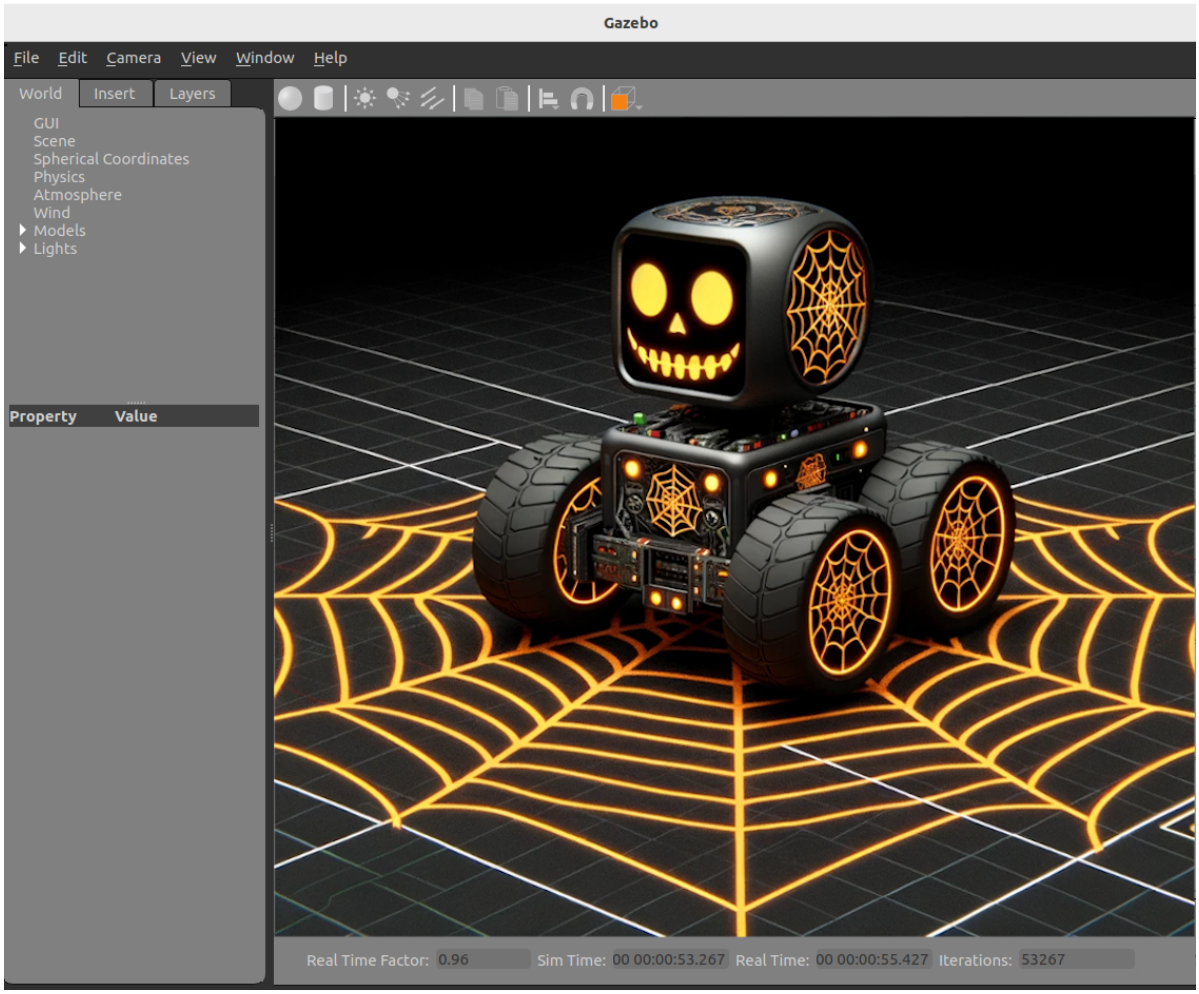
Simulación y Control de Robots con ROS 2 y Gazebo



Eugenio Ivorra

En esta práctica, se pretende explorar las técnicas de simulación robótica con ROS 2 y Gazebo, la construcción y personalización de modelos robóticos y entornos, así como la correcta integración y lanzamiento de estos componentes. La habilidad para simular y controlar robots en un entorno virtual es crucial para el desarrollo y prueba de algoritmos antes de su implementación en robots físicos. Los objetivos específicos de este módulo son:

1. Entender los conceptos básicos de la simulación robótica en Gazebo y cómo ROS 2 interactúa con esta herramienta.

2. Aprender a personalizar el entorno de simulación, incluyendo la selección y modificación de mundos preexistentes.

3. Familiarizarse con la importación y manejo de modelos en Gazebo, lo que permite la inclusión de robots y otros objetos en la simulación.

4. Dominar la Interfaz Gráfica de Usuario de Gazebo para una interacción efectiva y eficiente con la simulación.

5. Desarrollar habilidades para construir, desde cero, un robot móvil simulado y su entorno, comprendiendo cada componente necesario.

6. Aprender a construir un archivo de lanzamiento para ejecutar simultáneamente varios nodos y configuraciones en ROS 2.

7. Comprender el proceso de compilación y lanzamiento de paquetes en ROS 2, integrando simulaciones de Gazebo y nodos de control.

8. Reflexionar sobre la importancia de la simulación en el ciclo de desarrollo robótico, y cómo las herramientas como ROS 2 y Gazebo facilitan este proceso.

