IMPLEMENTACIÓN DE ROBOTS EN O3DE Y CONEXIÓN CON ROS2

Requerimientos del sistema

O3DE usa para compilar CMake >=3.22.0 para ver la versión de Cmake:

cmake --version

Si es una versión más antigua hay que instalar la nueva versión:

sudo apt remove cmake

sudo apt install cmake

Se necesita instalar Clang >=12 para ello podemos instalar la versión de Clang default de Ubuntu 22.04 que es la versión 14.

sudo apt install libstdc++-12-dev clang

Otras dependencias necesarias:

sudo apt install libglu1-mesa-dev libxcb-randr0-dev libxcb-xinerama0 libxcb-xinput0 libxcb-xinput-dev libxcb-xfixes0-dev libxcb-xkb-dev libxkbcommon-dev libxkbcommon-x11-dev libfontconfig1-dev libpcre2-16-0 zlib1g-dev mesa-common-dev libunwind-dev libzstd-dev tix

Hay que instalar el sistema build Ninja para seguir los mismos pasos que he realizado:

sudo apt install ninja-build

Instalación

A continuación se explicará la forma de instalar o3de en un sistema Linux Ubuntu 22.04 en el cual está instalado ros2 humble.Hay dos tipos de instalación de O3DE:

- Descargar una versión precompilada del motor con unos determinados GEMS/PLUGINS que vienen ya instalados de serie mediante un .deb .
- Descargar el código fuente del motor desde Github y luego compilarlo junto a los GEMS que queramos añadirle a nuestro proyecto.

Vamos a realizar la instalación mediante la segunda opción ya que nos permite elegir la versión del motor que queremos utilizar,podemos añadir otras GEMS aparte de las que vienen por defecto con el precompilado y en general es una forma más flexible y personalizable de crear un motor de O3DE.Además,actualmente la versión que viene del motor precompilado está anticuada y usando la GEM de ROS2 se consigue una simulación con bajos FPS(9/12 usando solo un rb_watcher) frente a 50/60 FPS con el motor 4.2 de O3DE.

Instalar Git LFS

El repositorio de GitHub de O3DE utiliza el sistema Git Large File Storage (LFS) para almacenar archivos binarios grandes. Las siguientes instrucciones preparará tu PC para autenticar y descargar estos archivos automáticamente cuando clones, obtengas (fetch) o descargues (pull) desde el repositorio.

sudo apt install git-lfs

https://www.docs.o3de.org/docs/welcome-guide/setup/setup-from-github/

Fork and clone (dentro del directorio donde queremos guardar O3DE en mi caso es $HOME)

git clone https://github.com/YOUR-Github-USERNAME/o3de.git(HAY QUE DESCARGAR LA VERSIÓN DE LA BRANCH DEFALUT DEL 4 DE MARZO)

Esto creará un directorio o3de y un directorio oculto .o3de dentro del directorio donde se ha hecho el clone.

cd o3de

git lfs pull

Una vez hecho lo anterior tendremos el código fuente de un motor de O3DE. Para crear un proyecto que use el plugin de ROS2 para implementar robots tenemos que crear un proyecto que use ese motor y la GEM de ROS2 y luego compilar todo el proyecto.A continuación se explicarán los pasos para hacer el setup de la GEM de ROS2 y la creación de un proyecto.

https://www.docs.o3de.org/docs/user-guide/interactivity/robotics/project-configuration/

source /opt/ros/humble/setup.bash

Paquetes necesarios de ros2

sudo apt install ros-${ROS_DISTRO}-ackermann-msgs ros-${ROS_DISTRO}-control-msgs ros-${ROS_DISTRO}-nav-msgs ros-${ROS_DISTRO}-gazebo-msgs ros-${ROS_DISTRO}-xacro ros-${ROS_DISTRO}-vision-msgs

Clonar el repositorio donde se encuentra la GEM de ROS2

git clone https://github.com/o3de/o3de-extras

Registrar la GEM de ROS2 en O3DE

export O3DE_HOME=${HOME}/o3de

export O3DE_EXTRAS_HOME=${HOME}/o3de-extras

${O3DE_HOME}/python/get_python.sh

${O3DE_HOME}/scripts/o3de.sh register --gem-path ${O3DE_EXTRAS_HOME}/Gems/ROS2

Los Templates(plantillas) de robótica pueden ayudarte a iniciar rápidamente tu proyecto de simulación. Te recomendamos que registres los Templates y sus Asset Gems, que descargaste con el repositorio o3de-extras.

cd ${O3DE_EXTRAS_HOME}

git lfs install && git lfs pull

${O3DE_HOME}/scripts/o3de.sh register --all-gems-path ${O3DE_EXTRAS_HOME}/Gems/

${O3DE_HOME}/scripts/o3de.sh register --all-templates-path ${O3DE_EXTRAS_HOME}/Templates/

Con el parámetro register del script o3de.sh lo que hacemos es modificar un archivo llamado o3de_manifest.json en el que se describe la cantidad de motores de los que dispone O3DE y sus rutas , las GEMS,Templates y Assets que se pueden utilizar,etc. Es un archivo de configuración muy importante que si se modifica sin cuidado puede hacer que no se pueda compilar ningún proyecto.

Mi archivo o3de_manifest.json con 2 motores(el precompilado y el creado desde código fuente):


```
{
    "o3de_manifest_name": "miningdox",
    "origin": "/home/miningdox/.o3de",
    "default_engines_folder": "",
    "default_projects_folder": "",
    "default_gems_folder": "",
    "default_templates_folder": "",
    "default_restricted_folder": "",
    "default_third_party_folder": "/home/miningdox/.o3de/3rdParty",
    "projects": [
        "/home/miningdox/projects/MySimulationProject"
    ],
    "external_subdirectories": [
        "/home/miningdox/o3de-extras/Gems/AudioEngineWwise",
        "/home/miningdox/o3de-extras/Gems/AzQtComponentsForPython",
        "/home/miningdox/o3de-extras/Gems/LevelGeoreferencing",
        "/home/miningdox/o3de-extras/Gems/MachineLearning",
        "/home/miningdox/o3de-extras/Gems/OpenXRVk",
        "/home/miningdox/o3de-extras/Gems/ProteusRobot",
        "/home/miningdox/o3de-extras/Gems/ROS2",
        "/home/miningdox/o3de-extras/Gems/RosRobotSample",
        "/home/miningdox/o3de-extras/Gems/WarehouseAssets",
        "/home/miningdox/o3de-extras/Gems/WarehouseAutomation",
        "/home/miningdox/o3de-extras/Gems/WarehouseSample",
        "/home/miningdox/o3de-extras/Gems/XR"
    ],
    "templates": [
        "/home/miningdox/o3de-extras/Templates/Multiplayer",
        "/home/miningdox/o3de-extras/Templates/Ros2FleetRobotTemplate",
        "/home/miningdox/o3de-extras/Templates/Ros2ProjectTemplate",
        "/home/miningdox/o3de-extras/Templates/Ros2RoboticManipulationTemplate"
    ],
    "restricted": [],
    "repos": [],
    "engines": [
        "/opt/O3DE/24.09.2",
        "/home/miningdox/o3de"
    ],
```

```
    "engines_path": {
        "o3de": "/home/miningdox/o3de",
        "o3de-sdk": "/opt/O3DE/24.09.2"
    }
}
```

Si se ha corrompido el archivo por algún motivo o lo hemos modificado mal,tenemos dos opciones:

1. Dejar el archivo vacío en cuanto a parámetros y registrar los motores ,gems,assets y templates que necesitamos que o3de reconozca mediante el script o3de.sh con el parámetro register.

