Práctica 2 - Turtlebot 4
Robótica de Servicios
Máster en Robótica e Inteligencia Artificial

En esta práctica utilizaremos el simulador Gazebo Ignition para simular el **Turtlebot 4**. Gazebo Ignition es la versión más reciente del simulador y ofrece características avanzadas. El simulador Gazebo Ignition se integra con ROS 2, lo que nos permite controlar el Turtlebot a través de los mismos comandos de `ros2 topic pub`. Es importante destacar que para utilizar Gazebo Ignition, se recomienda disponer de una GPU, ya que su rendimiento en sistemas sin GPU puede ser lento.

1. Turtlebot 4 en Gazebo Ignition

1.1. Instalación de Turtlebot 4

Simulación del Turtlebot4 utilizando Ignition Gazebo.

Visita el Manual de Usuario de Turtlebot4 para más detalles.

2. Instalación

Para instalar el simulador de Turtlebot4, sigue los siguientes comandos:

El primer paso consiste en instalar el Gazebo Ignition, en su versión Fortrees, lo vamos a hacer de paquete:

```
sudo apt-get update && sudo apt-get install wget
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -
    cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt-get install ignition-fortress
```

Ahora podemos instalar el software asociado a la simulación de Turtlebot también de paquete, simplemente lanza

```
sudo apt-get update
sudo apt-get install ros-humble-turtlebot4-simulator
```

Sin embargo instalar de paquete puede dar problemas

```
mkdir ~/turtlebot4_ws/src
git clone https://github.com/turtlebot/turtlebot4_simulator.git -b humble
```

Instalar las dependencias

```
cd ~/turtlebot4_ws
rosdep install --from-path src -yi
```

Compilar los paquetes :

```
source /opt/ros/humble/setup.bash
colcon build --symlink-install
```

A continuación, es necesario configurar el espacio de trabajo ejecutando `source /turtlebot4_ws/install/setup.bash` en la terminal o agregando ese comando al archivo `.bashrc` y luego cargando el archivo `.bashrc`.

2.1. Iniciando un Turtlebot 4

Luego, para iniciar Gazebo con un Turtlebot 4, usa el siguiente comando:

```
$ ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py
```

También puedes especificar un mundo particular utilizando el argumento `world`, con opciones como: `depot`, `maze`, `warehouse` (el valor por defecto es `warehouse`):

```
$ ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py world:=maze
```

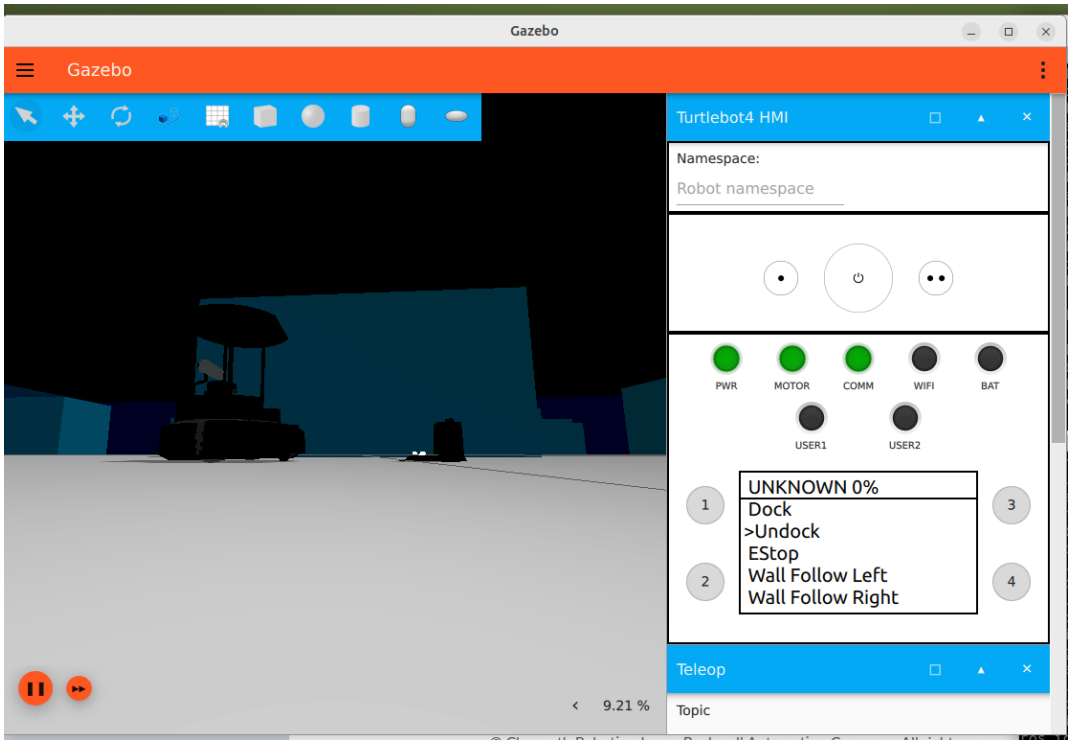

Figura 1: Modelo de mundo `maze` en Gazebo Ignition.

