



**UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL**

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO
CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y
AUTOMATISMO

TEMA:

Estudio del Sistema Operativo Robótico y Gazebo. Diseño de una
guía de ejercicios.

AUTOR:

Sánchez Valverde, Margarita del Rocío

Trabajo de Titulación previo a la obtención del título de
INGENIERA ELECTRÓNICA EN CONTROL Y AUTOMATISMO

TUTOR:

Ing. Philco Asqui, Luis Orlando Msc.

Guayaquil, Ecuador

15 de septiembre del 2022



**UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL**

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO
CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y
AUTOMATISMO

CERTIFICACIÓN

Certificamos que el presente trabajo de titulación fue realizado en su totalidad por, **Sánchez Valverde, Margarita del Rocío**, como requerimiento para la obtención del título de **INGENIERA ELECTRÓNICA EN CONTROL Y AUTOMATISMO**

TUTOR

Ing. Philco Asqui, Luis Orlando Msc.

DIRECTOR DE CARRERA

Ing. Bohórquez Escobar, Bayardo, MSc

Guayaquil, a los 15 días del mes de septiembre del año 2022



**UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL**

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO
CARRERA DE INGENIERÍA ELECTRONICA EN CONTROL Y
AUTOMATISMO

DECLARACIÓN DE RESPONSABILIDAD

Yo, Sánchez Valverde, Margarita del Rocío


DECLARO QUE:

El Trabajo de Titulación: **Estudio del Sistema Operativo Robótico y Gazebo. Diseño de una guía de ejercicios**, previo a la obtención del Título de, **Ingeniera Electrónica en Control y Automatismo** ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Titulación referido.

Guayaquil, a los 15 días del mes de septiembre del año 2022

EL AUTOR





**UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL**

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO
CARRERA DE INGENIERÍA ELECTRONICA EN CONTROL Y
AUTOMATISMO


AUTORIZACIÓN

Yo, **Sánchez Valverde, Margarita del Rocío**

Autorizo a la Universidad Católica de Santiago de Guayaquil, la **publicación** en la biblioteca de la institución del Trabajo de Titulación: **Estudio del Sistema Operativo Robótico y Gazebo. Diseño de una guía de ejercicios**, cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y total autoría.

Guayaquil, a los 15 días del mes de septiembre del año 2022

EL AUTOR



AGRADECIMIENTO

A Dios, por acompañarme todos los días, y darme la fortaleza necesaria

A mis padres y familia, quienes son la inspiración y ejemplo diario que forjaron mi camino para convertirme en la persona que soy, siempre me animaron a seguir adelante y luchar por mis metas

A mis amigas quienes siempre me impulsaron a hacer y ser más, gracias por ser esa motivación y alegría en mi vida, por apoyarme cuando más las necesito, por extender su mano en momentos difíciles y por el amor brindado cada día, de verdad mil gracias.

Agradezco a todos los docentes que, con su sabiduría, conocimiento y apoyo, motivaron a desarrollarme como persona y profesional en esta mi alma máter la Universidad Católica Santiago de Guayaquil.

DEDICATORIA

A todas las personas que desde sus trincheras luchan por una mejor sociedad, quienes con sus méritos y valentía se deslindan de las etiquetas sociales que los catalogan como "diferentes" y se toman espacios donde no siempre somos visibles.

A quienes ya no se encuentran en este plano terrenal, pero me vieron crecer, cuidaron y brindaron amor.



**UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL**

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO
CARRERA DE INGENIERÍA ELECTRONICA EN CONTROL Y
AUTOMATISMO

TRIBUNAL DE SUSTENTACIÓN

f.

ING. BOHÓRQUEZ ESCOBAR, BAYARDO, MSC

DIRECTOR DE CARRERA

f.

M. Sc. VÉLEZ TACURI, EFRAÍN OLIVERIO

COORDINADOR DEL ÁREA

f.

M. SC. CÓRDOVA RIVADENEIRA, LUIS SILVIO

OPONENTE

ÍNDICE GENERAL

RESUMEN	XIII
ABSTRACT	XIV
CAPÍTULO 1	2
Generalidades del trabajo de titulación	2
1.1. Justificación y alcance	2
1.2. Planteamiento del problema	2
1.3. Objetivos.....	3
1.3.1. Objetivo general	3
1.3.2. Objetivos específicos	3
1.4. Tipo de investigación	4
1.5. Metodología de investigación	4
CAPÍTULO 2	5
MARCO TEÓRICO	5
2.1 Fundamentos del sistema operativo robótico	5
2.1.1 Estructura del área de trabajo	5
2.1.2 Nivel del sistema de archivos	8
2.1.3 Nivel de gráfico de cálculo ROS.....	9
2.1.4 Simuladores	11
2.2 Fundamentos de la robótica móvil.....	13
2.2.1 Definición de Robot Móvil.....	13
2.2.2 Robots Móviles con Ruedas (RMR)	13
2.3 Robot Diferencial	16
2.3.1 Control de movimiento del robot móvil diferencial.....	17
2.3.2 Modelo cinemático del vehículo	18
2.4 Navegación y sensorización	19
2.4.1 Establecimiento de objetivo y operación	19
2.4.2 Esquemas de navegación	20
CAPITULO 3	25
FUNCIONALIDADES Y ESTRUCTURA DEL SOFTWARE	25
3.1 Análisis de los requerimientos previa simulación.....	25
3.1.1 Modelo Matemático	25
3.2 Estructura de configuración	26
3.3 Parámetros y desarrollo de Scripts.....	27
3.4 Estructura de los nodos para la comunicación	28
CAPITULO 4	30

4	PRACTICAS CON EL SOFTWARE	30
4.1	Instalaciones y elaboración de un robot diferencial	30
4.1.1	Requerimientos	30
4.1.2	Instalación de ROS y paquetes	30
4.1.3	Diseño de un robot móvil diferencial sin sensores	34
4.1.4	Simulación y movimiento del robot diferencial	39
4.2	Simulación de sensor líder y cámara monocular	46
4.2.1	Configuración del LIDAR y la cámara a bordo	46
4.2.2	Visualización de las mediciones del LIDAR y de imagen/video de la cámara	49
4.3	Simulación de actuadores	54
4.3.1	Configuración del servomotor a bordo	54
4.3.2	Herramienta para ajustar las ganancias del controlador usado por el servomotor	60
4.4	Lanzamiento de múltiples robots	62
	CAPITULO 5	66
	CONCLUSIONES Y RECOMENDACIONES	66
5.1	Conclusiones	66
5.2	Recomendaciones	67
	Bibliografía	68
	ANEXOS	70
	Diseño simple del robot diferencial	70
	Robot diferencial con sensor láser y cámara	74
	Robot diferencial con servomotor	81

ÍNDICE DE FIGURAS

Figura 2.1 Estructura de catkin	6
Figura 2.2 Ejemplo de package.xml	7
Figura 2.3 Forma de comunicación en ROS	9
Figura 2.4 Robot en el entorno Gazebo	12
Figura 2.5 Robot en el entorno Rviz	12
Figura 2.6 Tipos de locomoción en un robot móvil.....	13
Figura 2.7 Configuraciones cinemáticas de los RMR	14
Figura 2.8 Tipos de ruedas	14
Figura 2.9 Configuración típica de un robot móvil diferencial	17
Figura 2.10 Esquema de robot diferencial	18
Figura 2.11 Arquitectura de navegación mínima en un robot móvil	20
Figura 2.12 Diagrama del control de navegación.....	21
Figura 2.13 Navegación tipo Blanche	22
Figura 2.14 Diagrama de navegación del Navlab II	24
Figura 3.1 Plano de referencias	25
Figura 3.4 Estructura del proyecto	26
Figura 3.5 Estructura de los scripts.....	27
Figura 3.6 Comunicación ROS - Simulador	28
Figura 4.1 Configuración de fuentes y llaves de ROS	31
Figura 4.2 Espacio de trabajo creado	32
Figura 4.3 Ruta agregada en archivo bashrc.....	33
Figura 4.4 Importación de paquetes a utilizar	33
Figura 4.5 Código de estructura del cuerpo del robot.....	35
Figura 4.6 Elementos que componen el tag visual	35
Figura 4.7 Componentes del elemento joint	36
Figura 4.8 Marco de coordenadas y cuerpo del robot.....	36
Figura 4.9 Código de las llantas del cuerpo del robot.....	38
Figura 4.10 Código de la rueda tipo caster	39
Figura 4.11 Lanzamiento de robot usando la terminal.....	40
Figura 4.12 Robot inicializado en Gazebo	40
Figura 4.13 Marco de referencias del robot	41
Figura 4.14 Eje de rotación de cada uno de los eslabones del robot.....	41
Figura 4.15 Centros de masas de robot.....	42
Figura 4.16 Inercias de robot	42
Figura 4.17 Inicialización del robot en rviz	43
Figura 4.18 visualización del robot en rviz	43
Figura 4.19 Visualización de marcos de referencia en rviz.....	44
Figura 4.20 Tópicos del Robot.....	44
Figura 4.21 Asignación de velocidad lineal y angular	44
Figura 4.22 Movimiento del robot realizado en posición (b).....	45
Figura 4.23 Movimiento del robot iniciado en posición (A).....	45
Figura 4.24 Visualización de la velocidad del robot	45
Figura 4.25 Visualización de referencias en movimiento en rviz.....	46
Figura 4.26 código para configuración general del robot	47
Figura 4.27 código para modificar las llantas del robot.....	47
Figura 4.28 Código para configuración del sensor laser.....	48
Figura 4.29 Código de configuración de la cámara.....	49
Figura 4.30 Simulación del robot con sensor y cámara	50
Figura 4.31 Visualización de la simulación iniciada	50

Figura 4.32 Simulación con objetos para lectura del sensor.....	51
Figura 4.33 Representación de la lectura del sensor laser mediante RViz..	51
Figura 4.34 Tópicos de la cámara del robot.....	52
Figura 4.35 Publicación de velocidad lineal y angular para el robot.....	52
Figura 4.36 Imágenes capturada por cámara del robot	53
Figura 4.37 Lectura de mediciones del sensor laser.....	53
Figura 4.38 Código inicial del archivo xacro del robot con servomotor	54
Figura 4.39 Configuración de las masas.....	55
Figura 4.40 Configuración de parámetros físicos del servomotor.....	56
Figura 4.41 Configuración del motor.....	57
Figura 4.42 Configuración de la cámara en servomotor	57
Figura 4.43 Inicialización del del robot con sensores y servomotor	58
Figura 4.44 Visualización del robot con sensores y servomotor	58
Figura 4.45 Visualización en RViz con sus marcos de referencias.....	59
Figura 4.46 Tópicos generados por el servomotor.....	59
Figura 4.47 Ingreso de posición deseada para el servomotor	59
Figura 4.48 Resultado del valor ingresado para el servomotor.....	60
Figura 4.49 Lectura de posición, velocidad y torque del servomotor	60
Figura 4.50 Inicialización de la herramienta rqt_gui.....	61
Figura 4.51 Activar la lectura del servomotor.....	61
Figura 4.52 Visualización grafica de las lecturas del servomotor.....	62
Figura 4.53 Código para configurar un segundo robot.....	62
Figura 4.54 Configuración de la localización del segundo robot.....	63
Figura 4.55 Configuración general de los robots a ejecutar.....	63
Figura 4.56 Visualización de ambos robots en ejecución	64
Figura 4.57 Tópicos generados por el robot 1 y 2.....	64
Figura 4.58 Lectura de posición, velocidad y torque de los robots	65

RESUMEN

El presente trabajo se desarrolla para la obtención del título de ingeniería electrónica en control y automatismo, el abordaje que se toma en consideración es la creación de guías y pautas para el aprendizaje de robótica general mediante el cual los estudiantes de la Facultad Técnica para el Desarrollo de la Universidad Católica Santiago de Guayaquil adquieran estos conocimientos a través del desarrollo de simulaciones con distintos escenarios que se presentan dentro de la robótica móvil terrestre para ello se busca utilizar las herramientas presentes en el software Gazebo y el control de este por medio de los comandos de ROS. Para lo cual en se emplea la recolección de información referente al funcionamiento de las tecnologías anteriormente mencionadas y relevante al proceso de enseñanza-aprendizaje, además de toda aquella información fundamental al aspecto técnico para proceder a realizar y detallar a través de un manual de prácticas con el simulador.

Palabras claves: Robots, móviles, ROS, simulación, Gazebo, algoritmos.

ABSTRACT

The present work is developed to obtain the title of electronic engineering in control and automatism, the approach that is taken into consideration is the creation of guides and guidelines for the learning of general robotics through which the students of the Technical faculty for the development of the Catholic University of Santiago de Guayaquil acquire this knowledge through the development of simulations with different scenarios that are presented within the terrestrial mobile robotics, for this purpose it is sought to use the tools present in the Gazebo software and its control through the commands of ROS. For which the collection of information regarding the operation of the technologies and relevant to the teaching-learning process is used, in addition to all that information fundamental to the technical aspect to proceed to carry out and detail through a manual of practices with the simulator.

Keywords: Robots, mobiles, ROS, simulation, Gazebo, algorithms.

CAPÍTULO 1

GENERALIDADES DEL TRABAJO DE TITULACIÓN

1.1. Justificación y alcance

Entre los desafíos que enfrentan muchos estudiantes de las materias de robótica es el desarrollo de conocimientos en las distintas herramientas disponibles en la rama de la robótica, siendo uno de estos el software ROS para adquirir los conocimientos específicos de este framework constan de dos etapas para fundamentales: la Fase 1 es aprender los conceptos básicos y las técnicas de programación, mientras que la Etapa 2 se trata de usar ROS para controlar su propio robot por medio de un simulador en una primera instancia.

Las simulaciones proporcionan características que permiten que el robot se desenvuelva cómo se comporta en un entorno particular, por ellos el aprender a utilizar estos simuladores tiene un gran potencial tanto dentro como fuera de las aulas de clases ya que le proporciona al estudiante o investigador una idea general de cómo se comportará el robot y las necesidades bajo las cuales se tolerará de forma tangible.

En este trabajo se busca entender el funcionamiento de la tecnología ROS de forma global, gracias a la realización de una serie de pruebas y experimentos con el robot móvil en el simulador Gazebo que sirva para generar un manual de usuario para los estudiantes.

1.2. Planteamiento del problema

La robótica es una de las manifestaciones de la tecnología cuya aplicación se ha extendido a muchos contextos diferentes de la vida humana. Además de diversas aplicaciones industriales, está presente para facilitar y mejorar actividades como: vuelo de drones, investigación del mundo submarino, limpieza de piscinas, exploración espacial-atmosférica por robot y demás.

Hoy en día, la educación requiere una transición hacia un sistema de aprendizaje que permita a los estudiantes generar y adquirir conocimientos de manera espontánea dentro de este contexto tecnológico previamente descrito. Sin embargo, esto requiere el uso de herramientas específicas para ayudar a lograr el tipo de educación que desea aplicar.

Una de las problemáticas dentro de los laboratorios y asignaturas en las que se estudian estas herramientas es la poca existencia bibliográfica y de guías de alto valor en español o aterrizadas al nivel de los conocimientos de los estudiantes de la facultad, para contrarrestar esto se plantea este material didáctico que se centra en una línea de robótica en la que se utilizan aplicaciones con ROS y Gazebo para actuar como fuente de asesoramiento y desarrollo en los campos de sistemas dinámicos, control y robótica.

1.3. Objetivos

1.3.1. Objetivo general

Elaborar un manual para simulaciones de un robot móvil diferencial mediante el uso del Sistema Operativo Robótico, Gazebo y complementos para la utilización de los estudiantes de la carrera Ingeniería Electrónica en Control y Automatismo.

1.3.2. Objetivos específicos

- Recopilar la información pertinente en las herramientas informáticas previamente establecidas para simular sistemas robóticos reconociendo las rutas de aprendizaje.
- Diseñar modelos de simulación virtual que contengan todo lo necesario para ejecutar las pruebas y su documentación.
- Desarrollar un manual técnico o de laboratorio que explique de manera didáctica el funcionamiento y características técnicas para los estudiantes.

1.4. Tipo de investigación

El presente trabajo de investigación se basa en el tipo teórico con una orientación analítica y documental, se considera que el mayor aporte de la investigación bibliográfica está en el proceso de investigación con el acopio de criterios, conocimientos y experiencias de profesionales e investigadores, que se obtiene utilizando fuentes bibliográficas como tesis, revistas, artículos científicos y técnicos, conferencias web, sitios web de enciclopedias, códigos. En el estudio analítico se determinan los materiales básicos y las características que componen un robot móvil terrestre de tipo diferencial para cumplir con los requerimientos en los distintos escenarios de utilización.

1.5. Metodología de investigación

El siguiente trabajo se estructurará en tres etapas: la primera está relacionada con la investigación, manejo e instalación de paquetes del framework ROS y el simulador Gazebo, así como los mejores accionadores del mismo. Como segunda etapa se tiene el estudio y caracterización del robot a simular esto incluye su estructura, dinámica y forma de adquirir los datos del entorno, finalmente se tiene la simulación del caso estudiado y los resultados obtenidos con el mismo, esto con el fin de realizar las escenarios y prácticas para desarrollar una el manual para los estudiantes.

CAPÍTULO 2

MARCO TEÓRICO

2.1 Fundamentos del sistema operativo robótico

Este apartado muestra que el Sistema Operativo Robótico (ROS) tiene tres niveles conceptuales: el nivel de estructura, el sistema de archivos y el nivel de gráfico de los cálculos ROS; tales permiten en la simulación establecer los recursos necesarios que conforman la precisión entre el robot y la mensajería concretando la ejecución del robot, llegar así al movimiento y comportamiento del robot con las propiedades dinámicas y estáticas frecuentes en el entorno.

Este Sistema Operativo Robótico es un middleware de robot, es decir, un conjunto de marcos para el desarrollo de software de robot. ROS se desarrolló originalmente en 2007 como “switchyard” del Laboratorio de Inteligencia Artificial de Stanford para respaldar el proyecto Robot de Inteligencia Artificial de Stanford (STAIR2). Desde 2008, se ha seguido produciendo un gran desarrollo en Willow Garage, un instituto de robótica que reúne a más de veinte organizaciones que colaboran en un modelo de desarrollo integrado.

El ROS es de código abierto, basado en una colección de herramientas y bibliotecas de software que favorecen a la creación de aplicaciones robóticas, a partir de controladores hasta algoritmos modernos y herramientas de desarrollo potentes. (ROS.org, 2022)

2.1.1 Estructura del área de trabajo

Para trabajar con el sistema principalmente se emplea la creación de carpetas y subcarpetas que conformarán las rutas se sugiere mantener una estructura de trabajo de la siguiente forma:

Build: Contiene los archivos utilizados en la compilación de los comandos `catkin_make` que contendrá los paquetes de codificación fuente necesarios a compilar bajo el uso de `CMakeList.txt`

Devel: En este archivo se recopilan todas las librerías utilizables como los ejecutables que se han creado bajo las compilaciones realizadas.

