Guia para replicar mi progreso en Ignition

Joel Ramos Beltrán

Requisitos previos

-Instalar ros2_humble: Recomiendo hacerlo por debs usando este tutorial https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html

-Instalar Gazebo Ignition: Recomiendo hacerlo desde Binarios usando este tutorial https://gazebosim.org/docs/fortress/install_ubuntu/

-Instalar los plugins del controlador ign_ros2_control https://index.ros.org/p/ign_ros2_control/#humble-overview

sudo apt install ros-humble-ros-gz-sim

sudo apt install ros-humble-ros-gz-bridge

sudo apt install ros-humble-ros2-control ros-humble-controller-manager

sudo apt install ros-humble-ros2-controllers ros-humble-joint-state-broadcaster

sudo apt install ros-humble-ign-ros2-control

Pasos iniciales

-Clonar mi repositorio de “Robotnik_prácticas” de internet. Este es el enlace: https://github.com/JoelRamosBeltran/Robotnik_practicas

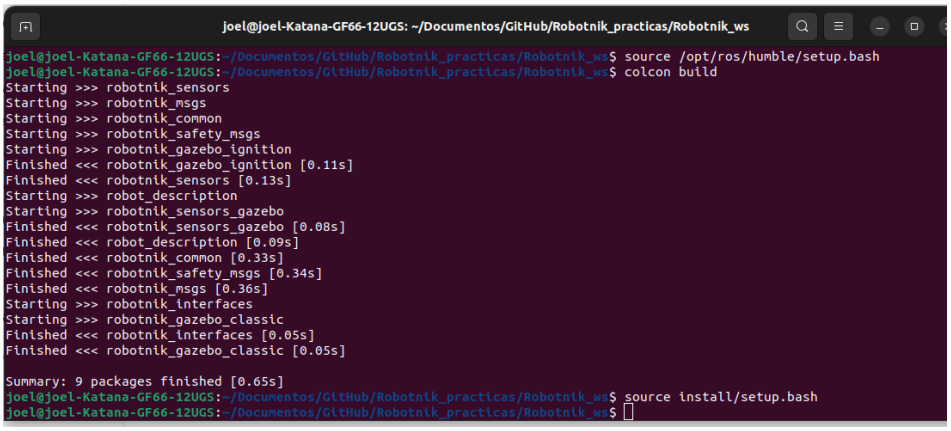
-Abrir una terminal en Robotnik_ws

-Instalar el controlador de Robotnik
sudo apt install src/robot_simulation/debs/ros-humble-robotnik-controller_0.0-20250131.153235-local_amd64.deb

-Realizar un “source /opt/ros/humble/setup.bash”

-Compilar con “colcon build”

-Realizar un “source install/setup.bash”

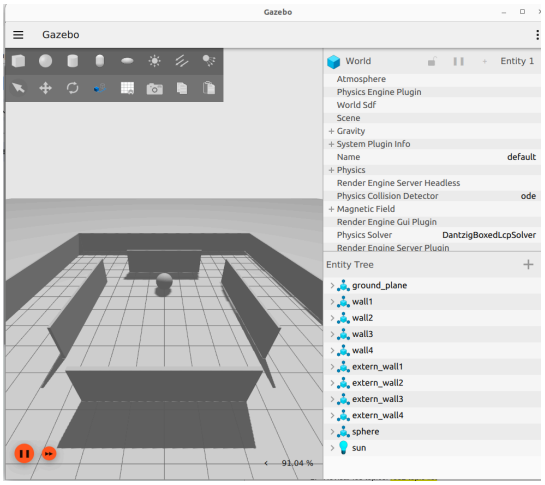


Procedimiento para probar los robots.

