto puede ejecutar el siguiente comando en la terminal (Figura 3-7).

Figura 3-7: Comando para agregar las variables de entorno ROS.

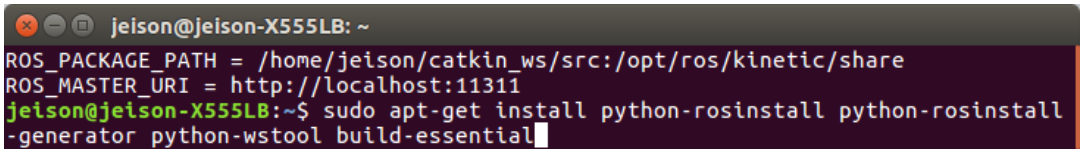


Fuente: Figura hecha por los autores

- Dependencias para construir paquetes

En este punto, aún hay varias herramientas que se necesitan para administrar los espacios de trabajo ROS. Para eso se puede ejecutar el siguiente comando (Figura 3-8).

Figura 3-8: Comando para instalar dependencias

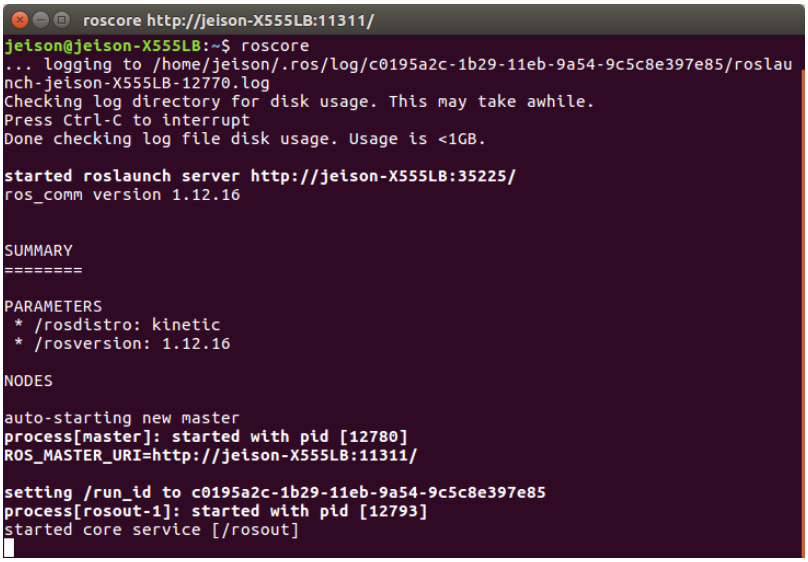


Fuente: Figura hecha por los autores

- Probar la configuración

Para esto, se debe ejecutar el comando roscore en la terminal. Roscore es el proceso principal de gestión de los sistemas ROS. Al ejecutar el comando debe aparecer algo como la Figura 3-9.

Figura 3-9: Comando roscore.



Fuente: Figura hecha por los autores

3.3 Creación del espacio de trabajo (catkin_ws)

3.3.1 Directorio catkin_ws

Lo primero que se debe hacer es crear un directorio o carpeta con el nombre del espacio de trabajo que queremos realizar. Para esto se utilizan los siguientes comandos:

$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin_make

El comando catkin_make es una herramienta para compilar espacios de trabajo catkin. Al ejecutar este comando por primera vez se crea el CmakeLists.txt en la carpeta src y aparecerá en el directorio las carpetas build y devel. Estas carpetas son de compilación y desarrollo del espacio de trabajo.

Antes de continuar con la verificación del espacio de trabajo, ejecute el siguiente comando:

$ source devel/setup.bash

3.3.2 Verificación del espacio de trabajo

Este paso es muy sencillo, para asegurarse de que el espacio de trabajo está correctamente configurado, se debe verificar con el siguiente comando:

echo $ROS_PACKAGE_PATH

3.3.2.1 Creación carpeta tesis_difrobott

Esta carpeta se crea en la carpeta src del espacio de trabajo (catkin_ws), para posterior a eso incluirle los paquetes de la descripción del robot y el entorno a diseñar.

3.4 Diseño del Robot Móvil Diferencial

3.4.1 Diseño en SolidWorks

Lo primero que se realizó fue la selección del Software tipo CAD/CAM llamado SolidWorks para el diseño del robot móvil diferencial. En este software se diseñó cada pieza del robot y al final se hizo el ensamblaje de estas piezas.

La primera pieza en realizar fue el chasis (caparazón) del robot, el cual quedo como se muestra en la Figura 3-10.

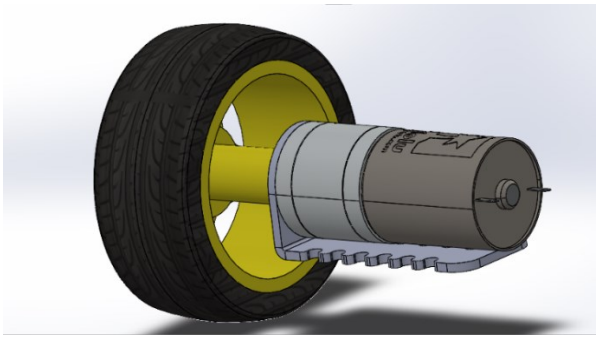
Figura 3-10: Chasis del robot móvil diferencial diseñado.

Fuente: Figura hecha por los autores.

Posterior a esto, se realizó el diseño de las ruedas laterales y las dos Ball Caster. Esto se hizo con distintas herramientas que ofrece el software SolidWorks. Sin embargo, no se

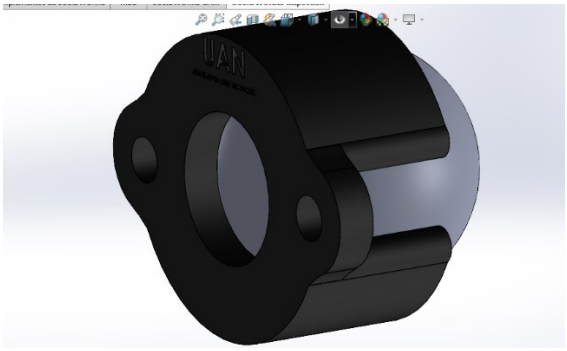
entrará en detalle de cuales son estas, ya que no es el punto central de este trabajo. Después de darle color a las partes y hacerle el grabado de la UAN, el diseño final se puede apreciar en la Figura 3-11 para las ruedas y en la Figura 3-12 para las Ball Caster.

Figura 3-11: Diseño ruedas laterales del robot móvil diferencial



Fuente: Figura hecha por los autores.

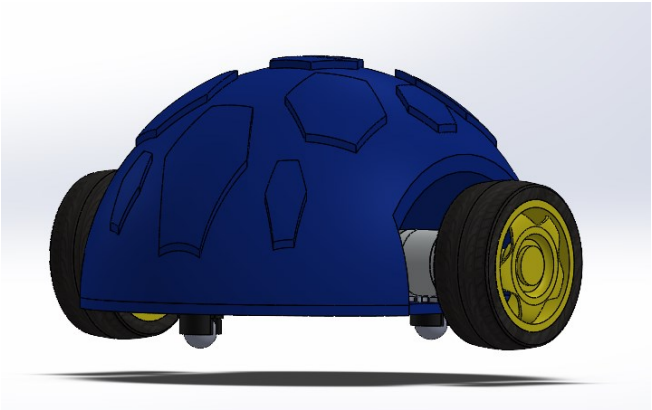
Figura 3-12: Diseño Ball Caster



Fuente: Figura hecha por los autores.

Por último, se realizó el ensamblaje de las piezas. El resultado final fue un robot móvil tipo diferencial, vea la Figura 3-13.

Figura 3-13: Robot móvil diferencial diseñado



Fuente: Figura hecha por los autores.

3.4.2 Exportación del robot en formato URDF

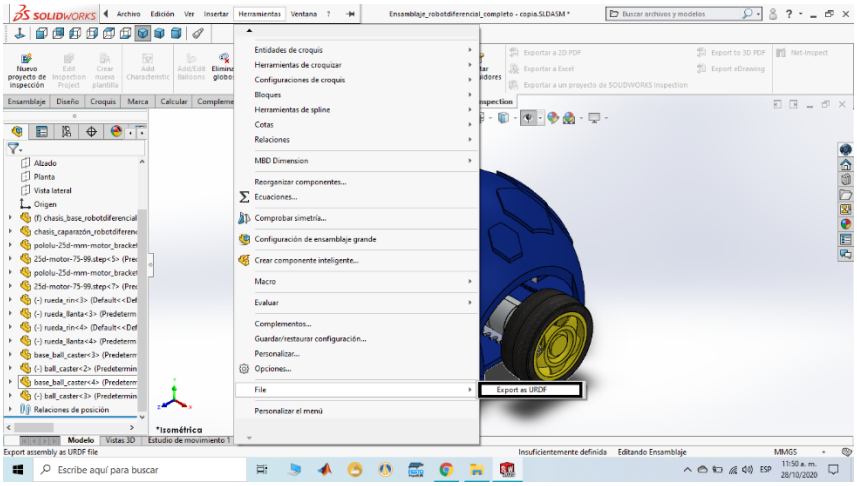
Luego de haber finalizado el diseño del robot en configuración diferencial se exportó desde SolidWorks en formato URDF. Es importante recordar que en el capítulo dos se habló del complemento necesario (SolidWorks Exporter to URDF) para realizar esta exportación.

A continuación, se muestra el paso a paso de la exportación del robot en formato URDF.

- Exportar como URDF

Primero diríjase a la barra de menús en la opción “Herramientas” seleccione “file” y por último “Export as URDF” (Figura 3-14).