Archivo o3de_manifest.json sin parámetros únicamente con la ruta al motor o3de:

```
{
  "o3de_manifest_name": "miningdox",
  "origin": "/home/miningdox/.o3de",
  "default_engines_folder": "",
  "default_projects_folder": "",
  "default_gems_folder": "",
  "default_templates_folder": "",
  "default_restricted_folder": "",
  "default_third_party_folder": "/home/miningdox/.o3de/3rdParty",
  "projects": [],
  "external_subdirectories": [],
  "templates": [],
  "restricted": [],
  "repos": [],
  "engines": [
    "/home/miningdox/o3de"
  ],
  "engines_path": {
    "default": "/home/miningdox/o3de",
    "custom": ""
  }
}
```

2. Borrar la carpeta o3de y ./o3de , clonar otra vez o3de y realizar todos los registers necesarios mencionados anteriormente.(Opción recomendada ya que nos aseguramos que esté todo igual que al inicio y podemos seguir los pasos en orden)

Una vez tenemos O3DE configurado con los elementos necesarios,vamos a crear un proyecto.Tenemos 2 formas de crear proyectos.Pero explicaremos solo la primera ya que la primera se podría usar para automatizar el proceso y la segunda es más intuitiva y da problemas como por ejemplo que se puede usar para crear proyectos pero no deja abrirlos, además de que puede corromper el manifest fácilmente.

1. **Mediante línea de comandos.**

Nombre del proyecto

export PROJECT_NAME=MySimulationProject

Definición de la ruta dónde se va a almacenar y creación.

export PROJECT_PATH=${HOME}/projects/${PROJECT_NAME}

${O3DE_HOME}/scripts/o3de.sh create-project --project-path $PROJECT_PATH --template-path ${O3DE_EXTRAS_HOME}/Templates/Ros2ProjectTemplate

Compilación

cd $PROJECT_PATH

cmake -B build/linux -G "Ninja Multi-Config" -DLY_DISABLE_TEST_MODULES=ON -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -DLY_STRIP_DEBUG_SYMBOLS=ON

cmake --build build/linux --config profile --target ${PROJECT_NAME} Editor ${PROJECT_NAME}.Assets

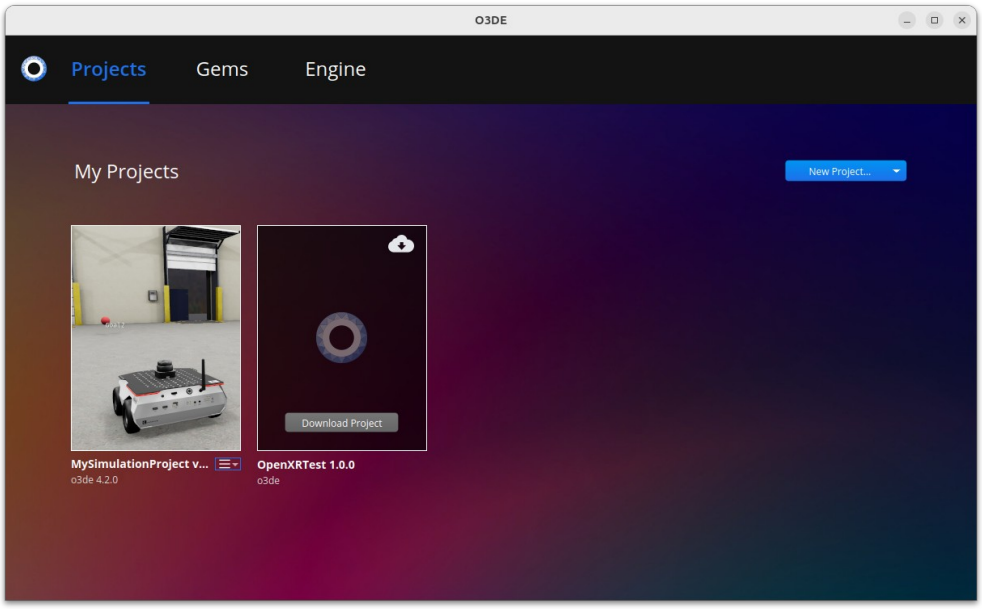
Nota: Antes de la versión 24.09.0, PhysX 5 era experimental y se compilaba durante el proceso de compilación del código fuente del motor. Si estás utilizando la versión 23.10.3 o una versión anterior, necesitarás especificar una bandera adicional: -DAZ_USE_PHYSX5:BOOL=ON.

cmake -B build/linux -G "Ninja Multi-Config" -DLY_DISABLE_TEST_MODULES=ON -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -DLY_STRIP_DEBUG_SYMBOLS=ON -DAZ_USE_PHYSX5:BOOL=ON