-Recomiendo abrir 4 terminales (todas ellas en el directorio Robotnik_ws), y que en las cuatro se hagan los dos sources (los de los pasos iniciales)

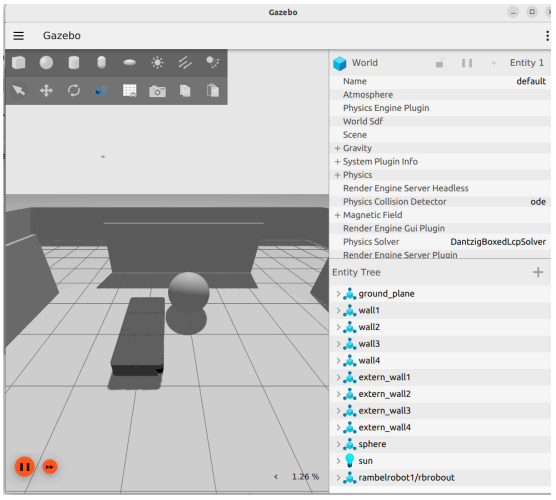
-En la primera terminal ejecutaremos el mundo

ros2 launch robotnik_gazebo_ignition spawn_world.launch.py



-En la segunda terminal ejecutaremos el spawneo del robot

ros2 launch robotnik_gazebo_ignition spawn_robot.launch.py robot:=rbrobout
o
ros2 launch robotnik_gazebo_ignition spawn_robot_dinamicbridge.launch.py robot:=rbrobout



-En la tercera terminal ejecutaremos el cmd_vel

ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r /cmd_vel:=/robot/robotnik_base_controller/cmd_vel

-Y la cuarta la utilizaremos de manera auxiliar, como por ejemplo:

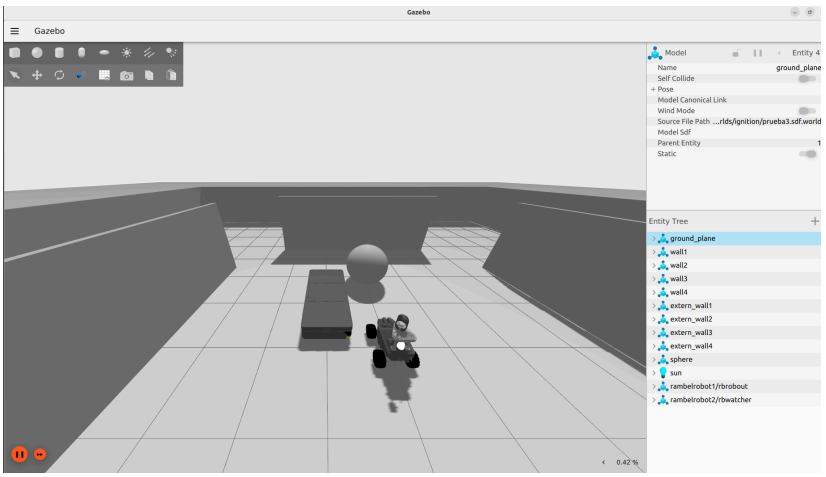
1. Revisar los topics: ros2 topic list
2. Spawnear un segundo robot:
   ros2 launch robotnik_gazebo_ignition spawn_robot.launch.py robot:=modelo_robot

(En principio he intentado usar la mayor cantidad de rutas relativas, pero es probable que falten algunas, así que habría que modificar estas, si saltan error, por las tuyas).

Después de esto, deberías poder desplazar correctamente el robot por el escenario, recibiendo los datos de sus sensores en los rviz2 abiertos con cada spawneo de robot.

El comando entero de spawneo es este:

ros2 launch robotnik_gazebo_ignition spawn_robot_dinamicbridge.launch.py robot:=modelo_robot namespace:=nombre_único_del_robot x:=0 y:=0 z:=0



Componentes de la simulación.

Antes de nada, voy a explicar un poco como funciona Ignition. El simulador Gazebo Ignition 6 se conforma de un archivo world, el cual es el mundo, y los archivos URDF de los robots.