Figura 3-14: Exportar como URDF

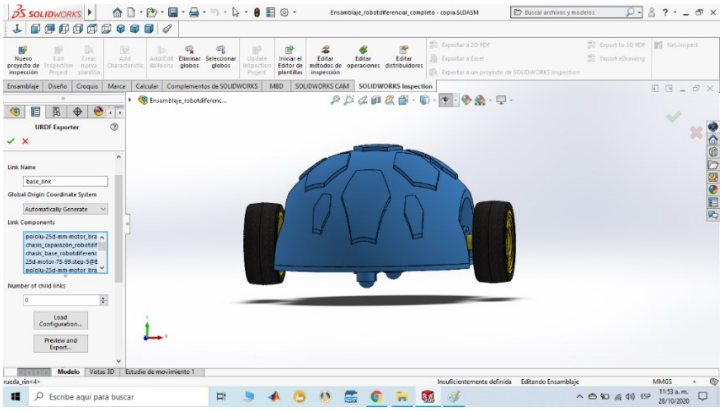


Fuente: Figura hecha por los autores.

- Configuración base_link (Parent)

Las articulaciones se definen en términos de padres e hijo. Esta articulación es el enlace principal, aquí se seleccionan las partes del robot que son parte de este enlace. En este caso se seleccionó toda la parte del chasis, los motores con sus bases y las Ball Caster. Las partes que se seleccionan toman un color azul más claro (Figura 3-15).

Figura 3-15: Configuración base_link

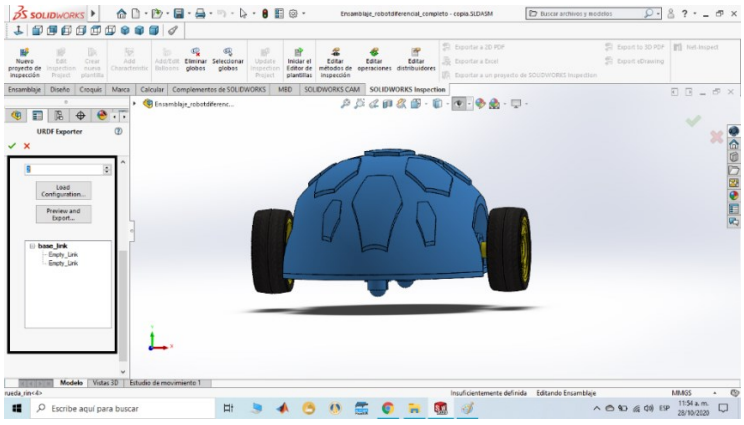


Fuente: Figura hecha por los autores.

- Agregar hijos (Child)

En esta parte se seleccionan las dos ruedas laterales como enlaces secundarios para crear los hijos (Figura 3-16).

Figura 3-16: Agregar hijos

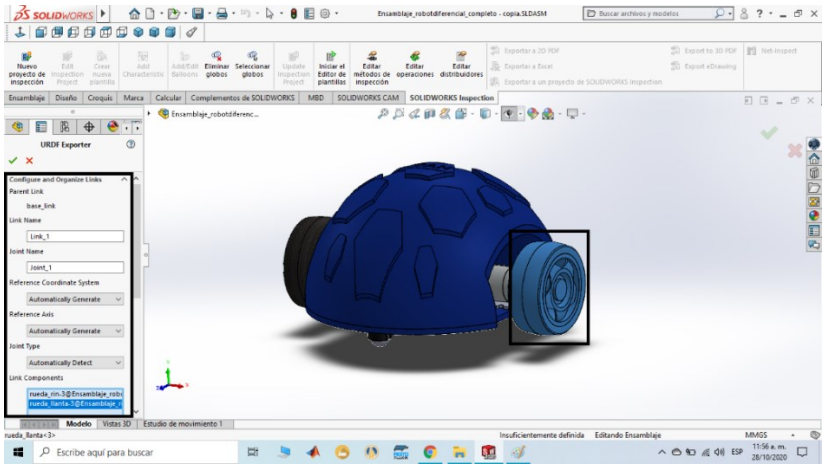


Fuente: Figura hecha por los autores.

- Configuración y organización de los Links

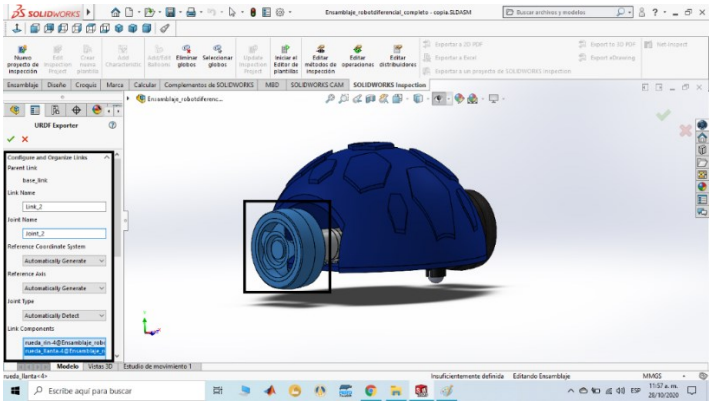
En la Figura 3-17 se muestra el proceso de configuración del hijo 1, esto se realiza de la misma manera para el hijo 2 (Figura 3-18). Aquí se le da un nombre por separado a cada enlace secundario (Link_Name) y a la articulación (Joint_Name) de cada hijo; teniendo en cuenta que el enlace principal tiene por nombre base_link. Es importante no colocar el mismo nombre para los hijos.

Figura 3-17: Configuración hijo 1



Fuente: Figura hecha por los autores.

Figura 3-18: Configuración hijo 2



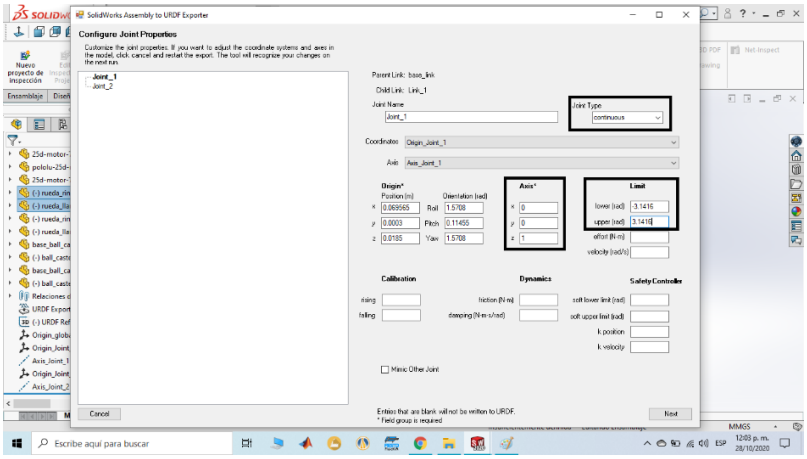
Fuente: Figura hecha por los autores

- Configuración propiedades Joint

Esta parte es muy importante ya que aquí se configuran las propiedades de los Joints creados. Lo primero que se realiza es asignar el tipo de joint de acuerdo al requerimiento que se tenga. Los cuatro tipos de Joints soportados por el exportador SW2URDF son de revolución, prismáticos, continuos y fijos. Para este caso se asignó el tipo continuo porque

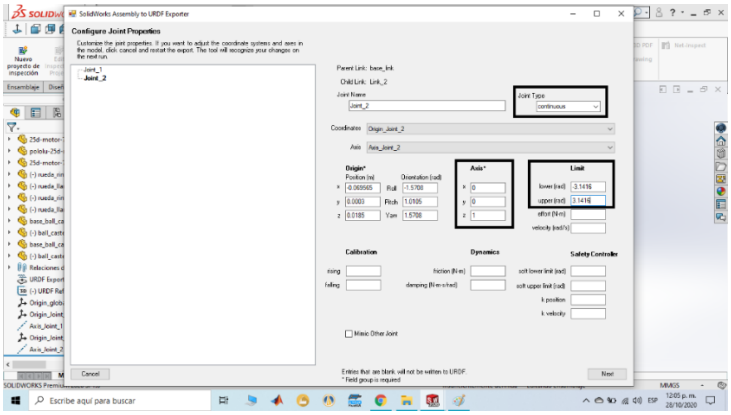
es el que mejor se representa para una rueda, que gira continuamente, como es el caso en nuestro robot. Esta articulación no requiere límites de velocidad y esfuerzo, los límites que se observan (Figura 3-19) son en relación al giro, se colocó un límite de -3,1416 a 3,1416. Por último, en la Figura 3-20 también se muestra la configuración de los ejes de rotación del joint, 2 el cual en este caso es el eje z.

Figura 3-19: Configuración propiedades joint 1



Fuente: Figura hecha por los autores.

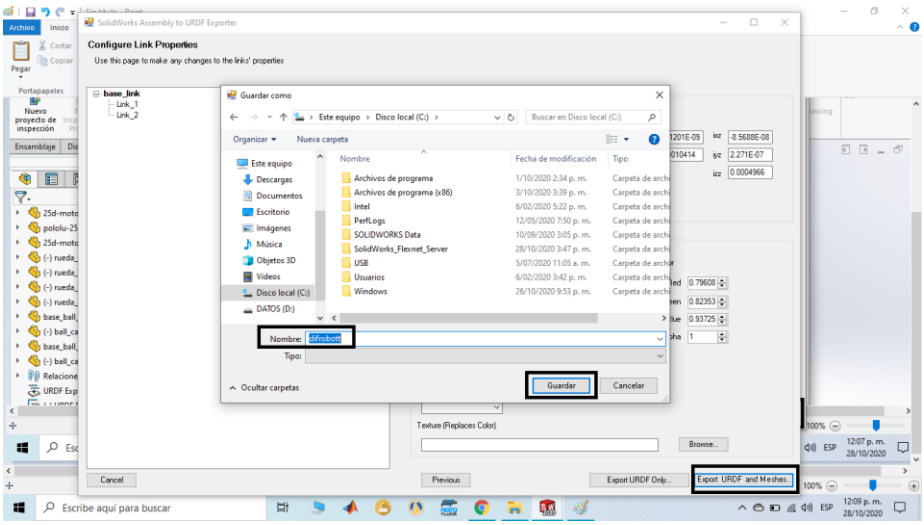
Figura 3-20: Configuración propiedades joint 2



Fuente: Figura hecha por los autores.

