IMPLEMENTACIÓN DE ROBOTS EN COPPELIA Y CONEXIÓN CON ROS2

Instalación

https://www.coppeliarobotics.com/
Hay diferentes tipos de simuladores para descargar:

- Versión lite: para uso comercial pero sin todas las capacidades de edición.
- Versión Pro: todas las funcionalidades y para fines comerciales.
- Versión edu: todas las funcionalidades pero sin fines comerciales.

Hay que instalar la versión Edu.

Abrir Coppelia

Para poder usar ros2,en la terminal de inicio del simulador hay que hacer:

1. **source /opt/ros/humble/setup.bash**
2. **cd /mi/directorio/instalacion/de/coppelia**
3. **./coppeliaSim.sh**

Importación de modelos

Para importar un modelo de robot a partir de un urdf hay que acceder en la esquina superior izquierda a **Modules/Importers/URDF Importer** y buscar el archivo urdf en el sistema.Una vez hecho esto,se habrá añadido el robot a la escena de la simulación.

Guardar un modelo modificado

Para guardar un modelo modificado en la escena(solo el modelo,no la escena completa) hay que:

1. Acceder al elemento raíz del modelo dentro de la jerarquía de entidades de la parte izquierda haciendo doble click.
2. Asignarle la propiedad model al objeto con : **Scene_Object_Properties/Object/Model definition/Object is model**
3. File/Save model as/CoppeliaSim model

El modelo se guardará en formato ttm, un archivo binario.

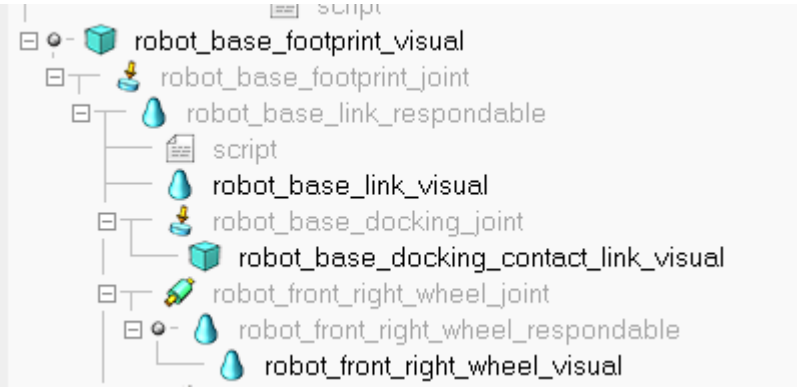
Conseguir la referencia a una entidad de la simulación en un script.

Para acceder a los datos y métodos que tiene un objeto en Coppelia necesitamos su referencia/handle.Para conseguir su referencia se utiliza el método:

sim.getObject('ruta_al_objeto')

La ruta puede ser relativa,desde dónde se encuentra el script en la jerarquía de entidades, o absoluta,desde la raíz de la escena(el objeto que contiene todos los demás objetos).

Ejemplos de acceso a la referencia de objetos dada esta jerarquía de objetos:



***El objeto robot_base_footprint_visual es el objeto inmediatamente posterior a la raiz.**

Ruta relativa: sim.getObject('../robot_front_right_wheel_joint')

En esta ruta, los 2 puntos se refieren al objeto padre del script,es decir robot_base_link_respondable y lo que va después de la barra es el objeto hijo de el objeto de la izquierda de la barra.

Ruta absoluta: robotHandle = sim.getObject('/robot_base_footprint_visual/robot_base_footprint_joint/ robot_base_link_respondable')

Esta ruta accede al handle del objeto robot_base_link_respondable.

Sensores

En Coppelia Sim se dispone de una gran variedad de sensores, tanto plugins (localizados en **Model_browser/components/sensors**) o como sensores nativos del simulador que podemos visualizar haciendo **click derecho en cualquier objeto de la jerarquía de entidades y luego Add**

El procedimiento general para añadir un sensor a un frame de la jerarquía es el siguiente:

Si es una cámara rgb, sensor de fuerza o sensor de proximidad.