El archivo de mundo contiene toda la estructura y componentes del mismo, como suelo, paredes, y otros objetos. Además, en él se introducen también determinados plugins que posteriormente son útiles para algunos elementos de los robots, como el plugin de sensores.

Los XACRO son URDFS modulares, los cuales funcionan referenciándose entre ellos. Esto permite reutilizar elementos entre robots, por ejemplo, sus ruedas o sus sensores y además, permite utilizar argumentos a la hora de llamarlos.

Voy a dividir los archivos de la simulación en 3 tipos:

- Launchers
- XACROS
- Otros elementos (params, bridges, ejecutables externos, mundo,...)

Launchers:

En el paquete Robotnik_Simulation podemos encontrar varios launchers:

- **World**: Este launcher es el encargado de lanzar el mundo de Gazebo.
- **robot_spawner**: Este es el encargado de spawnear el robot en la simulación usando un bridge genérico por modelo de robot.
- **spawn_robot_dinamicbridge**: Este es una versión más moderna del de arriba, utilizando solo un archivo bridge dinámico para el funcionamiento con ROS.

Voy a detallar la estructura del DinamicBridge ya que es el más completo y el que mejores resultados da, ya que es totalmente dinámico para cualquier tipo de robot sin necesidad de crear un bridge genérico de topics entre Ignition y Ros para cada uno.

Los argumentos, primera parte del launcher, son los que describen el robot que queremos spawnear. Es decir, su modelo, su nombre único, su posición en el mundo, etc.

La siguiente parte del launcher son las acciones que se ejecutarán al lanzarlo. Estas son:

- **robot_state**: Esta acción se encarga de llamar a otro launcher (perteneciente al paquete Robot_description) el cual se encargará de publicar en el topic robot_description la descripción de las transformadas del robot.

- **robot_spawner**: Esta activa el nodo cuya función es spawnear el robot en el mundo de Ignition. Utiliza para ello el robot_description que se publicará con la acción anterior.

- **joint_state_broadcaster**: Activa el nodo encargado de spawnear el controlador “joint_state_broadcaster” el cual publica el estado de las joints móviles del robot.

- **joint_trajectory_controller**: Activa el nodo encargado de spawnear el controlador encargado de la trayectoría de un posible brazo manipulador colocado sobre el robot.

- **robotnik_controller**: Este nodo tiene la función de spawnear el controlador creado por Robotnik, cuya función es mover el robot, es decir, hacer que el robot se desplace gracias a sus ruedas.

- **rviz2**: Este nodo tiene la función de abrir automáticamente rviz2 con una configuración creada con anterioridad.

- **bridge_yaml_creator**: Esta acción se encarga de ejecutar el programa encargado de crear el nuevo archivo bridge de topics para el robot que queremos usar.

- **ros_gz_bridge**: Este nodo se encarga de crear el bridge entre Ignition y Ros2 a partir del archivo creado en la acción anterior.

- **otros**: También hay acciones de secuencia, creadas para controlar el orden de ejecución de las acciones anteriores.

El otro launcher necesario para la ejecución de la simulación, y de hecho, el que es necesario lanzar primero, es el encargado de spawnear el mundo, es decir “spawn_world”.
Estas son sus acciones:

- **gazebo_ignition_launch_group**: Esta acción está conformada por un conjunto de otras dos acciones, pero la funcionalidad principal es abrir el mundo de Gazebo Ignition. Una de las acciones tiene el deber de activar la simulación a tiempo real con ROS, y la otra de abrir la interfaz gráfica.

- **imuclock_gz_bridge**: Este es un bridge de topics que se ejecuta al lanzar el spawner del mundo, el cual permite la inmediata transmisión del clock de la simulación.

XACROS

robot_description: En este paquete se encuentran los XACROs referentes a la estructura de todos los robots de Robotnik. En ello incluye su base y sus ruedas, además, también se encuentran los plugins de los controladores y la definición de sus joints como hardware.

robotnik_sensors: En este paquete se encuentran los XACROs de todos los sensores de los robots de Robotnik. Ahí se incluye la definición de la estructura y funcionalidad de los sensores, tanto para Ignition como para Classic.