Src: Bajo esta carpeta se encontrará la programación del robot, es decir, la creación del código y llamado a los paquetes. (Domínguez, 2019)

Para comprender mejor la dinámica de trabajo y su concepción, lo anteriormente descrito se ve plasmado en la figura

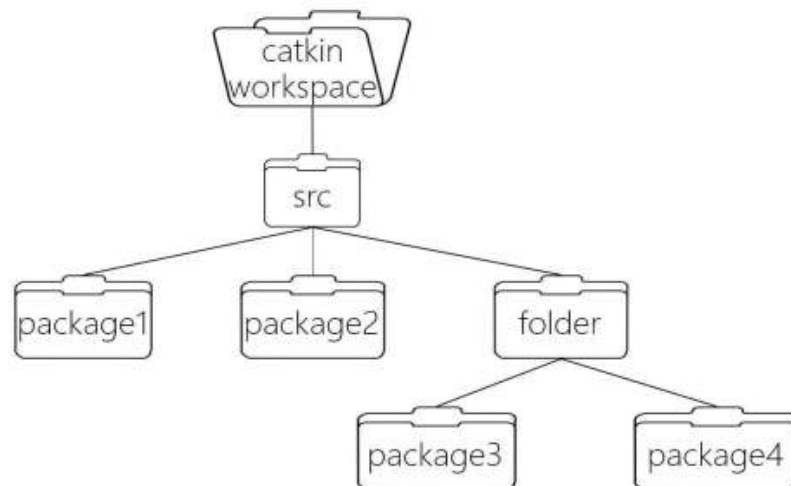


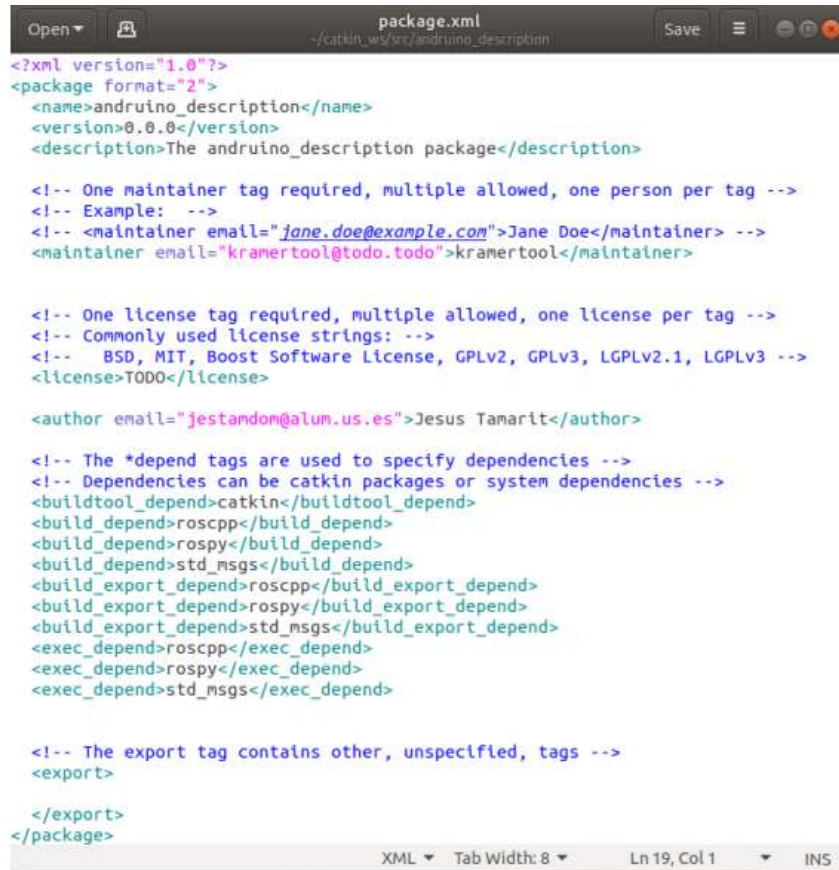
Figura 2.1 Estructura de catkin
Fuente: (Domínguez, 2019)

El modelo del código con el que se desarrolla se estructura en distintos paquetes, que son las unidades que ejecuta ROS, los mismos se le implementa descripciones como nodos, mensajes, servicios, librerías, etc.

Bajo este concepto cuando se requiera crear un paquete se debe ingresar a la consola el siguiente código:

```
$ catkin_create_pkg nombre_paquete dependencias
```

El nombre será asignado bajo el requerimiento, orden y necesidad del usuario, la o las dependencias serán otros sub-paquetes que se desglosen de la inicial para un funcionamiento adecuado y estas pueden ser por ejemplo roscpp, std_msgs u otros.



```
<?xml version="1.0"?>
<package format="2">
  <name>andruino_description</name>
  <version>0.0.0</version>
  <description>The andruino_description package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="kramertool@todo.todo">kramertool</maintainer>

  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>

  <author email="jestandom@alum.us.es">Jesus Tamarit</author>

  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <exec_depend>roscpp</exec_depend>

  <!-- The export tag contains other, unspecified, tags -->
  <export>

</export>
</package>
```

Figura 2.2 Ejemplo de package.xml
Fuente: (Dominguez, 2019)

En la imagen se puede ver el archivo package.xml donde se están estableciendo las características del paquete y datos informativos estándar como lo son el nombre del proyecto, nombre del autor, versión, correo de contacto, dependencia, etc. Mientras que la carpeta src contiene el código, el mismo puede estar el lenguaje C++ o Python también se puede describir estos archivos como nodos puesto que en el transcurso de la ejecución será ejecutados para el proyecto.

2.1.2 Nivel del sistema de archivos

2.1.2.1 Paquetes en ROS

Un paquete es la unidad organizativa principal del software en ROS, un paquete que puede contener procesos de tiempo de ejecución de ROS, bibliotecas dependientes de ROS, archivos de configuración, conjuntos de datos u otros tipos de elementos que son útiles para organizar en unión. Los elementos en la versión ROS se denominan paquetes, el elemento de compilación más atómico y detallado que puede erigir y lanzar o ejecutar es un paquete. (ROS.org, 2019)

2.1.2.2 Meta Packages

Estos son paquetes especiales que se utilizan para figurar un grupo de paquetes vinculados entre sí, los meta paquetes a menudo se usan como un marcador de posición compatible con versiones anteriores para pilas de ros build convertidas. (ROS.org, 2019)

2.1.2.3 Manifiestos de paquetes

Los manifiestos son tipos de paquetes con extensión package.xml, que brindan metadatos en un paquete, donde incluye el nombre, la versión, la descripción, la información de la licencia, las dependencias y otra información en cuanto a los paquetes exportados. La extensión package.xml es un nuevo formato de manifiesto de paquete, debido a que solamente es adecuado para paquetes que usan el sistema actualizado de compilación Catkin. (ROS.org, 2019)

2.1.2.4 Repositorios

Esta es una colección de paquetes donde interviene un común Sistema de Control de Versiones (VCS), aquellos paquetes que comparten un VCS también conlleva la misma versión, y a su vez realizar el lanzamiento juntos con el uso de la herramienta de automatización de lanzamiento de Catkin. Los

repositorios generalmente se asignan a pilas de rosbuid convertidas y pueden contener sólo un paquete. (ROS.org, 2019)

2.1.2.5 Tipos de mensajes

Los mensajes no sólo determinan las estructuras de datos de solicitud sino también la respuesta para los servicios en ROS, los cuales se almacenan en `my_package/msg/MyMessageType.msg`. La descripción del servicio define las estructuras de datos de solicitud y respuesta para los respectivos servicios en ROS, y se almacenan en `my_package/srv/MiTipoDeServicio.srv`. (ROS.org, 2019)

2.1.3 Nivel de gráfico de cálculo ROS

Este nivel del sistema operativo robótico es una red peer-to-peer de procesos ROS que procesan datos entre sí, la base del gráfico de cómputo de ROS son nodos, maestros, servidores de parámetros, mensajes, temas, servicios, bolsas, maestro de acuerdo con la distribución mostrada en la Figura 2.3, estos conceptos son implementados en el repositorio `ros_comm`.

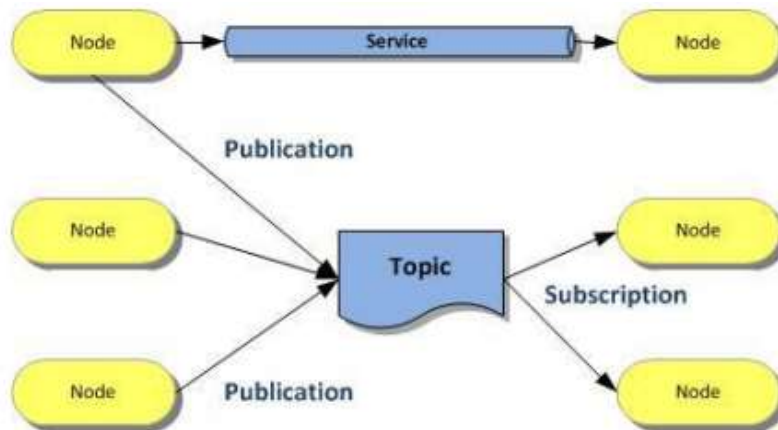


Figura 2.3 Forma de comunicación en ROS
Fuente: (Mazzari, 2022)

- Nodos: Estos son los procedimientos para realizar los cálculos, ROS tiene un diseño modular a una escala muy detallada. Un sistema de control de robot generalmente recibe muchas llamadas, por ejemplo, un botón controla el telémetro láser, un botón controla el motor de la

rueda, uno hace la localización, uno se destaca como planificador de ruta, un nodo que proporciona una vista gráfica del sistema, descrito usando un cliente ROS de biblioteca, como roscpp o rospy.

- Maestro: Proporciona nombres y búsquedas para el resto del nivel gráfico; sin un maestro, los nodos no pueden encontrarse, intercambiar mensajes o llamadas de servicio.
- Servidor de parámetros: Sirve para el almacenamiento de datos por clave en una ubicación central, y es parte del maestro.
- Mensajes: Los nodos se comunican por mensajes, que es una estructura de datos, correspondiente a espacios o campos escritos, se permiten los tipos primitivos estándar (Int, Float, Boolean, Char, entre otros), de manera similar para las tablas de tipo primitivo, los mensajes pueden incluir matrices arbitrariamente anidadas y estructuras
- Temas: El mensaje se pasa a través de un sistema de transporte con semántica de publicación o suscripción; un botón que envía un mensaje al publicarlo en un tema determinado. El tema se debe al nombre que se le da para identificar el contenido del mensaje, puede haber varios publicadores y suscriptores para un mismo tema y nodo puede publicar y/o suscribirse a varios temas. La idea principal es desacoplar la producción de la información de consumo, lógicamente cada bus tiene un nombre y cualquiera puede conectarse al bus para enviar o recibir mensajes siempre que sean de la tipología correcta.
- Servicios: El modelo de publicación/suscripción es un modelo de comunicación muy flexible, donde las solicitudes o respuestas se realizan a través de servicios, definidos por un par de estructuras de mensajes, una para la solicitud y otra para la respuesta. Un nodo proveedor de servicios designado y un cliente usan el servicio enviando un mensaje de solicitud y esperando una respuesta; Las bibliotecas de clientes de ROS a menudo presentan esta interacción al programador como si fuera una llamada a un procedimiento remoto.
- Bolsas: Es un formato para almacenar datos de mensajes ROS, Bag es un importante mecanismo para almacenar datos, como datos de

sensores, que pueden ser difíciles de recopilar, pero necesarios para el desarrollo y prueba de algoritmos.

- **Máster:** Actúa como un servicio de nombres a nivel gráfico del ROS, almacena los asuntos y el servicio de información de registro para los nodos, que se comunican con el maestro para proporcionar información de registro, similar a la información que se puede recibir en otros nodos registrados y establecer una conexión según corresponda. Máster vuelve a llamar cuando estos botones cambian la información de registro original solo proporciona información de búsqueda, como servidores DNS; Establecen la conexión a través de un protocolo de conexión acordado, el protocolo más común usado en el ROS se llama TCROS, que usa sockets TCP/IP estándar. (ROS.org, 2019).

2.1.4 Simuladores

2.1.4.1 Definición e importancia

Una parte importante de la robótica es la pre-implementación que se desarrolla a través de las simulaciones, en estas se realiza la depuración de su código y modelado matemático, de esta forma se evaden errores que podrían representar costos más altos o reducir significativamente la vida útil de su robot. Además, es interesante saber qué robot usar, ya que es fácil seleccionar un robot con una función específica con una sin la necesidad de realizar una inversión inicial y con errores mínimos.

2.1.4.2 Gazebo

Este es un software de simulación gratuito, disponible actualmente solo para los sistemas operativos basados en Linux y funciona como una herramienta robusta, con capacidad para simular ambientes exteriores e interiores, con interfaz gráfica programable. Esto permite cambiar el motor de rigidez y física de la simulación de una manera muy sencilla, permitiendo adaptar lo probado para más situaciones. (Open Robotics, 2021)

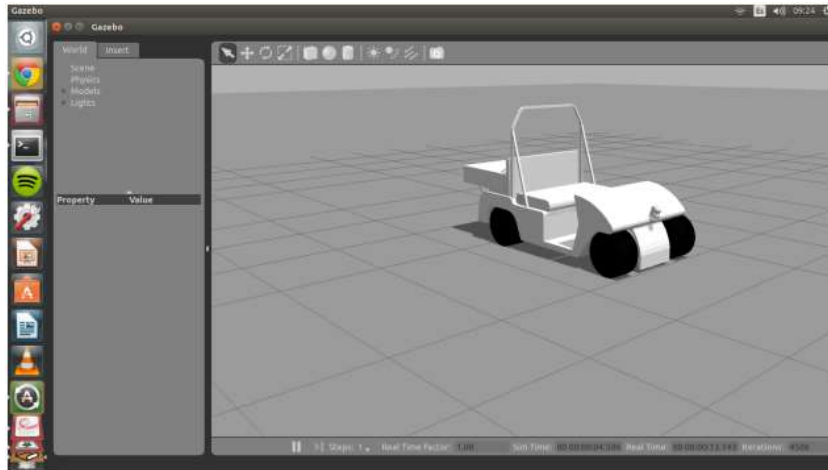


Figura 2.4 Robot en el entorno Gazebo
Fuente: (Open Robotics, 2021)

2.1.4.3 Rviz

Rviz es un visor gráfico que puede representar un entorno 3D de robots, sensores y algoritmos. Puede configurarse fácilmente para cada tipo de robot, por lo que puede mostrar una cantidad considerable en la pantalla o entorno gráfico, el número de variables ROS con un enfoque particular en las variables 3D. Estos datos siempre se representan en relación con el sistema de referencia, también proporciona una serie de complementos y paneles que puede configurar y modificar para realizar las funciones que desea realizar. (ROS, 2018)

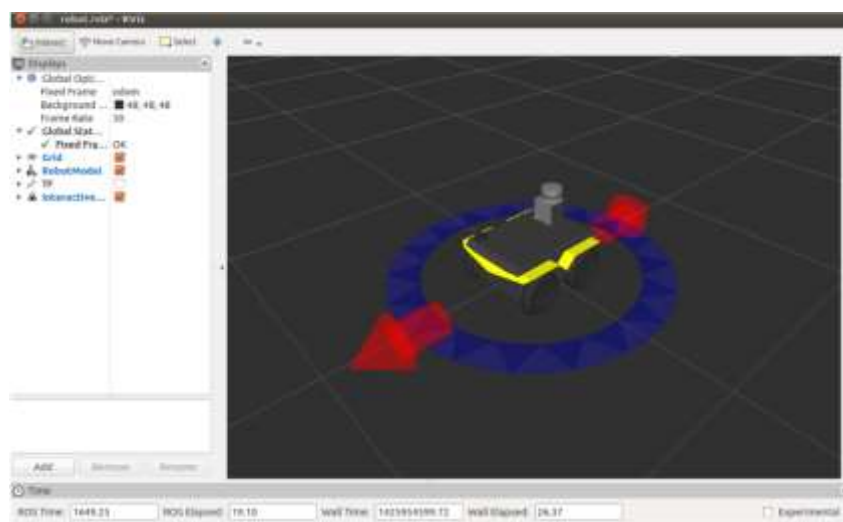


Figura 2.5 Robot en el entorno Rviz
Fuente: (Open Robotics, 2021)

2.2 Fundamentos de la robótica móvil

2.2.1 Definición de Robot Móvil

Los robots móviles se pueden definir como un sistema electromecánico con la capacidad de moverse automáticamente sin estar sujeto desde algún punto, para ello disponen de sensores que les permite monitorizar su entorno, posición, origen y destino, por lo general su control se da en un lazo cerrado.

Su movimiento está garantizado gracias a dispositivos de movimiento, como ruedas, pasadores, ruedas dentadas, etc. (Hernández, Ríos, & Bueno, 2016)



Figura 2.6 Tipos de locomoción en un robot móvil
Fuente: (Cuenca, 2019)

En la figura 2.6 se aprecia la clasificación según el tipo de locomotora utilizada; en general se utilizan tres medios de transporte son: por ruedas presente en el literal (a), por patas literal (b) y por orugas literal (c)

2.2.2 Robots Móviles con Ruedas (RMR)

Como generalidad se puede establecer que es un vehículo con la capacidad de movilizarse de manera autónoma gracias a las operaciones de las ruedas distribuidas en el robot, dentro de sus principales cualidades frente a otras tipologías de robots móviles se encuentra su eficiencia energética en zonas que no producen desgastes como lo son las superficies lisas y firmes (Hernández, Ríos, & Bueno, 2016)

2.2.2.1 Configuraciones cinemáticas de los RMR

Existen diferentes configuraciones cinéticas para el RMR que dependen principalmente de la aplicación en la que se enfoca el RMR; en general sin embargo están disponibles las siguientes configuraciones: Ackerman, triciclo clásico, tracción diferencial, skid steer, síncrona y tracción omnidireccional.

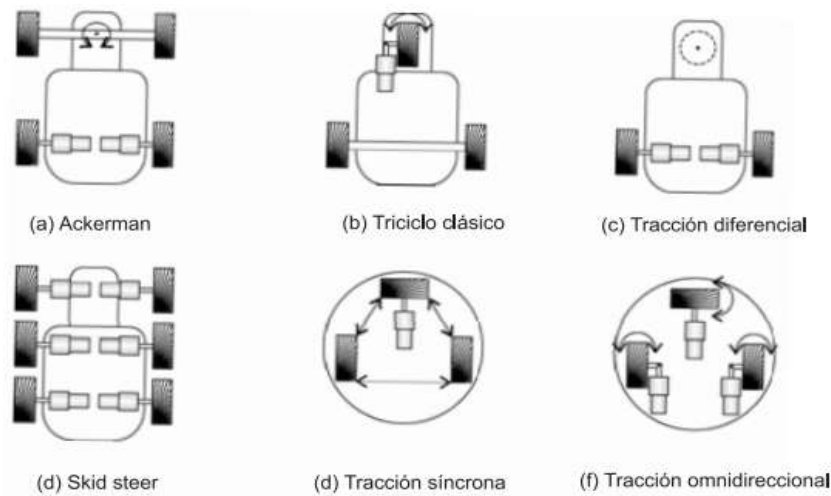


Figura 2.7 Configuraciones cinemáticas de los RMR
Fuente: (García & Silva, 2007)

Dentro de las mismas características bajo el margen de configuración cinemática se utilizan cuatro tipos de ruedas para su desplazamiento que son las siguientes: convencionales, tipo castor, ruedas de bolas y omnidireccionales, se pueden observar en la figura 2.8



Figura 2.8 Tipos de ruedas
Fuente: (García & Silva, 2007)

2.2.2.2 Actuadores en los RMR

Para la descripción de los actuadores que se manejan en estos robots, los más usados en la robótica móvil son los de corriente directa (CD), por el argumento de que su modelo es lineal, lo cual permite de forma enorme su control, y especialmente los de imán persistente ya que el voltaje de control es aplicado al circuito de armadura y el circuito de campo es excitado de manera sin dependencia. Hay dos tipos de motores de CD de imanes permanentes: con escobillas y sin escobillas. Ambos tipos ofrecen ventajas similares sin embargo el motor sin escobillas tiene ventajas significativas sobre el motor con escobillas tales como:

- a) porque son sin escobillas no hay necesidad de reemplazo de escobillas o mantenimiento debido a los residuos de ellos.
- b) no presentan chispas de cepillos por lo que pueden considerarse más seguros en ambientes que contienen vapores o líquidos inflamables.
- c) El ruido debido a la conmutación mecánica de las escobillas se reduce en gran medida mediante la conmutación electrónica.
- d) El motor sin escobillas alcanza velocidades de hasta 50.000 rpm frente a unos 5.000 rpm.

Si bien estas ventajas parecen favorecer a los motores sin escobillas, existen algunas desventajas importantes que pueden cambiar la tendencia:

- a) En los motores sin escobillas, el sentido de giro no se puede invertir cambiando la polaridad de sus terminales, lo que se suma a su complejidad, manejo y costo.
- b) los motores sin escobillas son más caros.
- c) se requiere un sistema adicional de conmutación electrónica.
- d) El controlador de movimiento de un motor sin escobillas es más costoso y complejo que un motor con escobillas comparable.

Al igual que en el arreglo cinemático, al modelar un motor DC se asumen varias consideraciones, de esta manera se puede determinar que el único rozamiento presente es el viscoso, aunque en realidad están involucrados otros tipos de rozamiento no lineal, sin embargo, esta suposición es válida cuando seleccionando motores con efectos friccionales no lineales muy pequeños. (Antón, 2019)

2.2.2.3 Control de los RMR

Desde el punto de vista de la teoría de control, estos sistemas se enmarcan en la llamada región controlable de los sistemas amorfos, que se caracterizan por tener un número de grados de libertad controlables inferior al número total de grados de libertad, en el caso de RMR con tracción diferencial, el número total de grados de libertad es 3, (posición x , y y su dirección ϕ), sin embargo, solo puede controlar el movimiento hacia adelante y hacia atrás, así como su dirección, deja un descontrolado desplazamiento horizontal.

