Guía de Uso NavBot

Jeremy S Rivera

Enero 2026

1. Introducción

La siguiente Guía de Uso tiene como objetivo explicar cómo clonar el repositorio y cómo utilizar de manera adecuada las herramientas que tiene para ofrecer.

2. Prerrequisitos

Antes de acceder al repositorio es necesario haber revisado la documentación de ROS 2 Jazzy Jalisco, especialmente la guía de instalación, conceptos y tutoriales que proporciona la página.

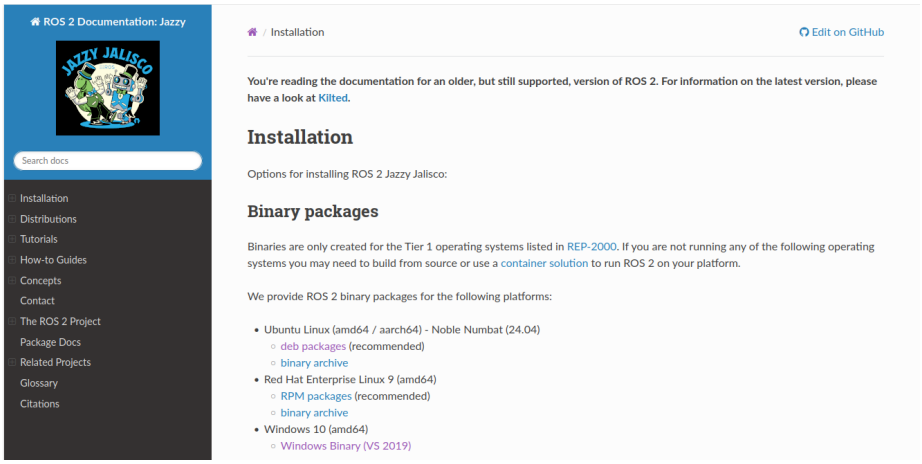

Figura 1: Página oficial de documentación de ROS 2.

Además de otros paquetes de navegación y simulación que se encuentran detallados en el repositorio de GitHub.

Una vez desarrollados los pasos guía en el repositorio que corresponden al clonado y construcción del paquete con la librería `colcon build`, podemos continuar con la guía para implementar su uso de forma adecuada.

3. Uso del paquete

Dentro del paquete `src/nav_bot/launch` se encuentran 3 archivos scripts codificados en Python que son esenciales para la implementación de la simulación, en especial el archivo `navigation.launch.py`. Al ejecutarse, este abre RViz2 con un mapa guardado por defecto en la codificación del script. Este mapa guardado se llama `mapa_mundo_prueba.yaml`. Además, el script ejecuta internamente el código de `sim.launch.py`, que abre nuestro mundo en Gazebo definido en la ruta del archivo: `mundo_prueba.sdf`.

En caso de querer realizar la simulación con un mundo diferente, estos serán los principales archivos que se deberán modificar: las rutas en `sim.launch.py` y `navigation.launch.py`. Se debe prestar atención a las especificaciones del nuevo mundo del que queramos generar un mapa, ya que este debe contar con la siguiente línea de código dentro del archivo SDF en la sección de plugins:

```
<plugin name='gz::sim::systems::Sensors' filename='gz-sim-sensors-system'>
```

```
# Ruta al mapa que acabamos de mover
map_file = os.path.join(pkg_nav_bot, 'maps', 'mapa_mundo_prueba.yaml')

# Ruta a los parámetros de Nav2 (Usaremos los por defecto de momento)
nav2_params = os.path.join(pkg_nav2_bringup, 'params', 'nav2_params.yaml')

# 2. Incluir nuestra simulación (Gazebo + Robot)
# Nota: Asegúrate de que 'sim.launch.py' es el nombre correcto de tu archivo anterior
sim_launch = IncludeLaunchDescription(
    PythonLaunchDescriptionSource(
        os.path.join(pkg_nav_bot, 'launch', 'sim.launch.py')
    ),
```

Figura 2: Estructura del archivo navigation.launch.py.

Después de haber construido el paquete siguiendo la documentación en GitHub, al ejecutar el comando:

```
ros2 launch nav_bot navigation.launch.py
```

se nos abrirá RViz2 con nuestro mapa cargado, y Gazebo con el mundo referenciado en la ruta `sim.launch.py`.