Luego de haber configurado los joints procedemos a guardar el archivo (Figura 3-21).

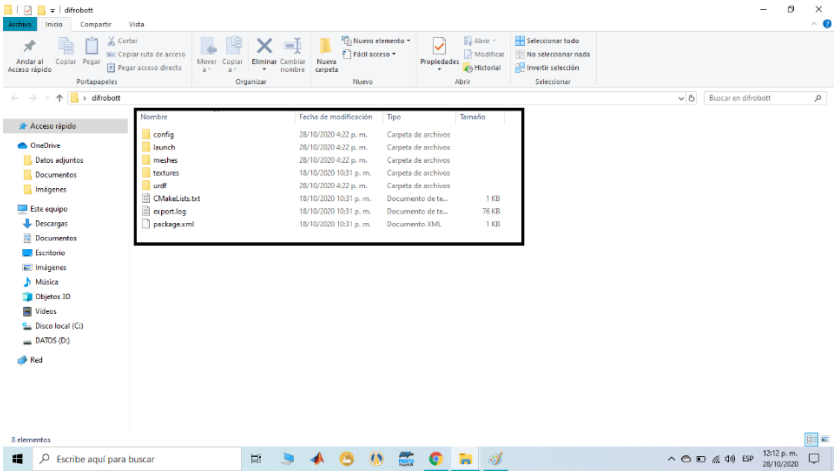
Figura 3-21: Guardar el archivo Exporter



Fuente: Figura hecha por los autores.

Por último, verificamos el contenido de la carpeta exportada, en este caso llamada **difrobott** y como se puede observar ya se creó el paquete (Figura 3-22).

Figura 3-22: Contenido de la carpeta exportada



Fuente: Figura hecha por los autores.

3.4.3 Creación del paquete difrobott_descriptions

Es muy importante crear un paquete en la carpeta src/tesis_difrobott/src del espacio de trabajo (catkin_ws); para crearlo se escribe en la terminal de Linux el comando catkin_create_pkg de la siguiente manera:

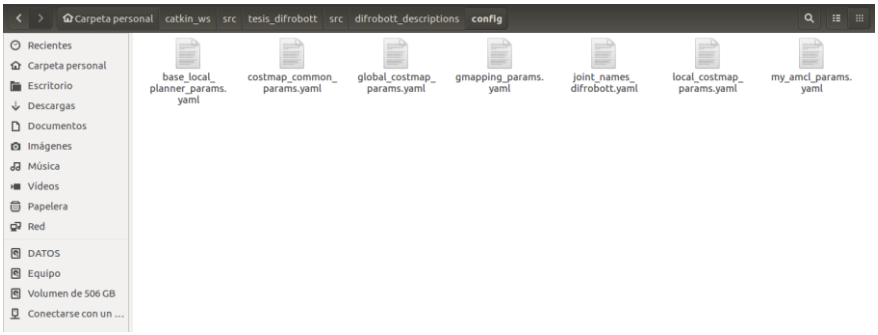
$ catkin_create_pkg difrobott_descriptions std_msgs rospy controller_manager joint_state_controller robot_state_publisher.

Este paquete contiene la descripción del robot en siete carpetas generadas por el exporter de SolidWorks: config, launch, maps, meshes, src, textures y urdf.

3.4.3.1 Contenido carpeta config

Esta carpeta contiene siete archivos *.yaml. Los archivos base_local_planner_params, costmap_common_params, global_costmap_params y local_costmap_params se deben configurar antes de poder ejecutar el nodo move_base. Estos definen los parámetros necesarios para la planificación de ruta del robot. Los tres archivos faltantes contienen la descripción de los joints (joint_names_difrobott), el paquete amcl para localizar el robot en el mapa creado y los parámetros del nodo slam_gmapping (Figura 3-23).

Figura 3-23: Contenido carpeta config

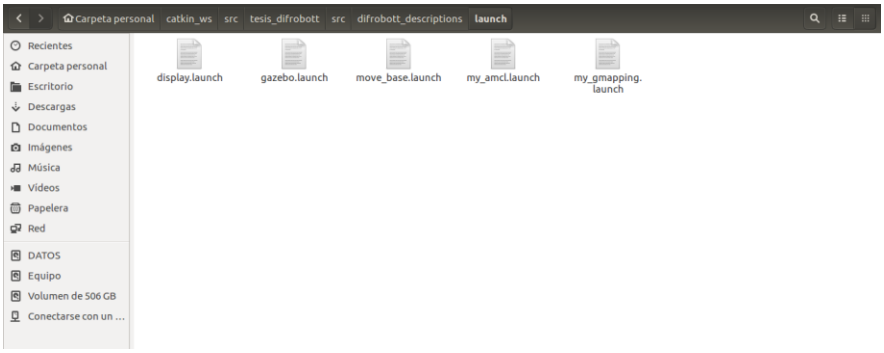


Fuente: Figura hecha por los autores.

3.4.3.2 Contenido carpeta launch

Esta carpeta contiene cinco archivos *.launch: display.launch, gazebo.launch, move_base.launch. my_amcl.launch y my_gmapping.launch. El primer archivo contiene el código para lanzar el modelo del robot en RViz. El segundo archivo es uno de los más importantes ya que nos permite lanzar el robot y el entorno. Por último, los archivos restantes son únicamente para la ejecución de cada configuración de los nodos amcl, move_base y slam_gmapping (Figura 3-24).

Figura 3-24: Contenido carpeta launch paquete difrobott_description

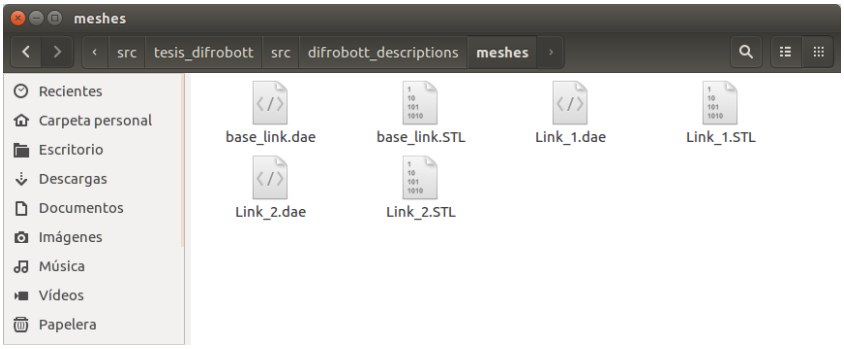


Fuente: Figura hecha por los autores.

3.4.3.3 Contenido carpeta meshes

Aquí están los archivos *.stl generados por el exporter de SolidWorks (Figura 3-25).

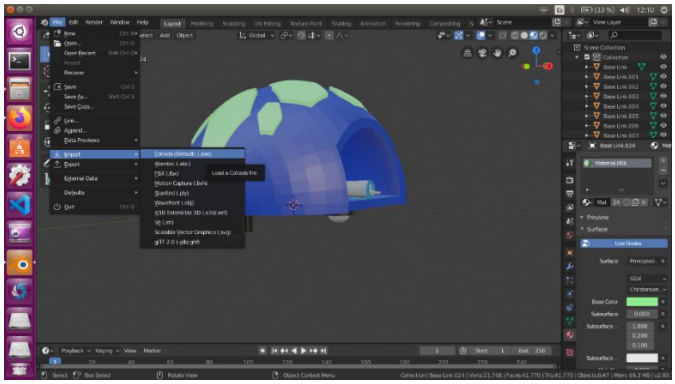
Figura 3-25: Contenido carpeta Meshes



Fuente: Figura hecha por los autores.

Como en estos formatos se excluyen la información de color y texturas de los objetos, se usa el software CAD Blender. Con este software se soluciona el problema de la información de los colores y texturas exportando en formato *.dae los objetos 3D creados, en este caso el robot (Figura 3-26).

Figura 3-26: Exportar en formato *.dae



Fuente: Figura hecha por los autores.

3.4.3.4 Contenido carpeta URDF

Aquí se especifica las propiedades cinemáticas y dinámicas del robot en el archivo URDF, recordemos que este formato de archivo XML nos permite describir todos los elementos de un robot. Además, se incluye el ROS Control plugin, Gazebo Transmission y differential_drive_controller para modelar la cinemática del robot en Gazebo.

3.5 Diseño del entorno en Gazebo

3.5.1 Creación del paquete difrobott_gazebo

Este paquete también debe estar incluido en la carpeta src/tesis_difrobott/src del espacio de trabajo (catkin_ws); para la creación del paquete es con el mismo comando catkin_create_pkg:

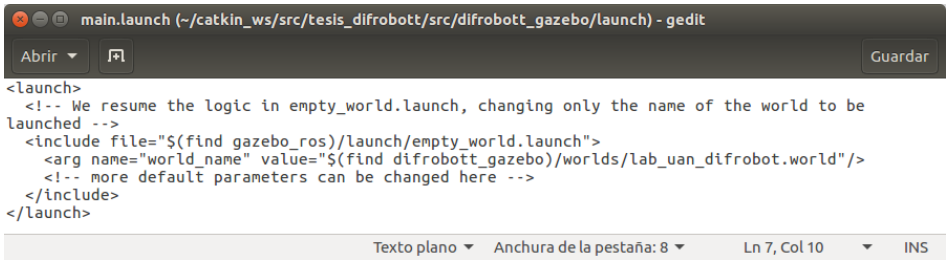
$ catkin_create_pkg difrobott_gazebo std_msgs rospy controller_manager joint_state_controller robot_state_publisher

Este paquete contiene cuatro carpetas: launch, models, src y worlds. Cada una de estas carpetas tiene una misión importante para la construcción del entorno.

