Taller 5: ROS2

Robótica Móvil

2^{do} cuatrimestre 2025

1 Introdución

El objetivo de este taller es familiarizarlos con el marco de trabajo (framework) **ROS2** (Robot Operating System 2), el simulador **CoopeliaSim** e introducir las herramientas para el desarrollo de aplicaciones de Robótica Móvil.

1.1 Uso de Docker

Docker es una plataforma que permite crear, distribuir y ejecutar aplicaciones dentro de entornos aislados llamados contenedores (*containers*). Su funcionamiento se basa en *imágenes*, que son plantillas inmutables con todo lo necesario para ejecutar un programa: el sistema operativo, las librerías, dependencias, configuraciones y el propio código. Un contenedor es una instancia en ejecución de una imagen. A modo de analogía, se puede pensar a la imagen como si fuera una receta, y a cada contenedor como un plato preparado de esa receta. Pueden tenerse varios contenedores creados a partir de la misma imagen, cada uno independiente de los demás. Aunque no son lo mismo, un contenedor de Docker puede entenderse como una máquina virtual ligera, que no incluye un sistema operativo completo, pero permite aislar y ejecutar aplicaciones de manera independiente del resto del sistema.
El uso de Docker tiene varias ventajas para el desarrollo de software:

- **Aislamiento**: cada contenedor tiene sus propias versiones de librerías y paquetes, sin interferir con otros contenedores ni con el sistema operativo base.
- **Portabilidad**: si un programa se desarrolla dentro de un contenedor con ciertas librerías y configuraciones, se puede compartir la imagen y ejecutarla en cualquier otra computadora con Docker, manteniendo exactamente la misma funcionalidad, sin necesidad de instalar o adaptar librerías localmente. Ésto evita el clásico "en mi computadora funciona".
- **Ligereza**: a diferencia de las máquinas virtuales, los contenedores comparten el kernel del sistema, lo que los hace más rápidos de iniciar y menos pesados.
- **Facilidad de despliegue**: una misma imagen puede usarse para crear múltiples contenedores en diferentes entornos (desarrollo, pruebas, producción) de manera consistente.

De esta forma, Docker permite empaquetar una aplicación con todo lo que necesita para funcionar y ejecutarla en cualquier lugar. Nosotros utilizaremos estas ventajas para trabajar con ROS2 a lo largo del cuatrimestre.
En las computadoras del laboratorio ya se cuenta con una imagen de ROS2 instalada, junto con el simulador CoppeliaSim y las demás herramientas que estudiaremos en la materia. Para confirmar que la imagen de ROS2 se encuentre activa, pueden utilizar el comando:

```
$ docker images
```

Si la imagen `ros2_robotica` se encuentra creada, se puede directamente generar un contenedor de la misma y comenzar con el desarrollo de los ejercicios.
En el Campus de la materia se encuentra disponible tanto un Dockerfile que configura y permite crear la imagen que vamos a utilizar, y un archivo `start-docker.sh`, que define una serie de macros que permiten crear tanto la imagen como los contenedores. Para esto utilice en una terminal, situada en la misma ruta en que se encuentra este archivo, el comando:

```
$ bash start-docker.sh <INSTRUCTION>
```

donde `<INSTRUCTION>` es una de las siguientes opciones:

- `build`: permite crear la imagen `ros2_robotica` a partir del Dockerfile, por ejemplo si quieren resolver los talleres en su propia computadora (NO utilizar en las computadoras del laboratorio).
- `start`: crea un contenedor `ros2_robotica` a partir de la imagen `ros2_robotica` creada previamente.
- `open`: abre el contenedor `ros2_robotica` previamente creado. Recordar que pueden abrirse múltiples terminales dentro del contenedor usando este comando en distintas terminales.
- `stop`: cierra/finaliza el contenedor `ros2_robotica` previamente creado.
- `down`: cierra/finaliza el contenedor `ros2_robotica` y posteriormente, lo borra.