Matemáticamente decno que el sistema está sujeto a límites de tarifa no integrados, lo que significa que su plan de velocidad es limitado. En términos generales, el control de movimiento de RMR se puede clasificar en cuatro tareas básicas; ubicación, planificación de rutas, la pista en sí y evitación de obstáculos. (Fernández, Aracil, & Armada, 2015)

2.3 Robot Diferencial

Un robot móvil diferencial consta de dos ruedas activas conectadas directamente o a través de una transmisión de engranajes a dos servomotores, los cuales se mueven cuando se aplica un voltaje a su entrada. Cuando se aplican voltajes, se inducen velocidades angulares en las ruedas, lo que provoca cambios en la dirección del robot, de modo que, si los voltajes son iguales, el robot se mueve en línea recta, y cuando permanece uno de los voltajes más grandes, el robot describe una trayectoria curva.

En el estudio de los modelos dinámicos de robots móviles diferenciales se establece que algunos sirven para diseñar controladores con su respectiva validación de manera experimental, también para el diseño de los modelos

dinámicos conforme a un robot móvil diferencial de tres ruedas desde el punto de vista de la mecánica de Lagrangiana, con verificación experimental. En la literatura, varios artículos han estudiado los robots móviles diferenciales desde el punto de vista de la cinemática y la dinámica, centrándose en el diseño de controladores y utilizando la mecánica de Lagrangiana para construir modelos. (Cardona, Leal, & Ramirez, 2018)

2.3.1 Control de movimiento del robot móvil diferencial

Para el diseño del controlador se utilizan los conocimientos del modelo dinámico del robot y las herramientas de análisis y diseño que proporciona la teoría de control. Con una estructura particular sobre el robot de movilidad diferencial, es necesario determinar los movimientos que requiere cada rueda del robot, estas ruedas son accionadas por el motor, para poder llevarlo a la posición y orientación requerida en el espacio de trabajo. Este tipo de comando está destinado a asegurar que la trayectoria de seguimiento del robot $q(t)$ sea lo más cercana posible a la trayectoria propuesta por el comando de movimiento que es la trayectoria $q_d(t)$ deseada.

La cinética se ocupa de la descripción analítica del movimiento espacial del robot en función del tiempo y, en particular, de las relaciones entre la posición y orientación finales del robot y sus valores de coordenadas.

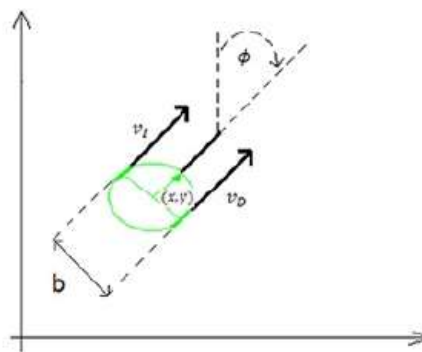


Figura 2.9 Configuración típica de un robot móvil diferencial
Fuente: (Hernández, Ríos, & Bueno, 2016)

En la figura se aprecia el tradicional vehículo de tracción diferencial, el robot se mueve por la acción de dos ruedas motrices independientes (VI y VD, respectivamente); para el planteamiento de un modelo cinético de este tipo de robot, se utilizan las siguientes suposiciones para ayudar a simplificar el modelo:

- El movimiento del robot es efectuado sobre una superficie plana.
- El eje guía es perpendicular al plano.
- La rueda gira libremente.
- El robot no posee piezas flexibles.
- Durante pequeños intervalos cuando la dirección es constante, el vehículo se moverá de un punto a otro a lo largo de una circunferencia.
- El robot se establece como un cuerpo rígido y cualquier parte móvil del timón se moverá de acuerdo con los comandos de control de posición.

Todos estos supuestos se cumplen razonablemente en la mayoría de los robots móviles modernos. El deslizamiento es realmente el mayor problema al que se enfrenta cualquier robot móvil cuando se trata de establecer la posición automática correcta. (Hernández, Ríos, & Bueno, 2016)

2.3.2 Modelo cinemático del vehículo

Este sistema es comparable a un robot con dos ruedas paralelas, el cambio de dirección se logra por la diferencia en la velocidad de rotación de cada rueda. El modelo del sistema diferencial se utilizará la figura 2.10 que cuenta con el esquema del robot como base para desarrollar las siguientes ecuaciones (Sierra-García, & Santos, 2022)

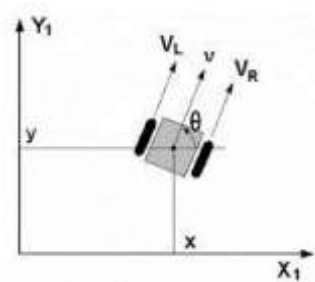


Figura 2.10 Esquema de robot diferencial
Fuente: (Sierra-García & Santos, 2020)

Según (Sierra-García & Santos, 2020) el modelo responde a las siguientes ecuaciones:

$$v = \frac{1}{2} \cdot (v_D + v_I) \quad (1)$$

$$\omega = \frac{1}{L} \cdot (v_D - v_I) \quad (2)$$

$$R_{curva} = \frac{L}{2} \cdot \frac{(v_D + v_I)}{(v_D - v_I)} \quad (3)$$

$$v = \omega \cdot R_{curva} \quad (4)$$

$$\omega = \dot{\beta} \quad (5)$$

Donde v es la velocidad equivalente de la combinación [m/s], v_D , v_I es la velocidad en línea recta de las ruedas derecha e izquierda, respectivamente [m/s], ω es la velocidad de rotación [rad], R_{curva} es la distancia a partir del punto entre las dos ruedas y el centro instantáneo de rotación (CIR) [m]; y L es la distancia entre las ruedas [m].

2.4 Navegación y sensorización

Navegar por un robot móvil implica tareas que crean un mundo abstracto basado en percibir el entorno a través de sensores, planificar un camino suave para llegar a un punto de destino elegido y guiar el vehículo con referencias integradas. Al mismo tiempo, el vehículo puede interactuar con ciertos elementos del entorno. Por lo tanto, el concepto de operaciones se define como la programación de herramientas a bordo que le permiten realizar tareas específicas. (Ortiz, Vásquez, & Muñoz, 2018)

2.4.1 Establecimiento de objetivo y operación

Los robots móviles deben contar con una arquitectura que coordine con precisión y eficiencia los diversos elementos a bordo es decir sus sistemas de detección, control de movimiento y operaciones, para llevar a cabo la misión. El diseño de esta arquitectura depende en gran medida de su aplicación específica, pero en la Fig. 2.11 se puede ver una estructura estándar o generalizada. (Ortiz, Vásquez, & Muñoz, 2018)

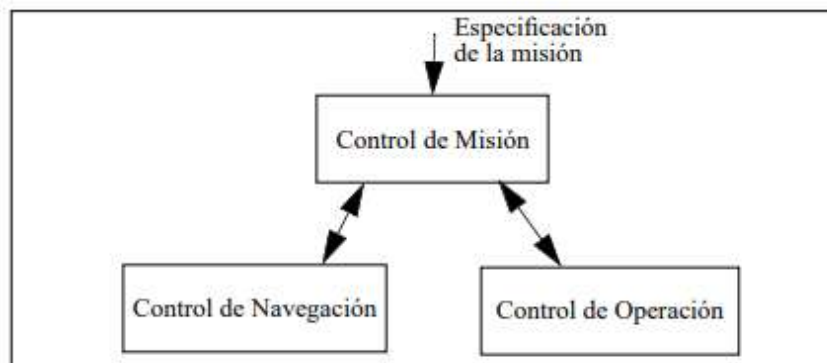


Figura 2.11 Arquitectura de navegación mínima en un robot móvil
Fuente: (Ortiz, Vásquez, & Muñoz, 2018)

Esta secuencia de acciones debe realizarse de forma exitosa en cada una de sus iteraciones para que se pueda llevar a cabo el objetivo establecido por la misión para ello como se observa en la imagen se va a establecer como entrada las especificaciones y limitaciones para la misión, seguido a ello en el bloque determinado como control de misión se llevara a cabo las validaciones de la problemática recibida como entrada para plantear la estrategia de resolución, lo que lo lleva a convertirse en un plan de navegación y de operación.

2.4.2 Esquemas de navegación

Como se ha establecido en anterioridad la navegación no es otra cosa más que el desplazamiento a través de un camino por medio de diferentes posiciones, para lograr esto se desarrollan las siguientes etapas.

1. Precepción del mundo: Gracias a la implementación de sensores se realiza la asimilación del entorno para la creación del mapa.
2. Planificación de rutas: En este ítem se crear una secuencia metódica y sistemática de metas que se pretende lograr. Estas actividades se deducen utilizando un modelo o mapa del entorno obtenido gracias a la etapa previa, una descripción de la tarea a realizar y algún tipo de procedimiento estratégico.

3. Generación de camino: Se define una función que intercala continuamente la serie de metas construida por el programador, luego procede a generar el camino.
4. Seguimiento de la trayectoria: Realiza el movimiento siguiendo la trayectoria creada por el control adecuado de los actuadores.

Si bien estas tareas se pueden realizar de forma independientes deben llevar el orden anteriormente descrito ya que existe una correlación entre cada una de estas, esto constituye una estructura básica de control de navegación (Camino & Rojas, 2019)

Para lo anteriormente descrito se puede ilustrar mediante el siguiente diagrama

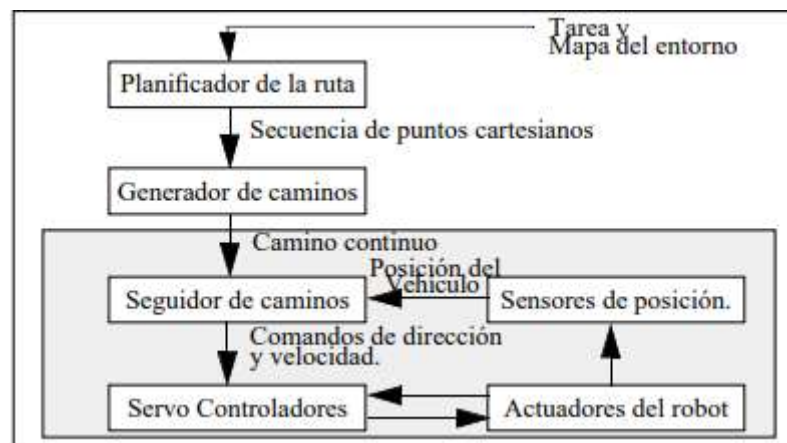


Figura 2.12 Diagrama del control de navegación
Fuente: (Ortiz, Vásquez, & Muñoz, 2018)

En este diagrama expuesto se inicializa a partir del mapa y tareas especificadas, es decir, cuál es el objetivo de la misión, a partir de esta información se planifica la ruta se da a partir de una secuencia de puntos cartesianos que se establezcan como ruta, si este conjunto logra satisfacer los requisitos de la tarea establecida inicialmente sin que esta ruta presente obstrucciones para el desplazamiento procede a ejecutarse el generador de caminos que proporcionará las referencias que seguirá el seguidor de caminos para direccionar los servo controladores por medio de los comandos de dirección y velocidad, recibiendo una retroalimentación por parte de los

sensores de posición del vehículo hacia el seguidor de caminos, esto aplica siempre y cuando el camino elegido sea uno continuo y libre de obstáculos.

Sin embargo, es posible que el modelo de entorno disponible para el robot adolezca de ciertos defectos al omitir algunos de sus detalles. El esquema presentado anteriormente no funciona, ya que no garantiza la construcción de un camino sin obstáculos. Por ello, es necesario introducir nuevos elementos en la estructura básica de control para paliar este defecto y para ello se presentan otros tipos de esquemas de navegación.

2.4.2.1 Navegación tipo Blanche

Los diagramas de navegación se utilizan en aplicaciones donde la información sobre el entorno de trabajo varía desde un conocimiento perfecto del mismo hasta cierto grado de incertidumbre, los diagramas presentados en el sistema corresponden al diagrama desplegado en el robot móvil Blanche

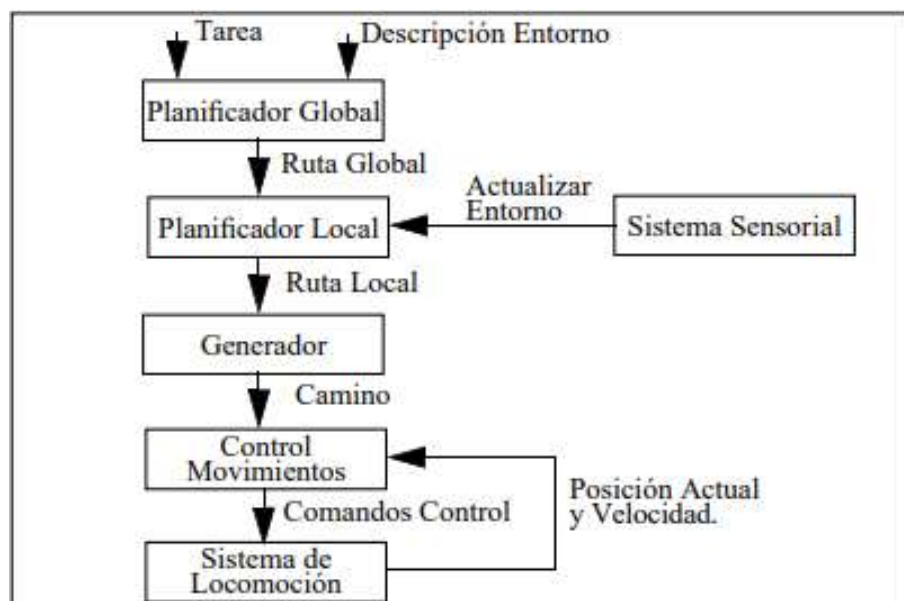


Figura 2.13 Navegación tipo Blanche
Fuente: (Ortiz, Vásquez, & Muñoz, 2018)

Este esquema se compone en una línea de acción general y con una estructura básica su diferencial se encuentra en la partición de sus tareas en dos subrutinas la planificación global y loca, de las cuales la primera es de tipo análoga con un módulo de planificación que edifica la ruta del camino libre de

obstáculos vista anteriormente en base a la digitalización del entorno mediante la información receptada por los sensores al no encontrarse este camino libre se activa la planificación local se acciona al receptar datos del sistema sensorial y a razón del radio de alcance de estos se actualiza el modelo del entorno para que se tome la decisión de realizar un replanificación de la ruta local. (Ortiz, Vásquez, & Muñoz, 2018)

Este esquema es ideal para la adaptación de entornos cambiantes para comprender mejor la diferencia de estos bloques se pueden establecer los siguientes conceptos:

- **Planificación Global:** En esta actividad se construye o planifique una ruta para llevar al robot a cada subobjetivo definido por el control de la misión, de acuerdo con la especificación del problema a resolver. Esta planificación es una aproximación a la ruta final a seguir, ya que a la hora de realizar esta acción no se tienen en cuenta los detalles del entorno local del vehículo. (Berrio, 2008)
- **Planificación Local:** En esta actividad lo que se busca es superar los obstáculos en el camino globalmente en un entorno local para que el robot determine el camino real a seguir. El modelo de entorno local se construye fusionando la información proporcionada por los sensores externos del robot móvil. (Berrio, 2008)

2.4.2.2 Navegación tipo Navlab II

Cuando se habla de navegación en exteriores el reconocimiento del entorno puede ser pobre por lo cual se requiere la mayor utilización de la planificación local, en la navegación del robot móvil Navlab II la configuración empleada fue la siguiente, en primera instancia la navegación está completamente delegada al planificador local, de modo que el camino a seguir se construye dinámicamente a medida que avanza la navegación.

El esquema usó el sistema sensorial para más uso que en el caso de Blanche, y fue responsable de la coordinación de la percepción, la

planificación y el control del vehículo para dirigirlo a lo largo de la ruta prevista y, al mismo tiempo, observar el entorno y hacer la pista de obstáculos. (Cantillo & Berrio, 2014)

La interacción de cada módulo en este esquema de navegación local se muestra en la siguiente figura.

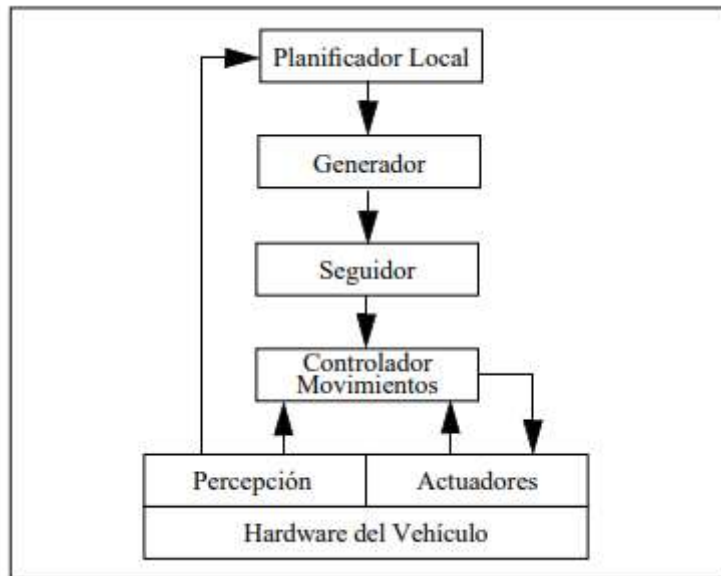


Figura 2.14 Diagrama de navegación del Navlab II
Fuente: (Ortiz, Vásquez, & Muñoz, 2018)

Este diagrama planea una navegación basada en el uso generalizado de sensores de bajo costo, como transductores ultrasónicos, sensores infrarrojos, sensores táctiles y más. para responder con flexibilidad al entorno, perdiendo relevancia el concepto de planificación y trazado. Esta tendencia se basa en una arquitectura de dependencias, una arquitectura que se descompone en módulos que se especializan en realizar tareas individuales, llamadas comportamientos.

Esta es una descomposición vertical del problema de navegar con eficacia en entornos dinámicos donde su conocimiento es impreciso. En cada intervalo de navegación el sistema sensorial, según la información extraída del entorno local del robot, activa uno o varios comportamientos simples que suman sus actuaciones, de suerte que el comportamiento final resulta una mezcla de los simples activados(Cantillo & Berrio, 2014)

CAPITULO 3

FUNCIONALIDADES Y ESTRUCTURA DEL SOFTWARE

3.1 Análisis de los requerimientos previa simulación

Establecido el tipo de robot que se va a simular, sus características y objetivos se procede a realizar los modelados que se deben de cumplir para que esto se realice, para el caso de este escrito se trabajara bajo los siguientes parámetros.

- Robot móvil diferencial
- Control teledirigido

3.1.1 Modelo Matemático

Se realiza un modelado matemático con el fin de comprender los accionamientos que se vean reflejados en la simulación y que estos correspondan al cálculo esperado.

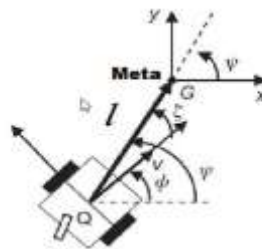


Figura 3.1 Plano de referencias
Fuente: (Cárdenas & Méndez, 2020)

Bajo la representación gráfica presentada en la figura se da lectura en coordenadas polares obteniendo:

$$L = \sqrt{(xrd - xr)^2 + (yrd - yr)^2}$$

$$\zeta = \psi - \phi$$

$$\psi = \text{atan2}((yrd - yr), (xrd - xr))$$

Siendo (L) la distancia entre el punto inicial y el punto final, esta se encuentra en distancia euclidiana, el ángulo (ψ) es el ángulo de orientación

que se puede obtener en el destino, y el ángulo (ζ) viene dado por la diferencia de orientación (ψ) y (Φ) del robot.

También se observa que, entre el punto actual y el punto final, se forma un triángulo rectángulo. Por lo tanto, la tangente se usa para encontrar este ángulo, donde se dice que la tangente es igual al cateto opuesto ($yr_d - yr$) sobre el adyacente ($xrd - xr$).

Planteado esto se obtiene las ecuaciones para el modelado diferencial

$$\dot{L} = -v \cos(\zeta)$$

$$\dot{\zeta} = -\omega + \left(\frac{v}{l}\right) \sin(\zeta)$$

$$\dot{\psi} = \left(\frac{v}{l}\right) \sin(\zeta)$$

3.2 Estructura de configuración

El desarrollo general se debe llevar a cabo con un enfoque modular ya que lo que se realiza en la unión y comunicación de los distintos grupos de archivos que permiten al usuario entender y configuración del robot por medio de simuladores y la influencia del entorno sobre ese.