3.5.1.1 Contenido carpeta launch

Esta carpeta contiene un código escrito en Python en formato .xml, lo que nos ayuda a lanzar el entorno (Figura 3-27).

Figura 3-27: Contenido carpeta launch

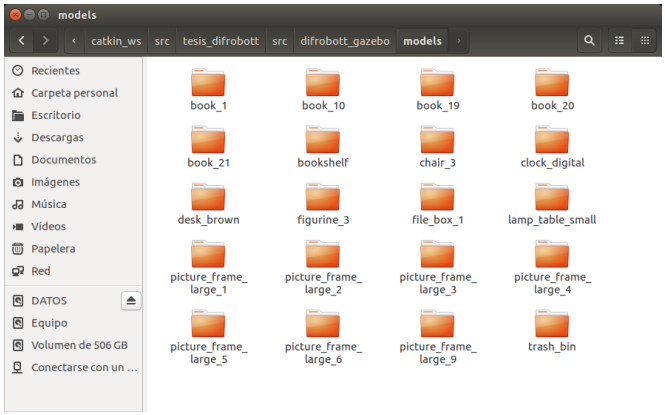


Fuente: Figura hecha por los autores.

3.5.1.2 Contenido carpeta models

Esta carpeta contiene todos los modelos 3D que se agregaron al entorno como cuadros, mesas, libros, sillas, etc. Por cada modelo se crea una carpeta (Figura 3-28). Más adelante se explica cómo incluir los modelos 3D en el entorno.

Figura 3-28: Contenido carpeta models



Fuente: Figura hecha por los autores.

3.5.1.3 Contenido carpeta worlds

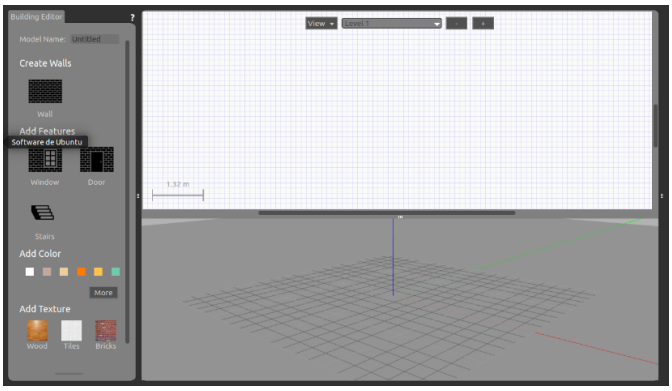
Este nombre se le da porque en ella existe un archivo *.world que describe la colección de objetos (como mesas, luces, sillas y estructuras) y parámetros globales por medio del formato SDF. Este formato es un de tipo XML y es capaz de describir todos los objetos estáticos y dinámicos, la luz ambiental y las propiedades físicas del entorno

3.5.2 Creación archivo *.world

3.5.2.1 Construcción estructural del entorno

Para la construcción estructural del entorno (paredes, ventanas y puertas) se utiliza la herramienta propia de Gazebo Building Editor (Figura 3-29).

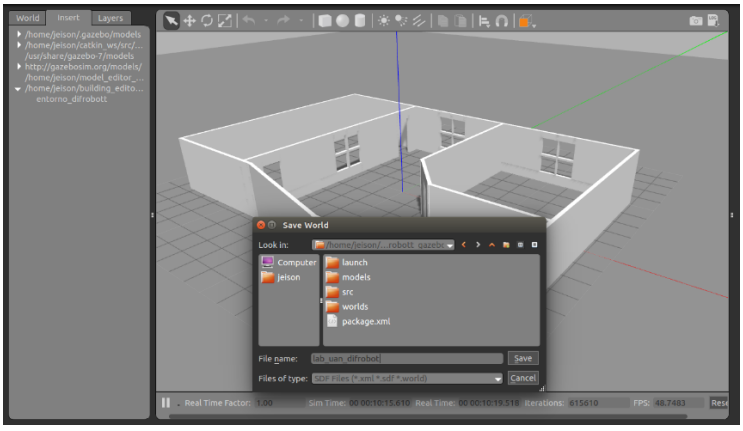
Figura 3-29: Building Editor



Fuente: Figura hecha por los autores.

Después de haber realizado la construcción estructural del entorno se procede a guardar el archivo con extensión *.world (Figura 3-30). En este archivo se describe la colección de objetos (como mesas, luces, sillas y estructuras) y parámetros globales que se van agregando al entorno.

Figura 3-30: Guardar archivo *.world



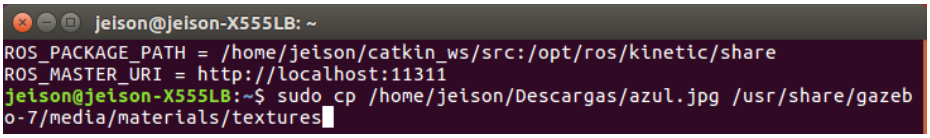
Fuente: Figura hecha por los autores.

3.5.3 Aplicación de texturas al entorno

3.5.3.1 Aplicación textura paredes

Gazebo ofrece un grupo de texturas por defecto, estas texturas se pueden visualizar en la carpeta propia de Gazebo llamada textures. Sin embargo, para el entorno diseñado se agregó una nueva textura (Figura 3-31).

Figura 3-31: Comando para pegar imagen a textures



Fuente: Figura hecha por los autores.

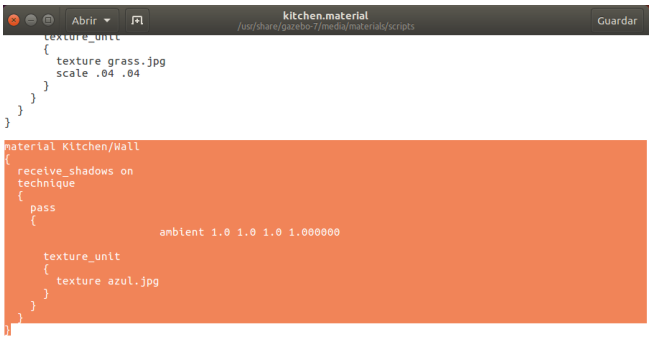
Además, se tuvo que modificar el Script kitchen.material ubicado en la carpeta llamada scripts. Para esto se debe solicitar permisos de administrador utilizando el siguiente comando en la terminal de Linux (Figura 3-32).

Figura 3-32: Comando para modificar el script

Fuente: Figura hecha por los autores.

Luego de haber solicitado los permisos de administrador, el script se puede editar. Lo único que se debe hacer es cambiar el nombre de la textura por la que se agregó (Figura 3-33).

Figura 3-33: Script kitchen_material

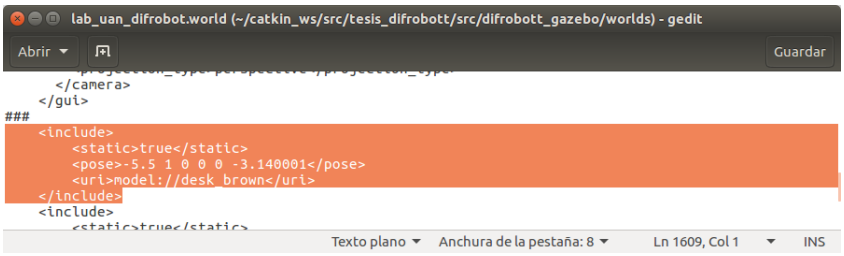


Fuente: Figura hecha por los autores.

3.5.3.2 Agregar modelos 3D al entorno

Para agregar los modelos 3D en el entorno se debe editar el código del script que se encuentra en la carpeta worlds (Figura 3-34). Muchos de los modelos que se usaron se encuentran en (Rasouli & Tsotsos, 2017).

Figura 3-34: Incluir modelos



Fuente: Figura hecha por los autores.

3.6 Teleoperación del robot en el entorno

Antes de comenzar en el proceso de lanzamiento del robot y el entorno en Gazebo, se debe instalar un paquete propio de ROS para poder teleoperar el robot por teclado.

Para eso se usa el siguiente comando:

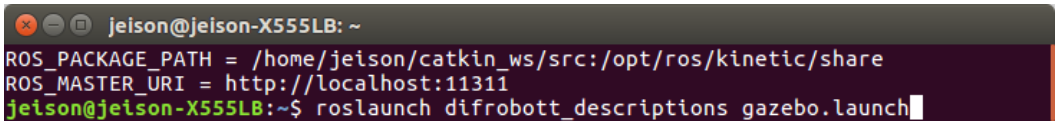
$ sudo apt-get install ros-kinetic-teleop-twist-keyboard

$ catkin_make

3.6.1 Lanzamiento del robot en Gazebo

Ya teniendo todo el entorno y el robot configurados adecuadamente, se procede a ejecutar el archivo de lanzamiento gazebo.launch mencionado en la sesión 3.4.3.2 Contenido carpeta launch en la terminal de Linux. Esto se realiza mediante el comando mostrado en la Figura 3-35.

Figura 3-35: Comando para lanzar gazebo.launch.



Fuente: Figura hecha por los autores

Ya en este punto el robot se puede visualizar en el entorno, sin embargo, no se puede generar ningún movimiento.