Para confirmar que un contenedor fue creado exitosamente, se puede consultar la lista de contenedores activos usando:

```
$ docker ps -a
```

Si en la lista de contenedores aparece `ros2_robotica`, entonces éste fue creado satisfactoriamente.
Importante: dentro del archivo `start-docker.sh`, en la instrucción de `start`, hay un parámetro de configuración que permite compartir un volumen con el contenedor, es decir, que exista una carpeta compartida visible desde adentro del contenedor y desde afuera del mismo (el sistema base de la computadora). Esto permite poder leer o modificar archivos fuera del contenedor y que los cambios sean vistos dentro de éste, y viceversa. Esta configuración dentro de la instrucción tiene la forma:

```
-v <DIR>:/root/ros2_ws/src/robotica
```

donde se establece la ruta de una carpeta compartida vista desde afuera (`<DIR>`, antes del ":") y desde dentro (`/root/ros2_ws/src/robotica`, luego del ":") del contenedor. Deben modificar la ruta vista **desde afuera** por aquella que contenga los archivos que quieren tener disponibles dentro del contenedor (por ejemplo, los archivos para la simulación en este taller).
Por una cuestión de permisos de administrador, se recomienda crear primero la carpeta del volumen a compartir, con los archivos del taller dentro, y luego crear el contenedor. De esta forma, dentro del contenedor creado ya estarán visibles los contenidos del taller.
Ejemplo: si se arma una carpeta en `/home/Estudiante/robotica/Taller5` con los contenidos de este taller, entonces se podría editar el comando como sigue:

```
-v /home/Estudiante/robotica:/root/ros2_ws/src/robotica
```

de forma que, dentro del contenedor, pueda encontrarse en la dirección `/root/ros2_ws/src/robotica` la carpeta Taller5.

1.2 Instalación de la imagen en computadora personal

Para la instalación de la imagen de Docker con la que trabajaremos en la materia es necesario, en primer lugar, instalar Docker, y después descargar tanto el archivo `Dockerfile` como `start-docker.sh` provistos en el Campus de la materia. Ambos archivos deben estar en un mismo directorio. Luego, para compilar la imagen, en el directorio en que se encuentren ambos archivos debe utilizarse el comando:

```
$ bash start-docker.sh build
```

Esto compilará la imagen de Docker con ROS2 y CoppeliaSim ya instalados. Recomendamos utilizar Docker en lugar de instalar ROS2 y CoppeliaSim directamente en sus computadoras personales, para tener entre todos un entorno de trabajo compatible, con las mismas versiones de todas las librerías a utilizar, y no correr riesgos por dependencias entre paquetes.

1.2.1 Configurar variables de entorno de ROS2

Nota: Estas instrucciones fueron ya realizadas en tiempo de compilación de la imagen de Docker, por lo que en principio no es necesario realizarlas en caso de uso de esta images, pero les dejamos la sección por si deciden instalar ROS2 de manera nativa en sus computadoras personales sin utilizar Docker. Nosotros **no** recomendamos esta opción, sino utilizar Docker y la imagen que se genera a partir del DockerFile que está subido en el Campus de la materia, si quieren trabajar en sus computadoras personales.
Dado que uno podría tener varias versiones de ROS y/o ROS2 instaladas es necesario seleccionar la versión que se utilizará y configurar las variables de entorno correspondientes. El ejemplo está dado para la versión **Humble** de ROS2, que es la que proponemos utilizar para la materia, pero es equivalente para cualquier otra versión. Para esto es necesario ejecutar las siguientes líneas de comando **por única vez**:

```
$ echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

De esta manera, las variables de entorno de ROS2 serán accesibles para toda nueva consola de comandos. Se recomienda cerrar y volver a abrir toda consola de comandos abierta para asegurar la carga de las variables de entorno necesarias por ROS2.
Para confirmar que las variables de entorno estén correctamente configuradas es posible ejecutar:

```
$ printenv ROS_DISTRO
```

Debería responder: `humble`.

1.3 Configurar el workspace de ROS2 con Colcon

Colcon es el nombre del sistema de compilación y manejo de paquetes integrado en ROS2. Los workspaces de Colcon facilitan trabajar con varios paquetes al mismo tiempo y definen un entorno unificado para la ejecución de diversos nodos.
La materia provee los paquetes necesarios para cada taller, los cuales son necesarios compilar. Estos paquetes deben copiarse en el workspace de ROS2. Para poder configurar el workspace y compilar los paquetes primero se debe ejecutar en la raíz del mismo (`/root/ros2_ws`) los siguientes comandos:

```
$ colcon build
$ source ./install/setup.bash
```

Como resultado se crearán varias carpetas. Un workspace se compone principalmente de 4 directorios:

- **/build**: Contiene los archivos generados durante la compilación de los paquetes.
- **/log**: Contiene logs de ejecución y compilación.
- **/install**: Contiene los paquetes instalados y configurados. Desde aquí se ejecutan nodos y se corren archivos.
- **/src**: Contiene los paquetes con los cuales se trabajará. Estos proveen definiciones de mensajes, nodos, directivas de compilación y establecen dependencias entre paquetes. Aquí se colocan los paquetes que se desarrollan o descargan.

Es posible listar los paquetes presentes en el entorno de desarrollo Colcon utilizando el comando:

```
$ colcon list
```

Aún cuando los paquetes hayan compilado de manera exitosa, es posible tener varios workspaces Colcon creados. Por lo que es necesario configurar las variables de entorno del workspace actual ejecutando en la raíz del workspace:

```
$ source install/setup.bash
```

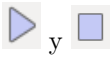
Nota: Esta configuración solo es válida en la consola de comandos en la que se ejecutó, por lo que si se cierra la consola es necesario volver a ejecutar este comando de configuración para el workspace con el que se desea trabajar.

Ejercicio 1: Interfaz con CoopeliaSim

La materia provee de una versión de CoppeliaSim previamente configurada para comunicarse con ROS2 de manera de poder publicar y suscribirse a tópicos.
Es posible iniciar CoopeliaSim ejecutando:

```
$ coppeliaSim.sh
```

Una vez abierto el simulador deben cargar la escena del ejercicio clickeando en: **File** → **Open scene...** y abrir el archivo:
`/root/ros2_ws/src/robotica/.../ros_intro/coopeliaSim/ros_intro.ttt`
donde (...) refleja la ruta dentro del volumen compartido en el contenedor hasta la carpeta `ros_intro`, incluida en los contenidos entregados en este taller.
La escena presentada se compone de un único robot **Pioneer**. Para iniciar y detener la simulación deben hacer click en los botones:

 y

Luego de haber iniciado la simulación de la escena, se pide:

a) Utilizar comandos de ROS2 para descubrir que tópicos se suscribe y publica el CoppeliaSim.

b) Controlar el robot enviando comandos de velocidad, mediante mensajes de tipo: `geometry_msgs/msg/Twist`. Realizar de dos formas distintas: enviando un comando desde consola que publique en el tópico correcto, y creando un nodo que publique en el tópico correcto. En el primer caso, puede publicarse un mensaje en algún tópico utilizando un comando del tipo:

```
ros2 topic pub -r 10 <TOPIC> geometry_msgs/msg/Twist
                     "{linear: {x: 0.0, y: 0.0, z: 0.0},
                      angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

que envía a una frecuencia de 10 Hz el mensaje especificado. Para el segundo caso, pueden utilizar el script base `publicar_velocidad.py` que se encuentra en el Campus de la materia. Este script de python crea un nodo publicador, de forma análoga a lo visto en clase para `.cpp`, que puede ejecutarse directamente utilizando

```
python3 publicar_velocidad.py
```

Utilizar scripts de este tipo permite crear nodos para leer y escribir en tópicos de forma sencilla, sin la necesidad de crear paquetes de Ros2 y compilarlos para efectuar pruebas rápidas.

c) Replicar el inciso b) para enviar comandos de velocidad al robot de manera que se encuentre constantemente girando en circulo.