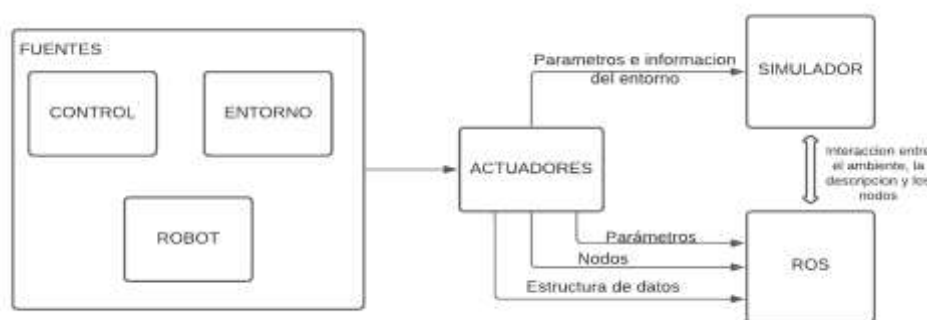


Figura 3.2 Estructura del proyecto
Fuente: Autor

En la figura 3.4 se puede dimensionar como se encuentra configurado los elementos que conforman el proyecto en los cuales se tiene: En primera

instancia las fuentes que son aquellos archivos fuentes que contienen la información del comportamiento y descripción que corresponde al ambiente o entorno de la simulación, al robot y su control. Estos archivos son editables y pueden ser adaptados a los diferentes requerimientos para el comportamiento del robot.

Como segundo grupo de acción constan las estructuras de integración que absorben y procesan los elementos fuentes, estas estructuras al igual que las anteriormente mencionadas también son modificables ya que trabajan con parámetros que pueden ser cambiados como lo son la velocidad con la que se procesa, los tiempos de ejecución y demás requerimientos que impacten en el hardware cuando se lleva a la implementación.

3.3 Parámetros y desarrollo de Scripts

Cuando se habla de parámetro y scripts se hace referencia a la forma en la que se establece la información que se tendrá como características del robot y del entorno que lo rodee, para ello y con el objetivo estudiar el comportamiento de este se desarrollan varios scripts para cada una de las partes.

Como se describió en el marco teórico ROS trabaja a partir de paquetes que contiene toda la información del proyecto lo que reduce en gran medida las fuentes de información que intervienen y evitan los errores en la compilación.



Figura 3.3 Estructura de los scripts
Fuente: Autor

La figura 3.5 contiene las secciones que hacen posible la ejecución de forma simultánea, el descrito como archivos Shell se presenta con la extensión *.sh y en este se encuentran localizados los comando que se ejecutan en el terminal por medio de este se accionan varios programas y archivos ordenadamente, para realizar esto sin perturbar la configuración predeterminada se acciona el siguiente código.

3.4 Estructura de los nodos para la comunicación

Como se estableció al inicio de este capítulo una de las características asignadas fue que el robot pueda ser teleoperador o navegar de manera autónoma, para ambas de las actividades se requiere el envío y recepción de información que accione los motores del robot.

Para poder generar esto la información enviada se realiza en formato de mensajes de clase geométrica, este tipo de mensajes contiene uno o más variables cuya característica es vectorial con variables definidas como el torque, fuerza, velocidad y aceleración correspondiente a cada uno de los ejes. Para el accionamiento tele operado se predefinen las teclas de computador que controlaran las variables de velocidad y fuerza, misma actividad produce el movimiento del robot.

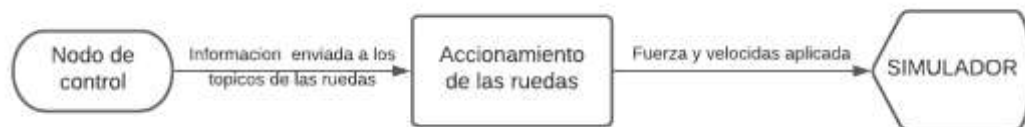


Figura 3.4 Comunicación ROS - Simulador

Fuente: Autor

Para llevar a cabo esta estructura por medio de la consola se debe crear un plugin o descargar uno disponible en los repositorios de ROS que posean relación en la actividad a realizarse para controlarlos, para el caso de este estudio se hizo uso de un plugin prediseñado el cual genera para cada rueda tres tópicos que permite controlar los movimientos.

Por medio de un sensor elegido responde bajo la creación o utilización un plugin que lee los datos del ambiente y envía esta data al tópico para la toma de decisión con ROS.

El sensor escogido para trabajar en esta ocasión es el laser hokuyo mediante el cual se realiza un mapeo estándar mediante el complemento rviz para la retroalimentación de la información.

CAPITULO 4

PRACTICAS CON EL SOFTWARE

4.1 Instalaciones y elaboración de un robot diferencial

4.1.1 Requerimientos

En este capítulo se establecen las pautas para configurar el sistema operativo robótico y sus complementos para las simulaciones a realizar. Uno de los pre-requisitos a tener en cuenta es que la plataforma Gazebo solo funciona de forma adecuada en sistemas operativos Linux, además se debe saber que se puede instalar Gazebo en su última versión o instalar los paquetes de ROS que contienen Gazebo 7, bajo esta premisa se desarrolla bajo los siguientes softwares:

- Ubuntu 20.04
- ROS Noetic
- Gazebo 7

4.1.2 Instalación de ROS y paquetes

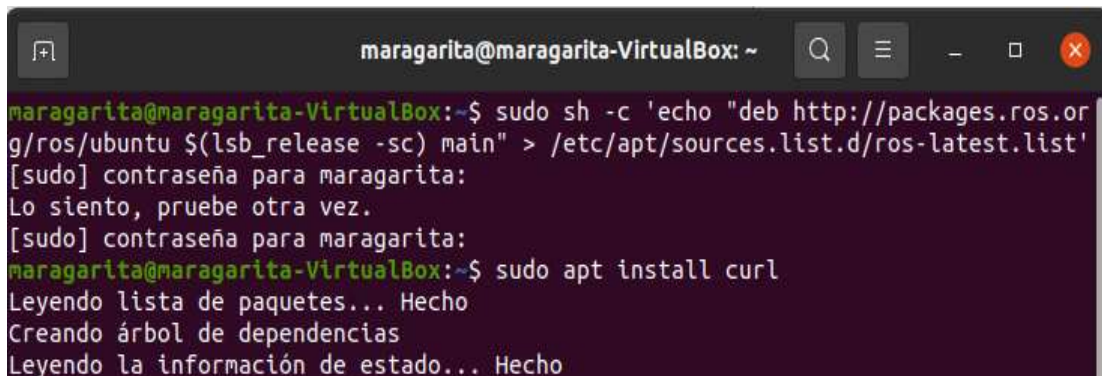
Para generalizar la instalación los códigos correspondientes a cada sistema operativo se pueden localizar en la página oficial de ROS donde se debe selección y seguir las instrucciones según el sistema operativo con el que se esté trabajando, para esta guía se cómo se encuentra especificado en el punto anterior se trabajara con Ubuntu

Para estos los códigos utilizados son los siguientes los cuales son ingresados en el terminal del computador:

- Configuración de la lista de fuentes. - Con estos el computador podrá aceptar software de packages.ros.org.

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

- Configuración de llaves: sudo apt install curl



```
maragarita@maragarita-VirtualBox: ~
maragarita@maragarita-VirtualBox:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
[sudo] contraseña para maragarita:
Lo siento, pruebe otra vez.
[sudo] contraseña para maragarita:
maragarita@maragarita-VirtualBox:~$ sudo apt install curl
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

Figura 4.1 Configuración de fuentes y llaves de ROS
Fuente: Autor

Inicializado los componentes se procede a realizar la instalación, dentro de la misma terminal ya aperturada se ingresa los siguientes códigos:

- sudo apt actualizar: Con esta instrucción se verifica que los componentes se encuentren actualizados a su última versión
- sudo apt install ros-noetic-desktop-full: Se instalan todas las dependencias disponibles
- sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential : Nos permite crear paquetes ROS y otras dependencias.

4.1.2.1 Creación de catkin y clonación de paquetes

En esta sección se podrá observar cómo construir un espacio de trabajo para trabajar con ROS, para esto utilizando el terminal y se utiliza el comando:

- `Mkdir -p ~/my_ws/src`

El comando `mkdir` indica que se va a crear un directorio y el nombre asignado para el espacio de trabajo es “my_ws” que se puede modificar a elección de la persona que esté diseñando.

Desde esta carpeta creada de ingresa:

- `Catkin_make`

Compilando el workspace aunque sin paquetes con esto se puede ver que se nos crean dos nuevas carpetas que son “build” y “devel”

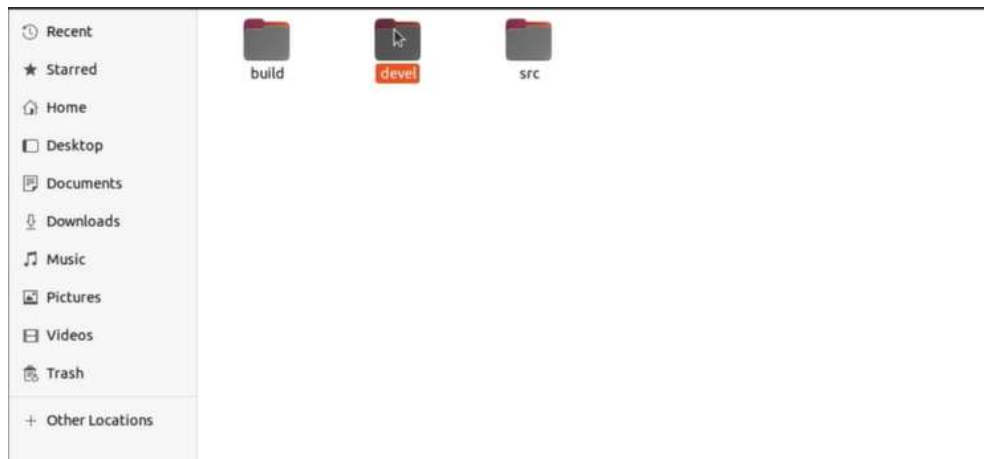
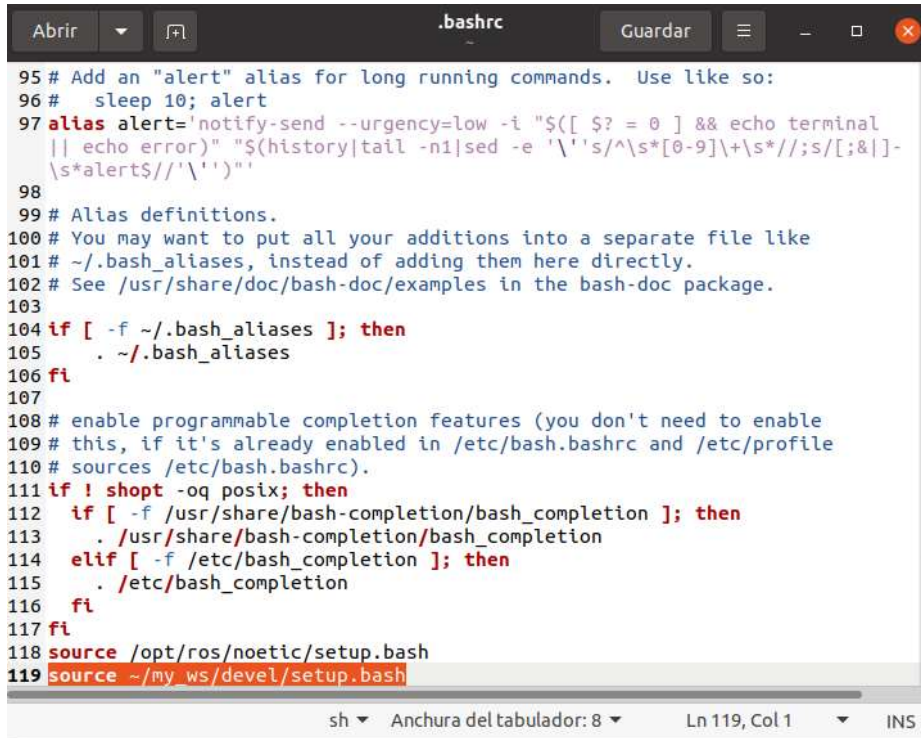


Figura 4.2 Espacio de trabajo creado
Fuente: Autor

Utilizando el comando `control +h` muestro los archivos ocultos y abre el archivo llamado `.bashrc`, en este se agrega una nueva ruta con el nuevo catkin creado como se muestra a continuación.



```
95 # Add an "alert" alias for long running commands. Use like so:
96 #   sleep 10; alert
97 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$//'\`)'"'
98
99 # Alias definitions.
100 # You may want to put all your additions into a separate file like
101 # ~/.bash_aliases, instead of adding them here directly.
102 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
103
104 if [ -f ~/.bash_aliases ]; then
105   . ~/.bash_aliases
106 fi
107
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -oq posix; then
112   if [ -f /usr/share/bash-completion/bash_completion ]; then
113     . /usr/share/bash-completion/bash_completion
114   elif [ -f /etc/bash_completion ]; then
115     . /etc/bash_completion
116   fi
117 fi
118 source /opt/ros/noetic/setup.bash
119 source ~/my_ws/devel/setup.bash
```

Figura 4.3 Ruta agregada en archivo bashrc
Fuente: Autor

Una vez guardado se presiona ctrl +h para volver a ocultar los archivos.

El paquete de ROS que se utilizará será el “my_ugv_descriptor” este se copia y pega en el workspace creado anteriormente en la siguiente ruta:
my_ws/src

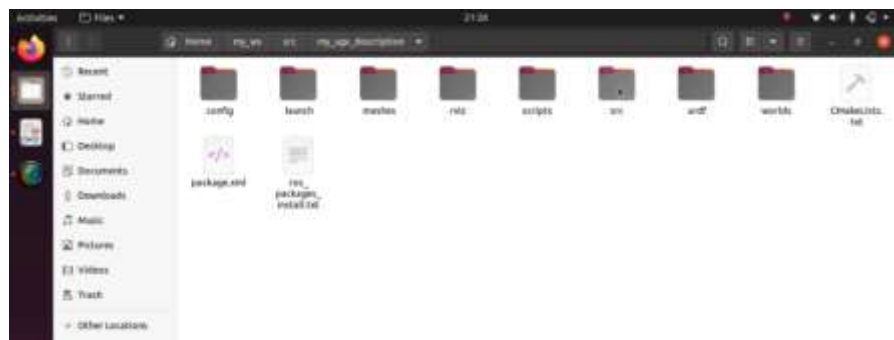


Figura 4.4 Importación de paquetes a utilizar
Fuente: Autor

En la carpeta de paquete my_ugv_description, dentro existe una carpeta llamada urdf.

En esta se encuentran los siguientes Archivo xacro.

- `simple_diff_robot_urdf` punto xacro.

Este es el archivo principal que carga el otro archivo y contiene solamente elementos de tipo URDF como eslabones y juntas.

- `simple_diff_robot_gazebo`, punto xacro.

Contiene especificaciones de gazebo, propiedades físicas, eslabones, color de los eslabones y los plugin de gazebo.

Este archivo abre el simulador gazebo utilizando un mundo vacío y engendra el robot móvil correspondiente.

4.1.2.2 Comandos básicos de ROS para mover al robot tipo diferencial.

Para mover el robot móvil tipo diferencial, dos velocidades son necesarias, la velocidad traslacional y la velocidad rotacional V y W , respectivamente, a través del tópico llamado `cmd_vel`, usando el tipo de mensaje llamado `geometry_msgs/Twist` en sus campos `linear` punto X y `angular` punto Z , respectivamente. La posición y orientación usando cuaterniones del robot móvil se puede obtener a través del tópico llamado `odom` utilizando el tipo de mensaje `nav_msgs/Odometry`.

4.1.3 Diseño de un robot móvil diferencial sin sensores

Para tener una mayor noción de cómo se encuentra la estructura y funcionamiento del robot se explicara en grandes rasgos el archivo `simple_diff_robot_urdf.xacro`.

```

<?xml version="1.0"?>
<robot name="diff_robot" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find my_upv_description)/urdf/simple_diff_robot_gazebo.xacro"/>
  <link name="base_footprint"/>
  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0 0 0.01" rpy="0 0 0"/>
  </joint>
  <link name="base_link"> <!-- Important: Every visual, collision and inertial tag has its own origin -->
    <visual>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.14 0.14 0.03"/>
      </geometry>
    </visual>
    <collision>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.140 0.140 0.03"/>
      </geometry>
    </collision>
    <inertial>
      <origin xyz="0 0 0.015" rpy="0 0 0"/>
      <mass value="0.25e-01"/>
    </inertial>
  </link>
</robot>

```

Figura 4.5 Código de estructura del cuerpo del robot
Fuente: Autor

1. En la primera línea se indica que esto es un lenguaje XML
2. Dentro de la etiqueta robot se va a definir todo nuestro robot se asigna un nombre y se indica que se va a incluir otro archivo en la carpeta o RDF llamado simple_diff_robot_gazebo.
3. Se crea el primer enlace llamado base_foot_print y el primer joint o junta, como se muestra, cada link debe contener las etiquetas de inercia visual y colisión, como se muestra en la figura donde cada una de estas etiquetas contiene su propio origen, es decir, su propio marco de referencia

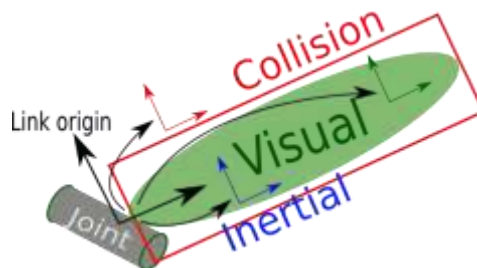


Figura 4.6 Elementos que componen el tag visual
Fuente: (ROS,2021)

De la misma manera para el elemento joint, Este tiene un origen y se especifica la relación entre el padre e hijo igual que otros parámetros dinámicos.

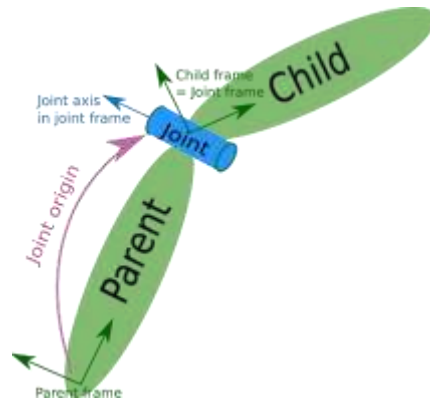


Figura 4.7 Componentes del elemento joint
Fuente: (ROS,2021)

4. Se define un enlace llamado base_Joint de tipo fijo, el cual asigna una relación entre el primer eslabón llamado base_Footprint y el siguiente eslabón que se llamará base_link. La traslación y rotación entre estos dos marcos se especifica con el tag origin.
5. Se utiliza para definir el cuerpo de robot la etiqueta visual con un origen, el cual está desfasado tres centímetros sobre el eje x

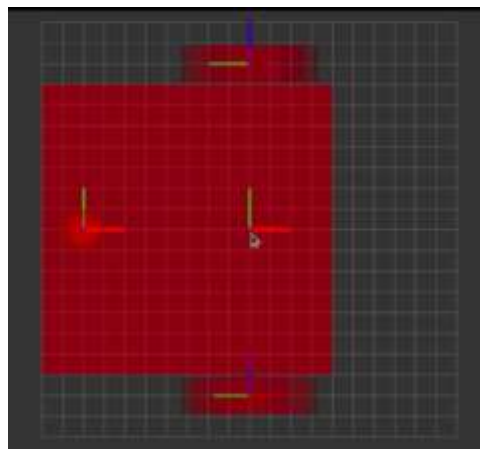


Figura 4.8 Marco de coordenadas y cuerpo del robot
Fuente: Autor

El marco de coordenadas inicial llamado base footprint es el punto medio del eje de las ruedas.

Entonces, para generar un bloque que tiene un centroide desfasado sobre el eje X hacia atrás, es necesario indicar ese desfasamiento de esta manera, el centroide de bloque que compone el cuerpo de robot se localiza atrás del eje de las ruedas.

6. Se genera una caja de 14 centímetros por 14 centímetros y 3 de altura. Para englobar este cuerpo de robot con la etiqueta colisión se toma en cuenta el mismo desplazamiento sobre el eje X y Z que se mostró para el tag visual y las dimensiones de la geometría son exactamente las mismas de esta manera, el origen del tag visual coincide con el origen del tag colisión.

7. Referente al tag de inercia, se tiene un origen igual que el mostrado para el elemento base_link diferente que los dos anteriores. Esto es porque, debido a que se utiliza un plugin para modelar el robot tipo diferencial, entonces sí se asigna un desplazamiento sobre el eje X se tiene un efecto no deseado en el comportamiento del robot, en cuanto al eje Z, el desplazamiento indicado no presenta una un deterioro en el desempeño del robot. En la etiqueta inercia se define también el valor de la masa y la matriz de inercia correspondiente a este elemento.

Hasta el momento se tiene definido un robot que sólo está compuesto por un bloque a continuación se definen las ruedas, los eslabones y las juntas.