2.Mediante Gui

Abrimos project manager

cd o3de/build/linux/bin/profile/o3de



Abrir nuestro proyecto

source /opt/ros/humble/setup.bash

cd projects/MySimulationProject/build/linux/bin/profile

./Editor

Importación de modelos

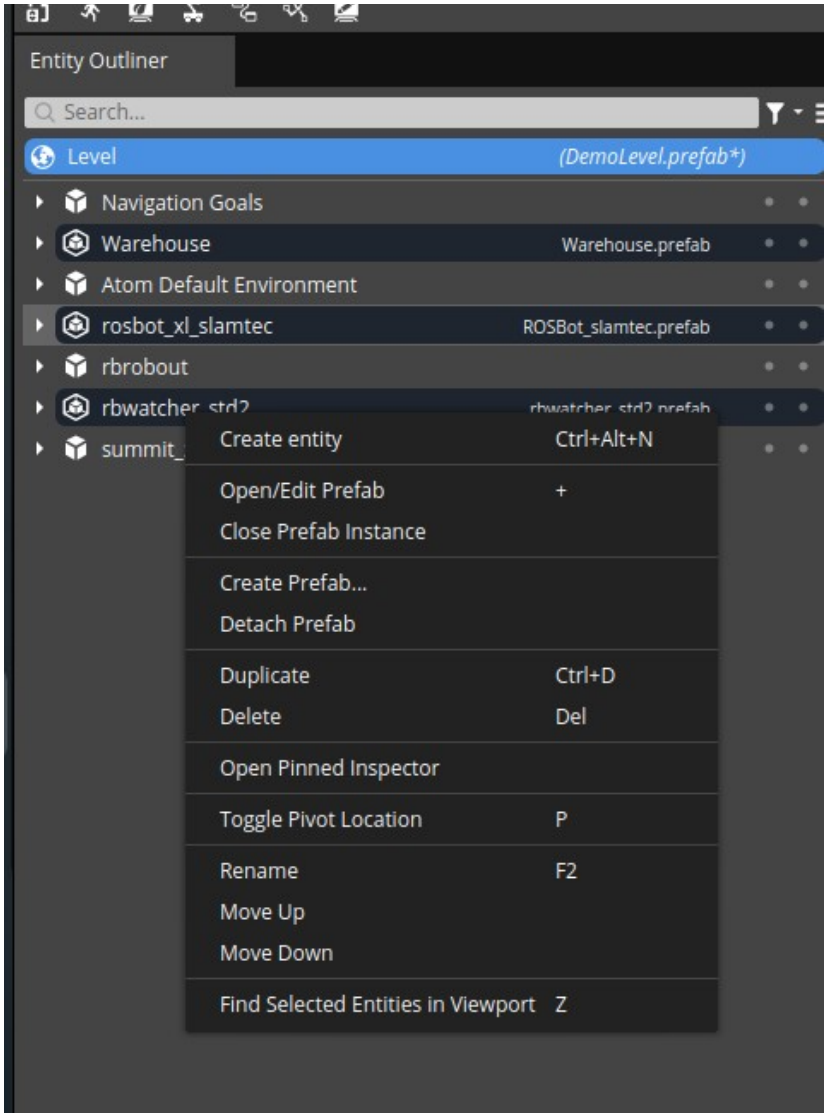
Para importar un modelo de robot a partir de un urdf hay que hacer click en la esquina superior izquierda **en el siguiente botón**  y buscar el archivo urdf en el sistema.Una vez hecho esto,se habrá añadido el robot a la escena de la simulación.El modelo del robot resultante no tendrá apenas sensores y si los tiene gran parte de ellos tendrá los frames girados.

Guardar un modelo modificado

No es muy intuitivo cómo guardar un prefab importado en el mundo y editado después de ser importado.

1. Desvincular prefab del nivel en el Editor
2. clic derecho en el ICONO(sino no aparece el menú) de la entidad dentro de Entity Outliner + Detach Prefab(me he dado cuenta que el menú sale cuando le da la gana,así que hay que darle doble click a la entidad hasta que salga).

Menú que tiene que salir:



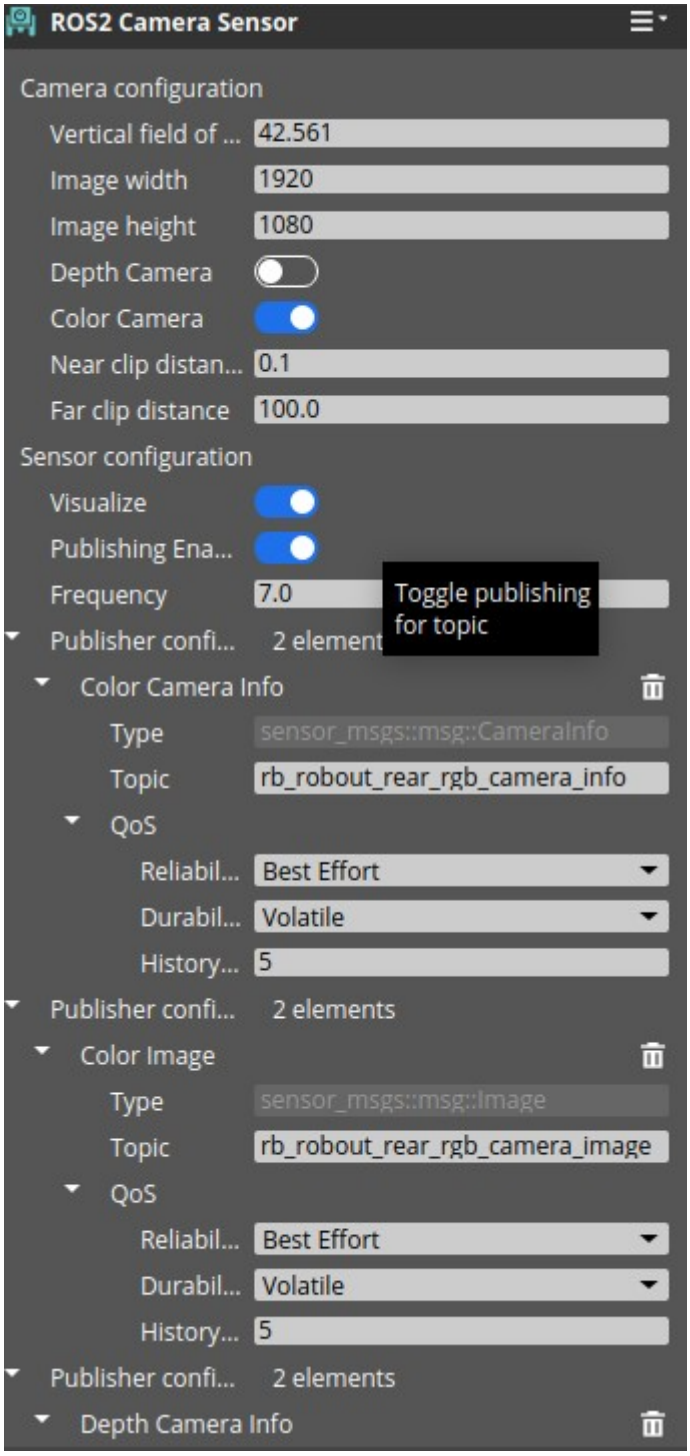
3. Click derecho en la entidad desvinculada + Create Prefab

Sensores

En general agregar sensores no es muy complicado en o3de con la GEM de ROS2 ya que disponemos de todos los sensores necesarios para implementar un robot en la simulación.