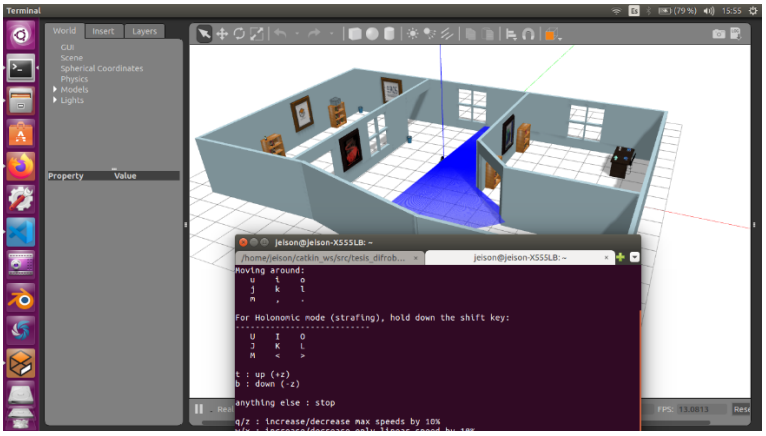
3.6.1.1 Teleoperar el robot

Después de haber lanzado los archivos anteriores, se procede a lanzar el comando para poder teleoperar el robot:

$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Debe mostrar los comandos (teclas) para teleoperar el robot (Figura 3-36).

Figura 3-36: Teleoperación del robot



Fuente: Figura hecha por los autores.

3.7 Mapping

Otro paso importante es la realización del mapa en donde el robot va a realizar las dos trayectorias. Lo primero es instalar el paquete SLAM GMAPPING:

$ sudo apt-get install ros-kinetic-slam-gmapping

Por otro lado, cuando ya se tenga instalado lo anterior se necesita lanzar el entorno de gazebo con el comando mencionado en la sesión 3.6.1. De igual forma en una nueva ventana, se lanza el paquete teleop_twist_keyboard como se explica en la sesión anterior, Tambien se lanza el nodo slam_gmapping con el siguiente comando:

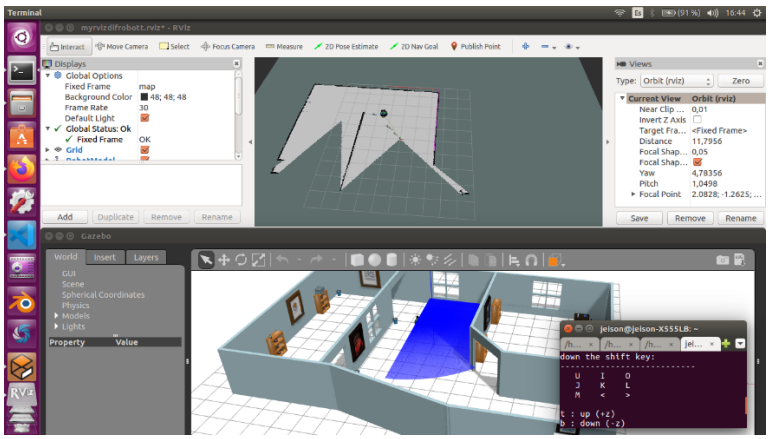
$ roslaunch difrobott_descriptions my_gmapping.launch

Por último, se lanza la interfaz RViz en otra ventana con el comando:

$ roslaunch difrobott_descriptions display.launch

Posterior a esto, se comienza a realizar el mapa 2D del entorno (Figura 3-37). Este mapeo es posible realizarlo con el algoritmo gmapping implementado en el nodo slam_gmapping utilizando los datos que almacena el láser scanner referente a la posición del robot.

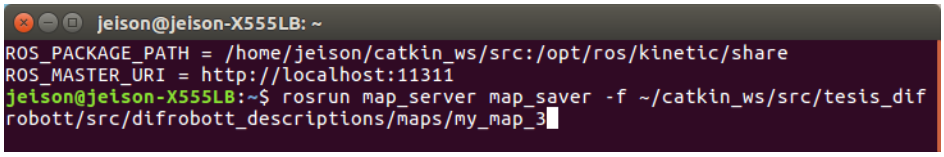
Figura 3-37: Creación del mapa



Fuente: Figura hecha por los autores.

Lo último es guardar el mapa que hayamos creado, para esto se usa el siguiente comando en una nueva ventana de la terminal de Linux (Figura 3-38).

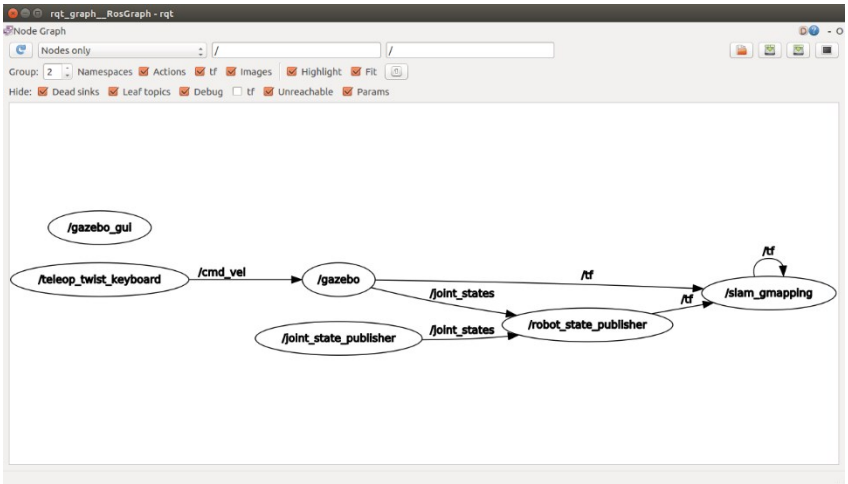
Figura 3-38: Comando para guardar el mapa



Fuente: Figura hecha por los autores.

En resumen, para entender un poco mejor cómo funciona este proceso de creación del mapa observemos el gráfico de computación ROS, aquí detalla la información que recibe el nodo slam_gmapping respecto al posicionamiento y localización del robot (Figura 3-39).

Figura 3-39: Información nodo slam_gmapping



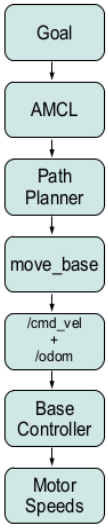
Fuente: Figura hecha por los autores.

3.8 Navigation

Para la generación de las dos trayectorias se usó el nodo move_base y el nodo amcl, estos nodos operan de la mano para que el robot sea capaz de llegar hasta la meta propuesta, es decir que planifican e implementan una trayectoria para el robot.

Esto se logra entender mejor mirando el siguiente diagrama (Figura 3-40).

Figura 3-40: Diagrama planificación de ruta

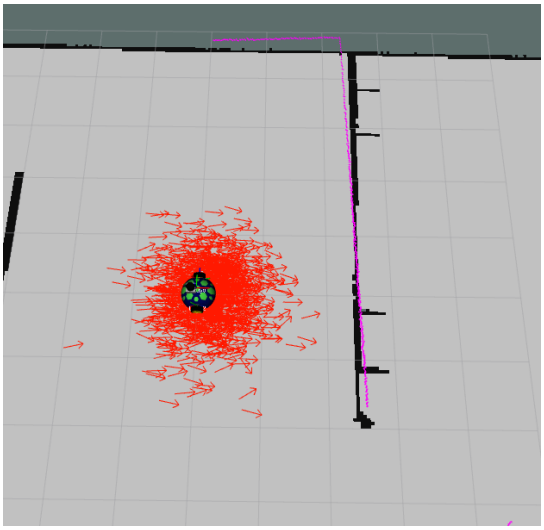


Fuente: Adaptado de ROS By Example (Goebel, 2013).

Básicamente este diagrama sucede internamente en el algoritmo cuando le decimos a nuestro robot una meta específica, es decir que se dirija desde su posición de origen a un punto determinado en el mapa. Primero, en el bloque número uno se fija un “Goal” o “Semantic Goal”, es decir se le comunica al robot en un lenguaje natural para que el entienda a qué lugar debe ir para que realice una acción o simplemente que llegue ahí.

En el bloque numero dos ya entra en acción el nodo amcl, proveniente del paquete AMCL de ROS. Este nodo permite obtener la ubicación del robot en un mapa 2D mediante el escaneo del láser. El cual descarta información que no tengan coincidencia con las lecturas obtenidas por el mismo en el entorno y con esto crea alrededor del robot un grupo de partículas que él considera coincidentes (Flechas), estimando la verdadera localización del robot (Figura 3-41).

Figura 3-41: Localización del robot



Fuente: Figura hecha por los autores.

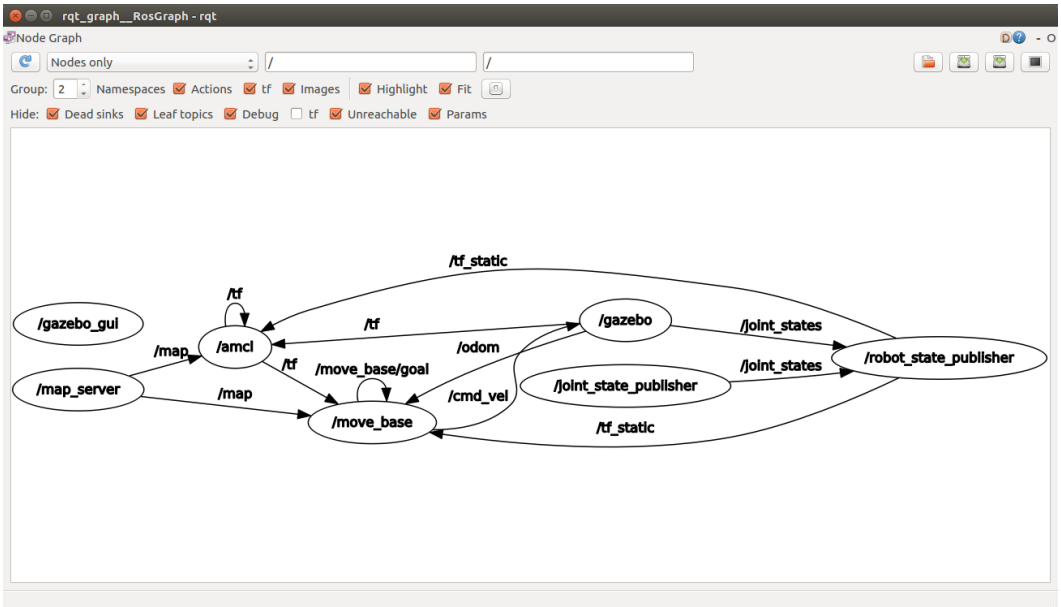
En el bloque número tres, una vez se tiene la correcta estimación de la localización del robot en el mapa, el algoritmo planifica la mejor ruta que debe realizar el robot para que este pueda llegar correctamente al punto determinado. Hasta ahora el algoritmo ya ha fijado la mejor ruta para el robot, pero todavía no está sujeto a que llegue ahí correctamente. Es por esto que en el bloque número cuatro, el nodo amcl trabaja en conjunto con el nodo move_base, ya que este último necesita dos archivos importantes

que ayudan a almacenar información del mapa en general (Local_costmap & Global_costmap).