```

    iyy="1.4e-03" iyz="0"
    iiz="2.695e-03" />
</inertial>
</link>
[!-- Wheels definition (joints and links) --]
<!-- 8 -->
<joint name="wheel_left_joint" type="continuous">
  <parent link="base_link"/>
  <child link="wheel_left_link"/>
  <origin xyz="0 0.00 0.023" rpy="-1.57 0 0"/>
  <axis xyz="0 0 1"/>
</joint>
<!-- 9 -->
<link name="wheel_left_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <!-- 10 -->
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0" />

```

Figura 4.9 Código de las llantas del cuerpo del robot

Fuente: Autor

8. Se crea una junta llamada Wheel_left_Join de tipo continuo, esto es porque cada rueda va a rotar indefinidamente. El padre es la base, de esta manera las ruedas seguirán al cuerpo base del robot.
9. El enlace para la rueda izquierda se definirá a continuación, la traslación y rotación entre los marcos de referencia se muestran con el tag original, en este caso, si se especifica una rotación sobre el eje X, R minúscula de menos 90 grados. Esto es para qué, para pasar del marco base al marco de cada rueda, se aplica una rotación, de manera que se obtenga el eje de rotación Z de cada rueda.
10. Se utiliza un cilindro para simular la rueda, especificando su altura y su radio, asimismo, para el tag colision coincide como lo que se configura para el tag visual, también se configura los parámetros de masa e inercia para dicho elemento.

A continuación, se definirá la rueda de tipo caster

```
1:-- Caster wheel definition (joint and link) -->
<joint name="caster_joint" type="fixed">
  <parent link="base_link"/>
  <child link="caster_link"/>
  <origin xyz="-0.08 0 0" rpy="0 0 0"/>
</joint>

<link name="caster_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.01"/>
    </geometry>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.01"/>
    </geometry>
  </collision>
  <!-- Since the caster wheel is only a point of contact, inertial tag can be omitted -->
</link>
```

Figura 4.10 Código de la rueda tipo caster
Fuente: Autor

11. Se define una cultura llamada Caster_joint, de tipo fijo, atada al elemento llamado base_link. Con un desplazamiento de – 8 centímetros sobre el eje X. El centroide para el elemento visual de esa rueda tipo astor coincide con el marco de referencia de la junta llamada Caster_link, se utiliza en esta ocasión un elemento llamado esfera, un radio, un centímetro.

12. En el tag de colisión se propone una esfera idéntica que la mostrada anteriormente para el tag visual, en esta ocasión, dado que este tipo de rueda solo sirve como un punto de contacto, el tag de inercia se puede omitir.

De esta manera ya se tiene el robot móvil tipo diferencial.

4.1.4 Simulación y movimiento del robot diferencial

En una nueva terminal, utilizando las teclas control + t, se lanza el archivo lanzador roslaunch, el nombre del paquete y el archivo lanzador que quiero utilizar en este momento.

```

/home/margarita/my_ws/src/my_ugv_description-20220719T2315...
margarita@margarita-VirtualBox:~$ cd my_ws
margarita@margarita-VirtualBox:~/my_ws$ roslaunch my_ugv_description simple_diff_r
obot.launch
... logging to /home/margarita/.ros/log/223a8fd0-1143-11ed-9b63-8dbbb5cd175e/roslau
nch-margarita-VirtualBox-3641.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://margarita-VirtualBox:34517/

SUMMARY
~~~~~
PARAMETERS
* /gazebo/enable_ros_network: True
* /robot_description: <?xml version="1...
* /robot_state_publisher/publish_frequency: 50.0
* /robot_state_publisher/tf_prefix:
* /roscpp: noetic
* /rosversion: 1.15.14
* /use_sim_time: True

```

Figura 4.11 Lanzamiento de robot usando la terminal
Fuente: Autor

De esta manera se agrega y se engendra un robot personalizado, como modelos se tiene el plano horizontal y el robot que se asignó con un nombre, este robot está compuesto por diferentes enlaces, juntas y el plugin.

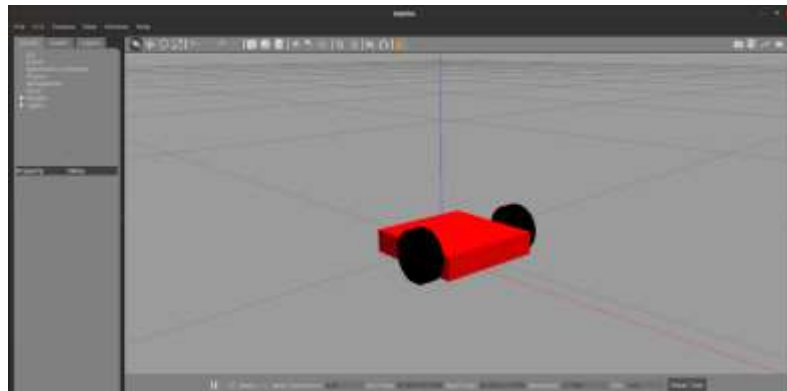


Figura 4.12 Robot inicializado en Gazebo
Fuente: Autor

Utilizando la barra de herramientas opción vistas marcando la opción transparente y link frames, se puede visualizar el marco de referencia llamado base Footprint, el punto medio del eje de las ruedas con una altura cero, el cual coincide con el eje global.

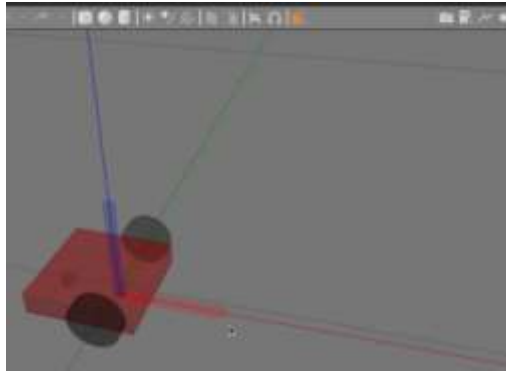


Figura 4.13 Marco de referencias del robot
Fuente: Autor

Asimismo, si se marca la opción joint, se visualizan los marcos de referencia de cada rueda de la junta que configuran dicha rueda. También se muestra el eje de rotación de cada uno de los eslabones que componen la rueda, como se especificó anteriormente, los ejes de revolución de las ruedas son paralelos.

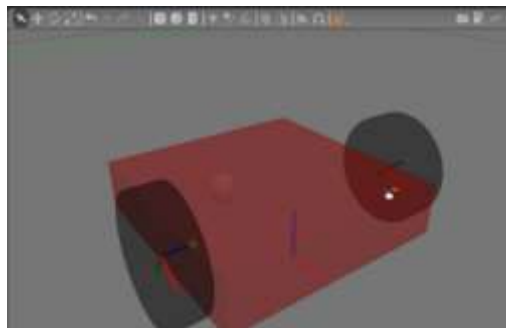


Figura 4.14 Eje de rotación de cada uno de los eslabones del robot
Fuente: Autor

En caso de que se requiera visualizar los centros de masas se pueden habilitar con la opción center of mass para el cuerpo principal del robot y el centro de masa de cada rueda.

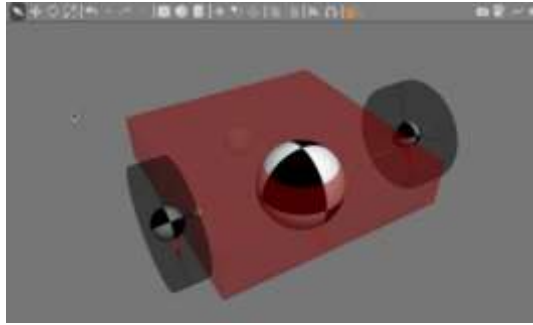


Figura 4.15 Centros de masas de robot
Fuente: Autor

Finalmente, con la opción inercias, se visualiza las inercias que se configuran para nuestro robot.

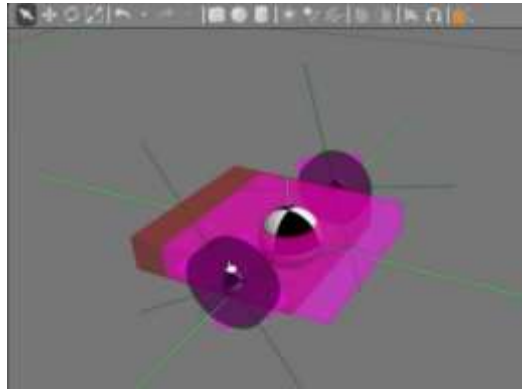


Figura 4.16 Inercias de robot
Fuente: Autor

Debido a que se configura que no hay un desplazamiento en el tag inercia, entonces para Gazebo este robot tiene propiedades físicas, como se muestra en color rosa, diferente a lo visualizado utilizando el tag visual. El tag visual tiene un desplazamiento hacia atrás, lo que implica que el robot va a parecer que está recorrido hacia atrás, no se tiene una inercia para la rueda castor, como se especificó anteriormente.

Utilizando el comando rviz en la terminal como se muestra a continuación se llama al programa

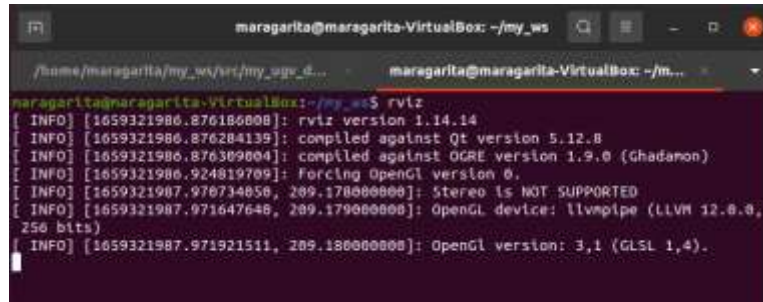


Figura 4.17 Inicialización del robot en rviz
Fuente: Autor

Utiliza el marco de referencia al que quiero referir, el cual se llama como se configuró en el plugin y agrego el plugin para visualizar el nuevo.

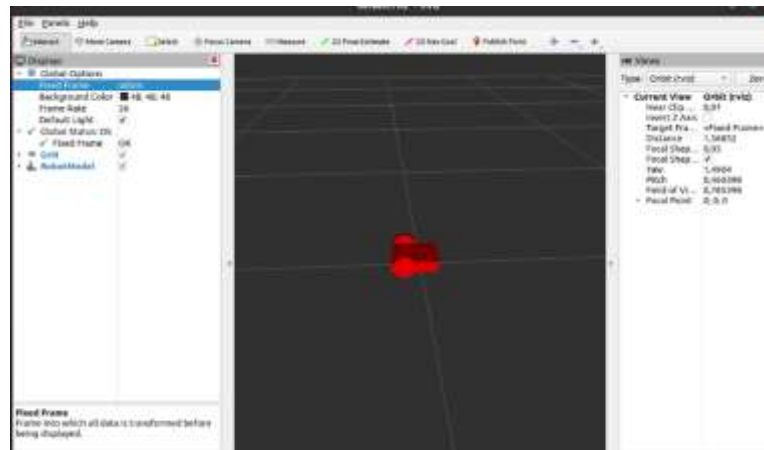


Figura 4.18 visualización del robot en rviz
Fuente: Autor

Asimismo, agregó el plugin del árbol de transformaciones TF, con esto se visualizan los marcos de referencia que componen a mi robot, se pueden marcar para visualizar todos o marcar un punto de referencia específico que se desea revisar.

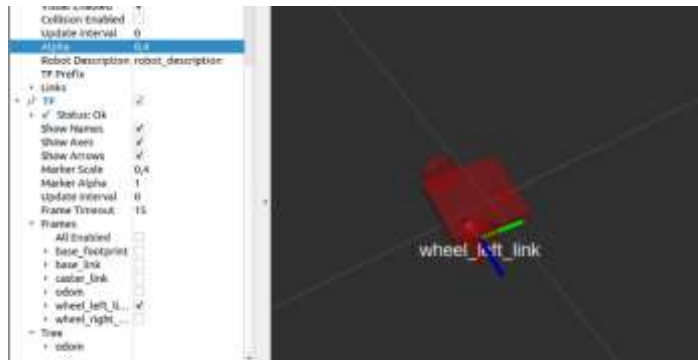


Figura 4.19 Visualización de marcos de referencia en rviz
Fuente: Autor

Para mover un robot en una nueva terminal se hace una lista de tópicos, se muestra el tópico, `cmd_vel` para asignar velocidades de control,

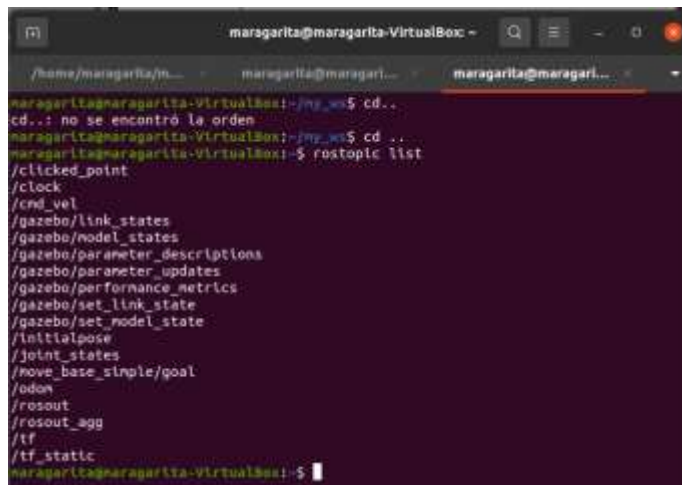


Figura 4.20 Tópicos del Robot
Fuente: Autor

A continuación, se publica una velocidad lineal y angular, con la tecla TAB se completa el comando y se le da play a la simulación.

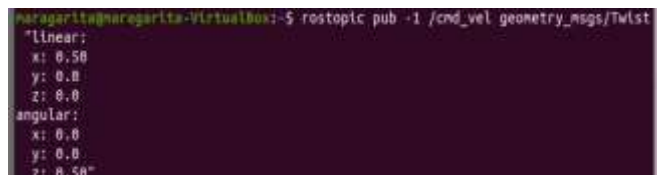


Figura 4.21 Asignación de velocidad lineal y angular
Fuente: Autor

La velocidad lineal que se tiene es 0.45 y la velocidad angular de 0.48 muy próxima a los 0.5 de forma paralela se muestra la rotación de las ruedas en rviz

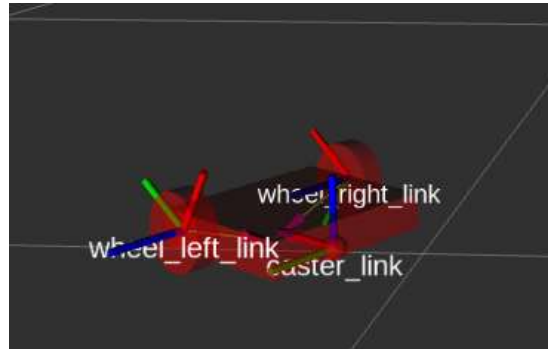


Figura 4.25 Visualización de referencias en movimiento en rviz

Fuente: Autor

4.2 Simulación de sensor líder y cámara monocular

4.2.1 Configuración del LIDAR y la cámara a bordo

Los archivos que se utilizaran para esta práctica son los siguientes:

- `Sensors_diff_robot_urdf.xacro`: Es el actor principal que carga el archivo siguiente y contiene sólo elementos de tipo o URDF, por ejemplo, eslabones, juntas, similar al archivo sacro visto anteriormente
- `sensors_diff_robot_gazebo.xacro`: Contiene especificaciones de gazebo, las propiedades físicas, el color de los eslabones para visualizarse en gazebo y los plugins de gazebo (robot y sensores).

A continuación, se explica el código del robot tipo diferencial con sensores a bordo

1. De la misma manera se asigna un nombre para el robot y se indica que se va a utilizar el segundo archivo.

```
1 <?xml version="1.0"?>
2 <robot name="sensors_diff_robot" xmlns:xacro="http://ros.org/wiki/xacro">
3   <xacro:include filename="$(find my_ugv_description)/urdf/sensors_diff_robot_gazebo.xacro"/>
4
5   <!-- Robot Gazebo Plugins -->
6   <xacro:my_robot_sensors />
7
8
9   <link name="base_footprint"/>
10
11   <joint name="base_joint" type="fixed">
12     <parent link="base_footprint"/>
13     <child link="base_link"/>
14     <origin xyz="0 0 0.01" rpy="0 0 0"/>
15   </joint>
16   <link name="base_link">
17     <visual>
18       <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
19       <geometry>
20         <box size="0.14 0.14 0.03" />
21       </geometry>
22     </visual>
```

Figura 4.26 código para configuración general del robot
Fuente: Autor

2. En esta ocasión se va a utilizar un macro para definir los sensores, el código es similar al mostrado para el caso de robot más simple tipo diferencial, se agregó un mesh para visualizar una llanta con más detalles en lugar de sólo utilizar un cilindro.

```
<!-- Wheels definition (joints and links) -->
<joint name="wheel_left_joint" type="continuous">
  <parent link="base_link"/>
  <child link="wheel_left_link"/>
  <origin xyz="0 0.08 0.023" rpy="-1.57 0 0"/>
  <axis xyz="0 0 1"/>
</joint>
<link name="wheel_left_link">
  <visual>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <!--cylinder length="0.018" radius="0.033"/-->
      <mesh filename="package://my_ugv_description/meshes/husky_wheel.stl" scale="0.18
0.18 0.18"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
```

Figura 4.27 código para modificar las llantas del robot
Fuente: Autor

3. En la definición de el sensor para el sensor LIDAR o radar se crea su junta llamada scan joint de tipo fija, es importante que esta junta

debe de estar lo suficientemente arriba para evitar posibles interferencias en la lectura del sensor. Se configura que será parte del marco llamado base Link, también se define una mallado para visualizar el sensor utilizando un cat llamado o hukuyo.dae.

```

<!-- SENSORS definition -->
<!-- LIDAR -->
<joint name="scan_joint" type="fixed"> <!-- The scan_joint must be high enough to avoid
possible interferences -->
  <parent link="base_link"/>
  <child link="base_scan"/>
  <origin xyz="0.07 0 5(0.03+0.08)" rpy="0 0 0"/>
</joint>
<link name="base_scan">
  <visual>
    <origin xyz="0 0 -0.015" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://my_ugv_description/meshes/hokuyo.dae" scale="1 1 3" />
      <!--cylinder radius="0.035" length="0.1" /-->
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 -0.005" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="0.035" length="0.11"/>
    </geometry>
  </collision>
  <!-- Since a sensor is used, the inertial tag can be omitted -->
</link>

```

Figura 4.28 Código para configuración del sensor laser
Fuente: Autor

4. En cuanto a la cámara, su junta y su enlace se muestran a continuación, una observación importante que se debe resaltar es que tanto el nombre del enlace como el nombre de la junta deben de ser el mismo usados en el plugin. El nombre de la junta de la Cámara Camera RGB Óptica es de tipo fijo, en esta ocasión la cámara está desfasada 3 centímetros hacia delante y 4 centímetros hacia arriba, es importante mencionar que se considera el eje X de este de esta junta como el eje óptico de la cámara, el elemento de la cámara se le asigna el nombre camera_rgb_frame.

En cuanto el CAT que se mostrará para la cámara, utilizó el CAT de un sensor ultrasónico llamado Max_sonar_ez4.

En cuanto a la etiqueta de colisión, define un bloque y no se especifican parámetros físicos, ya que solo es un sensor.

```
<!-- The joint and link of the camera. Note: Both link name and joint name must be the
same as the ones used in the plugin -->
<joint name="camera_rgb_optical_joint" type="fixed">
  <origin xyz="0.03 0 0.04" rpy="0 0 0"/> <!-- X axis is considered as the optical axis -->
  <parent link="base_link"/>
  <child link="camera_rgb_frame"/>
</joint>
<link name="camera_rgb_frame">
  <visual>
    <origin xyz="-0.015 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.04 0.02"/>
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>
  <visual>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://my_ugv_description/meshes/max_sonar_ez4.dae" scale="1 1
0.8" />
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>

  <collision>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.03 0.04 0.02"/>
    </geometry>
  </collision>
  <!-- Since a sensor is used, the inertial tag can be omitted -->
</link>

/robot
```

Figura 4.29 Código de configuración de la cámara
Fuente: Autor

4.2.2 Visualización de las mediciones del LIDAR y de imagen/video de la cámara

Para realizar esta visualización se procede a lanzar el archivo explicado en el punto anterior usando la terminal con el comando: `roslaunch my_ugv_description sensors_diff_robot.launch`.

Como se especificó, la simulación inicia pausada, el robot es el que se muestra a continuación, con sus enlaces, juntas y plugin, los parámetros de la simulación utilizando el archivo se muestran aquí y los parámetros de la

física también, el robot es el mismo utilizado para práctica anterior al cual se le agregaron mallas para las ruedas, el sensor LIDAR y la cámara.