Índice general

1 Introducción

Gazebo Classic es un simulador de robótica 3D de código abierto que proporciona una interfaz para diseñar y simular robots, así como para interactuar con ellos en escenarios de mundo virtual. A lo largo de este documento, introduciremos los conceptos básicos de la interfaz, como las funciones de los botones y cómo navegar en la escena. Puede obtener más información y acceder al proyecto en su página oficial https://classic.gazebosim.org/

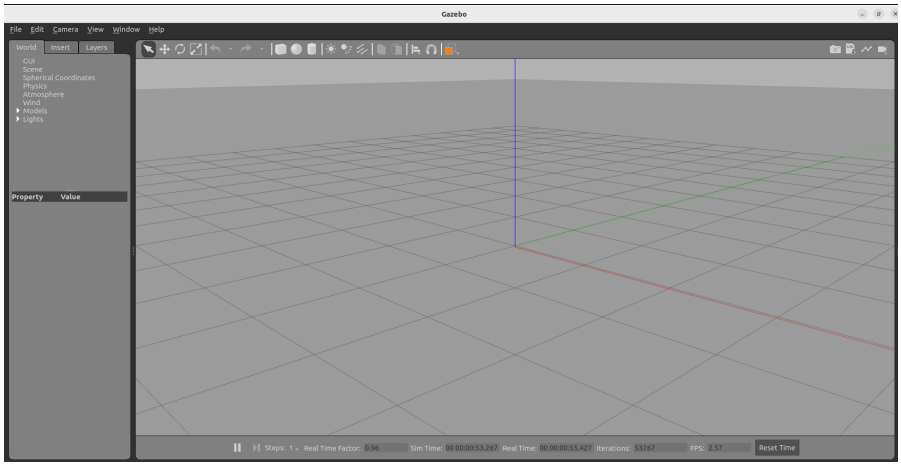


Figura 1.1: Interfaz inicial de Gazebo Classic

Para ejecutar Gazebo (figura 1.1, simplemente utiliza el siguiente comando en la terminal:

```
gazebo --verbose
```

Aconsejo el argumento –verbose para que nos de mucha más información acerca de posibles errores en modelos y escenas.

1.1 Cambiar el mundo inicial

Por defecto carga un mundo vacío (`empty_sky.world`) pero se puede poner que cargue un mundo que ya hayamos creado o alguno de los que vienen por defecto. Para realizar esta carga podemos ejecutar directamente Gazebo con el argumento de la ruta del mundo o utilizar el servidor de Gazebo. Para cargarlo un mundo concreto pondríamos:

```
gazebo <world_filename>
```

El `<world_filename>` puede ser:

- Relativo al directorio actual,
- Una ruta absoluta, o
- Relativo a un componente de ruta en `GAZEBO_RESOURCE_PATH`.
- `worlds/<world_name>`, donde `<world_name>` es un mundo que está instalado con Gazebo.

Por ejemplo, para usar el `simple_arm.world` que viene con Gazebo, use el siguiente comando:

```
gazebo worlds/simple_arm.world
```

1.2 Importando Modelos a Gazebo

Para enriquecer tus simulaciones, puedes importar modelos 3D a Gazebo. Aquí te mostramos cómo hacerlo:

Existen varios repositorios y bibliotecas en línea donde puedes obtener modelos compatibles con Gazebo, como la base de datos de modelos de Gazebo. Si deseas añadir modelos extra los puedes poner en la carpeta `home/user/.gazebo/models`. Sin embargo, Gazebo ya tiene disponible el acceso a un amplio repositorio que puedes acceder directamente desde la interfaz (figura 1.2

1. Abre Gazebo y ve a la pestaña Insertar en el panel izquierdo.
2. Aquí tendrás acceso a tus modelos locales y dos repositorios
3. Haz clic en la escena para colocar el modelo en la posición deseada.

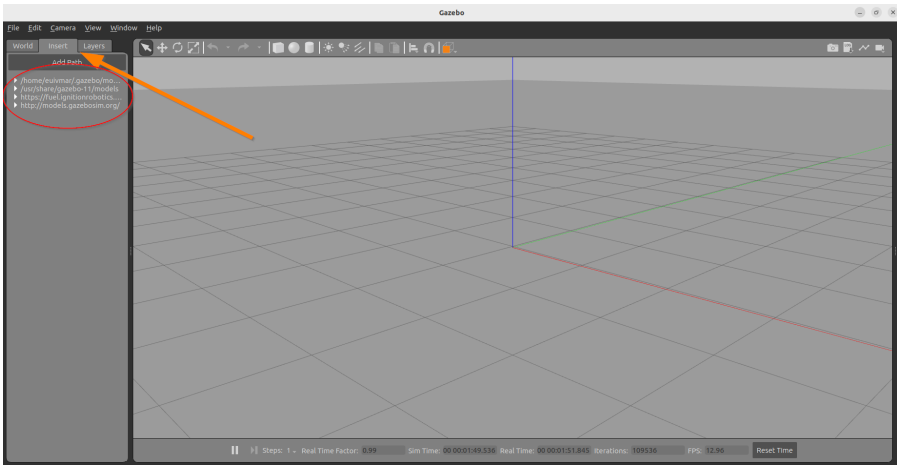


Figura 1.2: Añadiendo modelos a Gazebo

1.3 Interfaz Gráfica de Usuario (GUI)

La GUI se compone de los siguientes elementos: la escena, los paneles izquierdo y derecho, la barra de herramientas y el menú tal y como se ve en la imagen figura 1.3