ROS2
Ackermann Control
Ackermann Vehicle Model
FingerGripperComponent
GripperActionServerComponent
Image Encoding Conversion Component
JointsArticulationControllerComponent
JointsManipulationEditorComponent
JointsPIDControllerComponent
JointsPositionsEditorComponent
JointsTrajectoryComponent
Manual Motor Controller
PID Motor Controller
ROS2 Camera Sensor
ROS2 Contact Sensor
ROS2 Frame
ROS2 GNSS Sensor
ROS2 Imu Sensor
ROS2 Lidar 2D Sensor
ROS2 Lidar Sensor
ROS2 Odometry Sensor
ROS2 Robot Control
ROS2 Spawn Point
ROS2 Spawner
ROS2 Wheel Odometry Sensor
Rigid Body Twist Control
Skid Steering Twist Control
Skid Steering Vehicle Model
VacuumGripperComponent
Wheel Controller

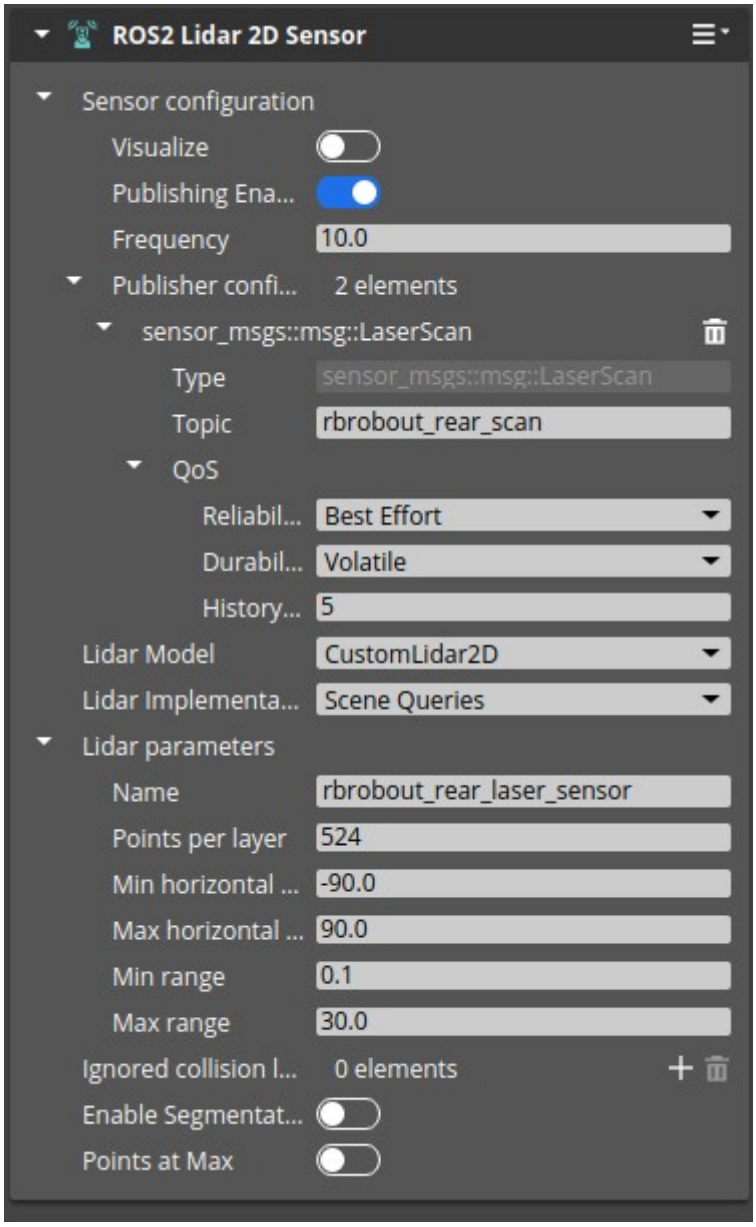
Cámaras



Este complemento tiene los parámetros Depth camera y Color camera, si están activados se utilizarán los parámetros inferiores que describen ambas cámaras.Si están desactivados, aunque los parámetros están definidos no se utilizan y no hay ningún tipo de cámara(no hay retroalimentación visual de su activación en los parámetros,como por ejemplo que estén oscurecidos).Básicamente tenemos que especificar su resolución,el nombre del tópico en el

que publican y la política de Qos de los mensajes.Hay que visualizar cada cámara mediante rviz para asegurarnos que esté orientada adecuadamente.

LIDAR 2D



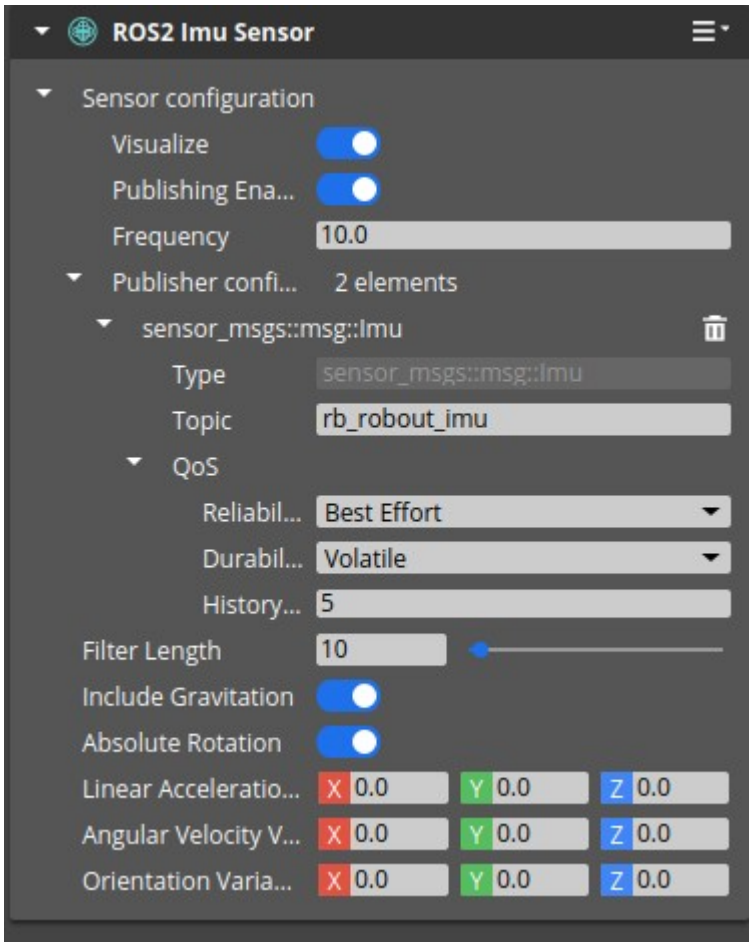
Para implementar el LIDAR 2D usaremos este componente,el importer URDF no detecta este tipo de sensor,hay que agregarlo manualmente al objeto correspondiente.
Además de configurar su tópico de publicación,tenemos que definir lo siguiente:

- **Points per layer:** los puntos que usa para detectar los objetos en su única capa.
- **Min/max horizontal range:** el ángulo de apertura del sensor, el ángulo 0 corresponde a la unión entre el plano x e y.
- **Min/Max range** : la distancia a la que detecta los objetos.