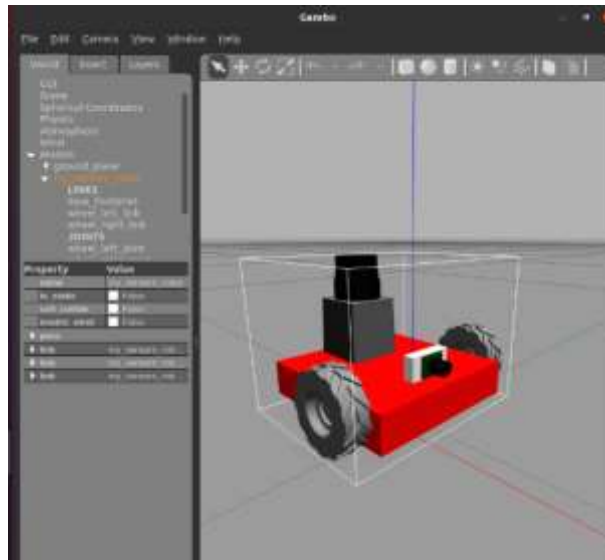


Figura 4.30 Simulación del robot con sensor y cámara
Fuente: Autor

Iniciada la simulación se pueden visualizar los rayos azules que representan la medición del sensor LIDAR utilizando el argumento configurado en el archivo xacro.

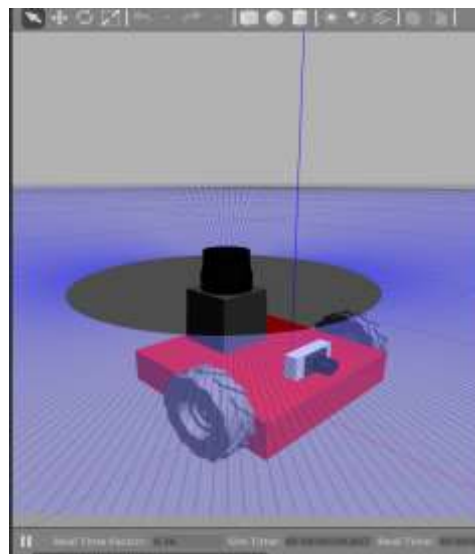


Figura 4.31 Visualización de la simulación iniciada
Fuente: Autor

En la imagen mostrada anteriormente el sensor aún no se encuentra midiendo nada, a continuación, se agregan distintos objetos para obtener mediciones por parte del sensor

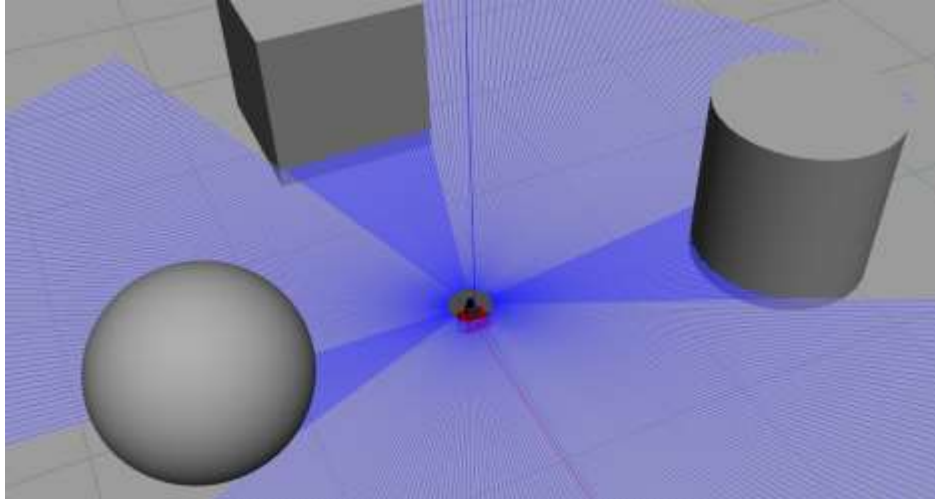


Figura 4.32 Simulación con objetos para lectura del sensor
Fuente: Autor

Una vez que agrego diferentes elementos, entonces ya se tendrá una lectura de salida y se puede mostrar en RViz, lo que se ve en la imagen a continuación es la representación de lo que registra el robot a través del sensor LIDAR.

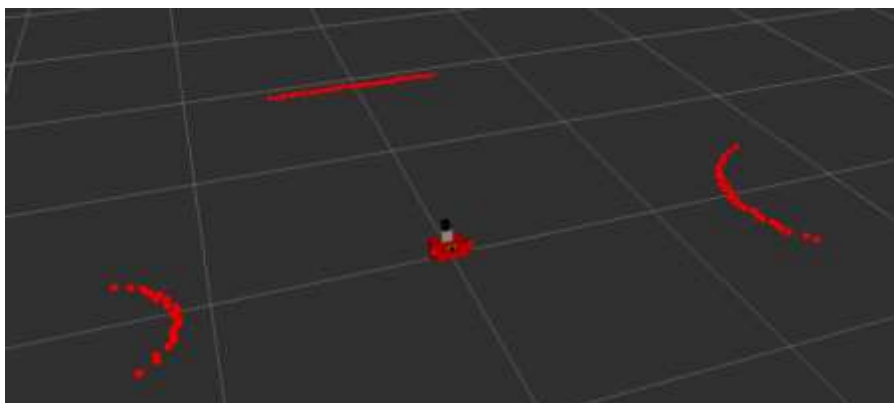


Figura 4.33 Representación de la lectura del sensor laser mediante RViz
Fuente: Autor

A continuación, se genera una lista de tópicos y se muestran los tópicos siguientes para visualizar la imagen capturada por la cámara, se utiliza el plugin image donde se especifica el nombre del tópico en cuestión. Se tiene el tópico cmd_vel para asignar velocidades de control, el tópico odom para suscribirse a su odometría, tópicos para visualizar las medidas del sensor LIDAR.

```
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clicked_point
/clock
/cmd_vel
```

Figura 4.34 Tópicos de la cámara del robot
Fuente: Autor

Por medio de estos se publica una velocidad para asignar movimiento al robot, la lectura del sensor LIDAR no se modifica, aunque el robot esté en movimiento. Esto es por se configura el marco de referencia global, de esta manera, las lecturas del sensor se transforman para siempre mostrarse con respecto al marco de referencia inercial.

```
maragarita@maragarita-VirtualBox:~$ rostopic pub -1 /cmd_vel geometry_msgs/Twist
"linear:
  x: 0.080
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.40"
publishing and latching message for 3.0 seconds
```

Figura 4.35 Publicación de velocidad lineal y angular para el robot
Fuente: Autor

4.3 Simulación de actuadores

4.3.1 Configuración del servomotor a bordo

De manera similar a los robots móviles previos, en la carpeta fuera de F se localizan dos archivos de tipo xacro `sensor_diff_robot_urdf.xacro`, el cual incluye también el tag de transmisión y sensor DIF robot de acero con el plugin de `ros_control`, el cual requiere el argumento llamado `robot_ns` desde el archivo lanzador.

Adicionalmente, en la carpeta `config` se localiza el siguiente archivo de configuración llamado `servomotor_controller.yaml`, este es el archivo de configuración que contiene las ganancias del controlador PID y la configuración del controlador. Para sintonizar las ganancias del controlador PID utilizado para controlar la posición del servomotor a bordo del robot móvil tipo diferencial, la herramienta llamada `rqt_gui` se puede utilizar.

Ahora se procede a explicar el archivo xacro para un robot tipo diferencial con un servomotor a bordo. Se define el nombre para el robot y también el argumento llamado `PI`. Se indica que se va a utilizar otro archivo xacro, de la misma manera, utiliza un marco para definir los plugin del robot y el controlador.

```
<?xml version="1.0"?>
<robot name="servomotor_diff_robot" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:arg name="pi" default="3.14159"/>
  <xacro:include filename="$(find my_ugv_description)/urdf/servomotor_diff_robot_gazebo.xacro"/>
```

Figura 4.38 Código inicial del archivo xacro del robot con servomotor
Fuente: Autor

Debido a que se va a utilizar un elemento para el motor, el cual tiene inercia, es necesario modificar los valores de más masa e inercia para hacerlos más grande y de esta manera tener un desempeño aceptable para el movimiento de robot con el sensor a bordo de otra manera, es posible que el robot se vea afectado por el sensor, dado que este tiene masa no despreciable en comparación a la masa del robot, de manera que se obtenga el movimiento aproximado del modelo cinemático de robot, ya que se utiliza un plugin.

```
<origin xyz="0 0 0.015" rpy="0 0 0"/> &!-- Note: A plugin is used to move the robot. To stabilize the mobile robot due to the servo_link has inertia, a larger mass and inertia are used -->
<mass value="${5*8.25e-01}"/>
<inertia ixx="${5*1.4e-03}" ixy="0" izx="0"
        iyy="${5*1.4e-03}" iyz="0"
        izz="${5*2.695e-03}" />
</inertia>
</link>
```

Figura 4.39 Configuración de las masas
Fuente: Autor

El resto de configuración para lo que falta de robot se modificó para utilizar un sensor cámara a bordo del servomotor.

Es necesario definir primero los parámetros, el enlace, transmisión y sub-juntura del servomotor. A la junta del servomotor se le llamó servo_joint de tipo revolución, es decir, tiene un eje de revolución, pero tiene límites en su posición. El eje de rotación de este servo motor está desfasado cuatro centímetros del sobre el eje X del cuerpo de robot, se define como padre el elemento base y el hijo es el elemento servo, al eje de revolución es el eje Z, como se utiliza un tipo de junta revolución.

Es necesario definir sus parámetros de amortiguamiento y fricción para permitir el movimiento del ser motor, tanto el amortiguamiento como los

parámetros de fricción deben ser bastante bajos o 0, en este caso se agregó amortiguamiento para reducir las oscilaciones en el estado estacionario.

En el tag `limit effort` hace referencia al torque máximo del motor, `Velocity`, la velocidad máxima que lo harían es sobre sí mismo y `lower` que hace referencia a la posición mínima y máxima de revolución, respectivamente

```
<!-- SERVO MOTOR and CAMERA definition -->
<!-- Servomotor (joint, link and transmission) -->
<joint name="servo_joint" type="revolute">
  <origin xyz="0.04 0 0.04" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="servo_link"/>
  <axis xyz="0 0 1"/>
  <dynamics damping="0.001" friction="0.0"/> <!-- To allow the servo_link movement, both damping and friction parameters must
  be very low or zero. In this case, I add damping to reduce the oscillations in the steady state -->
  <limit effort="1" velocity="1" lower="-${pi/2}" upper="${pi/2}"/> <!-- Limit tag is required due to a revolute type joint
  s used -->
</joint>

<link name="servo_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.04 0.02"/>
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>
```

Figura 4.40 Configuración de parámetros físicos del servomotor
Fuente: Autor

Es muy importante mencionar que debido a que se utilizarán `ros_control` para controlar su posición, es indispensable configurar sus parámetros de inercia, masa y matriz de inercia.

Asimismo, dado que se va a utilizar los controles, es necesario asignar un elemento de transmisión al motor donde un tipo se utiliza transmisión interfaz simple, se indica que la junta para esa transición será `servo_joint`, `hardwareInterface` y `for jointInterface`.

En cuanto al activador utilizado, se utiliza el nombre motor, por poner un ejemplo y en cuanto a la etiqueta de reducción mecánica, puede ser omitida, realmente no se ve un cambio en la circulación con esto se define completamente el motor.

```
<transmission name="tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="servo_joint">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction><!-- This tag does not any change in simulation -->
  </actuator>
</transmission>
```

Figura 4.41 Configuración del motor
Fuente: Autor

Ahora se define la junta y el eslabón para la cámara que estará a bordo del servomotor, tanto el nombre de la voz como la junta para la cámara deben de ser los mismos que se utilizan en el plugin como se explicó, se define la junta para la cámara de tipo fijo en este caso se utiliza como padre el eslabón del servo y no el eslabón del cuerpo principal de robot, de esta manera, la cámara se moverá con el motor. El resto de configuración es similar que en el caso anterior.

```
<!-- The joint and link of the camera. Note: Both link name and joint name must be the same as the ones used in the plugin -->
<joint name="camera_rgb_optical_joint" type="fixed">
  <origin xyz="0.03 0 0.02" rpy="0 0 0"/> <!-- X axis is considered as the optical axis -->
  <parent link="servo_link"/> <!-- Note: The camera link is attached to the servo link -->
  <child link="camera_rgb_frame"/>
</joint>
<link name="camera_rgb_frame">
  <visual>
    <origin xyz="-0.015 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.04 0.02"/>
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>
  <visual>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://my_ugv_description/meshes/max_sonar_ez4.dae" scale="1 1 0.8" />
    </geometry>
  </visual>
```

Figura 4.42 Configuración de la cámara en servomotor
Fuente: Autor

Para lanzar el robot con sensores y servomotor se inicializa el código en una nueva terminal como se muestra a continuación.

```
maragarita@maragarita-VirtualBox:~$ roslaunch my_ugv_description servomotor_diff_robot.launch
... logging to /home/maragarita/.ros/log/cebf4928-115f-11ed-9b63-8dbbb5cd175e/roslaunch-maragarita-VirtualBox-8347.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://maragarita-VirtualBox:46559/

SUMMARY
*****
PARAMETERS
 * /gazebo/enable_ros_network: True
 * /robot_description: <?xml version="1...
 * /robot_state_publisher/publish_frequency: 50.0
 * /robot_state_publisher/tf_prefix:
 * /roscpp: noetic
 * /rosversion: 1.15.14
 * /servo/joint_position_controller/joint: servo_joint
 * /servo/joint_position_controller/pid/d: 0.0
```

Figura 4.43 Inicialización del del robot con sensores y servomotor
Fuente: Autor

La simulación inicia corriendo y se visualiza el robot con sus elementos juntas y plugins.

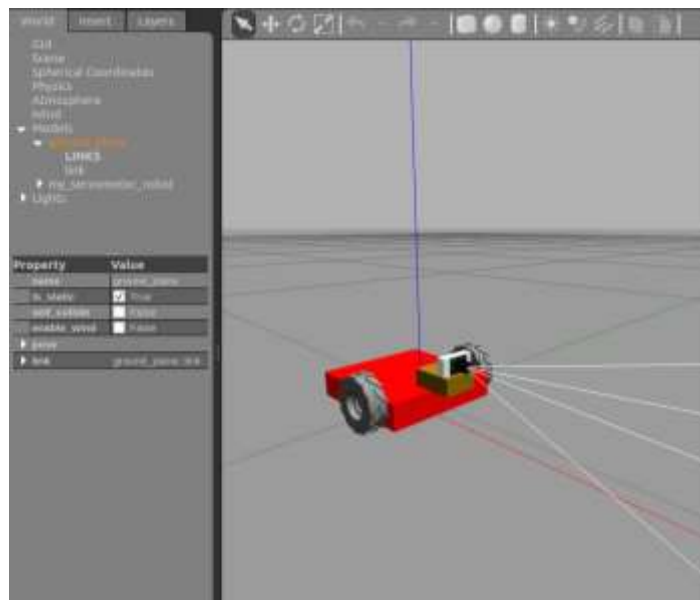


Figura 4.44 Visualización del robot con sensores y servomotor
Fuente: Autor

En cuanto a Rviz, presenta al Robot con los marcos de referencia y la imagen de la Cámara.

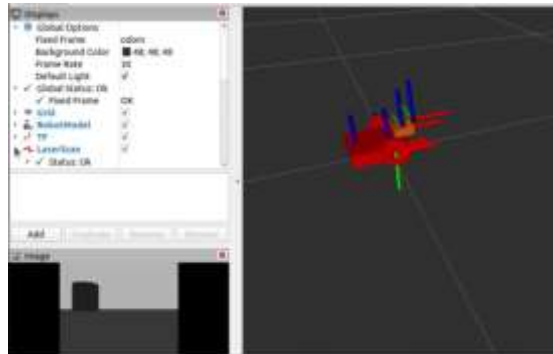


Figura 4.45 Visualización en Rviz con sus marcos de referencias
Fuente: Autor

Es indispensable visualizar los marcos que componen el robot para corroborar la correcta estructura de este. Si genera una lista de tópicos. Y se muestran tópicos precedidos por el servomotor.



Figura 4.46 Tópicos generados por el servomotor
Fuente: Autor

En el tópico servo joint position controller common es donde se publica el valor de consigna de la posición angular para el servomotor. A continuación, como se observa en la imagen se publica una posición deseada para el servomotor



Figura 4.47 Ingreso de posición deseada para el servomotor
Fuente: Autor

El valor de la posición angular de ser motor debe de estar dada en radianes positivo en el sentido antihorario.



Figura 4.48 Resultado del valor ingresado para el servomotor
Fuente: Autor

En el tópic `joint state` se muestran los estados de las juntas donde se está utilizando `ros_control` cuando se muestra la posición, velocidad y torque para el servo motor.

```
maragarita@maragarita-Virtu
/home/maragarita/m... maragarita@maragari...
- servo_joint
position: [0.0000946979596178355]
velocity: [-4.43135412781005e-13]
effort: [0.0]
---
header:
  seq: 4707
  stamp:
    secs: 94
    nsecs: 797800000
  frame_id: ''
name: ''
- servo_joint
position: [0.0000946942552428220]
velocity: [-4.3125904299028013e-13]
effort: [0.0]
---
```

Figura 4.49 Lectura de posición, velocidad y torque del servomotor
Fuente: Autor

4.3.2 Herramienta para ajustar las ganancias del controlador usado por el servomotor

Para sintonizar las ganancias del controlador PID utilizado para controlar la posición del servomotor a bordo del robot móvil diferencial la

herramienta rqt_gui se puede utilizar, mediante el siguiente código se abre la herramienta como se muestra a continuación

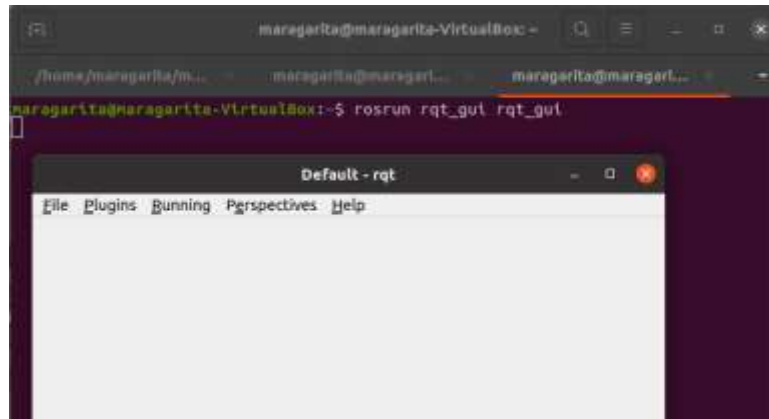


Figura 4.50 Inicialización de la herramienta rqt_gui
Fuente: Autor

En el menú plugin se elige tópicos y se selecciona publicador de mensajes, con esto se va a publicar un valor de consigné al ser un motor como lo realicé usando la terminal.

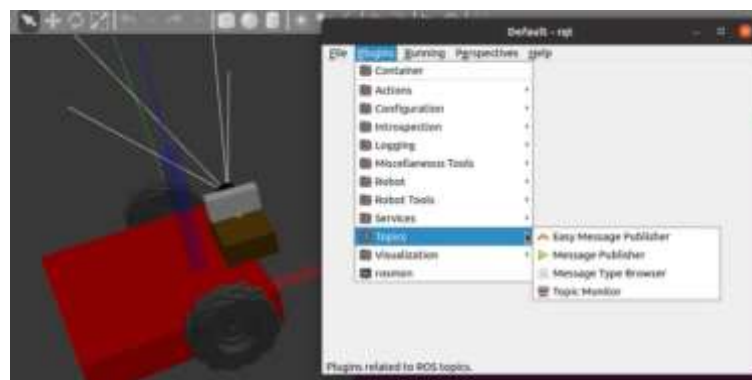


Figura 4.51 Activar la lectura del servomotor
Fuente: Autor

Se da clic en el símbolo de más, ahora, para graficar el desempeño en visualización, se elige Plot. aunque selecciona el tópico servo chain position controller State, se puede reducir la cantidad de variables que se muestran en la gráfica.