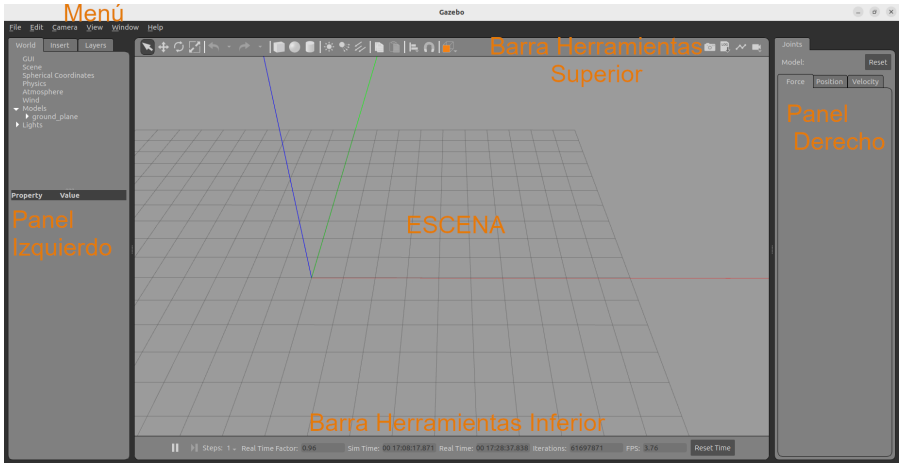


Figura 1.3: GUI Gazebo

1.3.1 La Escena

La Escena en Gazebo es el espacio virtual en el que se lleva a cabo la simulación. Es la parte principal del simulador donde los objetos y entidades son representados, animados y con los que el usuario puede interactuar. Dentro de la escena de Gazebo, existen varios componentes esenciales:

- **Modelos:** Son representaciones de objetos, robots o cualquier otro elemento que se quiera simular. Estos modelos pueden tener física asociada, permitiendo simular características como masa, fricción y colisiones.
- **Luces:** Gazebo permite la simulación de diferentes fuentes de luz, incluyendo luces puntuales, direccionales y focos. Estas luces afectan la apariencia de los modelos en la escena y pueden simular condiciones de iluminación variadas.
- **Cámara:** Las cámaras en Gazebo permiten al usuario ver la simulación desde diferentes perspectivas. Es posible configurar múltiples cámaras y cambiar entre ellas.
- **Física:** Gazebo cuenta con un motor de física que simula las interacciones entre objetos, como colisiones, gravedad y otras fuerzas. El usuario puede ajustar parámetros físicos para obtener una simulación más precisa o para probar diferentes condiciones.
- **Plugins:** Los plugins son extensiones que añaden funcionalidades específicas a la simulación. Pueden ser usados para controlar robots, interactuar con sensores y actuar sobre la escena en tiempo real.

El objetivo de la escena es ofrecer un entorno realista donde los usuarios puedan simular robots y otros sistemas, permitiéndoles probar, desarrollar y validar algoritmos y comportamientos en un ambiente controlado antes de su implementación en el mundo real

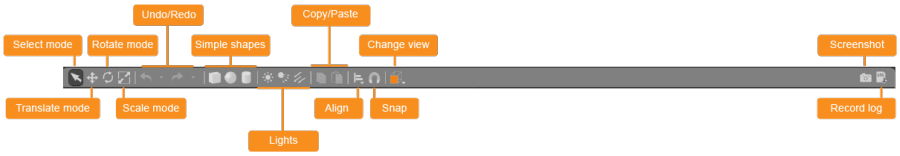


Figura 1.4: Barra de herramientas superior de Gazebo

Figura 1.5: Barra de herramientas inferior de Gazebo

1.3.2 Los Paneles

Los paneles, tanto izquierdo como derecho, pueden mostrarse, ocultarse o redimensionarse. Para ello, basta con arrastrar la barra que los separa de la Escena.

Panel Izquierdo

Este panel se muestra por defecto al iniciar Gazebo. En él, se encuentran tres pestañas principales:

- MUNDO: Muestra los modelos actuales en la escena. Permite modificar parámetros del modelo y cambiar el ángulo de vista de la cámara.
- INSERTAR: Se utiliza para añadir nuevos objetos a la simulación.
- CAPAS: Organiza y muestra los diferentes grupos de visualización. Esta característica puede estar vacía en muchos casos.

Panel Derecho

Por defecto, este panel está oculto. Al abrirlo, permite interactuar con partes móviles de un modelo seleccionado.

1.3.3 Barras de Herramientas

La interfaz de Gazebo contiene dos Barras de Herramientas. Una se encuentra encima y la otra debajo de la Escena. A continuación, se describen sus principales características:

Barra Superior: Incluye opciones para seleccionar, mover, rotar y escalar objetos, entre otras funciones.

Barra Inferior: Muestra información sobre la simulación, como el tiempo de simulación y su relación con el tiempo real.

1.3.4 El Menú

Gazebo, como muchas aplicaciones, cuenta con un menú principal en la parte superior. Algunas de sus opciones se replican en las Barras de Herramientas.