1. Click derecho en el objeto que tiene el frame donde queremos situar nuestro sensor.
2. Add
3. Añadir sensor.

Si es un sensor dentro de components/sensors:

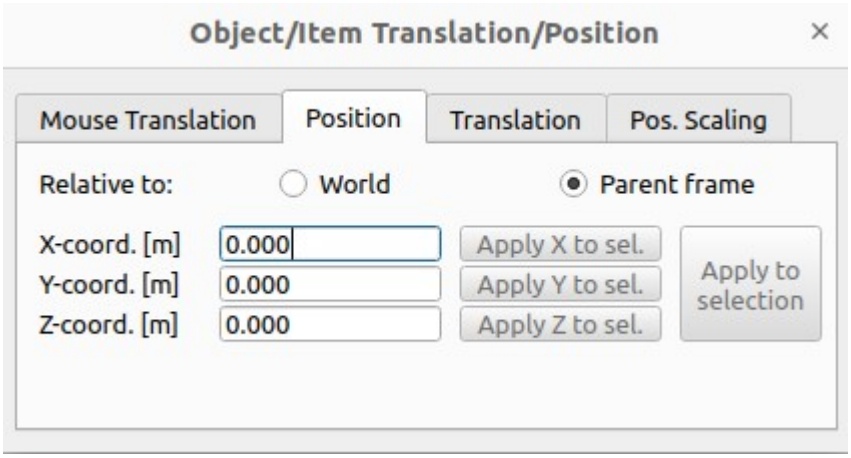
1. Click en el sensor y arrastrar hasta el elemento de la jerarquía al que lo queremos adjuntar.
2. Si solo queremos usar sus propiedades hay que eliminar la interacción física de su modelo 3D con el mundo. Accedemos al objeto con doble click, Scene_object_properties/Object_special_properties y dejamos las casillas Collidable y Detectable sin marcar.
3. (Opcional) Para que el sensor sea invisible tenemos que acceder a Scene_object_properties/Object_special_properties/Model_definition/ Model_properties y marcar la casilla Model is not visible.

Finalmente tenemos que dejar el sensor posicionado en el centro del frame y orientado de la misma forma que el frame,sino,las mediciones del sensor no serían respecto del frame que queremos sino de su propio frame y necesitaríamos publicar un tf adicional a ros2.

Para posicionar el sensor en el centro del frame al que se ajunta:

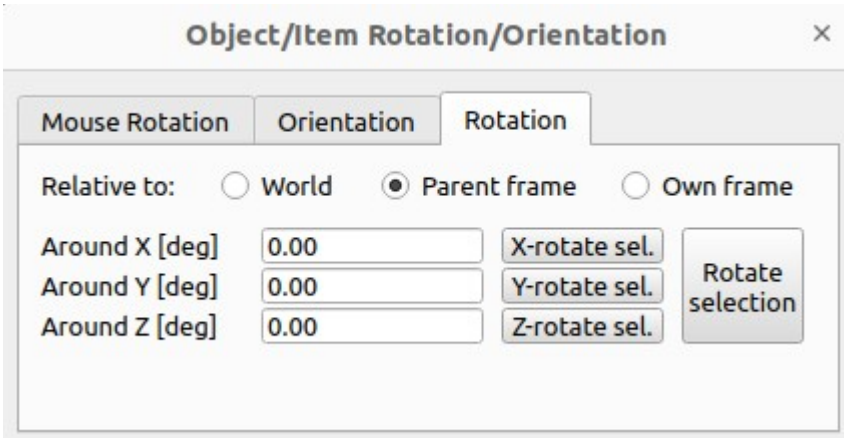
1. Click en el sensor
2. Click en este botón de la parte superior. 

3. Poner a 0 todas las coordenadas respecto al frame padre.



Para orientar el sensor en la misma orientación que su frame padre:

1. Click en el sensor
2. Click en este botón de la parte superior. 
3. Poner a 0 todas las rotaciones respecto al frame padre.



Cámaras

Para añadir una cámara:

1. Click izquierdo en elemento de la jerarquía.
2. Click derecho una vez esté seleccionado.
3. Añadir script a la cámara con Add/Script/simulation_script/Non-threaded/Python
4. Copiar estructura de **camerascript** , cambiar tópico de publicación y el frame_id por los que se desean ,y cambiar el nombre a la propiedad(contador del robot) al que pertenece.