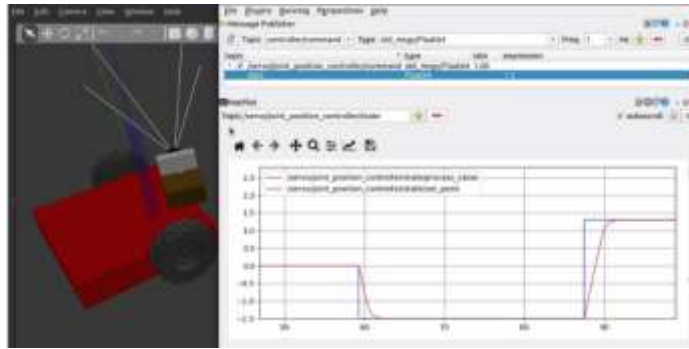


Figura 4.52 Visualización grafica de las lecturas del servomotor
Fuente: Autor

Es importante mencionar que la línea en rojo, que es el valor de la posición angular, converge a su valor deseado utilizando una saturación de velocidad. Dado que ya se sintonizar el controlador, obtengo un error en estado estacionario bastante aceptable, pero si fuera el caso de sintonizar el controlador de manera dinámica, es necesario utilizar la opción dynamic reconfigure

4.4 Lanzamiento de múltiples robots

Para visualizar múltiples robots correctamente en rviz se necesita correr el nodo que publica correctamente el árbol de transformación TF, sin embargo, para lanzar varios robots en gazebo solamente se necesita utilizar el tag group, como se muestra a continuación.

```
<!-- Spawn and use a second robot in Gazebo -->
<arg name="robot_name2" default="my_robot2"/>
<arg name="robot_ns2" default="servo2"/>
<arg name="robot_posture2" default="-x 1.0 -y 0 -Y 0"/><!--If a collision between the robots exists, the second robot will not be spawned-->
```

Figura 4.53 Código para configurar un segundo robot
Fuente: Autor

Se configura el nombre para el otro robot y el namespace para este y la posición inicial, es importante notar que es necesario que no exista una colisión entre los robots, porque de esta manera el segundo robot no se va a

engendrar, para este nuevo robot se utilizó otro archivo de configuración llamado Servo motor Controller 2 con extensión yaml, el cual se localiza en la carpeta llamada config.

```
<rosparam file="$(find my_ugv_description)/config/servomotor_controller2.yaml" command="load"/>
<group ns="$(arg robot_ns2)">
  <param name="robot_description" command="$(find xacro)/xacro -inorder $(find my_ugv_description)/urdf/
  servomotor_diff_robot.urdf.xacro robot_ns:=$(arg robot_ns2)"/>

  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-param robot_description -urdf $(arg robot_posture2) -model
  $(arg robot_name2)"
    respawn="false" output="screen"/>

  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" args="joint_state_controller joint_position_controller"/>
</group>
```

Figura 4.54 Configuración de la localización del segundo robot
Fuente: Autor

En el namespace se configura Servo2, esto debe de ser diferente al namespace utilizado para el robot anterior. El resto del código es igual que el anterior, dado que el robot es el mismo y tiene las mismas configuraciones y los mismos nombres en las juntas, utilizando la etiqueta group_namespace se configura el espacio para el nuevo robot.

```
<!-- Spawn and use a second robot in Gazebo -->
<arg name="robot_name2" default="my_robot2"/>
<arg name="robot_ns2" default="servo2"/>
<arg name="robot_posture2" default="-x 1.0 -y 0 -Y 0"/><!-- If a collision between the robots exists, the second robot will
not be spawned -->

<rosparam file="$(find my_ugv_description)/config/servomotor_controller2.yaml" command="load"/>

<group ns="$(arg robot_ns2)">
  <param name="robot_description" command="$(find xacro)/xacro -inorder $(find my_ugv_description)/urdf/
  servomotor_diff_robot.urdf.xacro robot_ns:=$(arg robot_ns2)"/>

  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-param robot_description -urdf $(arg robot_posture2) -mu
  $(arg robot_name2)"
    respawn="false" output="screen"/>

  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" args="joint_state_controller joint_position_controller"/>
  <!-- Note that the group tag is used, so the ns parameter is not required -->

  <!--node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false"
  output="screen">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" value="" />
  </node>
</group-->

</launch>
```

Figura 4.55 Configuración general de los robots a ejecutar
Fuente: Autor

De esta manera se evita colisionar entre los nombres de los tópicos, el resto de código es el mismo que se utiliza para un solo robot. Es necesario utilizar el nodo para cargar el controlador utiliza el Tag Group el parámetro NS

no se requiere, a diferencia que, en el caso anterior, donde no se utiliza el tag group, si es necesario utilizar el argumento NS para generar un namespace. Finalmente se ejecuta el nodo para publicar los estados del robot, se guarda y lanza este archivo, una vez procesado el comando ya se tienen engendrados dos robots, el primer robot llamado Servo motor robot y el segundo robot llamado Mi robot.

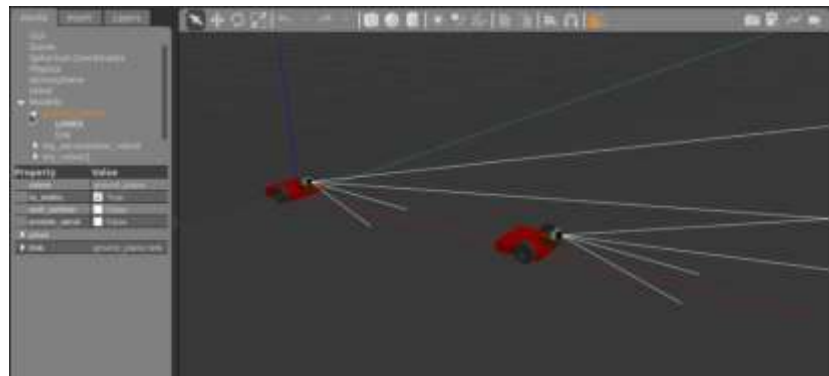


Figura 4.56 Visualización de ambos robots en ejecución
Fuente: Autor

Se genera una lista de tópicos bajo los comandos explicados con anterioridad, se presenta la lista de tópicos pertenecientes a Robot 1 y para el robot 2 se tienen los mismos tópicos, pero utilizando el namespace Servo 2 para el tópico de su cámara, tal como se muestra a continuación

```

/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/robon
/robon
/robon_agg
/servo/joint_position_controller/command
/servo/joint_position_controller/pid/parameter_descriptions
/servo/joint_position_controller/pid/parameter_updates
/servo/joint_position_controller/state
/servo/joint_states
/servo2/camera/parameter_descriptions
/servo2/camera/parameter_updates
/servo2/camera/rgb/camera_info
/servo2/camera/rgb/image_raw
/servo2/camera/rgb/image_raw/compressed
/servo2/camera/rgb/image_raw/compressed/parameter_descriptions
/servo2/camera/rgb/image_raw/compressed/parameter_updates
/servo2/camera/rgb/image_raw/compressedDeath

```

Figura 4.57 Tópicos generados por el robot 1 y 2
Fuente: Autor

CAPITULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- Este proyecto respeta con su propósito originalmente propuesto en sus objetivos. Elaborar un manual para simulaciones de un robot móvil diferencial mediante el uso del Sistema Operativo Robótico, Gazebo y complementos para la utilización de los estudiantes de la carrera Ingeniería Electrónica en Control y Automatismo con pautas explicativas del código para futuras modificaciones intencionada de los estudiantes en su proceso de adquisición de conocimientos.
- Se diseñó tres modelos de simulación virtual con las herramientas y metodología explicadas en el desarrollo de este proyecto en primera instancia se realizó la simulación de un robot móvil diferencial sencillo, con estos conocimientos adquiridos se plantea una segunda simulación en este caso al robot realizado inicialmente se le incorporan un sensor laser y una cámara monocular, para concluir en una tercera etapa donde al robot se le agrega un servomotor que servirá para hacer girar la cámara previamente incorporada, cada una de estas tres practicas se le describe el código, sus especificaciones y funcionalidades para que puedan ser modificadas a requerimientos específicos.
- Se desarrolló una guía teórica y práctica para aplicaciones del "Robot Diferencial". Donde se sientan las bases para el desarrollo de varios robots en el aula, en esta implementación de guías teóricas y prácticas para la simulación de robots, con trayectorias definidas para cada modelo y los métodos convencionales como los controladores PID y sus derivados agregan comportamiento a las respuestas de este método.

5.2 Recomendaciones

Si bien el simulador planteado en esta documentación cuenta con una configuración compleja cuya curva de aprendizaje es “empina” generalmente y principalmente en algún modelo robótico que no se encuentre incluido en la paquetería por defecto, es recomendable incentivar a los estudiantes a aprender de forma activa estas herramientas puesto que permiten pasar los desarrollos teóricos a robótica física con esfuerzo mínimos ya que cuenta con simulaciones suficientemente realistas que siguen un modelo físico muy precisos que incluyen fuerzas de aceleración, fricción, viento gravedad entre otras variables destacables.

En cuanto a la simulación, se ha visto cómo crear tres modelos distintos basados en un mismo robot con una configuración por defecto de un mundo vacío se invita al usuario a modificar estos mundos con diversas herramientas libres disponibles en la web y posteriormente insertar estos mundos dentro del simulador Gazebo. También se encuentran paquetes donde se han insertado varios robots dentro del mundo especificado en el simulador Gazebo, el cual ayudará a desarrollar futuros algoritmos para crear comportamientos multirobots.

Puede usar varias herramientas ROS utilizadas en esta documentación para trabajar en el diseño y desarrollo de comportamientos más complejos con el robot, estos comportamientos se pueden desarrollar utilizando Rospay, la biblioteca de cliente ROS de Python y más para escalar la complejidad y autonomía por ejemplo para el análisis de las nubes de puntos estas actividades generan más información y conocimientos e incentivan una conducta autodidacta para el alumnado.

BIBLIOGRAFÍA

- Antón, A. O. (2019). *Universidad Politécnica de Madrid*. Obtenido de Algoritmo de control de un robot omnidireccional de tres ruedas: https://oa.upm.es/58728/1/TFM_Alberto_Ortega_Anton.pdf
- Barrientos, V. R., García, J. R., & Silva, R. (2007). Robots Móviles: Evolución y Estado del Arte. *Polibits*, (35),12-17.
- Berrio, J. (2008). *Navegación planificada de un robot móvil*. Obtenido de <https://red.uao.edu.co/bitstream/handle/10614/6373/T04375.pdf?sequence=1&isAllowed=y>
- Camino, S., & Rojas, M. (2019). *Repositorio carrera de ingeniería en electrónica*,. obtenido de “sistema aéreo terrestre basado en robótica cooperativa para detección de minas antipersonales: <http://repositorio.espe.edu.ec/bitstream/21000/21589/1/T-ESPE-040892.pdf>
- Cantillo Molina, S. J., & Berrio Pérez, J. S. (2014). Planned navigation of a Lego NXT robot. *2014 9th Computing Colombian Conference (9CCC)*. <https://doi.org/10.1109/columbiancc.2014.6955348>
- Cárdenas, C., & Méndez, A. (2020). *Documentación de aplicaciones robóticas en ROS*. Obtenido de Repositorio Institucional: <http://repository.unipiloto.edu.co/handle/20.500.12277/6889>
- Cardona, J., Leal, J., & Ramírez, J. (2018). Modelado Matemático de la Posición del Centro de Masa de un Robot de Tracción Diferencial. Un Enfoque desde la Mecánica Lagrangiana. *Información tecnológica*, 29(6), 307-320. Obtenido de <https://dx.doi.org/10.4067/S0718-07642018000600307>

- Domínguez, J. T. (2019). *Modelado 3D de un robot móvil en ROS y Gazebo*. Sevilla: Dpto. Ingeniería de Sistemas y Automática Escuela Técnica Superior de Ingeniería.
- Fernández, R., Aracil, R., & Armada, M. (2015). Control de Tracción en Robots Mviles con Ruedas. *Revista Iberoamericana de Automática e Informática industrial*, 393–405.
- Hernández Millán, G., Ríos Gonzales, L., & Bueno López, M. (2016). Implementación de un controlador de posición y movimiento de un robot móvil diferencial. *Tecnura*, 20 (48), 123-136. doi:<https://doi.org/10.14483/udistrital.jour.tecnura.2016.2.a09>
- Roberto, S., Sierra-García, J., & Santos, M. (2022). Modelado de unAGV híbrido triciclo-diferencia. *Revista Iberoamericana de Automática e Informática Industrial*, 19, 84-95. doi:<https://doi.org/10.4995/riai.2021.14622>
- ROS.org. (2019). *ROS wiki*. Obtenido de Concepts: <http://wiki.ros.org/ROS/Concepts>
- ROS.org. (2022). *ROS wiki*. Obtenido de Guia ROS: https://docs-ros-org.translate.goog/en/foxy/?_x_tr_sch=http&_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sc
- Mazzari, V. (2022). *ROS: Définition du Robot Operating System, middleware pour la robotique*. Génération Robots. <https://www.generationrobots.com/fr/ros-robot-operating-system-3/>
- Sierra-García, J., & Santos, M. (2020). Mechatronic modelling of industrial AGVs: A complex system architecture. *Complexity*, 16.
- Ortiz, L. F., Vásquez, M., & Muñoz, N. D. (2018). Navegación de robots móviles en entornos condiscinuidades: una revisión. *Politécnica*, 103-115.

ANEXOS

Diseño simple del robot diferencial

```
<?xml version="1.0"?>
<robot name="diff_robot" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find
my_ugv_description)/urdf/simple_diff_robot_gazebo.xacro"/>

  <link name="base_footprint"/>
  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0 0 0.01" rpy="0 0 0"/>
  </joint>

  <link name="base_link"> <!-- Important: Every visual, collision
and inertial tag has its own origin -->
    <visual>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.14 0.14 0.03" />
      </geometry>
    </visual>

    <collision>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.140 0.140 0.03"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0.015" rpy="0 0 0"/> <!-- Since a plugin
is used, an offset along its X axis can affect the robot behavior
-->
      <mass value="8.25e-01"/>
      <inertia ixx="1.4e-03" ixy="0" ixz="0"
        iyy="1.4e-03" iyz="0"
        izz="2.695e-03" />
    </inertial>
  </link>

  <!-- Wheels definition (joints and links) -->
  <joint name="wheel_left_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_left_link"/>
    <origin xyz="0 0.08 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>

  <link name="wheel_left_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
    </visual>
  </link>
</robot>
```

```

    <geometry>
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
  </collision>

  <inertial>
    <origin xyz="0 0 0" />
    <mass value="3e-02" />
    <inertia ixx="9e-06" ixy="0" ixz="0"
             iyy="9e-06" iyz="0"
             izz="1.6e-05" />
  </inertial>
</link>

<joint name="wheel_right_joint" type="continuous">
  <parent link="base_link"/>
  <child link="wheel_right_link"/>
  <origin xyz="0 -0.08 0.023" rpy="-1.57 0 0"/>
  <axis xyz="0 0 1"/>
</joint>

<link name="wheel_right_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder length="0.018" radius="0.033"/>
    </geometry>
  </collision>

  <inertial>
    <origin xyz="0 0 0" />
    <mass value="3e-02" />
    <inertia ixx="9e-06" ixy="0" ixz="0"
             iyy="9e-06" iyz="0"
             izz="1.6e-05" />
  </inertial>
</link>

<!-- Caster wheel definition (joint and link) -->
<joint name="caster_joint" type="fixed">
  <parent link="base_link"/>
  <child link="caster_link"/>

```

```

    <origin xyz="-0.08 0 0" rpy="0 0 0"/>
  </joint>

  <link name="caster_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <sphere radius="0.01"/>
      </geometry>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <sphere radius="0.01"/>
      </geometry>
    </collision>
    <!-- Since the caster wheel is only a point of contact,
inertial tag can be ommited -->
  </link>

</robot>

<?xml version="1.0"?>
<robot name="diff_robot" xmlns:xacro="http://ros.org/wiki/xacro">

  <gazebo reference="base_link">
    <material>Gazebo/Red</material> <!-- Color of the base link
in Gazebo -->
  </gazebo>

  <gazebo reference="wheel_left_link"> <!-- Physical properties
setup -->
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>500000.0</kp>
    <kd>10.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl>
    <material>Gazebo/Black</material>
  </gazebo>

  <gazebo reference="wheel_right_link">
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>500000.0</kp>
    <kd>10.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl>
    <material>Gazebo/Black</material>
  </gazebo>

```

```

    <gazebo reference="caster_link">
      <mu1>0.1</mu1>
      <mu2>0.1</mu2>
      <kp>1000000.0</kp>
      <kd>100.0</kd>
      <minDepth>0.001</minDepth>
      <maxVel>1.0</maxVel>
      <material>Gazebo/White</material>
    </gazebo>

    <gazebo>
      <plugin name="diff_controller"
filename="libgazebo_ros_diff_drive.so"> <!-- Plugin for a
differential mobile robot -->
        <commandTopic>cmd_vel</commandTopic>
        <odometryTopic>odom</odometryTopic>
        <odometryFrame>odom</odometryFrame>
        <odometrySource>world</odometrySource> <!-- world or
encoder -->
        <publishOdomTF>true</publishOdomTF> <!-- Publish the fixed
frame called odom, default = true-->
        <robotBaseFrame>base_footprint</robotBaseFrame>
        <publishWheelTF>true</publishWheelTF> <!-- default = false.
To show the spinning wheels attached to the robot. Thus,
joint_state_publisher node must be omitted -->
        <publishTf>true</publishTf> <!-- default = true-->
        <publishWheelJointState>>false</publishWheelJointState>
        <legacyMode>>false</legacyMode>
        <updateRate>30</updateRate>
        <leftJoint>wheel_left_joint</leftJoint>
        <rightJoint>wheel_right_joint</rightJoint>
        <wheelSeparation>0.160</wheelSeparation>
        <wheelDiameter>0.066</wheelDiameter>
        <wheelAcceleration>1</wheelAcceleration>
        <wheelTorque>10</wheelTorque>
        <rosDebugLevel>na</rosDebugLevel>
      </plugin>
    </gazebo>

</robot>

```

Robot diferencial con sensor láser y cámara

```
<?xml version="1.0"?>
<robot name="sensors_diff_robot"
xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find
my_ugv_description)/urdf/sensors_diff_robot_gazebo.xacro"/>

  <!-- Robot Gazebo Plugins -->
  <xacro:my_robot_sensors />

  <link name="base_footprint"/>

  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0 0 0.01" rpy="0 0 0"/>
  </joint>
  <link name="base_link">
    <visual>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.14 0.14 0.03" />
      </geometry>
    </visual>
    <collision>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.140 0.140 0.03"/>
      </geometry>
    </collision>
    <inertial>
      <origin xyz="0 0 0.015" rpy="0 0 0"/> <!-- Since a plugin
is used, an offset along its X axis can affect the robot
behavior -->
      <mass value="8.25e-01"/>
      <inertia ixx="1.4e-03" ixy="0" ixz="0"
        iyy="1.4e-03" iyz="0"
        izz="2.695e-03" />
    </inertial>
  </link>

  <!-- Wheels definition (joints and links) -->
  <joint name="wheel_left_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_left_link"/>
    <origin xyz="0 0.08 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>
  <link name="wheel_left_link">
    <visual>
      <origin xyz="0 0 0" rpy="1.57 0 0"/>
      <geometry>
        <!--cylinder length="0.018" radius="0.033"/-->

```

```

        <mesh
filename="package://my_ugv_description/meshes/husky_wheel.stl"
scale="0.18 0.18 0.18"/>
    </geometry>
</visual>
<collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
        <cylinder length="0.018" radius="0.033"/>
    </geometry>
</collision>
<inertial>
    <origin xyz="0 0 0" />
    <mass value="3e-02" />
    <inertia ixx="9e-06" ixy="0" ixz="0"
            iyy="9e-06" iyz="0"
            izz="1.6e-05" />
</inertial>
</link>

<joint name="wheel_right_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_right_link"/>
    <origin xyz="0 -0.08 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
</joint>
<link name="wheel_right_link">
    <visual>
        <origin xyz="0 0 0" rpy="1.57 0 0"/>
        <geometry>
            <!--cylinder length="0.018" radius="0.033"/-->
            <mesh
filename="package://my_ugv_description/meshes/husky_wheel.stl"
scale="0.18 0.18 0.18"/>
                </geometry>
            </visual>
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <cylinder length="0.018" radius="0.033"/>
                </geometry>
            </collision>
            <inertial>
                <origin xyz="0 0 0" />
                <mass value="3e-02" />
                <inertia ixx="9e-06" ixy="0" ixz="0"
                        iyy="9e-06" iyz="0"
                        izz="1.6e-05" />
            </inertial>
        </link>

<!-- Caster wheel definition (joint and link) -->
<joint name="caster_joint" type="fixed">
    <parent link="base_link"/>
    <child link="caster_link"/>
    <origin xyz="-0.08 0 0" rpy="0 0 0"/>
</joint>