2.2. Iniciar el movimiento del Turtlebot 4

Para mover el Turtlebot 4, sigue estos pasos:

- Haz clic en el botón **Play** (abajo a la izquierda).
- Desacopla (undock) el Turtlebot seleccionando el botón 4 y luego elige la opción de *undock* usando el botón 1.

- Espera hasta que el desacople esté completo. Si intentas ejecutar un comando mientras el desacople no ha finalizado, el simulador mostrará un mensaje como: *Ignoring velocities commanded while an autonomous behavior is running*.

3. Modificaciones del mundo en Gazebo Ignition

3.1. Añadir formas

Es fácil añadir formas 3D al mundo de Gazebo. Selecciona una forma desde el panel superior y colócala en el mundo.

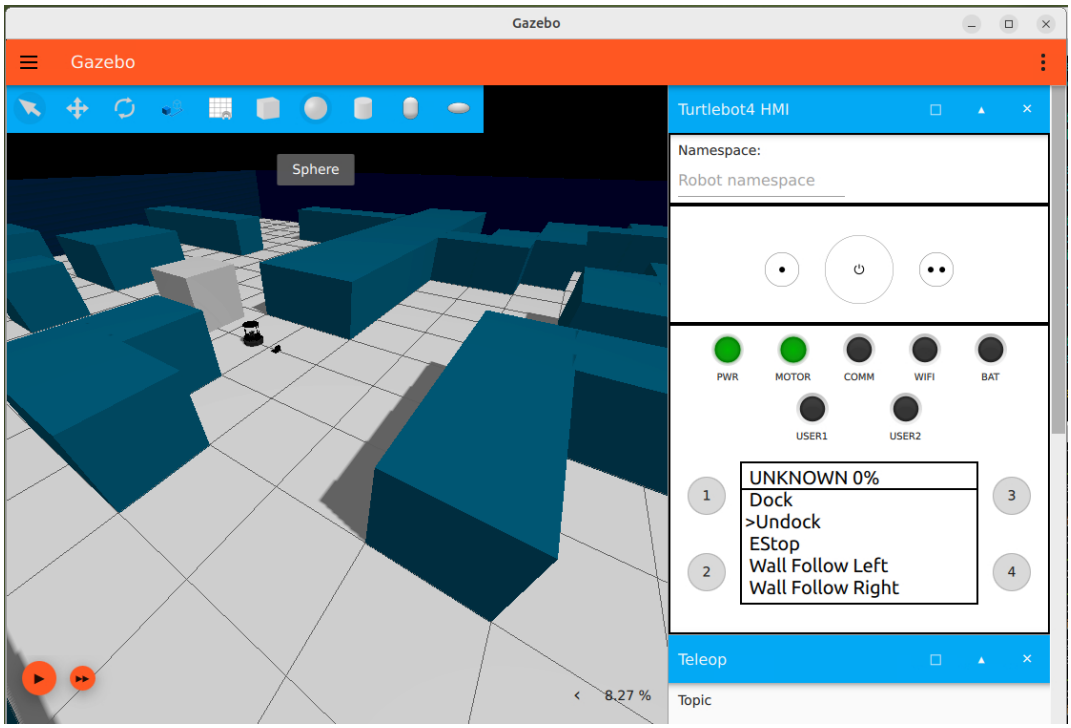

Figura 2: Añadir formas en Gazebo Ignition.

Luego, puedes hacer doble clic sobre la forma para ajustar sus propiedades, como la posición, orientación, escala, etc.

4. RViz con Gazebo Ignition

Para observar los datos de los sensores del Turtlebot, puedes usar RViz. Con el simulador Gazebo ya en ejecución, utiliza el siguiente comando para abrir RViz con el modelo del Turtlebot:

```
$ ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py rviz:=true
```

En RViz, puedes cambiar entre diferentes vistas desde el panel de *Views*. Además, en las *Opciones Globales*, el *Fixed Frame* determina el punto de referencia para la visualización. Seleccionar *odom* te permite fijar la vista en la posición original del robot.