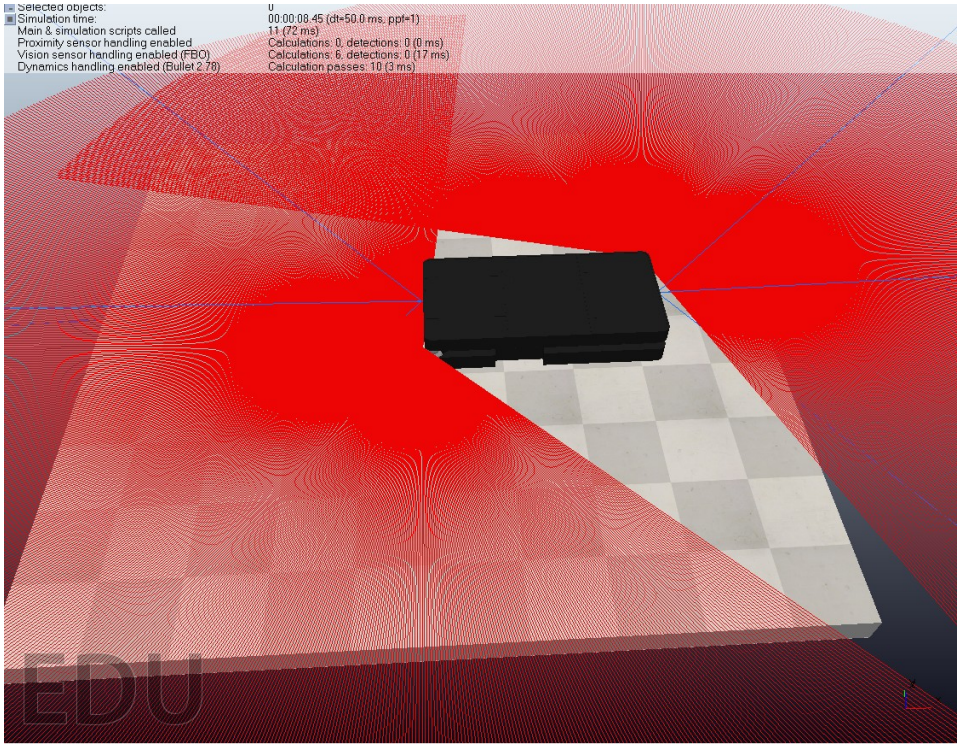
LIDAR 2D

Para la implementación de un LIDAR 2D se ha probado a usar un proximity sensor de tipo cilíndrico, pero solo detecta la posición del objeto más cercano(solo un rayo).Después se ha probado a usar un 2D Laser Scanner de la carpeta sensors, pero al no tener código para la comunicación con ros he optado finalmente por usar un fastHokuyo_ROS que ya tenía código para la comunicación con ros1 y solo he tenido que cambiar el plugin de ROS por el plugin de ROS2 en el script.Por último,he tenido que agregarle un retardo de 2 segundos en la función sensing() del script ya que al spawnear el robot, tardaba en iniciarse el sensor y al acceder a sus datos en las iteraciones iniciales no había ningún valor y se paraba la simulación.

Para agregarlo al robot se debe:

1. Arrastrar fastHokuyo_ROS al objeto al que se quiere adjuntar.
2. Colocarlo en la misma posición y orientación que su frame padre.
3. Cambiar el script default por **lidar2d_script** ,modificar su tópico de publicación y el frame_id por los que se desean ,y cambiar el nombre a la propiedad en la que se guarda el contador del robot al que pertenece.

El resultado de agregar 2 sensores LIDAR 2D al rb_robout debería verse así:



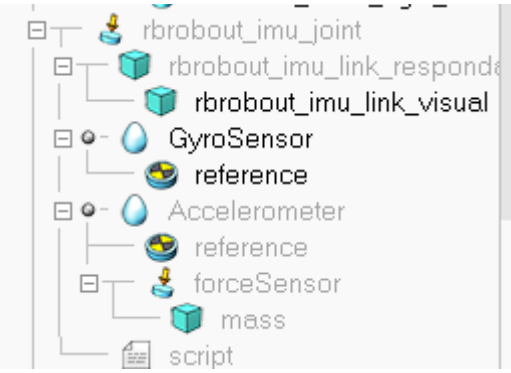
IMU

Para conseguir implementar un IMU,hemos combinado un Gyro Sensor(que proporciona las aceleraciones angulares) y un Accelerometer(que proporciona las aceleraciones lineales).

Pasos para implementarlo:

1. Añadir interfaz sensor_msgs/msg/Imu a **/rutadeinstalaciondeCoppelia/programming/ros2_packages/sim_ros2_interface/meta/interfaces.txt**
2. cd /rutadeinstalaciondeCoppelia/programming/ros2_packages/sim_ros2_interface
3. source /opt/ros/humble/setup.bash
4. colcon build --symlink-install
5. Hacer que ambos sensores sean hijos de imu_joint y que estén en su mismo frame posicionados y orientados como hemos explicado anteriormente.
6. Añadir el script **imu_script como hijo de imu_joint ,** cambiar el tópico de publicación ,y cambiar el nombre a la propiedad(contador del robot) al que pertenece.
7. La jerarquía debería verse como la figura de abajo.