- **Lidar Model**: con el valor CustomLidar2D para poder modificar los parámetros anteriores.

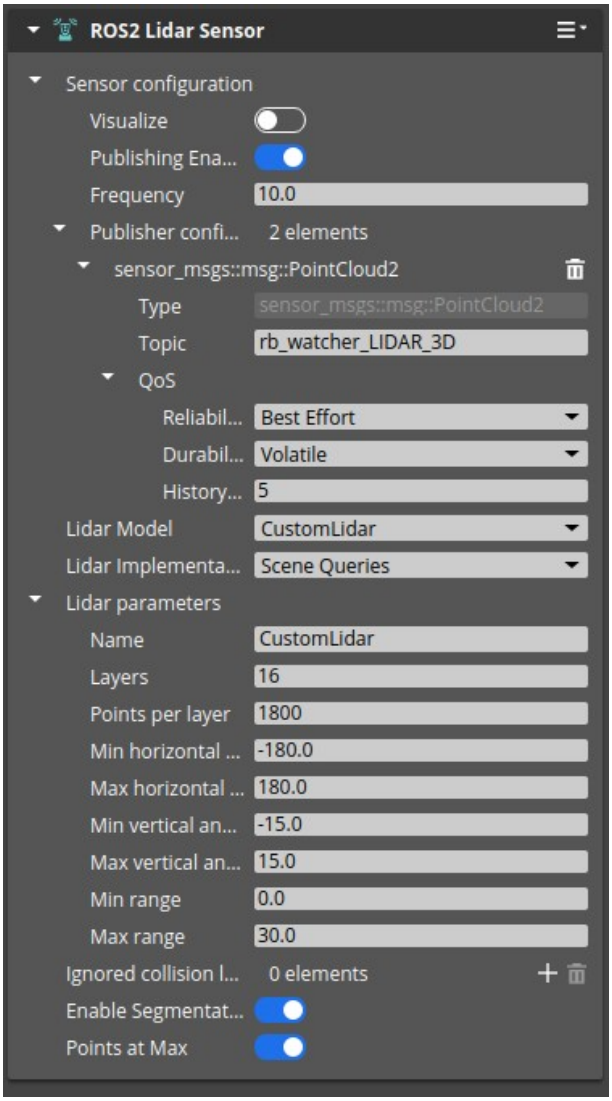
Importante: El parámetro **Publish enable** debe estar activado para publicar los datos.El parámetro **Visualize** no funciona ya que aún habilitado no muestra los puntos en la simulación.Si el LIDAR 2D lo situamos dentro de una malla colisiona con ella aún si no tiene colisionador,para solucionar esto tenemos que situar el LIDAR ligeramente fuera de la malla.Otra forma sería definir capas de colisión para todos los objetos que queremos que ignore el láser y adjuntarlos a ignored collision layer pero en documentación no he encontrado cómo se realiza.

IMU



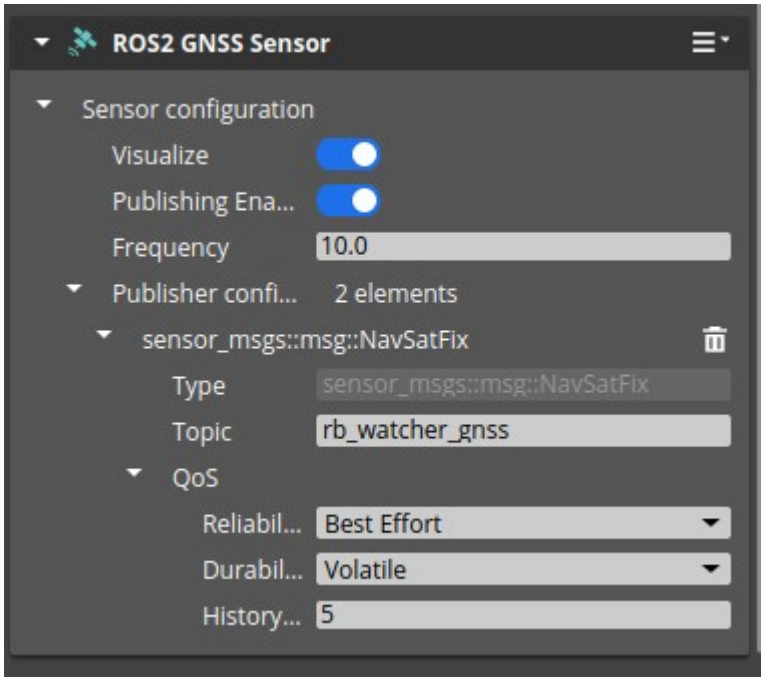
Por defecto el componente IMU viene configurado correctamente para su funcionamiento, pero podemos configurarlo para que tenga un comportamiento similar a uno real modificando las variaciones de su aceleración lineal, velocidad angular y de la orientación.La variación se refiere a la variabilidad aleatoria que se aplica a esos datos.

LIDAR 3D

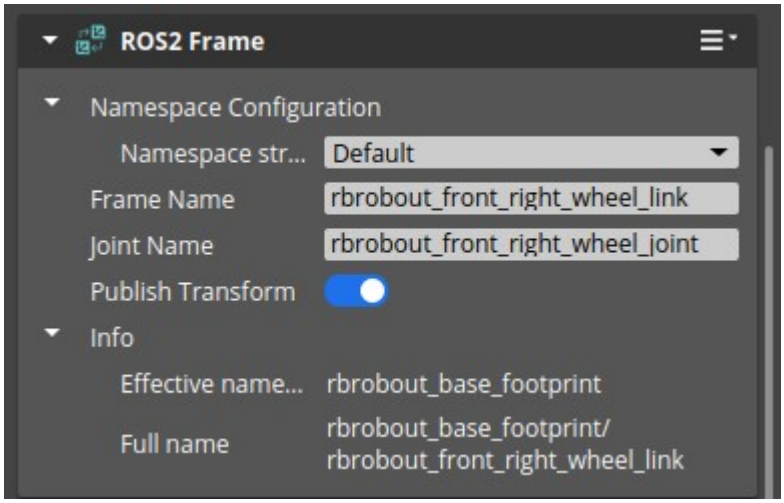


De manera similar al Lidar 2d tenemos que especificar que es un CustomLidar para poder personalizar sus especificaciones como las capas que tiene,los puntos por capa y sus rangos de alcance y apertura.