Finalizando, en los tres últimos bloques el nodo move_base envía la información de las velocidades y posicionamiento del robot para realizar el control del movimiento y con esto poniendo en marcha los motores.

Otra forma de ver cómo funciona los nodos y su comunicación es observando el grafico de computación ROS final del proyecto (Figura 3-42).

Figura 3-42: Gráfico computacional ROS final del proyecto



Fuente: Figura hecha por los autores.

4. Evaluación de resultados

A partir de los resultados encontrados, se puede evaluar cada objetivo de este trabajo. El objetivo general era controlar el seguimiento de dos trayectorias de un robot diferencial basado en el Sistema Operativo de Robótica (ROS).
Estos resultados guardan relación con lo expuesto por (Rivera et al., 2019), quienes señalan que el marco Gazebo-ROS facilita una rápida simulación y control de robots, gracias a que permite la reutilización de código para personalizarlo en función del robot y el entorno de simulación deseado. Esto es acorde con lo que se encuentra en este trabajo.

Otra de las cosas en las que se asemeja estos resultados es que, en el trabajo de los autores mencionados anteriormente, también se usa el Software SolidWorks para importar la geometría en formato urdf del robot diseñado. Uno de los datos más importantes que nos brinda esta exportación es la ubicación del origen de los hijos (child) con respecto a la base_link (parent). Esta información permite obtener una correcta simetría en las piezas del robot y por ende un correcto control de este (Tabla 4-1).

Tabla 4-1: Orígenes de los Link_Name

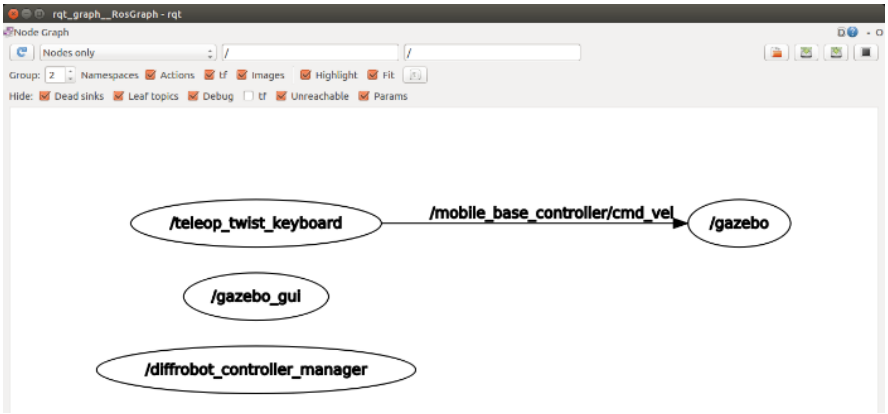
Link Name	Center of Mass X	Center of Mass Y	Center of Mass Z
Base_link	-0,000015159	0,092313	-0,000024405
Link_1	-2,4079E-08	1,1008E-07	0,062029
Link_2	-0,000000024	1,1011E-07	0,062876

Fuente: Tabla hecha por los autores.

Sin embargo, en nuestro caso se tuvo un problema respecto a la información de colores y texturas para el robot, porque al usar el plugin de SolidWorks SW2URDF generaba los modelos 3D en formato stl. Para solucionar esto se tuvo que usar Blender para darle textura y colores a cada elemento del robot (motores, chasis, ruedas, base y Ball Casters) y posterior a eso exportar en formato *.dae.

Con lo que respecta a la teleoperación del robot, algunos autores como (Araújo et al., 2014) exponen como trabajo futuro la integración de un algoritmo de navegación ROS con la adicción de un láser Hocuyo para dotar al robot con la capacidad de realizar 3D SLAM. Esto es lo que se realiza en este trabajo de grado, se descarga un paquete propio de ROS llamado teleop_twist_keyboard para la teleoperación del robot. Como se puede ver en Figura 4-1 se muestra un gráfico de computación ROS que describe de manera general lo que se lleva en el sistema. Hasta ahora el resultado es solo de teoperación, el nodo de teleoperación está ejecutándose y se comunica con el nodo Gazebo.

Figura 4-1: Nodo para teleoperación del robot.



Fuente: Figura hecha por los autores.

4.1 Prueba seguimiento de las dos trayectorias con el controlador move_base_controller

En un inicio para el control de movimiento del robot se utilizó el controlador move_base_controller, según el libro de (Goebel, 2013) el control de un robot móvil se puede hacer en una serie de niveles y ROS brinda métodos para todos estos, desde el control directo de los motores y la planificación de las trayectorias hasta la localización y mapeo del entorno. A partir de esto y que en este trabajo de grado se diseñó un robot móvil tipo diferencial se hace la primera prueba de seguimiento de las dos trayectorias en el entorno, como resultado el robot no podía realizar las trayectorias indicadas ya que el robot tenía un balanceo involuntario a causa del controlador. Este movimiento no solo afectaba a la realización de las trayectorias sino también a la hora de realizar el mapa del entorno en donde el robot se iba a ubicar.

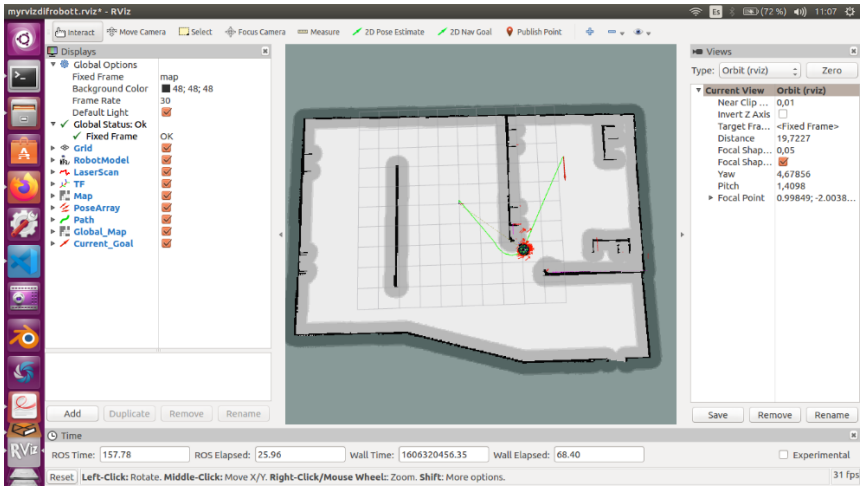
Para solucionar este problema, por lo primero que se optó fue por verificar el diseño del robot respecto a las Ball Caster o ruedas locas que se diseñaron. Sin embargo, se identificó que ese no era el problema sino el controlador que se estaba usando, pues este no ofrecía una funcionalidad adecuada para nuestro robot.

4.2 Prueba seguimiento de las dos trayectorias con el controlador differential_drive_controller

Gracias a que ROS y Gazebo cuentan con una comunidad internacional activa en diferentes plataformas virtuales como Youtube, The construct, Gazebo answer, wiki ROS, etc. Se pudo encontrar el controlador adecuado para nuestro robot, este controlador ya no presento problemas referentes al movimiento del robot y no generaba eventos involuntarios. No obstante, el robot aún no podía realizar las dos trayectorias indicadas en el entorno, esto se debía a que faltaba agregarse parámetros para la planificación de rutas. Para configurar estos parámetros nos guiamos por el libro titulado “ROS By Examples” de R. Patrick Goebel en donde explica los parámetros que requiere el nodo move_base para su funcionamiento.

La primera trayectoria que se realizó en el entorno la podemos observar en la Figura 4-2, ahí se visualiza la planificación de la ruta con una línea verde que ha elegido el algoritmo.

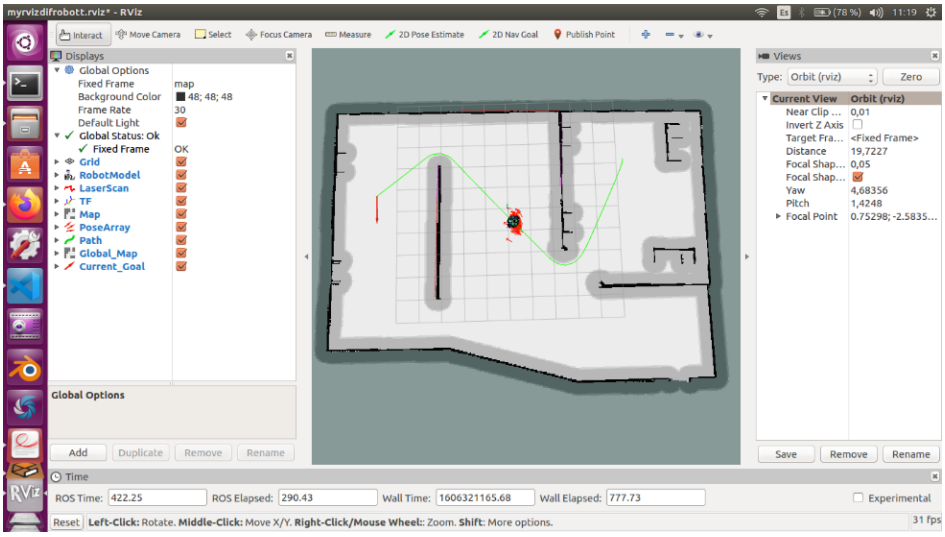
Figura 4-2: Trayectoria número 1



Fuente: Figura hecha por los autores.