En este punto se nos mostrará un error en el Display Map, en la sección de Message: `"No Map Received"`. Este error se genera porque, aunque nuestro mapa está cargado y podemos visualizar su estructura, el sistema necesita inicializarse. El error se soluciona automáticamente al utilizar la herramienta **2D**

Pose Estimate para configurar la posición actual del robot. Al utilizar esta herramienta se debe apuntar en dirección hacia donde mira el frente del robot:

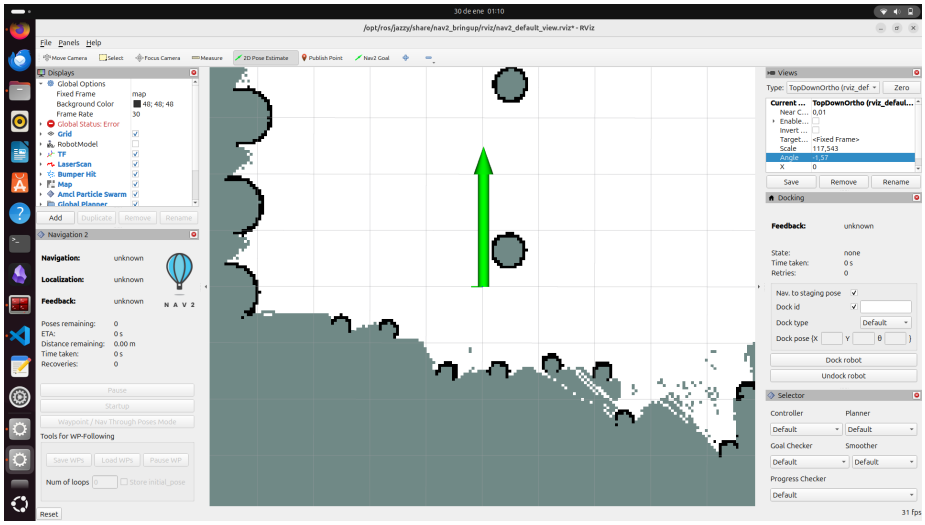

Figura 3: Uso de la herramienta 2D Pose Estimate en RViz2.

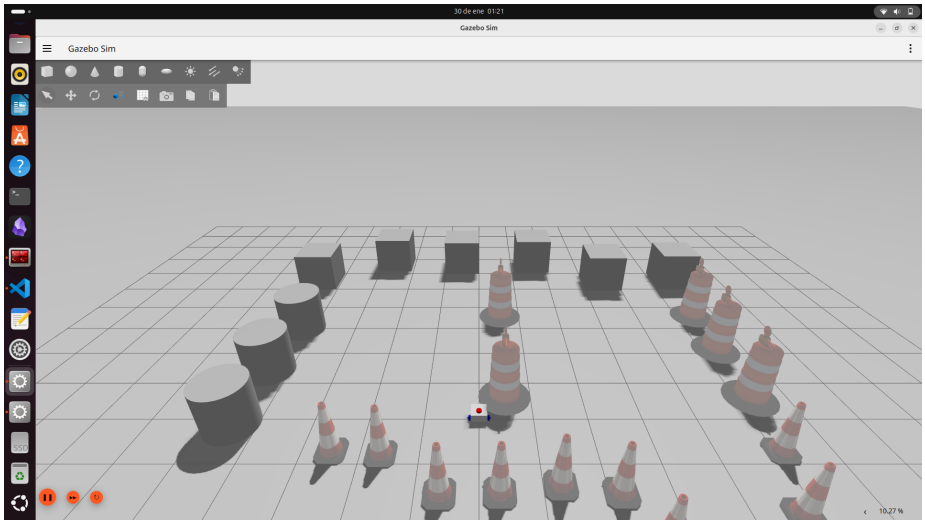

Figura 4: Vista del entorno de simulación en Gazebo.

4. Operación del Robot

El sistema de navegación implementado permite operar el robot móvil mediante tres modalidades distintas:

- **Teleoperación:** Control manual utilizando el paquete `teleop_twist_keyboard`.
- **Navegación por GUI:** Asignación de objetivos visuales mediante la herramienta *Nav2 Goal* en RViz.
- **Navegación por Script:** Ejecución de misiones autónomas programadas en Python.