```



```

<link name="caster_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.01"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.01"/>
    </geometry>
  </collision>
  <!-- Since the caster wheel is only a point of contact,
inertial tag can be ommited -->
  <!--inertial>
    <origin xyz="-0.08 0 0" />
    <mass value="5e-02" />
    <inertia ixx="2e-06" ixy="0" ixz="0" iyy="2e-06" iyz="0"
izz="2e-06" />
  </inertial-->
</link>

<!-- SENSORS definition -->
<!-- LIDAR -->
<joint name="scan_joint" type="fixed"> <!-- The scan_joint must
be high enough to avoid possible interferences -->
  <parent link="base_link"/>
  <child link="base_scan"/>
  <origin xyz="-0.07 0 ${0.03+0.06}" rpy="0 0 0"/>
</joint>
<link name="base_scan">
  <visual>
    <origin xyz="0 0 -0.015" rpy="0 0 0"/>
    <geometry>
      <mesh
filename="package://my_ugv_description/meshes/hokuyo.dae"
scale="1 1 1.3" />
      <!--cylinder radius="0.035" length="0.1" /-->
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 -0.005" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="0.035" length="0.11"/>
    </geometry>
  </collision>
  <!-- Since a sensor is used, the inertial tag can be ommited
-->
</link>

<!-- The joint and link of the camera. Note: Both link name and
joint name must be the same as the ones used in the plugin -->
<joint name="camera_rgb_optical_joint" type="fixed">

```

```

    <origin xyz="0.03 0 0.04" rpy="0 0 0"/> <!-- X axis is
considered as the optical axis -->
    <parent link="base_link"/>
    <child link="camera_rgb_frame"/>
</joint>
<link name="camera_rgb_frame">
  <visual>
    <origin xyz="-0.015 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.04 0.02"/>
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>
  <visual>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh
filename="package://my_ugv_description/meshes/max_sonar_ez4.dae"
scale="1 1 0.8" />
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>

  <collision>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.03 0.04 0.02"/>
    </geometry>
  </collision>
  <!-- Since a sensor is used, the inertial tag can be omitted
-->
</link>

</robot>

```

```

<?xml version="1.0"?>
<robot name="sensors_diff_robot"
xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:arg name="laser_visual" default="true"/> <!-- With true,
blue rays are shown in Gazebo-->
  <xacro:arg name="camera_visual" default="false"/> <!-- With
true, what the camera sees is shown en Gazebo-->

```

```

  <material name="orange"> <!-- Custom color (for RViz) -->
    <color rgba="{255/255} {108/255} {10/255} 1.0"/>
  </material>

```

```

  <gazebo reference="base_link">
    <material>Gazebo/Red</material> <!-- Color of the base link
in Gazebo -->
  </gazebo>

```

```

  <gazebo reference="wheel_left_link"> <!-- Physical properties
setup -->

```

```

    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>500000.0</kp>
    <kd>10.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl>
    <material>Gazebo/Grey</material>
  </gazebo>

```

```

  <gazebo reference="wheel_right_link">
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>500000.0</kp>
    <kd>10.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl>
    <material>Gazebo/Grey</material>
  </gazebo>

```

```

  <gazebo reference="caster_link">
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>1000000.0</kp>
    <kd>100.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>1.0</maxVel>
    <material>Gazebo/White</material>
  </gazebo>

```

```

  <gazebo>
    <plugin name="diff_controller"
filename="libgazebo_ros_diff_drive.so"> <!-- Plugin for a
differential mobile robot -->
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <odometrySource>world</odometrySource> <!-- world or
encoder -->
      <publishOdomTF>true</publishOdomTF> <!-- Publish the fixed
frame called odom, default = true-->
      <robotBaseFrame>base_footprint</robotBaseFrame>
      <publishWheelTF>true</publishWheelTF> <!-- default = false.
To show the spinning wheels attached to the robot. Thus,
joint_state_publisher node must be omitted -->
      <publishTf>true</publishTf> <!-- default = true-->
      <publishWheelJointState>>false</publishWheelJointState>
      <legacyMode>>false</legacyMode>
      <updateRate>30</updateRate>
      <leftJoint>wheel_left_joint</leftJoint>
      <rightJoint>wheel_right_joint</rightJoint>
      <wheelSeparation>0.160</wheelSeparation>
    </plugin>
  </gazebo>

```

```

    <wheelDiameter>0.066</wheelDiameter>
    <wheelAcceleration>1</wheelAcceleration>
    <wheelTorque>10</wheelTorque>
    <rosDebugLevel>na</rosDebugLevel>
  </plugin>
</gazebo>

  <!-- Only the Hokuyo mesh is used. The used parameters are
given by:
https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix\_
lds\_01/ -->
  <!-- My robot Sensor Plugins -->
  <xacro:macro name="my_robot_sensors">
    <gazebo reference="base_scan">
      <sensor type="ray" name="hokuyo_LIDAR_sensor">
        <pose>0 0 0 0 0 0</pose>
        <visualize>$(arg laser_visual)</visualize>
        <update_rate>5</update_rate>
        <ray>
          <scan>
            <horizontal>
              <samples>360</samples>
              <resolution>1</resolution><!-- deg-->
              <min_angle>0.0</min_angle>
              <max_angle>6.28319</max_angle>
            </horizontal>
          </scan>
          <range>
            <min>0.120</min>
            <max>3.5</max>
            <resolution>0.015</resolution>
          </range>
          <noise>
            <type>gaussian</type>
            <mean>0.0</mean>
            <stddev>0.01</stddev>
          </noise>
        </ray>
        <plugin name="laser_scan_controller"
filename="libgazebo_ros_laser.so"> <!-- Plugin for a LIDAR sensor
-->
          <topicName>scan</topicName>
          <frameName>base_scan</frameName>
        </plugin>
      </sensor>
    </gazebo>

    <!-- A Raspberry Pi Camera v2.1 is simulated. Link :
https://www.raspberrypi.org/documentation/hardware/camera/-->
    <gazebo reference="camera_rgb_frame">
      <sensor type="camera" name="Pi_camera">
        <always_on>true</always_on>
        <visualize>$(arg camera_visual)</visualize>
        <camera>
          <horizontal_fov>1.085595</horizontal_fov><!--rad-->
          <image>
            <width>640</width>

```

```

        <height>480</height>
        <format>R8G8B8</format>
    </image>
    <clip>
        <near>0.03</near>
        <far>100</far>
    </clip>
</camera>
    <plugin name="camera_controller"
filename="libgazebo_ros_camera.so"> <!-- Plugin for a monocular
camera -->
        <alwaysOn>true</alwaysOn>
        <updateRate>30.0</updateRate>
        <cameraName>camera</cameraName>
        <frameName>camera_rgb_optical_joint</frameName>
        <imageTopicName>rgb/image_raw</imageTopicName>
        <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
    >
        <hackBaseline>0.07</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
        <distortionK3>0.0</distortionK3>
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
    </plugin>
</sensor>
</gazebo>
</xacro:macro>

</robot>

```

Robot diferencial con servomotor

```
<?xml version="1.0"?>
<robot name="servomotor_diff_robot"
xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:arg name="pi" default="3.14159"/>
  <xacro:include filename="$(find
my_ugv_description)/urdf/servomotor_diff_robot_gazebo.xacro"/>

  <!-- Robot Gazebo Controller -->
  <xacro:servomotor_controller />

  <link name="base_footprint"/>

  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0 0 0.01" rpy="0 0 0"/>
  </joint>
  <link name="base_link">
    <visual>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.14 0.14 0.03" />
      </geometry>
    </visual>
    <collision>
      <origin xyz="-0.03 0 0.015" rpy="0 0 0"/>
      <geometry>
        <box size="0.140 0.140 0.03"/>
      </geometry>
    </collision>
    <inertial>
      <origin xyz="0 0 0.015" rpy="0 0 0"/> <!-- Note: A plugin
is used to move the robot. To stabilize the mobile robot due to
the servo_link has inertia, a larger mass and inertia are used -
->
      <mass value="{5*8.25e-01}"/>
      <inertia ixx="{5*1.4e-03}" ixy="0" ixz="0"
        iyy="{5*1.4e-03}" iyz="0"
        izz="{5*2.695e-03}" />
    </inertial>
  </link>

  <!-- Wheels definition (joints and links) -->
  <joint name="wheel_left_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_left_link"/>
    <origin xyz="0 0.08 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>
  <link name="wheel_left_link">
    <visual>
      <origin xyz="0 0 0" rpy="1.57 0 0"/>
      <geometry>
        <!--cylinder length="0.018" radius="0.033"/-->

```

```

        <mesh
filename="package://my_ugv_description/meshes/husky_wheel.stl"
scale="0.18 0.18 0.18"/>
    </geometry>
</visual>
<collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
        <cylinder length="0.018" radius="0.033"/>
    </geometry>
</collision>
<inertial>
    <origin xyz="0 0 0" />
    <mass value="3e-02" />
    <inertia ixx="9e-06" ixy="0" ixz="0"
            iyy="9e-06" iyz="0"
            izz="1.6e-05" />
</inertial>
</link>

<joint name="wheel_right_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_right_link"/>
    <origin xyz="0 -0.08 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
</joint>
<link name="wheel_right_link">
    <visual>
        <origin xyz="0 0 0" rpy="1.57 0 0"/>
        <geometry>
            <!--cylinder length="0.018" radius="0.033"/-->
            <mesh
filename="package://my_ugv_description/meshes/husky_wheel.stl"
scale="0.18 0.18 0.18"/>
                </geometry>
            </visual>
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <cylinder length="0.018" radius="0.033"/>
                </geometry>
            </collision>
            <inertial>
                <origin xyz="0 0 0" />
                <mass value="3e-02" />
                <inertia ixx="9e-06" ixy="0" ixz="0"
                        iyy="9e-06" iyz="0"
                        izz="1.6e-05" />
            </inertial>
        </link>

<!-- Caster wheel definition (joint and link) -->
<joint name="caster_joint" type="fixed">
    <parent link="base_link"/>
    <child link="caster_link"/>
    <origin xyz="-0.08 0 0" rpy="0 0 0"/>
</joint>

```

```

<link name="caster_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.01"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.01"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="-0.08 0 0" />
    <mass value="5e-02" />
    <inertia ixx="2e-06" ixy="0" ixz="0" iyy="2e-06" iyz="0"
izz="2e-06" />
  </inertial>
</link>

<!-- SERVOMOTOR and CAMERA definition -->
<!-- Servomotor (joint, link and transmission) -->
<joint name="servo_joint" type="revolute">
  <origin xyz="0.04 0 0.04" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="servo_link"/>
  <axis xyz="0 0 1"/>
  <dynamics damping="0.001" friction="0.0"/> <!-- To allow the
servo_link movement, both damping and friction parameters must be
very low or zero. In this case, I add damping to reduce the
oscillations in the steady state -->
  <limit effort="1" velocity="1" lower="-${-pi/2}"
upper="${pi/2}"/> <!-- Limit tag is required due to a revolute
type joint is used -->
</joint>

<link name="servo_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.04 0.02"/>
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.04 0.02"/>
    </geometry>
  </collision>

  <inertial> <!-- The inertial tag must be correctly set up to
obtain the desired link behavior -->
    <origin xyz="0 0 0" />

```



```

    <mass value="5e-02" />
    <inertia ixx="8.333e-06" ixy="0" ixz="0" iyy="1.2083e-05"
    iyz="0" izz="1.7083e-05" />
  </inertial>
</link>

<transmission name="tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="servo_joint">
    <hardwareInterface>hardware_interface/EffortJointInterface<
/hardwareInterface>
  </joint>
  <actuator name="motor">
    <hardwareInterface>hardware_interface/EffortJointInterface<
/hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction><!-- This tag
does not any change in simulation -->
  </actuator>
</transmission>

<!-- The joint and link of the camera. Note: Both link name and
joint name must be the same as the ones used in the plugin -->
<joint name="camera_rgb_optical_joint" type="fixed">
  <origin xyz="0.03 0 0.02" rpy="0 0 0"/> <!-- X axis is
considered as the optical axis -->
  <parent link="servo_link"/> <!-- Note: The camera link is
attached to the servo link -->
  <child link="camera_rgb_frame"/>
</joint>
<link name="camera_rgb_frame">
  <visual>
    <origin xyz="-0.015 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.04 0.02"/>
    </geometry>
    <material name="orange"/> <!--Color in RViz-->
  </visual>
  <visual>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh
filename="package://my_ugv_description/meshes/max_sonar_ez4.dae"
scale="1 1 0.8" />
    </geometry>
  </visual>

  <collision>
    <origin xyz="-0.01 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.03 0.04 0.02"/>
    </geometry>
  </collision>
  <!-- Since a sensor is used, the inertial tag can be omitted
-->
</link>

```

```

</robot>

<?xml version="1.0"?>
<robot name="servomotor_diff_robot"
xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:arg name="camera_visual" default="true"/> <!-- With
true, what the camera sees is shown en Gazebo-->
  <xacro:arg name="robot_ns" default="servo"/> <!-- The namespace
must coincide with the one given in "*.yaml" and "*.launch" files
-->

  <material name="orange"> <!-- Custom color (for RViz) -->
    <color rgba="{255/255} {108/255} {10/255} 1.0"/>
  </material>

  <gazebo reference="base_link">
    <material>Gazebo/Red</material> <!-- Color of the base link
in Gazebo -->
  </gazebo>

  <gazebo reference="wheel_left_link"> <!-- Physical properties
setup -->
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>500000.0</kp>
    <kd>10.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl>
    <material>Gazebo/Grey</material>
  </gazebo>

  <gazebo reference="wheel_right_link">
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>500000.0</kp>
    <kd>10.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl>
    <material>Gazebo/Grey</material>
  </gazebo>

  <gazebo reference="caster_link">
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <kp>1000000.0</kp>
    <kd>100.0</kd>
    <minDepth>0.001</minDepth>
    <maxVel>1.0</maxVel>
    <material>Gazebo/White</material>
  </gazebo>

  <gazebo reference="servo_link">
    <!--mu1>0.1</mu1>
    <mu2>0.1</mu2>

```

```

    <kp>50</kp>
    <kd>10</kd>
    <minDepth>0.001</minDepth>
    <maxVel>0.1</maxVel>
    <fdirl>1 0 0</fdirl-->
    <material>Gazebo/Gold</material>
  </gazebo>

```

```

  <gazebo>
    <plugin name="diff_controller"
      filename="libgazebo_ros_diff_drive.so"> <!-- Plugin for a
      differential mobile robot -->
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <odometrySource>world</odometrySource> <!-- world or
      encoder -->
      <publishOdomTF>true</publishOdomTF> <!-- Publish the fixed
      frame called odom, default = true-->
      <robotBaseFrame>base_footprint</robotBaseFrame>
      <publishWheelTF>true</publishWheelTF> <!-- default = false.
      To show the spinning wheels attached to the robot in Rviz -->
      <publishTf>true</publishTf> <!-- default = true-->
      <publishWheelJointState>true</publishWheelJointState>
      <legacyMode>>false</legacyMode>
      <updateRate>30</updateRate>
      <leftJoint>wheel_left_joint</leftJoint>
      <rightJoint>wheel_right_joint</rightJoint>
      <wheelSeparation>0.160</wheelSeparation>
      <wheelDiameter>0.066</wheelDiameter>
      <wheelAcceleration>1</wheelAcceleration>
      <wheelTorque>10</wheelTorque>
      <rosDebugLevel>na</rosDebugLevel>
    </plugin>
  </gazebo>

```

```

  <!-- A Raspberry Pi Camera v2.1 is simulated. Link :
  https://www.raspberrypi.org/documentation/hardware/camera/-->
  <gazebo reference="camera_rgb_frame">
    <sensor type="camera" name="Pi_camera">
      <always_on>true</always_on>
      <visualize>$(arg camera_visual)</visualize>
      <camera>
        <horizontal_fov>1.085595</horizontal_fov>
        <image>
          <width>640</width>
          <height>480</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.03</near>
          <far>100</far>
        </clip>
      </camera>

```

```

    <plugin name="camera_controller"
filename="libgazebo_ros_camera.so"> <!-- Plugin for a monocular
camera -->
    <alwaysOn>true</alwaysOn>
    <updateRate>30.0</updateRate>
    <cameraName>camera</cameraName>
    <frameName>camera_rgb_optical_joint</frameName>
    <imageTopicName>rgb/image_raw</imageTopicName>
    <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
>
    <hackBaseline>0.07</hackBaseline>
    <distortionK1>0.0</distortionK1>
    <distortionK2>0.0</distortionK2>
    <distortionK3>0.0</distortionK3>
    <distortionT1>0.0</distortionT1>
    <distortionT2>0.0</distortionT2>
    </plugin>
</sensor>
</gazebo>

<!-- ros_control plugin -->
<xacro:macro name="servomotor_controller">
    <gazebo>
        <plugin name="gazebo_ros_control"
filename="libgazebo_ros_control.so">
            <robotNamespace>$(arg robot_ns)</robotNamespace>
            <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSi
mType>
            <legacyModeNS>true</legacyModeNS>
        </plugin>
    </gazebo>
</xacro:macro>

</robot>

```

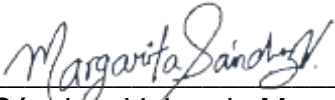
DECLARACIÓN Y AUTORIZACIÓN

Yo, Sánchez Valverde Margarita del Rocio, con C.C: # 0950544312 autora del trabajo de titulación: **Estudio del Sistema Operativo Robótico y Gazebo. Diseño de una guía de ejercicios**, previo a la obtención del título de **Ingeniera Electrónica en Control y Automatismo** en la Universidad Católica de Santiago de Guayaquil.

1.- Declaro tener pleno conocimiento de la obligación que tienen las instituciones de educación superior, de conformidad con el Artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de titulación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la SENESCYT a tener una copia del referido trabajo de titulación, con el propósito de generar un repositorio que democratice la información, respetando las políticas de propiedad intelectual vigentes.

Guayaquil, 15 de septiembre del 2022

f. 
Nombre: Sánchez Valverde Margarita del Rocio
C.C: 0950544312



REPOSITORIO NACIONAL EN CIENCIA Y TECNOLOGÍA		
FICHA DE REGISTRO DE TESIS/TRABAJO DE TITULACIÓN		
TÍTULO Y SUBTÍTULO:	Estudio del Sistema Operativo Robótico y Gazebo. Diseño de una guía de ejercicios.	
AUTOR (ES)	Sánchez Valverde, Margarita del Rocío	
REVISOR(ES)/TUTOR(ES) (apellidos/nombres):	Philco Asqui, Luis Orlando	
INSTITUCIÓN:	Universidad Católica de Santiago de Guayaquil	
FACULTAD:	Facultad de educación técnica para el Desarrollo	
CARRERA:	Ingeniería Electrónica en Control y Automatismo	
TÍTULO OBTENIDO:	Ingeniera Electrónica en Control y Automatismo	
FECHA DE PUBLICACIÓN:	15 de septiembre del 2022	No. DE PÁGINAS: 87
ÁREAS TEMÁTICAS:	Robótica, Mecatrónica	
PALABRAS CLAVES:	Robots, Móviles, ROS, Simulación, Gazebo, Algoritmos	
RESUMEN:		
<p>El presente trabajo se desarrolla para la obtención del título de ingeniería electrónica en control y automatismo, el abordaje que se toma en consideración es la creación de guías y pautas para el aprendizaje de robótica general mediante el cual los estudiantes de la Facultad Técnica para el Desarrollo de la Universidad Católica Santiago de Guayaquil adquieran estos conocimientos a través del desarrollo de simulaciones con distintos escenarios que se presentan dentro de la robótica móvil terrestre para ello se busca utilizar las herramientas presentes en el software Gazebo y el control de este por medio de los comandos de ROS. Para lo cual en se emplea la recolección de información referente al funcionamiento de las tecnologías anteriormente mencionadas y relevante al proceso de enseñanza-aprendizaje, además de toda aquella información fundamental al aspecto técnico para proceder a realizar y detallar a través de un manual de prácticas con el simulador.</p>		
ADJUNTO PDF:	<input checked="" type="checkbox"/> SI	<input type="checkbox"/> NO
CONTACTO CON AUTOR/ES:	Teléfono: 0982691946	E-mail: margarita.sanchez01@cu.ucsg.edu.ec / rmargarita1998@gmail.com
CONTACTO CON LA INSTITUCIÓN:	Nombre: Vélez Tacuri, Efraín Oliverio	
COORDINADOR DEL PROCESO DE UTE	Teléfono: (04) 2 206957 ext.5555	
	E-mail: efrain.velez@cu.ucsg.edu.ec / ute@cu.ucsg.edu.ec	
SECCIÓN PARA USO DE BIBLIOTECA		
Nº. DE REGISTRO (en base a datos):		
Nº. DE CLASIFICACIÓN:		
DIRECCIÓN URL (tesis en la web):		