Otros elementos

- **controller_params**: En estos archivos YAML se encuentran los parámetros de los controladores de cada robot.

- **rviz2_config**: Estos son archivos con configuraciones de rviz2 establecidas con anterioridad, las cuales se pueden usar para abrir el rviz con todos los ajustes necesarios para el robot que se quiere usar.

- **bridge_yaml_generator**: Este programa se encarga de leer todo los topics que genera Ignition y crear un archivo YAML con el bridge entre estos topics y ROS2, permitiendo así poder acceder a ellos desde fuera de la simulación.

- **bridges**: Los archivos YAML con los bridges entre los topics de Ignition y de ROS2

- **world**: El archivo que contiene la información del mundo de Gazebo Ignition que estamos lanzando.

Guía para migrar tu robot a Ignition

Para esta guía vamos a asumir que tienes un robot en URDF/XACRO funcional para Gazebo Classic.

Con esto en mente, voy a dividir esta guía en XACROS de sensores, Control y Bridges. Afortunadamente, de los XACROS de robot_description, sólo hay que modificar el urdf.xacro principal, todas las demás piezas del robot, en principio, no necesitan cambios a no ser que ya no funcionen de por sí en Classic.

XACRO general y sensores.

En cuanto al XACRO general del robot, los cambios necesarios se pueden realizar fácilmente usando de referencia los demás XACRO generales que hay creados. Lo más relevante que lo diferencia de los otros URDFs es algún parámetro con nombre diferente, algún argumento de más, y la importación, y posterior llamada, de un archivo nuevo de control, del que hablaremos posteriormente.

En cuánto a los Xacro de los sensores, lo principal que hay que saber es que, a diferencia de Gazebo Classic, aquí no hay plugins de ROS por sensor, si no que el plugin de los sensores se encuentra directamente en el archivo del mundo. Esto limpia bastante el código de los sensores, ya que el funcionamiento de publicación de topics está incorporado en los mismos argumentos del sensor de Ignition.

En robotnik_sensors hay bastantes XACROs de sensores ya migrados a Ignition que pueden servir de guía para transformar un sensor que no se encuentra allí, aunque en la mayoría de los casos sirve simplemente quitando el argumento plugin y añadiendo el frame y nombre del topic directamente sobre el sensor. En el robotnik_sensors subido a Github por la misma robotnik, hay más sensores migrados que en el mío.

```
<xacro:if value="${gazebo_ignition}">
  <gazebo reference="${frame_link}">
    <sensor type="gpu_lidar" name="${node_name}">
      <pose>0 0 0 0 0 0</pose>
      <visualize>false</visualize>
      <update_rate>${rate}</update_rate>
      <lidar>
        <scan>
          <horizontal>
            <samples>${horizontal_samples}</samples>
            <resolution>1</resolution>
            <min_angle>${-radians(180)}</min_angle>
            <max_angle>${radians(180)}</max_angle>
          </horizontal>
          <vertical>
            <samples>${vertical_samples}</samples>
            <resolution>1.0</resolution>
            <min_angle>${radians(min_angle)}</min_angle>
            <max_angle>${radians(max_angle)}</max_angle>
          </vertical>
        </scan>
        <range>
          <min>${min_range}</min>
          <max>${max_range}</max>
          <resolution>1</resolution>
        </range>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.03</stddev>
        </noise>
      </lidar>

      <topic> ${node_namespace}/${node_name}/scan </topic>
      <gz_frame_id> ${frame_link} </gz_frame_id>
    </sensor>
  </gazebo>
</xacro:if>

</xacro:macro>
```

XACRO de Control y parámetros.

Los XACRO de control funcionan de manera similar en Classic e Ignition, modificando simplemente el nombre del Plugin que estamos usando.