Podemos encontrar el archivo interfaces.txt con todas las interfaces utilizadas en: **interfaces.txt**.

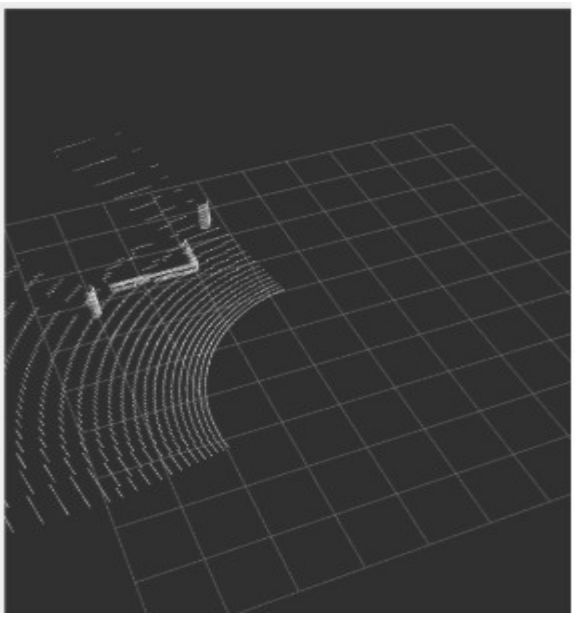


LIDAR 3D

Para implementarlo hemos usado el sensor VelodyneVPL16.Es un sensor que usa 4 cámaras para calcular la nube de puntos con las distancias a los objetos.Devuelve los puntos en un “array” unidimensional en el que cada punto tiene 3 elementos.La idea es coger esos puntos y pasárselos al formato pointCloud2 mediante la modificación del script default.

Pasos a seguir para implementarlo:

1. Situar el sensor en el frame adecuado.
2. Quitar las interacciones físicas del modelo 3D del sensor.
3. Cambiar el script default por **Velodyne_script**.
4. Modificar el tópico de publicación y el frame id para que vaya acorde a las tf publicadas ,y cambiar el nombre a la propiedad(contador del robot) al que pertenece.
5. Debería verse así en el rviz:



En la función de creación del sensor LIDAR3D(simVision.createVelodyneVPL16) podemos modificar los siguientes parámetros para optimizar la simulación:

- options: si options=1 no se muestra los puntos en el simulador,solo se verían en el tópico del rvizz, si options= 2+8 sí que se muestran
- frequency: si la reducimos,el ángulo en el que las cámaras calculan los puntos y que está girando se reduce.

GPS

Para añadir un GPS a nuestro robot,debemos usar el GPS que se encuentra en la carpeta sensors,le quitaremos la colisión y lo añadiremos al frame que queremos.A continuación,cambiaremos su script default por el script **gps_script** , el tópico de publicación al que queremos que publique ,y el nombre a la propiedad(contador del robot) al que pertenece.

TF

En ROS, TF (transform) es una biblioteca que permite rastrear múltiples marcos de referencia coordinados en el tiempo. Es fundamental para sistemas robóticos porque facilita saber dónde está cada parte del robot (o del entorno) en relación con otras en cualquier momento. Por ejemplo, permite transformar coordenadas de un sensor (como una cámara) al marco base del robot, o del robot al mundo. Gracias a TF, los robots pueden entender mejor su entorno y tomar decisiones informadas basadas en posiciones y orientaciones relativas.Para implementar un robot en CoppeliaSim y asegurar una comunicación adecuada con ROS 2, es fundamental publicar correctamente las **transformaciones (TF)** de todos los *frames* involucrados, es decir, las distintas partes del robot y sus referencias espaciales. Esto permite que ROS 2 conozca la posición y orientación relativa de cada componente en todo momento, lo cual es esencial para tareas como navegación, percepción y planificación de movimiento.Esto se consigue publicando en el tópico /tf la posición y orientación relativa de un frame hijo respecto del frame padre usando la interfaz tf2_msgs/msg/TFMessage.