GPS



TF



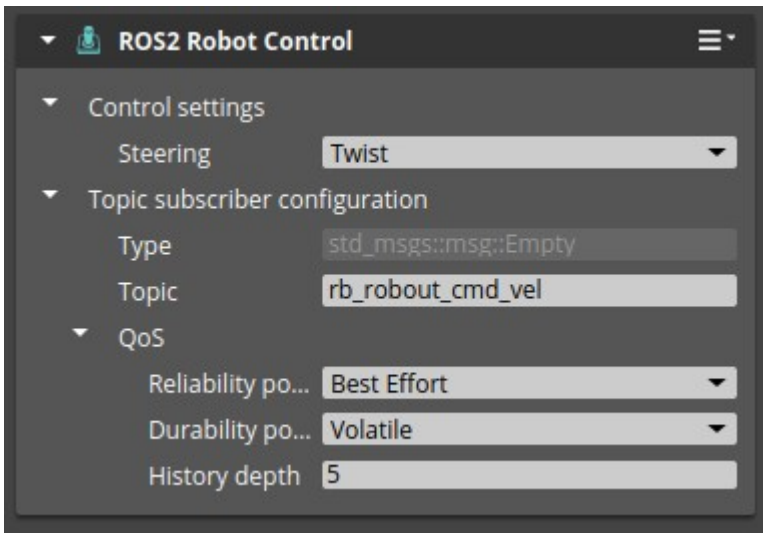
Para las tfs de los frames se usa el componente ROS2 Frame.Cuando se usa el importer se detectan todos los frames con tf y sus nombres.Lo que debemos modificar es el parámetro Namespace por el valor Default y darle el mismo nombre que el frame para no tener nombres de tópicos enlazados como los siguientes:

```
/rbrobout_base_footprint/rbrobout_front_scan
/rbrobout_base_footprint/rbrobout_imu_link/rb_robout_imu
/rbrobout_base_footprint/rbrobout_rear_scan
/robot_base_footprint/rb_watcher_cmd_vel
/robot_base_footprint/rb_watcher_gnss
/robot_base_footprint/robot_antenna_base_link/robot_top_3d_laser_base_link/robot_top_3d_laser_link/robot_top_3d_laser_sensor_
ensor/rb_watcher_LIDAR_3D
/robot_base_footprint/robot_front_rgbd_camera_base_link/robot_front_rgbd_camera_link/robot_front_rgbd_cameracolor_sensor/rb_w
tcher_front_camera_color
/robot_base_footprint/robot_front_rgbd_camera_base_link/robot_front_rgbd_camera_link/robot_front_rgbd_cameracolor_sensor/rb_w
tcher_front_camera_color_info
/robot_base_footprint/robot_front_rgbd_camera_base_link/robot_front_rgbd_camera_link/robot_front_rgbd_cameradepth_sensor/came
a_image_depth
/robot_base_footprint/robot_front_rgbd_camera_base_link/robot_front_rgbd_camera_link/robot_front_rgbd_cameradepth_sensor/dept
_camera_info
/robot_base_footprint/robot_imu_link/rb_watcher_imu
/robot_base_footprint/robot_top_ptz_camera_base_link/robot_top_ptz_camera_pan_link/robot_top_ptz_camera_tilt_link/robot_top_p
z_camera_frame_link/robot_top_ptz_camera_sensor_sensor/rb_watcher_top_camera
/robot_base_footprint/robot_top_ptz_camera_base_link/robot_top_ptz_camera_pan_link/robot_top_ptz_camera_tilt_link/robot_top_p
z_camera_frame_link/robot_top_ptz_camera_sensor_sensor/rb_watcher_top_camera_info
/rosout
/tf
/tf_static
```

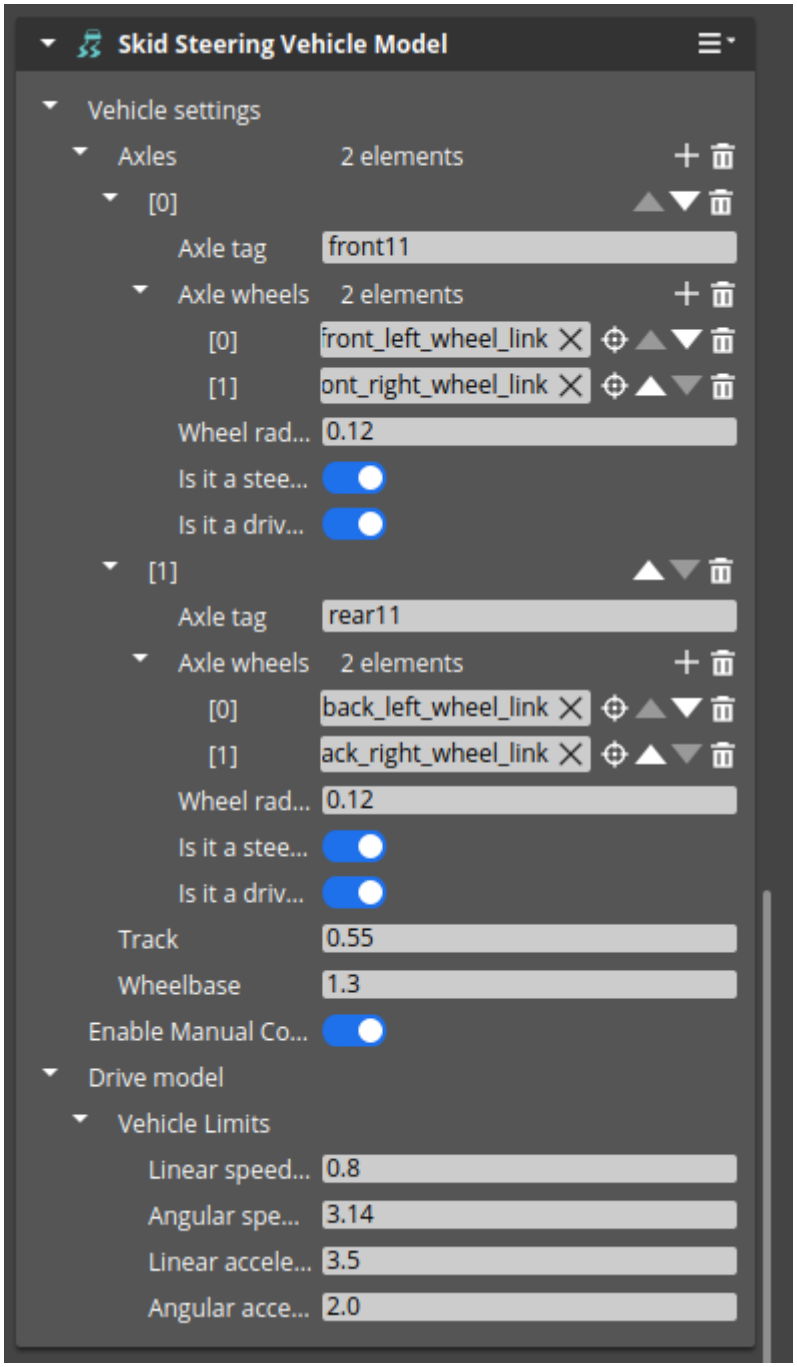
Control

Para el control debemos situar varios componentes en diferentes frames del robot.