—
<plugin name="ros2_control" filename="libgazebo_ros2_control.so">

pasa a ser

<plugin name="ign_ros2_control::IgnitionROS2ControlPlugin"
filename="ign_ros2_control-system">
—

y
—
<ros2_control name="GazeboSystem" type="system">
        <hardware>
        <plugin>gazebo_ros2_control/GazeboSystem</plugin>
        </hardware>

pasa a ser

<ros2_control name="IgnitionSystem" type="system">
        <hardware>
        <plugin>ign_ros2_control/IgnitionSystem</plugin>
        </hardware>
—

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://wiki.ros.org/xacro">
    <xacro:macro name="rbrobout_gz_ignition_control"  params="namespace prefix">
        <gazebo>
          <plugin name="ign_ros2_control::IgnitionROS2ControlPlugin" filename="ign_ros2_control-system">
            <robot_param>robot_description</robot_param>
            <robot_param_node>robot_state_publisher</robot_param_node>
            <parameters>$(find robot_description)/simulators/gazebo_ignition/rbrobout/rbrobout_controller_params.yaml</parameters>
            <ros>
                <namespace>/${namespace}</namespace>
            </ros>
          </plugin>
        </gazebo>

        <xacro:macro name="ros2_joint" params="interface jmin jmax name">
          <joint name="${name}">
            <command_interface name="${interface}">
              <param name="min">${jmin}</param>
              <param name="max">${jmax}</param>
            </command_interface>
            <state_interface name="position"/>
            <state_interface name="velocity"/>
            <state_interface name="effort"/>
          </joint>
        </xacro:macro>

        <ros2_control name="IgnitionSystem" type="system">
          <hardware>
              <plugin>ign_ros2_control/IgnitionSystem</plugin>
          </hardware>

          <xacro:ros2_joint interface="velocity" jmin="-10" jmax="10" name="robot_front_left_wheel_joint"/>
          <xacro:ros2_joint interface="velocity" jmin="-10" jmax="10" name="robot_front_right_wheel_joint"/>
          <xacro:ros2_joint interface="velocity" jmin="-20" jmax="20" name="robot_back_left_wheel_joint"/>
          <xacro:ros2_joint interface="velocity" jmin="-20" jmax="20" name="robot_back_right_wheel_joint"/>
          <xacro:ros2_joint interface="position" jmin="-20" jmax="20" name="robot_lift_ewellix_lift_top_joint"/>

        </ros2_control>
    </xacro:macro>
</robot>
```

En cuanto a los parámetros de control, estos se mantienen exactamente igual, sin modificación alguna, ya que estos dependen exclusivamente de los controladores, los cuales son externos a la simulación.

Bridges de topics entre Ignition y ROS2

En Ignition, los sensores publican en un espacio interno de Ignition, al cual se puede acceder con ign topic -l, esto impide el uso sencillo y rápido de esta información en otros nodos y sistema de ROS.

Para sacar los topics fuera de la simulación, se necesita hacer un Bridge, el cual es un puente que conecta los topics de Ignition con los de Ros2.

En Gazebo Classic, usando los plugins en los sensores, se podía fácilmente publicar en ROS2, pero al tener que trabajar con Bridges, el funcionamiento se hace mucho más engorroso, ya que tienes que establecer todos los topics que quieres convertir desde un inicio.

Los bridges se pueden inicializar de dos formas diferentes.

1. La primera es crear un Bridge de cada uno de los topics de los sensores. Esto, en robots con pocos topics publicando es eficaz, pero cuando escalamos a robots más complejos, se hace muy engorroso.

2. La segunda forma que existe es usando un yaml, en el que se encuentren todos los topics que queremos transformar ya establecidos. Esto es lo que Ignition quiere que uses para robots más complejos, aunque el peso sobre la simulación que tiene hace bajar el RTF muchísimo, principalmente cuando tienes topics de lidars o nubes de puntos, los cuales transmiten muchísimos datos en poco tiempo.