A continuación, se detalla el procedimiento para utilizar cada una de estas opciones.

4.1. Teleoperación Manual

La primera modalidad consiste en el control directo mediante teclado. Para ello, se emplea el nodo `teleop_twist_keyboard`, el cual traduce las pulsaciones de teclas en comandos de velocidad lineal y angular.

Para iniciar este nodo, se debe ejecutar el siguiente comando en una terminal:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Controles de Movimiento

La interfaz permite controlar el desplazamiento del robot utilizando las siguientes teclas:

- **U:** Avanzar con giro a la izquierda.
- **I:** Avanzar en línea recta.
- **O:** Avanzar con giro a la derecha.
- **J:** Rotación sobre el propio eje a la izquierda.
- **K:** Detener el movimiento (Stop).
- **L:** Rotación sobre el propio eje a la derecha.
- **M:** Retroceder (hacia la izquierda, atrás, derecha según combinación con $<$ o $>$).

Ajuste de Velocidades

Es posible modificar dinámicamente los parámetros de velocidad mediante los siguientes comandos:

- **q / z:** Aumentar o reducir la velocidad máxima general en un 10 %.
- **w / x:** Aumentar o reducir únicamente la velocidad lineal en un 10 %.
- **e / c:** Aumentar o reducir únicamente la velocidad angular en un 10 %.

Figura 5: Interfaz de terminal del nodo teleop_twist_keyboard.

4.2. Navegación por Interfaz Gráfica (GUI)

Esta modalidad permite comandar el robot a través de la interfaz visual de RViz2, aprovechando las capacidades del stack *Navigation 2*. A diferencia de la teleoperación, aquí el operador no controla los motores directamente, sino que establece objetivos de alto nivel.

Asignación de Metas (Nav2 Goal)

Para operar el robot, se utiliza la herramienta **Nav2 Goal** ubicada en la barra superior. Al hacer clic en un punto del mapa y arrastrar el cursor, se

define la pose objetivo (coordenadas x, y y orientación θ) hacia la cual el robot debe desplazarse.

Evasión Dinámica de Obstáculos

Una vez fijada la meta, el sistema traza una ruta óptima a través del mapa conocido. Una característica crítica de este modo es su capacidad de reacción ante cambios en el entorno:

1. Si se introduce un obstáculo imprevisto en el mundo de simulación (Gazebo) que interrumpe la trayectoria planificada.
2. El sensor LIDAR detecta el objeto y actualiza el mapa de costos local en RViz2.
3. *Navigation 2* replantea la ruta automáticamente en tiempo real, generando una trayectoria alternativa para evadir el obstáculo sin detener la misión.

Figura 6: Planificación de trayectoria y evasión de obstáculos en RViz2 usando Nav2 Goal.

4.3. Navegación por Script (Automatización)

Esta modalidad representa el nivel más alto de operación, permitiendo la automatización de tareas de navegación sin intervención manual continua. Se basa en el uso de la API de *Navigation 2* para Python.

Descripción de la Misión

Para esta demostración, se utiliza el script `mision_autonoma.py` alojado en el paquete del proyecto. La lógica de control implementada realiza la siguiente secuencia:

1. El robot se desplaza hacia un **Punto A** predefinido.
2. Durante el trayecto, el sistema reporta por consola la distancia restante hacia el objetivo en tiempo real.
3. Al llegar, el robot realiza una espera programada de 3 segundos.
4. Finalmente, el sistema recalcula la ruta y dirige al robot hacia el **Punto B**.

Ejecución del Script

Para iniciar la misión automática, es necesario abrir una nueva terminal. Asegúrese de cumplir con los prerrequisitos de instalación de librerías Python indicados en el repositorio.

El procedimiento de ejecución es el siguiente:

```
source install/setup.bash
python3 mision_autonoma.py
```

Al ejecutar el comando, el robot comenzará la navegación inmediatamente en el entorno de simulación.