Para la segunda trayectoria se eligió hacerla un poco más larga para así ver si el robot respondía correctamente, de igual forma el algoritmo genera una ruta para que el robot la siga y se identifica con una línea verde en el mapa (Figura 4-3).

Figura 4-3: Trayectoria número 2

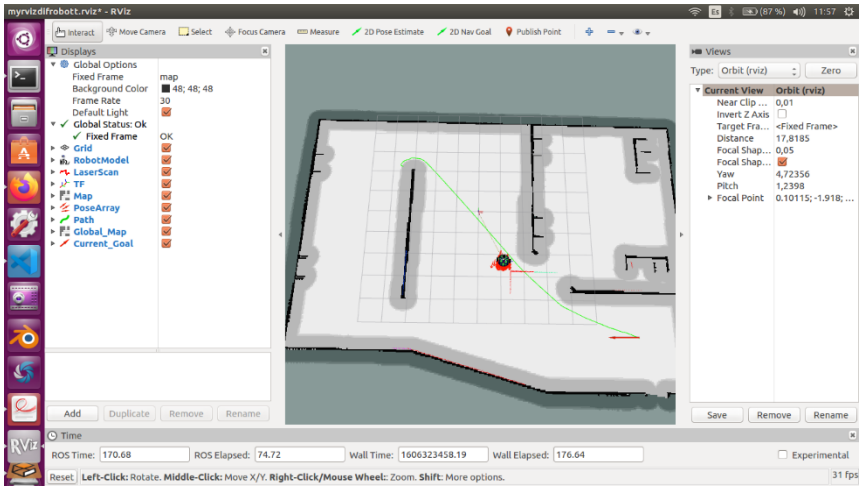


Fuente: Figura hecha por los autores

4.3 Prueba de evasión de obstáculos en el entorno

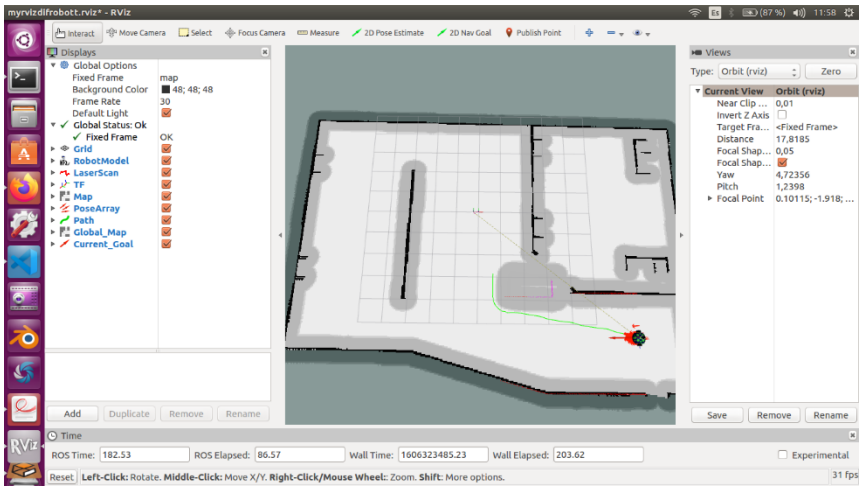
Por último, se quiso realizar una prueba de evasión de obstáculos para verificar la autonomía del robot con una tercera trayectoria (Figura 4-4). Para esto, se agregó en el entorno dos cubos que estuvieran en medio de la trayectoria que el robot iba a realizar; si vemos la Figura 4-5 el robot identifica los cubos y se genera una nueva ruta que evade el obstáculo (cubo). Esto demuestra que el robot puede evadir obstáculos y realizar más de dos trayectorias.

Figura 4-4: Generación tercera trayectoria



Fuente: Figura hecha por los autores

Figura 4-5: Evasión de obstáculos



Fuente: Figura hecha por los autores

5. Conclusiones y recomendaciones

5.1 Conclusiones

En lo que respecta a la construcción del robot, si se hacen sus dimensiones sin tener en cuenta las del entorno, se puede llegar a tener un problema con la visualización del robot, pues este se puede ver muy pequeño o demasiado grande.

Según las dos pruebas que se realizaron para el seguimiento de las trayectorias, concluimos que el controlador más adecuado para nuestro robot es el differential_drive_controller, ya que este resolvió el problema de los movimientos involuntarios del robot que provocaba fallos a la hora de hacer el mapa 2D del entorno y por ende las trayectorias.

Otra de las cosas que pudimos concluir es que con la implementación de los nodos amcl y el move_base se pueden realizar más de dos trayectorias y adicional a esto evadir obstáculos.

Al realizar el algoritmo de navegación se reduce el tiempo de desarrollo de los robots. Específicamente, gracias a que ROS permite la reutilización de código y programación de alto nivel. Esto quiere decir que podemos usar códigos ya creados para personalizarlos de acuerdo a las funciones que queramos cumpla el robot.

Por último, con la realización de este trabajo de grado se abren las puertas para que se realicen más trabajos con plataformas como ROS, sabemos que hoy en día presentamos una crisis a raíz de la pandemia COVID-19 y todo se está llevando a la virtualidad. Por esto, podemos concluir que ROS es una herramienta que se puede trabajar con los estudiantes desde casa y estudiar el comportamiento de diferentes tipos de robots de una manera versátil y entretenida.

5.2 Recomendaciones

Se recomienda la implementación del plugin de cámara en el robot ofrecida por gazebo llamada "camera_controller", la cual podría ser útil para afines de investigación.

Se recomienda en la implementación física del robot, tomar como base la documentación de este trabajo de grado y utilizar una cámara Kinect o Asus Xtion, ya que estas pueden proporcionar un escaneo adecuado.

También se recomienda realizar el curso básico de ROS en The Construct, esto le ayudará a afianzar sus conocimientos y a comprender mucho más este tema.

A. Anexo: Código difrobott.urdf

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This URDF was automatically created by SolidWorks to URDF
Exporter! Originally created by Stephen Brawner
(brawner@gmail.com)
     Commit Version: 1.5.1-0-g916b5db  Build Version:
1.5.7152.31018
     For more information, please see
http://wiki.ros.org/sw_urdf_exporter -->
<robot
  name="difrobott">
  <!-- chasis -->
  <link
    name="base_link">
    <inertial>
      <origin
        xyz="-7.1151E-06 0.033628 0.00022773"
        rpy="0 0 0" />
      <mass
        value="0.32421" />
      <inertia
        ixx="0.00074349"
        ixy="-1.1201E-09"
        ixz="-8.5688E-08"
        iyy="0.0010414"
        iyz="2.271E-07"
        izz="0.0004966" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 1.5708" />
      <geometry>
        <mesh

filename="package://difrobott_descriptions/meshes/base_link.dae"
scale="3 3 3"/>
      </geometry>
      <material
        name="">
        <color
```

```
          rgba="0.6 0.6 0.6 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 1.5708" />
      <geometry>
        <mesh

filename="package://difrobott_descriptions/meshes/base_link.dae"
scale="3 3 3"/>
      </geometry>
    </collision>
  </link>
  <!-- Left Wheel -->
  <link
    name="Link_1">
    <inertial>
      <origin
        xyz="6.2975E-08 -1.0172E-07 0.020781"
        rpy="0 0 0" />
      <mass
        value="0.045823" />
      <inertia
        ixx="1.7767E-05"
        ixy="-2.7362E-10"
        ixz="-2.3295E-11"
        iyy="1.7767E-05"
        iyz="3.7623E-11"
        izz="3.0451E-05" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh

filename="package://difrobott_descriptions/meshes/Link_1.dae"
scale="3 3 3"/>
      </geometry>
      <material
        name="">
        <color
          rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
```

```
        rpy="0 0 0" />
      <geometry>
        <mesh

filename="package://difrobott_descriptions/meshes/Link_1.dae"
scale="3 3 3"/>
      </geometry>
    </collision>
  </link>
  <!-- Joint between Chasis and Left Wheel-->
  <joint
    name="Joint_1"
    type="continuous">
    <origin
      xyz="0.0009 0.208695 0.0561"
      rpy="1.5708 -0.90266 3.1416" />
    <parent
      link="base_link" />
    <child
      link="Link_1" />
    <axis
      xyz="0 0 1" />
    <limit
      effort="10000"
      velocity="1000" />
    <joint_properties damping="1.0" friction="0.0" />
  </joint>
  <!-- Right Wheel -->
  <link
    name="Link_2">
    <inertial>
      <origin
        xyz="9.0756E-08 -7.7957E-08 -0.020781"
        rpy="0 0 0" />
      <mass
        value="0.045823" />
      <inertia
        ixx="1.7767E-05"
        ixy="-3.1023E-10"
        ixz="3.3566E-11"
        iyy="1.7767E-05"
        iyz="-2.8836E-11"
        izz="3.0451E-05" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
```

```
filename="package://difrobott_descriptions/meshes/Link_2.dae"
scale="3 3 3"/>
      </geometry>
      <material
        name="">
        <color
          rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh

filename="package://difrobott_descriptions/meshes/Link_2.dae"
scale="3 3 3"/>
      </geometry>
    </collision>
  </link>
  <!-- Joint between Chasis and Right Wheel-->
  <joint
    name="Joint_2"
    type="continuous">
    <origin
      xyz="0.0009 -0.208695 0.0561"
      rpy="1.5708 0.3016 3.1416" />
    <parent
      link="base_link" />
    <child
      link="Link_2" />
    <axis
      xyz="0 0 1" />
    <limit
      effort="10000"
      velocity="1000" />
    <joint_properties damping="1.0" friction="0.0" />
  </joint>