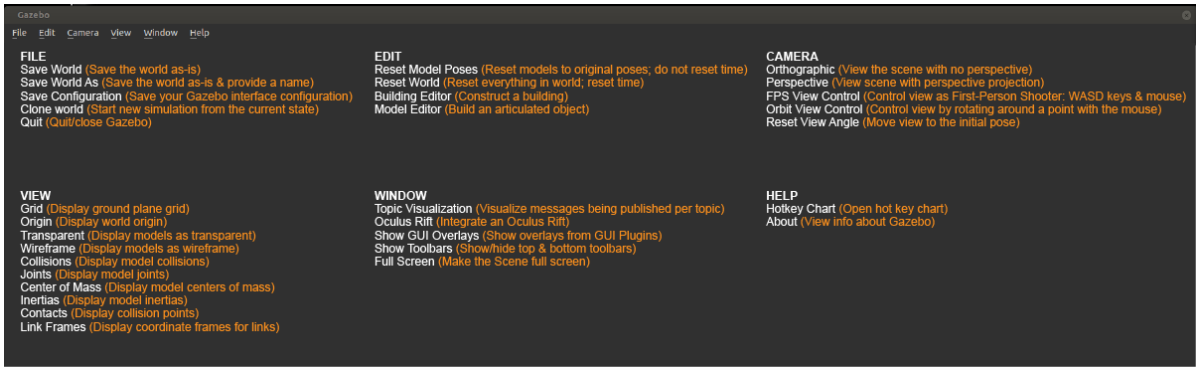


Figura 1.6: Opciones del menú

1.3.5 Controles del Ratón

El ratón es una herramienta esencial para navegar en la Escena. Se recomienda un ratón con rueda de desplazamiento para una experiencia óptima. En la figura 1.7 se muestran los controles del ratón.

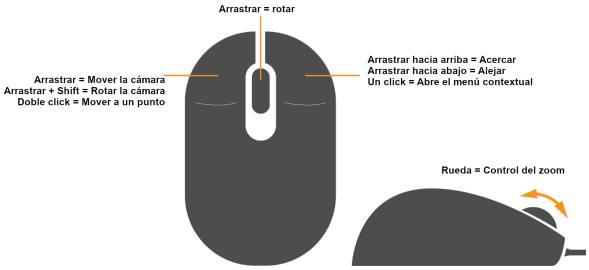


Figura 1.7: Controles del ratón

1.4 Prueba Tu Integración de ROS 2 y Gazebo

Para validar que la integración de ROS 2 y Gazebo ha sido realizada correctamente, ejecuta el siguiente comando en la terminal. Este comando lanzará Gazebo con una simulación de un vehículo diferencial:

```
gazebo --verbose
 ↪  /opt/ros/humble/share/gazebo_plugins/worlds/gazebo_ros_diff_drive_demo.world
```

Una vez que la simulación esté en marcha, deberías ver el vehículo diferencial en el entorno de Gazebo, tal como se muestra en la figura 1.8.

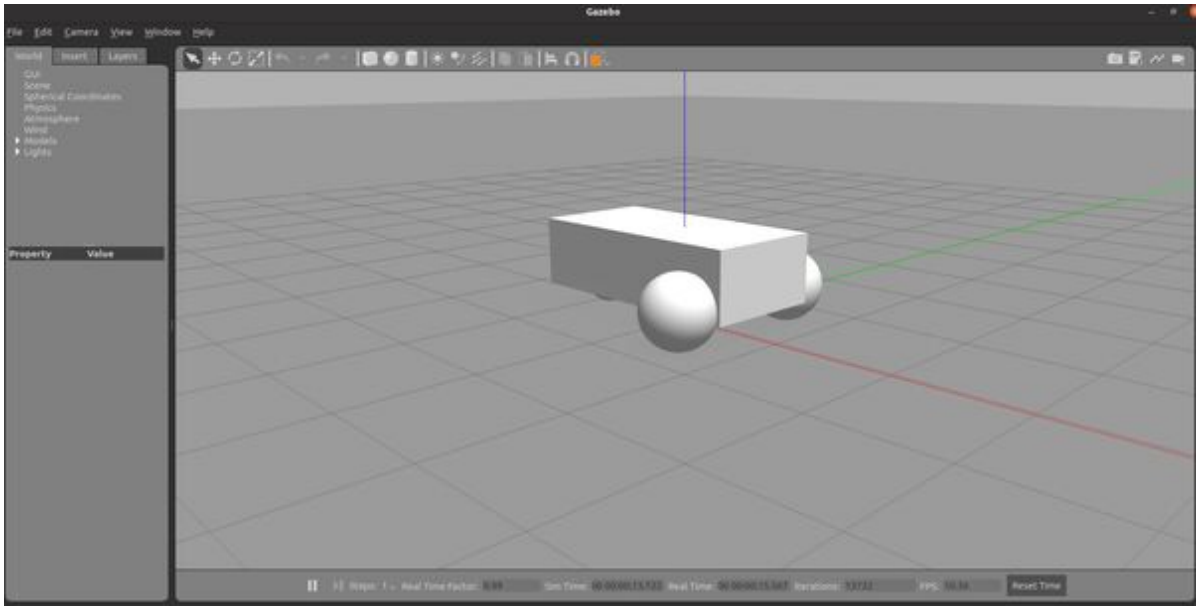


Figura 1.8: Vista del vehículo diferencial en Gazebo

Para comprender en detalle la configuración y características del vehículo diferencial en la simulación, puedes inspeccionar el archivo `.sdf` del mundo. Este archivo contiene descripciones y configuraciones del robot y su entorno en Gazebo:

```
/opt/ros/humble/share/gazebo_plugins/worlds/gazebo_ros_diff_drive_demo.world
```

Si deseas interactuar con el robot dentro de la simulación, puedes hacerlo a través de diferentes comandos de ROS 2. Por ejemplo, para hacer que el robot avance, ejecuta el siguiente comando:

```
ros2 topic pub /demo/cmd_demo geometry_msgs/Twist '{linear: {x: 1.0}}' -1
```

Este comando publicará un mensaje en el topic adecuado, instruyendo al robot para que se mueva hacia adelante.