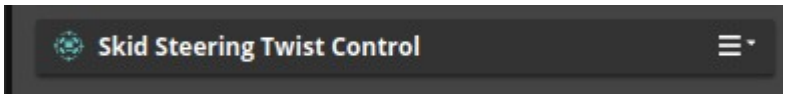
En base footprint



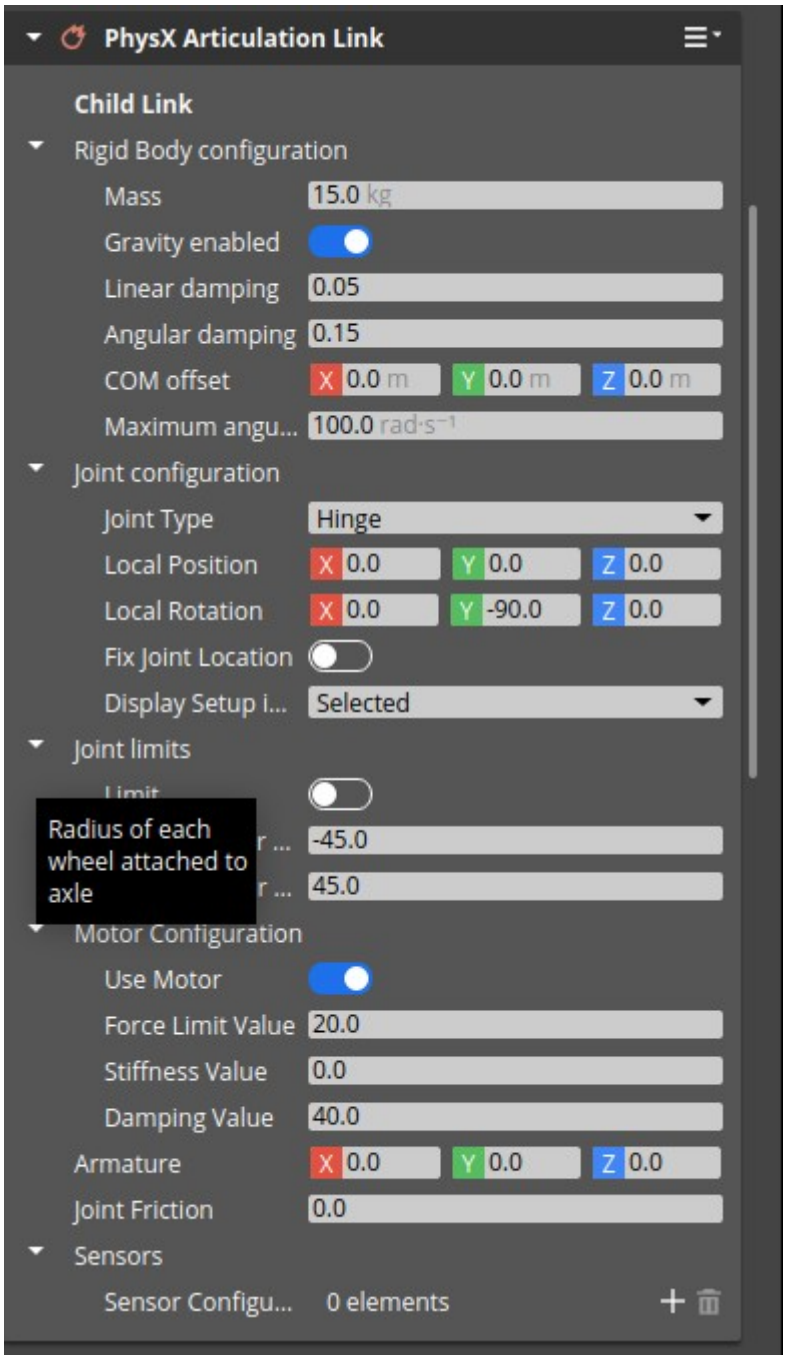
Componente **Robot Control** con el parámetro Steering con el valor Twist,el componente **Skid Steering Vehicle Model** y el componente **Skid Steering Twist Control**.



Debemos hacer clik en en icono + de Axles para agregar cada eje.Además en cada eje hay que especificar los objetos que se corresponden a las ruedas izquierda [0] y derecha [1].Para especificarlos se hace click en el icono de mirilla y después en la jerarquía de entidades se selecciona el objeto de la rueda.También tenemos que definir las longitudes en metros de :**Track** (distancia entre las ruedas del mismo eje, medida de lado a lado) y **Wheelbase** (la distancia entre el eje delantero y el eje trasero de un vehículo).



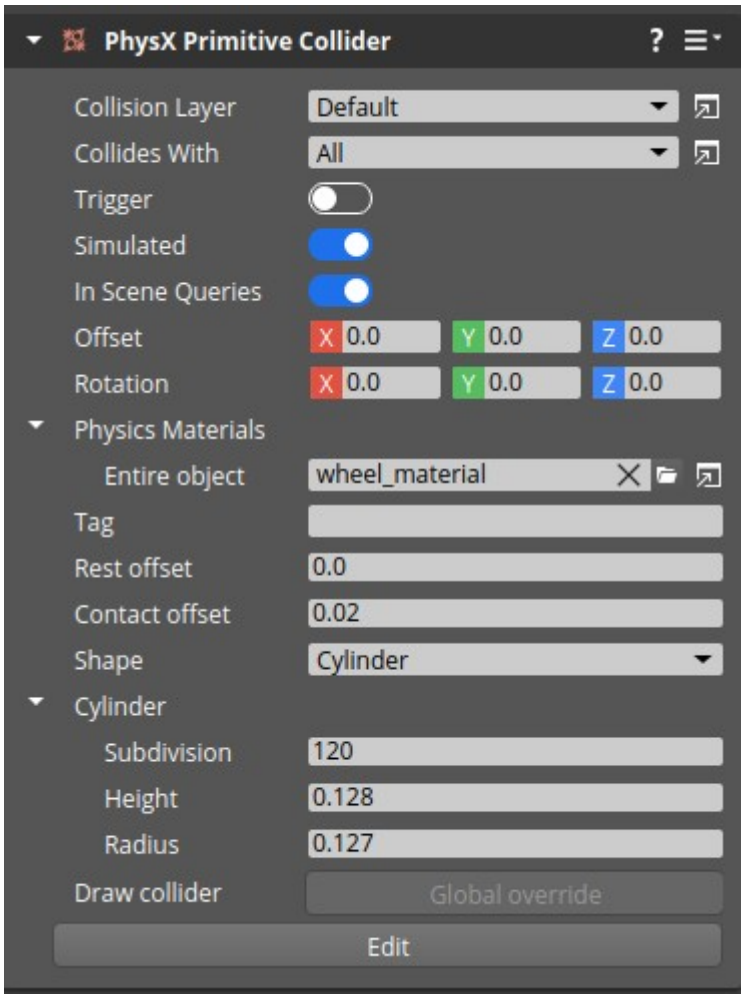
En cada rueda



En este componente tenemos que modificar:

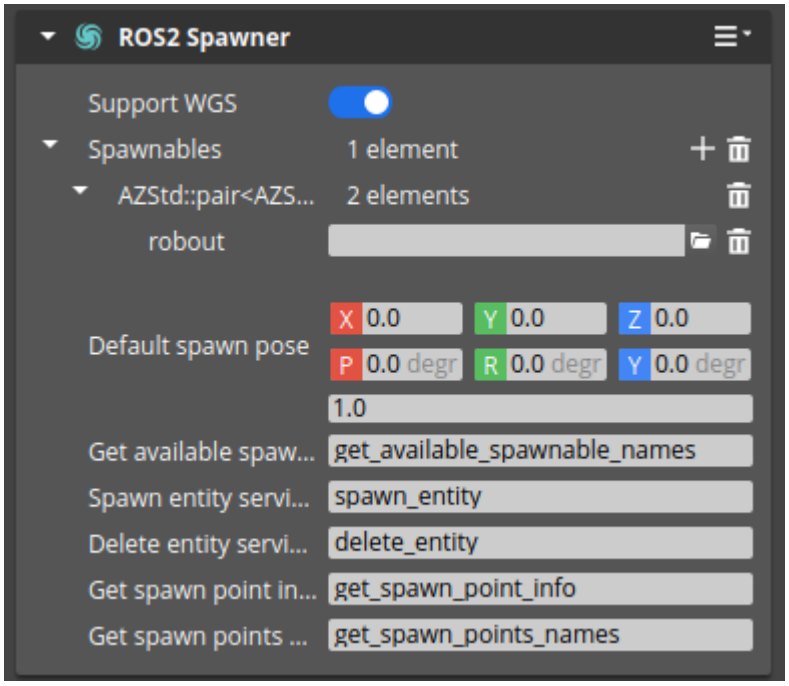
- **Local Rotation** para que al visualizar la rueda se muestre el giro en la dirección adecuada.Por defecto suele estar girada.
- Habilitar **Use Motor** y definir los parámetros **Force Limit Value** (para el límite de torque que puede aplicar el motor), dejar **Stiffness Value**(controla cuán fuerte empuja el sistema para volver a una posición original) a 0 y probar con un **Damping Value**(controla cuán rápido se detiene después de moverse) alto como por ejemplo 40.

Por defecto los parámetros no mencionados ya los coge correctamente del urdf del robot.

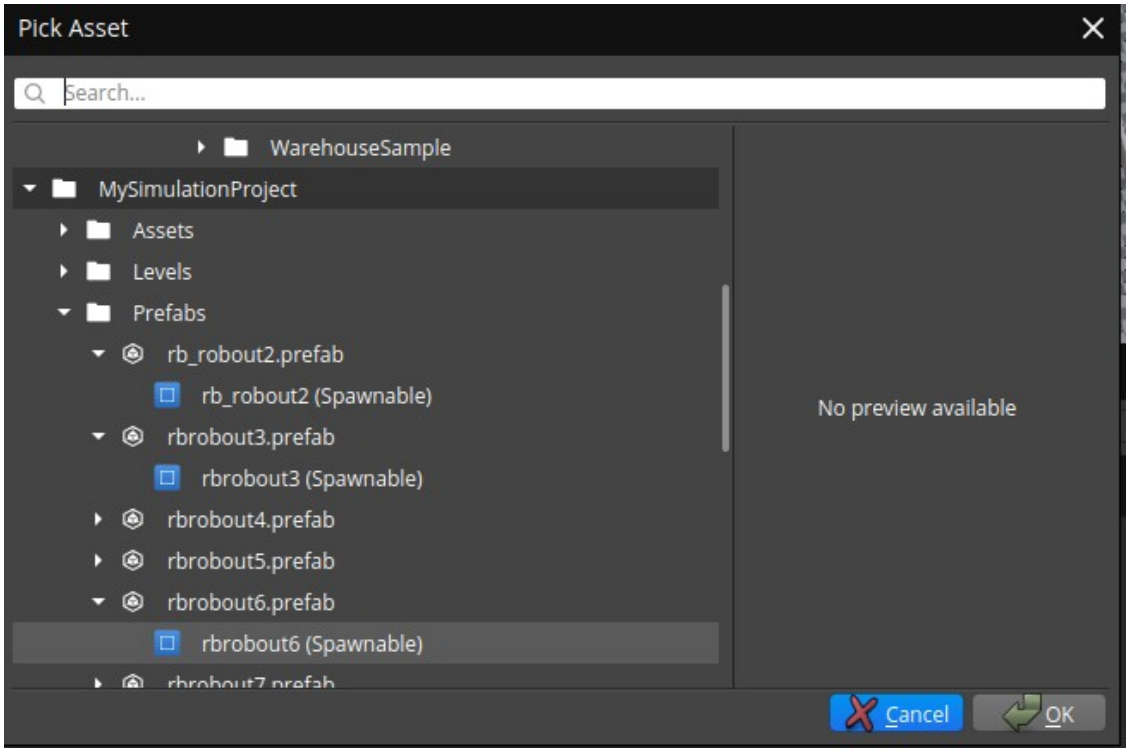


Spawn

Se agrega un elemento ROS2 Spawner al elemento del nivel donde se quiere que se pueda spawnear los modelos de los robots.



Haced click en el icono + en Spawneables y seleccionar el modelo que se quiere spawnear.



Se usa el spawner mediante el servicio de gazebo,por ejemplo:

$ ros2 service call /spawn_entity gazebo_msgs/srv/SpawnEntity "{name: 'watcher', initial_pose: {position:{ x: 5.0, y: -12.0, z: 0.0 }, orientation: { x: 0.0, y: 0.0, z: 0.0, w: 0.1 } } }"

Cómo usar los modelos y las escenas ya hechos

Para usar escenas:

1. Descargar la escena.
2. Moverla a MySimulationProject/Levels .
3. File/Open Level (manual)

Para usar robots:

1. Descargar modelo del robot.
2. Mover a MySimulationProject/Prefabs.
3. Click Derecho en Entity Outliner
4. Instantiate Prefab
5. Buscamos dónde se encuentra el prefab y lo seleccionamos.
6. Copiamos el contenido de la carpeta Assets en la carpeta assets de nuestro proyecto para tener todas las mallas necesarias para representar el robot.

ATENCIÓN: O3DE se bugea y se sale del Editor si se guarda algún parámetro desde el Editor.Para ello hay que asegurarse de guardar la escena cada poco tiempo ya que muchas veces de sale hasta de la sesión de Ubuntu.