  <joint name="hokuyo_joint" type="fixed">
    <axis xyz="0 1 0" />
    <origin xyz="0 0 0.315" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="hokuyo"/>
  </joint>

  <!--Hokuyo Laser -->
  <link name="hokuyo">
```

```
    <collision>
      <origin xyz="-0.05 0 -0.03" rpy="0 0 0"/>
      <geometry>
    <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>
    <visual>
      <origin xyz="-0.08 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh
filename="package://difrobott_descriptions/meshes/hokuyo.dae"
scale="3 3 3"/>
      </geometry>
    </visual>
    <inertial>
      <mass value="1e-5" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0"
izz="1e-6" />
    </inertial>
  </link>


   <!--              GAZEBO PLUGINS                    -->
   <gazebo>
    <plugin filename="libgazebo_ros_control.so"
name="gazebo_ros_control">
      <robotNamespace>/robot</robotNamespace>

<robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
      <legacyModeNS>true</legacyModeNS>
    </plugin>
  </gazebo>

  <!--                Gazebo Transmission
-->
  <transmission name="left_wheel_transmission">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="left_wheel_motor">
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
    <joint name="Joint_1">

<hardwareInterface>hardware_interface/EffortJointInterface</hardwa
reInterface>
    </joint>
  </transmission>
  <transmission name="right_wheel_transmission">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="right_wheel_motor">
      <mechanicalReduction>1</mechanicalReduction>
```

```
    </actuator>
    <joint name="Joint_2">

<hardwareInterface>hardware_interface/EffortJointInterface</hardwa
reInterface>
    </joint>
  </transmission>

  <!--                     differential_drive_controller
-->
  <gazebo>
    <plugin filename="libgazebo_ros_diff_drive.so"
name="differential_drive_controller">
      <leftJoint>Joint_1</leftJoint>
      <legacyMode>false</legacyMode>
      <rightJoint>Joint_2</rightJoint>
      <robotBaseFrame>base_link</robotBaseFrame>
      <wheelSeperation>0.41739</wheelSeperation>
      <wheelDiameter>0.0561</wheelDiameter>
      <publishWheelJointState>true</publishWheelJointState>
    </plugin>
  </gazebo>


  <!-- hokuyo -->
  <gazebo reference="hokuyo">
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>4</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>720</samples>
            <resolution>1</resolution>
            <min_angle>-1.570796</min_angle>
            <max_angle>1.570796</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.10</min>
          <max>6.0</max>
          <resolution>0.01</resolution>
        </range>
        <noise>
          <type>gaussian</type>
          <!--Noise parameters based on published spec for Hokuyo
laser
                achieving "+-30mm" accuracy at range < 10m.  A mean
of 0.0m and
```

```
               stddev of 0.01m will put 99.7% of samples within
0.03m of the true
               reading. -->
          <mean>0.0</mean>
          <stddev>0.01</stddev>
        </noise>
      </ray>
      <plugin name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
        <topicName>/difrobott/laser/scan</topicName>
        <frameName>hokuyo</frameName>
      </plugin>
    </sensor>
  </gazebo>

</robot>
```

B. Anexo: gmapping_params.yaml

```
base_frame: base_link
odom_frame: odom
map_update_interval: 5.0
maxUrange: 6.0
maxRange: 8.0
sigma: 0.05
kernelSize: 1
lstep: 0.05
astep: 0.05
iterations: 5
lsigma: 0.075
ogain: 3.0
lskip: 0
minimumScore: 200

srr: 0.01
srt: 0.02
str: 0.01
stt: 0.02
linearUpdate: 0.5
angularUpdate: 0.436
temporalUpdate: -1.0
resampleThreshold: 0.5
particles: 80
xmin: -1.0
ymin: -1.0
xmax: 1.0
ymax: 1.0

delta: 0.05
llsamplerange: 0.01
llsamplestep: 0.01
lasamplerange: 0.005
lasamplestep: 0.005
```

C. Anexo: move_base.launch

```
<launch>

  <master auto="start"/>

  <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">

    <param name="controller_frequency" value="10.0" />
    <param name="controller_patience" value="3.0" />

    <param name="oscillation_timeout" value="30.0" />
    <param name="oscillation_distance" value="0.5" />

    <rosparam file="$(find
difrobott_descriptions)/config/costmap_common_params.yaml"
command="load" ns="global_costmap" />
    <rosparam file="$(find
difrobott_descriptions)/config/costmap_common_params.yaml"
command="load" ns="local_costmap" />

    <rosparam file="$(find
difrobott_descriptions)/config/local_costmap_params.yaml"
command="load" />
    <rosparam file="$(find
difrobott_descriptions)/config/global_costmap_params.yaml"
command="load" />
    <rosparam file="$(find
difrobott_descriptions)/config/base_local_planner_params.yaml"
command="load" />

  </node>

</launch>
```

D. Anexo: my_amcl_params.yaml

```
use_map_topic: true
odom_model_type: diff
odom_frame_id: odom

gui_publish_rate: 10.0
min_particles: 500
max_particles: 2000
kld_err: 0.05
update_min_d: 0.25
update_min_a: 0.2
resample_interval: 1
transform_tolerance: 1.0

laser_max_beams: 60
laser_max_range: 12.0
laser_z_hit: 0.5
laser_z_short: 0.05
laser_z_max: 0.05
laser_z_rand: 0.5
```

Bibliografía

Araújo, A., Portugal, D., Couceiro, M. S., Sales, J., & Rocha, R. P. (2014). Desarrollo de un robot móvil compacto integrado en el middleware ROS. *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, *11*(3), 315–326. https://doi.org/10.1016/j.riai.2014.02.009

Barth, R., Baur, J., Buschmann, T., Edan, Y., Hellström, T., Nguyen, T., Ringdahl, O., Saeys, W., Salinas, C., & Vitzrabin, E. (2014). Using ROS for agricultural robotics : design considerations and experiences. *Second International Conference on Robotics and Associated High-Technologies and Equipment for Agriculture and Forestry, RHEA-2014. May 21-23, 2014 Madrid, Spain.*, 509–518. http://umu.diva-portal.org/smash/record.jsf?pid=diva2%253A720845&dswid=1722

Besseghieur, K. L., Trębiński, R., Kaczmarek, W., & Panasiuk, J. (2018). Trajectory tracking control for a nonholonomic mobile robot under ROS. *Journal of Physics: Conference Series*, *1016*(1). https://doi.org/10.1088/1742-6596/1016/1/012008

Cacace, J., & Joseph, L. (2018). *Mastering ROS for Robotics Programming, Second Edition : Design, build, and simulate complex robots using the Robot Operating System, 2nd Edition.* Packt Publishing.

Foote, T. (2019). *ROS Community Metrics Report*. *July*, 23. http://download.ros.org/downloads/metrics/metrics-report-2014-07.pdf

Gawryszewski, M., Kmiecik, P., & Granosik, G. (2017). V-REP and LabVIEW in the service of education. *Advances in Intelligent Systems and Computing*, *457*, 15–27. https://doi.org/10.1007/978-3-319-42975-5_2

Gazebo. (2014). *Gazebo: Tutorial: Principiante: Descripción general*. http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

Gil, J. R. (2017). *Desarrollo de un caso de estudio para la interacción entre humanos y robots móviles influida por las emociones*. https://riunet.upv.es/handle/10251/86468

Goebel, P. (2013). ROS by Example. In *Journal of Chemical Information and Modeling* (Vol. 53, Issue 9).

Joseph, L. (2015). Learning Robotics Using Python. In *Packt Publishing* (Vol. 44, Issue 8). https://doi.org/10.1088/1751-8113/44/8/085201

Karhumaa, M., Dereck, W., & Walker, J. (2015). AUTONOMOUS NAVIGATION PLANNING WITH ROS. *MICHIGAN TECHNOLOGICAL UNIVERSITY*, *1*, 1.

MetroRobots.com. (2020). *ROS Users of the World*. http://metrorobots.com/rosmap.html

Plaza Cano, M. del M. (2019). *DISEÑO DE TRAYECTORIAS PARA LA INTERCEPCIÓN DE OBJETOS MÓVILES SIGUIENDO CURVAS DE BÉZIER E IMPLEMENTACIÓN SOBRE ROBOTS LEGO*.

Pyo, Y., Cho, H., Ryuwoon, J., & Lim, T. (2017). *Robot Programming From The Basic Concept To Practical Programming and Robot Application*.

Quigley, M., Gerkey, B., & Smart, W. D. (2015). Programming Robots with ROS A Practical Introduction to the Robot Operating System. In *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Rasouli, A., & Tsotsos, J. K. (2017). *The Effect of Color Space Selection on Detectability and Discriminability of Colored Objects*. http://arxiv.org/abs/1702.05421

Rivera, Z. B., De Simone, M. C., & Guida, D. (2019). Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments. *Machines*, *7*(2), 42. https://doi.org/10.3390/machines7020042

robots.ros.org. (2020). *robots.ros.org*. https://robots.ros.org/

ROS.org. (2010). *ROS.org | Sobre ROS*. https://www.ros.org/about-ros/

ROS.org. (2020). *Nodos - ROS Wiki*. http://wiki.ros.org/Nodes

Vargas, H., Rosillón, K., Garcia, K., Arrieta, M., Tancredi, A., Bravo, S., Toro, E., Ordoñez, B., Núñez, G., Urdaneta, E., Villarreal, J. L., Mejías, J., & Rodríguez, R. (2019). Robótica educativa: Un nuevo entorno interactivo y sostenible de aprendizaje en la educación básica. *Revista Tecnológica-Educativa Docentes 2.0*, *7*(1), 51–64.

wiki.ros.org. (2020). *sw_urdf_exporter - ROS Wiki*. http://wiki.ros.org/sw_urdf_exporter