Ejercicio 2: Nodo de tele-operación

Implementaremos un nodo de ROS2 que nos permita enviar comandos de velocidad al robot simulado en CoppeliaSim utilizando el teclado.
Dentro del paquete `ros_intro` existe un directorio `/src/` donde encontrarán el código de fuente "esqueleto" de un nodo para traducir eventos del teclado a comandos de velocidad. Este nodo se llama `keys_to_twist` y se suscribe a los tópicos `/keyup` y `/keydown`. Los mensajes de dichos tópicos se reciben por los métodos `on_key_up(...)` y `on_key_down(...)` presentes en el archivo `keys_to_twist.cpp`.
Para compilar el nodo ejecuten desde la raíz del workspace :

```
$ colcon build
$ source install/setup.bash
```

Para iniciar correctamente el nodo es posible utilizar el comando `ros2 run`:

```
$ ros2 run ros_intro keys_to_twist
```

En otra terminal dentro del contenedor de Docker, correr:

```
$ ros2 run keyboard keyboard
```

Ésto abrirá una ventana que, al estar activa, capta las teclas presionadas del teclado y manda dicha información por los tópicos `/keyup` y `/keydown`, que serán leídos e interpretados por `ros_intro`, y enviados como comandos de velocidad por el tópico `/cmd_vel`.
Nota: Utilizar `Ctrl + C` para detener la ejecución de cualquier nodo.

Se pide:

a) Completar las áreas marcadas de código en los métodos `on_key_up(...)` y `on_key_down(...)` de manera de publicar un mensaje *Twist* por el tópico `/cmd_vel`.

   Nota: Las unidades en ROS2 son en metros por segundo para la velocidad lineal, y radianes por segundo para la velocidad angular.

b) Abrir el simulador CoppeliaSim, iniciar la simulación y ejecutar los nodos utilizando los comandos de `run` correspondientes. Visualizar el comportamiento del robot en la simulación ¿Qué nodos están tomando parte del sistema en ese momento?.

c) Teniendo en cuenta lo visto en la primera parte de la materia:
   ¿Qué nodo esta operando los motores de las ruedas? ¿Cúal es la diferencia en la forma de controlar el robot si la comparamos con el taller de control de motores con Arduino?

Este ejercicio replica el funcionamiento del paquete `teleop_twist_keyboard`. Utilizarlo para comparar el funcionamiento.

```
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Ejercicio 3: Visualizar en RViz2

RViz2 es una herramienta muy utilizada para visualizar la información que circula por los tópicos del sistema. Para iniciar RViz2 es necesario ejecutar el comando:

```
$ rviz2
```

Para poder visualizar en RViz2 al Pioneer que se está utilizando en CoppeliaSim, es necesario descargar desde éste su archivo de descripción `URDF`. Se trata de un archivo `XML` que describe la estructura física del robot, incluyendo sus enlaces (partes físicas), articulaciones (conexiones entre partes), propiedades geométricas, masa, inercia y apariencias visuales. Para poder visualzar el robot en RViz2, pueden exportar el archivo `URDF` correspondiente de CoppeliaSim clickeando en: **Modules** → **Exporter** → **URDF exporter...**.

Importante: Por defecto, el archivo de `.urdf` exportado por CoppeliaSim define a la base del cuerpo con el término `robot_base`, pero RViz2 lo espera con el término `base_link`, por lo que es necesario modificar en el archivo `.urdf` exportado las apariciones de este término.
Iniciar el simulador CoppeliaSim e iniciar la simulación. Realizar los pasos:

a) Agregar un modelo del robot a RViz2: **Add** → **RobotModel** → **OK**. En la configuración de RobotModel, cambiar "Description Source" a "File", y agregar la dirección del `urdf` exportado desde el simulador.

- Visualizar los marcos de referencia publicados por el sistema:
  Hacer click en **Add** → **TF** → **OK**.

  Nota: RViz2 necesita que se defina algún marco de referencia como "fijo". Seleccionen el marco fijo utilizando la opción desplegable:
  Global Options → **Fixed Frame** → **Seleccionar el marco deseado.**. Recomendamos utilizar tanto /map u /odom, ya que el resto de las transformaciones están referidas a estos dos marcos.

b) Listar los tópicos activos y visualizar la información odométrica provista por CoppeliaSim:
   Hacer click en **Add** → **By topic** → **Odometry**.

c) Utilizar el teclado para enviar comandos de velocidad al robot mientras se cambia el “marco fijo” en RViz2. ¿Que sucede con los marcos “no fijos” al moverse el robot? ¿Cómo es que se mueven?.

Para visualizar el árbol de transformaciones, puede utilizar el comando:

```
$ rqt
```

haciendo click en **Plugins** → **Visualization** → **TF Tree**.