2 Construcción de un Robot Móvil y su Entorno

En este capítulo, aprenderemos cómo crear un robot móvil autónomo desde cero usando Gazebo. Aprenderemos cómo crear un entorno para que el robot se mueva. También aprenderemos cómo integrar ROS 2 y Gazebo para que podamos controlar el robot enviándole comandos de velocidad. Esto es lo que construirás:

El tipo de robot que crearemos es un robot de almacén móvil autónomo con tracción diferencial. Construiremos el archivo SDF (Formato de Descripción de Simulación) desde cero. Nuestro robot simulado será similar al robot creado por Fetch Robotics, una empresa de robótica móvil con sede en Silicon Valley, California (figura 2.1).

Figura 2.1: Crédito: Fetch Robotics

2.1 Diseño del Robot de Almacén

Para simular nuestro robot de almacén móvil en Gazebo, primero necesitamos configurar y preparar sus archivos asociados, que incluyen la configuración, las mallas y el archivo SDF del robot.

2.1.1 Configuración del Modelo

Comienza creando un directorio para el modelo y un archivo de configuración:

```
mkdir -p ~/.gazebo/models/mobile_warehouse_robot
gedit ~/.gazebo/models/mobile_warehouse_robot/model.config
```

Para completar el archivo model.config, puedes utilizar la plantilla disponible en Poliformat. Esta plantilla proporciona campos predefinidos para detalles como el nombre del robot, la versión, el autor y una descripción detallada.

2.1.2 Obtención de Mallas del Robot

Para una representación visual más detallada y precisa, vamos a utilizar las mallas disponibles en el repositorio oficial de Gazebo. También hay una copia de estas mallas en Poliformat para tu comodidad.

Descarga las mallas del robot de almacén ejecutando:

```
cd ~/.gazebo/models
wget -q -R index.html,*.tar.gz --no-parent -r -x -nH
↪  http://models.gazebosim.org/warehouse_robot/
```

Dentro de la carpeta warehouse_robot, encontrarás un archivo llamado robot.dae, que es el modelo visual del robot. Lo referenciaremos más adelante en nuestro archivo SDF.

Adicionalmente, descarga las mallas del sensor Hokuyo utilizando:

```
cd ~/.gazebo/models
wget -q -R index.html,*.tar.gz --no-parent -r -x -nH
↪  http://models.gazebosim.org/hokuyo/
```

2.1.3 Archivo SDF del Modelo

Finalmente, descarga el archivo model.sdf del robot de almacén móvil desde Poliformat. Una vez descargado, colócalo en la carpeta `~/.gazebo/models/mobile_warehouse_robot/`. Este archivo define la estructura y comportamiento del robot dentro de la simulación.

2.2 Diseñando un Entorno de Almacén

Para proporcionar un escenario adecuado donde el robot pueda moverse y realizar sus tareas, diseñaremos un almacén. Este entorno servirá para simular las condiciones reales que el robot podría enfrentar en una situación práctica.

2.2.1 Preparación del Directorio

Comienza por crear un directorio específico para el modelo del almacén:

```
mkdir -p ~/.gazebo/models/small_warehouse
```

2.2.2 Archivo de Configuración del Modelo

A continuación, genera un archivo llamado model.config para configurar las propiedades básicas del modelo:

```
gedit ~/.gazebo/models/small_warehouse/model.config
```

Dentro de este archivo, añadirás varios campos descriptivos relacionados con el modelo del almacén. Estos incluyen el nombre del modelo, su versión, el autor (que en este caso eres tú), tu dirección de correo electrónico, y una breve descripción del almacén. Una vez hayas completado los campos, guarda y cierra el archivo.

```
<?xml version="1.0"?>
<model>
  <name>Small Warehouse</name>
  <version>1.0</version>
  <sdf version='1.4'>model.sdf</sdf>

  <author>
   <name>Automatic Addison</name>
   <email>automaticaddison@todo.todo</email>
  </author>

  <description>
    A small warehouse.
  </description>
</model>
```

Guarda el archivo y ciérralo.

2.2.3 Crea el archivo SDF

El siguiente paso implica la creación del archivo SDF, que corresponde al Formato de Descripción de Simulación. Esta es una representación esencial que define cómo se materializará y comportará el modelo small_warehouse dentro de Gazebo.

Para comenzar, descarga el archivo model.sdf disponible en Poliformat y posteriormente ubícalo en el directorio que has creado anteriormente:

```
~/.gazebo/models/small_warehouse/
```

Al inspeccionar el contenido del archivo model.sdf, notarás una extensa cantidad de líneas de código. Aunque pueda parecer complejo, esto ilustra el nivel de detalle y precisión con que Gazebo maneja la simulación, incluso para modelos aparentemente simples.

Adicionalmente, si deseas tener un mayor control o experimentar con la creación de tu propio modelo o mundo, Gazebo ofrece una interfaz gráfica intuitiva. Con la función de arrastrar y soltar, puedes agregar, mover o modificar componentes en tu simulación, y posteriormente guardar tu diseño en un archivo sdf. Esto proporciona flexibilidad tanto para los usuarios novatos como para los experimentados en la creación de mundos simulados.