```yaml
# bridge_params.yaml

- gz_topic_name: "imu/data"
  ros_type_name: "sensor_msgs/msg/Imu"
  gz_type_name: "ignition.msgs.IMU"
  direction: "GZ_TO_ROS"

- gz_topic_name: "front_laser/scan"
  ros_type_name: "sensor_msgs/msg/LaserScan"
  gz_type_name: "ignition.msgs.LaserScan"
  direction: "GZ_TO_ROS"

- ros_topic_name: "/clock"
  gz_topic_name: "/clock"
  ros_type_name: "rosgraph_msgs/msg/Clock"
  gz_type_name: "ignition.msgs.Clock"
  direction: "GZ_TO_ROS"

- gz_topic_name: "rear_laser/scan"
  ros_type_name: "sensor_msgs/msg/LaserScan"
  gz_type_name: "ignition.msgs.LaserScan"
  direction: "GZ_TO_ROS"

- gz_topic_name: "front_camera_color/color/image_raw"
  ros_type_name: "sensor_msgs/msg/Image"
  gz_type_name: "gz.msgs.Image"
  direction: "GZ_TO_ROS"

- gz_topic_name: "front_camera_color/color/camera_info"
  ros_type_name: "sensor_msgs/msg/CameraInfo"
  gz_type_name: "ignition.msgs.CameraInfo"
  direction: "GZ_TO_ROS"

- gz_topic_name: "rear_camera_color/color/image_raw"
  ros_type_name: "sensor_msgs/msg/Image"
  gz_type_name: "gz.msgs.Image"
  direction: "GZ_TO_ROS"
```

En Robotnik, un compañero y yo vimos que crear un archivo yaml para cada robot y cada situación era algo bastante engorroso, por lo que creamos un programa que lee los topics que publica en terminal Ignition con ign topic -l y crea un archivo yaml de forma automática. Este archivo, posteriormente, es pasado al nodo de ros_gz_bridge, el cual se encarga de crear el bridge para todos los topics del yaml.

Ahora bien, entonces ¿Hay que modificar algo? Pues según lo que quieras hacer. Si quieres usar un yaml creado por ti desde cero, deberías crearlo dentro de la carpeta config en el paquete robotnik_gazebo_ignition, en el directorio correspondiente a tu robot. Pero siempre puedes usar el programa automático de creación de yaml, que es con lo que funciona el launcher spawn_robot_dinamicbridge.py

Posibles errores

Estos son posibles errores que puede dar el proceso de migración den un nuevo robot:

1. **El controlador no se conecta**. Este problema se da cuando spawneas un robot nuevo, y en la terminal, sale en rojo diciéndote que no ha podido conectarse al robot. Esto puede ser debido por 3 situaciones:
    - Los parámetros del controlador son erróneos.
    - Los parámetros del controlador no coinciden con la versión del controlador instalada. Errores tales que el nombre del controlador no cuadra o algún argumento no existe.
    - Se ha spawneado el robot con un RTF de simulación extremadamente bajo. Esto puede ser causado, por ejemplo, al spawnear un segundo robot justo después de la creación de un bridge, ya que en esta situación el RTF suele bajar drásticamente. Para evitar esta situación, espera a que el RTF se estabilice para lanzar un nuevo robot (puede tardar varios minutos)

2. **El robot_description no va**. Este error se suele especificar su motivo de forma clara en terminal, pero aquí hay posibles motivos:
    - Las rutas de los XACROs son incorrectas.
    - XACROs de diferentes versiones de robot_description o robotnik_description (este cambio de nombre también puede ser un error) puede causar problemas de argumentos y cambios de nombres. Recomiendo seguir las indicaciones de la terminal para arreglar estos fallos.
    - Utilizas sensores de Gazebo_classic, por lo cual te pide plugins que no existen en tu instalación de Ignition.

Esto sería todo. Buena suerte.