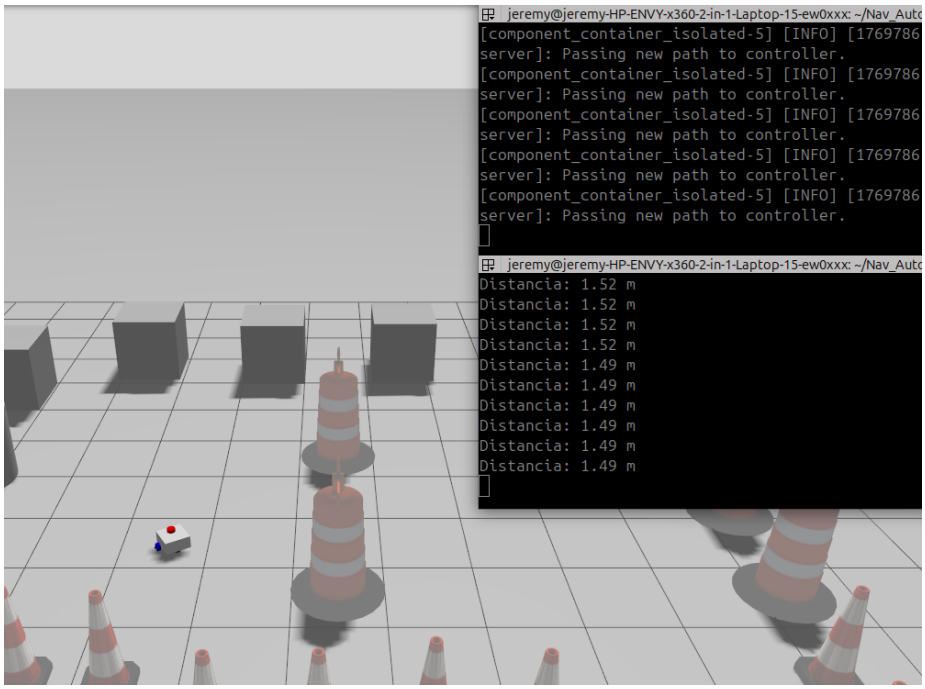

Figura 7: Ejecución del script en terminal mostrando el progreso de la misión.

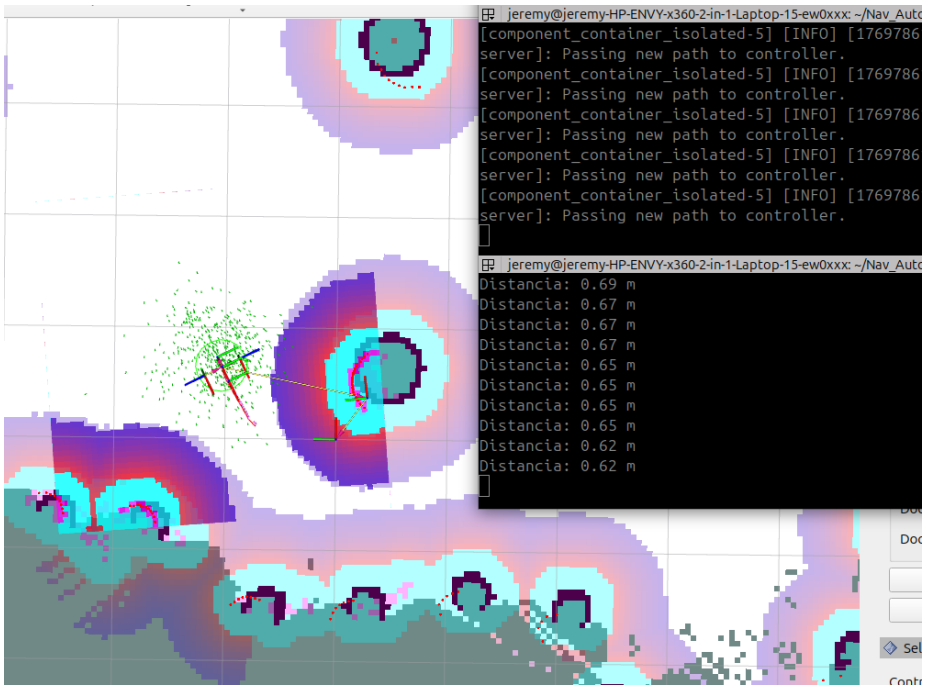

Figura 8: Visualización en RViz de la trayectoria autónoma generada por el script.

Personalización de la Misión

Si deseamos modificar la ruta autónoma, podemos editar el script `mision_autonoma.py` para redefinir los Puntos A y B, o incluso agregar nuevos destinos. Para realizar esto con precisión y evitar colisiones con obstáculos del mapa, es fundamental obtener coordenadas válidas.