2.3 Visualización y Prueba del Almacén y Robot

Para visualizar y probar el almacén y el robot que has creado dentro de Gazebo, sigue las instrucciones detalladas a continuación:

1. **Carga el Almacén**:
   - En el panel izquierdo de Gazebo, accede a la pestaña *"Insertar"*.
   - Localiza y selecciona `Small Warehouse`. Esto mostrará una representación del modelo de tu almacén.
   - Posiciona el modelo en el lugar deseado haciendo clic en el área correspondiente de la simulación.
2. **Carga el Robot**:
   - Siguiendo un proceso similar, busca y selecciona `Mobile Warehouse Robot`.
   - Tras añadirlo al entorno, el conjunto debería parecerse a lo ilustrado en la figura 2.2.
3. **Movimiento del Robot**:
   - Activa el movimiento del robot mediante la publicación en el *topic* adecuado. Podrás notar que su movimiento podría no ser completamente estable. Esta inestabilidad

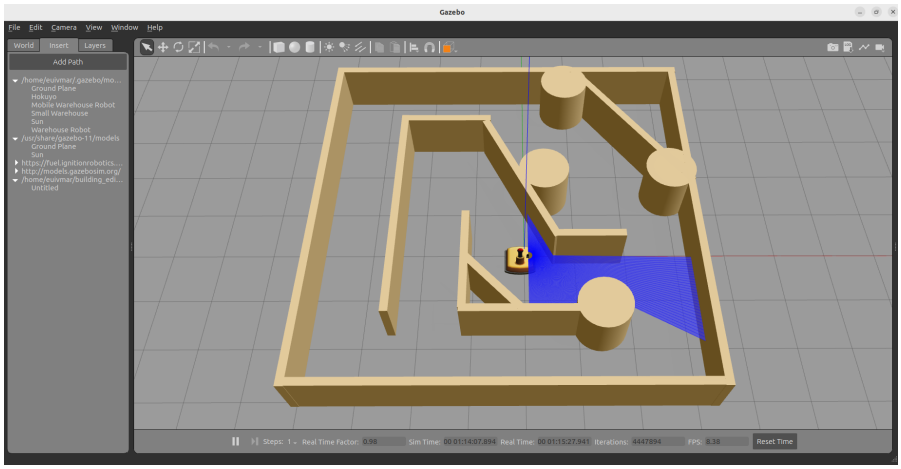


Figura 2.2: Representación del almacén y el robot configurados en Gazebo

puede deberse al sensor láser o a una falta de configuración adecuada de inercias y fricción.

- Para mejorar esto, edita el archivo `model.sdf` con las correcciones necesarias.

```
<?xml version='1.0'?>
<!--
  Gazebo ROS differential drive plugin
  ...
-->
<sdf version='1.4'>
  <model name="mobile_warehouse_robot">
     <static>false</static>

     <link name='chassis'>

         <pose>0 0 .04 0 0 0</pose>

         <!-- Inercia para el chasis -->
         <inertial>
           <mass>10</mass>
           <inertia>
             <ixx>0.05</ixx>
             <ixy>0</ixy>
             <ixz>0</ixz>
             <iyy>0.05</iyy>
             <iyz>0</iyz>
             <izz>0.05</izz>
           </inertia>
         </inertial>

         <collision name='collision'>
           <geometry>
             <box>
               <size>.4 .2 .1</size>
             </box>
```

```
            </geometry>
        </collision>

        <visual name='visual'>
            <pose> 0 0 0.02 0 0 0 </pose>
            <geometry>
              <mesh>
                <uri>model://warehouse_robot/meshes/robot.dae</uri>
                <scale>0.9 0.5 0.5 </scale>
              </mesh>
            </geometry>
        </visual>

        <!-- ... other parts of chassis ... -->

    </link>

    <link name="left_wheel">
        <pose>0.12 0.19 0.1 0 1.5707 1.5707</pose>

        <!-- Inercia para la rueda izquierda -->
        <inertial>
          <mass>2</mass>
          <inertia>
            <ixx>0.004</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.004</iyy>
            <iyz>0</iyz>
            <izz>0.004</izz>
          </inertia>
        </inertial>

        <collision name="collision">
          <geometry>
            <cylinder>
              <radius>.12</radius>
              <length>.08</length>
            </cylinder>
          </geometry>
          <surface>
            <friction>
              <ode>
                <mu>1.5</mu>
                <mu2>1.5</mu2>
              </ode>
            </friction>
          </surface>
        </collision>

        <visual name="visual">
          <geometry>
            <cylinder>
```

```
            <radius>.12</radius>
            <length>.08</length>
          </cylinder>
        </geometry>
      </visual>
    </link>

    <link name="right_wheel">
      <pose>0.12 -0.19 0.1 0 1.5707 1.5707</pose>

      <!-- Inercia para la rueda derecha -->
      <inertial>
        <mass>2</mass>
        <inertia>
          <ixx>0.004</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.004</iyy>
          <iyz>0</iyz>
          <izz>0.004</izz>
        </inertia>
      </inertial>

      <collision name="collision">
        <geometry>
          <cylinder>
            <radius>.12</radius>
            <length>.08</length>
          </cylinder>
        </geometry>
        <surface>
          <friction>
            <ode>
              <mu>1.5</mu>
              <mu2>1.5</mu2>
            </ode>
          </friction>
        </surface>
      </collision>

      <visual name="visual">
        <geometry>
          <cylinder>
            <radius>.12</radius>
            <length>.08</length>
          </cylinder>
        </geometry>
      </visual>
    </link>

    <!-- ... rest of your code with the laser range finder ... -->

    <!-- Connect laser range finder to the robot's body -->
```

```
        <joint type="fixed" name="laser_joint">
          <child>laser_link</child>
          <parent>chassis</parent>
        </joint>

        <!-- ... differential drive plugin and other configurations ...
        ↪  -->

      </model>
    </sdf>
```

4. **Reintroduce el Robot**:

   - Primero, elimina la instancia actual del robot en Gazebo: dirígete a la pestaña *"World"*, selecciona *"Models"*, y elimina `mobile_warehouse_robot`.

   - Luego, añade nuevamente el robot al entorno y observa su comportamiento. Con las modificaciones realizadas, deberías notar una mejora en la dinámica del robot.