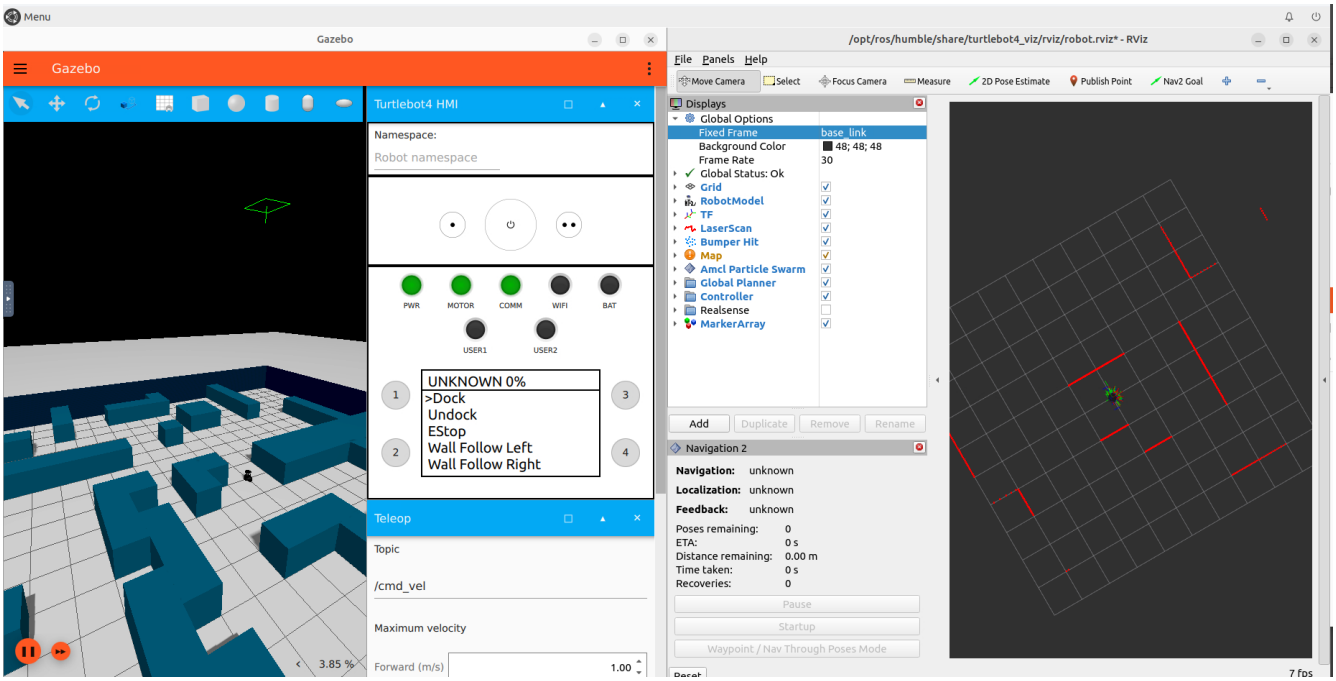

Figura 3: Visualización en RViz.

5. Control de Turtlebot mediante Tópicos

Una vez que el Turtlebot 4 esté en funcionamiento, puedes explorar los tópicos que permiten interactuar con él.

5.1. Tópicos

Puedes ver los tópicos que publica el Turtlebot usando el comando `ros2 topic list`:

```
$ ros2 topic list
```

Algunos de los tópicos importantes incluyen:

- `/cmd_vel`: Para comandar la velocidad del robot.
- `/odom`: Para obtener la odometría del robot.
- `/scan`: Para acceder a los datos del escáner láser.

5.2. Control desde la línea de comandos

Puedes enviar comandos de velocidad al Turtlebot desde la línea de comandos usando el tópico `/cmd_vel`. Por ejemplo, para mover el Turtlebot, publica el siguiente mensaje:

```
$ ros2 topic pub /cmd_vel geometry_msgs/msg/Twist '{linear: {x: 0.2}, angular: {z: -0.2}}'
    -1
```

Este comando hará que el robot avance con una velocidad lineal de 0.2 m/s y gire con una velocidad angular de -0.2 rad/s. Para detener el robot, simplemente publica un mensaje con velocidades lineales y angulares en cero:

```
$ ros2 topic pub /cmd_vel geometry_msgs/msg/Twist '{}' -1
```

5.3. Teleoperación del Turtlebot

También puedes controlar el Turtlebot usando teleoperación mediante el teclado. Para esto, usa el siguiente comando:

```
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

En una terminal separada, puedes monitorear los comandos enviados al Turtlebot con:

```
$ ros2 topic echo /cmd_vel
```

6. Nodo Python para controlar el Turtlebot

Aquí tienes un ejemplo de un nodo en Python que controla el Turtlebot:

```python
import math
import time
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist, Vector3

class CircleDrive(Node):
    def __init__(self, topic_name='/cmd_vel'):
        super().__init__('circle_drive')
        self.publisher_ = self.create_publisher(Twist, topic_name, 1)
        time.sleep(2.0)

    def drive(self, wait, motion = Twist()):
        self.publisher_.publish(motion)
        time.sleep(wait)
        self.publisher_.publish(Twist())

def main(args=None):
    rclpy.init(args=args)
    node = CircleDrive()
    motion = Twist(linear=Vector3(x=0.2), angular=Vector3(z=0.4))
    wait = 2 * math.pi / motion.angular.z
    node.drive(wait=wait, motion=motion)
```

Este código hace que el Turtlebot se mueva en un círculo completo y luego se detenga. El nodo publica un mensaje `Twist` en el tópico `/cmd_vel` para controlar el robot.