Procedimiento para la Obtención de Coordenadas:

1. Para visualizar en tiempo real las coordenadas que seleccionemos en RViz, debemos abrir una nueva terminal y ejecutar el siguiente comando:

```
ros2 topic echo /clicked_point
```

2. En la interfaz de RViz, ubique y seleccione la herramienta **Publish Point** en la barra superior.

3. Haga clic sobre el mapa en la ubicación deseada para el **Punto A** (Destino 1). Asegúrese de elegir una zona libre de obstáculos.

4. Observe la terminal abierta en el paso 1. Se imprimirán las coordenadas exactas del punto seleccionado (ver Figura 9).

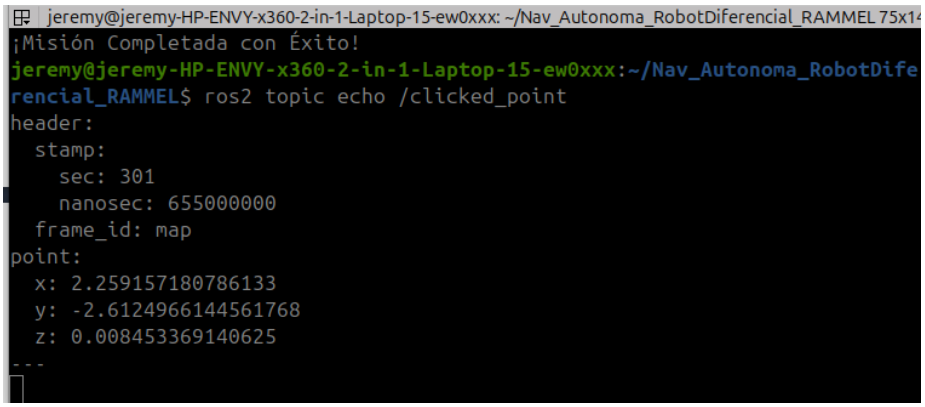

Figura 9: Visualización de coordenadas x, y, z en la terminal mediante el tópico /clicked_point.

5. Repita el procedimiento para el **Punto B** (Destino 2) o cualquier punto adicional (C, D, etc.) que desee incorporar a la misión.

Edición del Código:

Una vez obtenidas las coordenadas, abra el archivo `mision_autonoma.py` y reemplace los valores existentes en las variables de posición (`pose.position.x`, `pose.position.y`) con los nuevos datos obtenidos.

```python
# DEFINIR PUNTO A
goal_pose1 = PoseStamped()
goal_pose1.header.frame_id = 'map'
goal_pose1.header.stamp = nav.get_clock().now().to_msg()

# COORDENADAS A
goal_pose1.pose.position.x = 0.11024327576160431
goal_pose1.pose.position.y = 0.7243705987930298
goal_pose1.pose.orientation.w = 1.0

# Yendo al punto A
print("Dirigiendome al Punto A")
nav.goToPose(goal_pose1)

while not nav.isTaskComplete():
    feedback = nav.getFeedback()
    # Distancia Restante
    print(f'Distancia: {feedback.distance_remaining:.2f} m')

print("He Llegado al Punto A. Espero 5 segundos")
time.sleep(5)

# DEFINIR PUNTO B
goal_pose2 = PoseStamped()
goal_pose2.header.frame_id = 'map'
goal_pose2.header.stamp = nav.get_clock().now().to_msg()

# COORDENADAS B
goal_pose2.pose.position.x = 3.584080934524536
goal_pose2.pose.position.y = -3.270587205886841
goal_pose2.pose.orientation.w = 1.0
```

Figura 10: Edición de las coordenadas de destino dentro del script de Python.

Finalmente, guarde los cambios y vuelva a ejecutar el script en la terminal. Si el procedimiento se ha realizado correctamente, se habrá logrado una navegación autónoma personalizada capaz de evitar obstáculos en el mapa preconfigurado.

5. Referencias

- Documentación de ROS 2 Jazzy: `https://docs.ros.org/en/jazzy/index.html`
- Repositorio de GitHub: `https://github.com/jerimore/Nav_Autonoma_RobotDiferencial_RAMMEL.git`