2.4 Lanza tu Robot y Almacén usando ROS 2

Ahora que hemos creado nuestro robot y nuestro almacén, veamos cómo podemos lanzar estos elementos usando ROS 2.

2.4.1 Crea un Paquete

Abrimos una nueva ventana de terminal y naveguemos al directorio src de tu espacio de trabajo

Ahora vamos a crear un paquete llamado warehouse_robot_spawner_pkg.

```
ros2 pkg create --build-type ament_python warehouse_robot_spawner_pkg
```

2.4.2 Configuración del Paquete

Ahora que tenemos nuestro paquete, debemos modificar algunos archivos.

Abre tu archivo package.xml.

```
gedit package.xml
```

Rellena la descripción, tu dirección de correo electrónico y cualquier licencia que quieras usar. Luego guarda y cierra el archivo.

Ahora, abre tu archivo setup.py. Copia y pega estas líneas en él:

```
import os # Operating system library
from glob import glob # Handles file path names
from setuptools import setup # Facilitates the building of packages

package_name = 'warehouse_robot_spawner_pkg'

# Path of the current directory
cur_directory_path = os.path.abspath(os.path.dirname(__file__))

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),

        # Path to the launch file
        (os.path.join('share', package_name,'launch'),
        ↪  glob('launch/*.launch.py')),

        # Path to the world file
        (os.path.join('share', package_name,'worlds/'), glob('./worlds/*')),

        # Path to the warehouse sdf file
        (os.path.join('share', package_name,'models/small_warehouse/'),
        ↪  glob('./models/small_warehouse/*')),

        # Path to the mobile robot sdf file
        (os.path.join('share',
        ↪  package_name,'models/mobile_warehouse_robot/'),
        ↪  glob('./models/mobile_warehouse_robot/*')),

        # Path to the world file (i.e. warehouse + global environment)
        (os.path.join('share', package_name,'models/'), glob('./worlds/*')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='focalfossa',
    maintainer_email='focalfossa@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
          'spawn_demo = warehouse_robot_spawner_pkg.spawn_demo:main'
        ],
    },
)
```

Guarda el archivo y ciérralo.

Ahora, necesitamos agregar nuestros modelos. Crea una nueva carpeta dentro del directorio ~/ros_ws/src/warehouse_robot_spawner_pkg llamada models.

```
mkdir ~/dev_ws/src/warehouse_robot_spawner_pkg/models
```

Dirígete a tus modelos de Gazebo.

```
cd ~/.gazebo/models
```

Copia tus modelos en tu paquete:

```
cp -r small_warehouse ~/ros_ws/src/warehouse_robot_spawner_pkg/models
cp -r mobile_warehouse_robot ~/ros_ws/src/warehouse_robot_spawner_pkg/models
```

Mueve las mallas también:

```
cp -r hokuyo ~/ros_ws/src/warehouse_robot_spawner_pkg/models
cp -r warehouse_robot ~/ros_ws/src/warehouse_robot_spawner_pkg/models
```

Verifica que todo se haya copiado correctamente.

```
cd ~/ros_ws/src/warehouse_robot_spawner_pkg/models
dir
```

Ahora vamos a crear una carpeta llamada worlds. Queremos crear un archivo sdf aquí que maneje la generación del entorno de Gazebo y el almacén.

```
mkdir ~/ros_ws/src/warehouse_robot_spawner_pkg/worlds
```

Muévete dentro de esta carpeta:

```
cd ~/ros_ws/src/warehouse_robot_spawner_pkg/worlds
```

Abre un nuevo archivo llamado `warehouse.world`. Aunque este archivo tiene la extensión .world, sigue siendo un archivo sdf.

```
gedit warehouse.world
```

El código a poner (o descargar directamente el fichero) lo puedes encontrar en poliformat en la carpeta OTROS, haz clic en Guardar y luego ciérralo.

2.4.3 Crea un Nodo

Para interactuar con nuestro robot en el almacén a través de ROS 2, vamos a configurar un nodo específico.

1. **Accede a la carpeta del paquete**: `~/ros_ws/src/warehouse_robot_spawner_pkg/warehouse_robot_spawner_pkg/`

2. **Configura el nodo**:

   - Crea un archivo Python titulado `spawn_demo.py`.
   - Introduce el siguiente código en este archivo y guárdalo.

```python
"""
ROS 2 node to spawn a mobile robot inside a warehouse.

Author:
  - Addison Sears-Collins
  - https://automaticaddison.com
"""

import os # Biblioteca del sistema operativo
import sys # Biblioteca del entorno de tiempo de ejecución de Python
import rclpy # Biblioteca del Cliente de ROS para Python

# Biblioteca de gestión de paquetes
from ament_index_python.packages import get_package_share_directory

# Servicio de Gazebo para spawnear un robot
from gazebo_msgs.srv import SpawnEntity

def main():

    """ Función principal para spawnear un nodo de robot """
    # Obtener argumentos de entrada del usuario
    argv = sys.argv[1:]

    # Iniciar nodo
    rclpy.init()

    # Obtener la ruta del archivo para el modelo del robot
    sdf_file_path = os.path.join(
        get_package_share_directory("warehouse_robot_spawner_pkg"), "models",
```

```
        "mobile_warehouse_robot", "model.sdf")

    # Crear el nodo
    node = rclpy.create_node("entity_spawner")

    # Mostrar progreso en la ventana del terminal
    node.get_logger().info(
        'Creando cliente de servicio para conectarse a `/spawn_entity`')
    client = node.create_client(SpawnEntity, "/spawn_entity")

    # Obtener el servicio spawn_entity
    node.get_logger().info("Conectando al servicio `/spawn_entity`...")
    if not client.service_is_ready():
        client.wait_for_service()
        node.get_logger().info("...¡conectado!")

    # Obtener ruta al robot
    sdf_file_path = os.path.join(
        get_package_share_directory("warehouse_robot_spawner_pkg"), "models",
        "mobile_warehouse_robot", "model.sdf")

    # Mostrar ruta del archivo
    print(f"robot_sdf={sdf_file_path}")

    # Establecer datos para solicitud
    request = SpawnEntity.Request()
    request.name = argv[0]
    request.xml = open(sdf_file_path, 'r').read()
    request.robot_namespace = argv[1]
    request.initial_pose.position.x = float(argv[2])
    request.initial_pose.position.y = float(argv[3])
    request.initial_pose.position.z = float(argv[4])

    node.get_logger().info("Enviando solicitud de servicio a
    ↪  `/spawn_entity`")
    future = client.call_async(request)
    rclpy.spin_until_future_complete(node, future)
    if future.result() is not None:
        print('response: %r' % future.result())
    else:
        raise RuntimeError(
            'exception while calling service: %r' % future.exception())

    node.get_logger().info("¡Hecho! Cerrando el nodo.")
    node.destroy_node()
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

2.5 Crear un Archivo de Lanzamiento

El archivo de lanzamiento facilitará la ejecución de varios nodos y procesos relacionados. A continuación, vamos a crear un archivo de lanzamiento.

1. **Preparación del directorio**:

   ```
   mkdir ~/ros_ws/src/warehouse_robot_spawner_pkg/launch/
   cd ~/ros_ws/src/warehouse_robot_spawner_pkg/launch/
   ```

2. **Configuración del archivo**:

   - Crea y abre `gazebo_world.launch.py` para edición.
   - Agrega el siguiente código y, tras verificarlo, guárdalo y cierra el editor.

```python
# Copyright 2019 Open Source Robotics Foundation, Inc.
#
# Este es un aviso de copyright estándar que atribuye la propiedad del código a la
# Open Source Robotics Foundation (OSRF) y también especifica los términos de la licencia
# bajo los cuales se puede usar el código.

"""
Demo for spawn_entity.
Launches Gazebo and spawns a model
"""

# A bunch of software packages that are needed to launch ROS2
# Aquí se están importando los paquetes y módulos necesarios para lanzar ROS2 y trabajar con él.
import os
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir,LaunchConfiguration
from launch_ros.actions import Node
from launch.actions import ExecuteProcess
from ament_index_python.packages import get_package_share_directory

# Esta función es la definición principal que se utiliza para lanzar ROS2 y configurar
# cómo se inicializa y se ejecuta Gazebo.
def generate_launch_description():
    # Configura si Gazebo debería usar el tiempo de simulación o no.
    use_sim_time = LaunchConfiguration('use_sim_time', default='True')

    # Especifica el nombre del archivo de mundo que Gazebo debe cargar.
    world_file_name = 'warehouse.world'
```

```
    # Obtiene la ruta al directorio compartido para el paquete
    ↪  'warehouse_robot_spawner_pkg'.
    pkg_dir = get_package_share_directory('warehouse_robot_spawner_pkg')

    # Establece la variable de entorno GAZEBO_MODEL_PATH para que Gazebo
    ↪  pueda encontrar modelos.
    os.environ["GAZEBO_MODEL_PATH"] = os.path.join(pkg_dir, 'models')

    # Configura la ruta al archivo del mundo y al directorio de lanzamiento.
    world = os.path.join(pkg_dir, 'worlds', world_file_name)
    launch_file_dir = os.path.join(pkg_dir, 'launch')

    # Configura y lanza el proceso de Gazebo con ciertos argumentos y
    ↪  plugins.
    gazebo = ExecuteProcess(
            cmd=['gazebo', '--verbose', world, '-s', 'libgazebo_ros_init.so',
            '-s', 'libgazebo_ros_factory.so'],
            output='screen')

    # Crea y configura el nodo que se encargará de lanzar la entidad en
    ↪  Gazebo.
    spawn_entity = Node(package='warehouse_robot_spawner_pkg',
    ↪  executable='spawn_demo',
                        arguments=['WarehouseBot', 'demo', '-1.5', '-4.0',
                        ↪  '0.0'],
                        output='screen')

    # Retorna una descripción de lanzamiento que contiene todos los nodos y
    ↪  procesos
    # que deben ser lanzados.
    return LaunchDescription([
        gazebo,
        spawn_entity,
    ])
```

2.6 Compilar y lanzar el Paquete

Una vez todo configurado, debemos compilar y lanzar nuestro nodo.

1. Regresa al directorio raíz de tu espacio de trabajo y compila el paquete.
2. Inicializa tu entorno ROS con el comando ‘source‘.
3. Ejecuta el nodo usando el siguiente comando:

```
ros2 launch warehouse_robot_spawner_pkg gazebo_world.launch.py
```

Para maniobrar el robot dentro del almacén, puedes controlarlo usando el teclado. Realiza el siguiente mapeo de topics para lograrlo:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r
↪  cmd_vel:=/demo/cmd_vel
```

Bibliografía

Web de los tutoriales de ROS (2023). https://docs.ros.org/en/humble/Tutorials.html.

Web de ROS2 Humble (2023). https://docs.ros.org/en/humble/index.html.

Web del ayuda de Gazebo (2023). https://classic.gazebosim.org/tutorials?cat=guided_b&tut=guided_b2.

Web del tutorial de Gazebo (2023). https://automaticaddison.com/how-to-simulate-a-robot-using-gazebo-and-ros-2/.