Para conseguir pasar las tf de cada objeto del robot tenemos que seguir estos pasos:

1. Obtener todos los handles de los objetos de los que queremos publicar sus tf.Si tiene padres también tenemos que publicar las tf de sus padres ya que en Coppelia sólo se puede calcular la posición y orientación relativa a su frame padre inmediatamente superior.
2. Conseguir mediante el handle de cada objeto padre e hijo su posición y orientación relativa y rellenar los campos del mensaje de ros2 con la interfaz tf2_msgs/msg/TFMessage, también se necesitará los identificadores/nombres de los frames padre e hijo para rellenar correctamente el mensaje.Esto se realiza en el base_script (script principal) de cada robot con la función

   addTransform(parent_handle, parent_frame, child_frame, child_handle)

3. Publicar el mensaje completo en el tópico /tf

Control

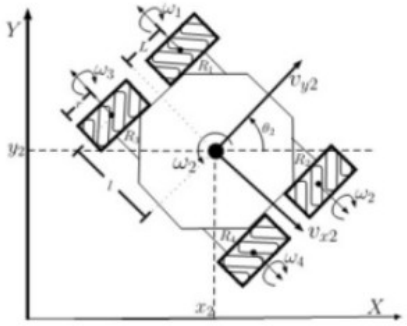
Para mover el robot usando el cmd_vel del robot tenemos que crear un subscriber a ese tópico y obtener la velocidad angular y lineal del mensaje recibido mediante un callback. Después tenemos que convertir esas velocidades(del robot) a las velocidades de las ruedas.Para ello tenemos que:

1. Obtener los handles de las ruedas
2. En el callback del subscriber tenemos que calcular las velocidades de las ruedas mediante las fórmulas cinemáticas del robot (depende de su configuración,en este caso ambos robots usan la configuración mecanum de 4 ruedas)

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & 1 & -(L+l)\\ -1 & 1 & (L+l)\\ -1 & 1 & -(L+l)\\ 1 & 1 & (L+l)\end{bmatrix}\begin{bmatrix}V_x\\ V_y\\ \omega\end{bmatrix} \quad (1)$$

En base a la Figura 2, Los parametros L, l y r, pertenecen al robot, para esta simulación, al KUKA YouBot [2], donde: r = 100 mm, L = 235.5 mm, l = 150 mm.

A este modelo matemático se le ingresan las velocidades en X, Y y rotación angular del robot, para obtener la velodiadad



3. Enviamos las velocidades a las joints de las ruedas con el método

   sim.setJointTargetVelocity(handle_de_la_rueda, velocidad)

Spawn

Para spawnear dinámicamente modelos se usa el plugin Remote API,que está activado por defecto en la versión edu de CoppeliaSim.

Se tiene que instalar la API de python con :

python -m pip install coppeliasim-zmqremoteapi-client

Pasos para spawnear robots:

1. Descargar spawn_model.py .
2. Modificar dentro la ruta del directorio donde se encuentran los modelos de los robots.
3. **python3 spawn_model.py modelo.ttm x y z roll pich yaw (la orientación en grados)**

Ej: python3 spawn_model.py rb_watcher.ttm 1.0 0.5 0.2 0 0 0

Para que al spawnear más de un robot del mismo tipo no publiquen en los mismos tópicos,lo que hemos hecho es leer o crear si no existe un fichero temporal con los contadores de los robots.De forma que al cargar un modelo se lee el contador y se pone al final de cada tópico y frame del robot como sufijo,y posteriormente se incrementa.Si se elimina el robot, reducimos ese contador en la función cleanup().Una vez leído el contador en el script base de cada robot,se extiende la variable como una “property” a todos los elementos de la escena de forma que cualquier elemento de la escena pueda leerla.

Cómo usar los modelos y las escenas ya hechos

Para usar escenas:

1. Descargar la escena.
2. Moverla a Directorio/installacion/de/Coppelia/scenes .
3. (Opción 1)(Dentro de CoppeliaSim) File/Open_Scene
4. (Opción 2) Descargar launch_scene.py y modificar la ruta del directorio de las escenas.

   source /opt/ros/humble/setup.bash
   python3 launch_scene.py nombre_de_la_escena.ttt

   Ej: python3 launch_scene.py escena_ligera.ttt

Para usar robots:

1. Descargar modelo del robot.
2. Mover a Directorio/installacion/de/Coppelia/models/robots/mobile.
3. (Opción 1) spawnearlos en la escena mediante spawn_script.py .

4. (Opción 2) Cargarlos en la escena antes de darle al play mediante File/Load_model.