Taller 7: Control a lazo abierto para el seguimiento de trayectorias

Robótica Movil

2^{do} cuatrimestre 2025

1 Introducción

El objetivo de este taller es explorar la generación y el seguimiento de trayectorias. Existen varías técnicas para esto, en principio trabajaremos con técnicas de control a lazo abierto. De la página de la materia pueden descargarse los paquetes que serán utilizados durante este taller.

Nota: Este taller requiere de los paquetes provistos en el Taller anterior, `robmovil_msgs` y `modelo_diferencial`, por lo que es necesario que estén completos y agregados al *workspace* para poder realizar los ejercicios.

Nota2: Este taller necesita que el simulador CoppeliaSim sea capaz de entender los mensajes del tipo `rosgraph_msgs/msg/Clock`, por lo que es necesario agregar este tipo de mensajes al paquete de interfaz `sim_ros2_interface`, tal como fue explicado en el enunciado del Taller 6.

1.1 Archivos `.launch.py` y configuración de parámetros

Los nodos de un paquete de ROS2, luego de haber sido correctamente instalados con Colcon, pueden ejecutarse de dos formas. La primera es utilizando la función de `ros2 run`, tal como vimos los talleres anteriores. Esta forma inicializa el nodo con la configuración determinada en los archivos en que éste fue definido (`.cpp` y `.hpp`, o bien `.py`, respectivamente). Para este taller se pueden utilizar:

```
$ ros2 run lazo_abierto trajectory_generator
$ ros2 run lazo_abierto trajectory_follower
```

La segunda forma es utilizando archivos `.launch.py`. Estos archivos permiten dinamizar la configuración de parámetros propios de los nodos pertenecientes a un paquete, y ejecutarlos simultáneamente. De esta manera, el sistema de *launch* ejecuta todos los nodos definidos en este archivo, y sobreescribe los parámetros de configuración declarados en los archivos de creación de los nodos (`.cpp`,`.hpp`, `.py`), lo cual permite reconfigurar rápidamente los nodos a ejecutar.

El archivo `.launch.py` debe encontrarse en la carpeta `launch`, ubicada en el directorio del paquete de ROS2. Para este taller en particular, se puede ejecutar de la forma:

```
ros2 launch lazo_abierto lazo_abierto.launch.py
```

Nota: Para utilizar el archivo `.launch.py` en ROS2, es necesario previamente compilar el paquete con Colcon desde el workspace `/root/ros2_ws` y cargar nuevamente el entorno del workspace:

```
$ colcon build --packages-select lazo_abierto
$ source install/setup.bash
```

En el archivo `.launch.py` de este taller se especifican los nodos que deben ser ejecutados junto con un apartado de configuración para el nodo de `trajectory generator`:

```
parameters=[  {'stepping': 0.1},
              {'trajectory_type': 'sin'},
              {'total_time': 50.0},
              {'amplitude': 1.0},
              {'cycles': 1.0},
```

```
            {'spline_waypoints': [...] }
        ]
```

El parámetro `trajectory_type` selecciona el tipo de método utilizado para generar la trayectoria. Puede ser "sin" si se quiere generar una trayectoria sinusoidal, o "spline" si se desean utilizar splines para seguir una secuencia de puntos determinada. Modificando los demás parámetros, se puede definir la forma en que el nodo `trajectory_generator` produce las trayectorias que el nodo `trajectory_follower` emplea para guiar el robot.

1.1.1 Trayectoria Sinusoidal

Para la generación de trayectorias sinusoidales se utilizan los siguientes parámetros:

1. **stepping:** controla la granularidad de muestreo de la función seno.
2. **total_time:** controla el tiempo total en que se debe recorrer la trayectoria.
3. **amplitude:** controla la amplitud de onda.
4. **cycles:** controla la cantidad de ciclos sinusoidales que deben efectuarse.

1.1.2 Trayectoria Generada con Splines

Para el caso de splines son necesarios los siguientes parámetros:

1. **stepping:** controla la granularidad de muestreo de los splines.
2. **spline_waypoints:** contiene puntos y tiempos utilizados para generar la trayectoria, según una lista con la estructura: $[time[s], x[m], y[m], orientation[rad]]$ (ver ejemplo en el archivo `.launch.py`).

1.2 Flujo de control

Los comandos de control de velocidad son publicados por el nodo `trajectory_follower` en base a la trayectoria notificada por el nodo `trajectory_generator`.

Independientemente de la granularidad de la trayectoria generada, `trajectory_follower` genera comandos de velocidad cada 0.01s. Para ésto inicializa un temporizador (`rclcpp::Time`) que ejecuta el método:

```
virtual bool control(const rclcpp::Time& t, double& v, double& w) = 0;
```

Las variables `v` y `w` serán, entonces, los comandos de velocidad lineal y angular que se deban publicar en cada momento. El cálculo de estos valores se deben realizar en base a la información que provea la trayectoria en los puntos temporalmente próximos mediante interpolación lineal.

Ejercicio 1: Enviar comandos de control

Se requiere realizar una interpolación lineal entre los comandos de velocidad publicados en los puntos de la trayectoria más proximos de manera de publicar comandos de velocidad acordes a la trayectoria requerida.
El archivo sobre el que deberán trabajar es `/lazo_abierto/src/FeedForwardController.cpp`, utilizando funciones senoidales para la generación de trayectorias. Para esto:

a) Inicializar `lazo_abierto/coppeliaSim/lazo_abierto.ttt` en el simulador CoppeliaSim.

b) Inicializar el visualizador RViz2, lanzar los nodos requeridos utilizando el archivo `.launch.py`, y comprobar que la trayectoria realizada por el robot es la adecuada.

En RViz2 es posible visualizar la trayectoria requerida agregando los mensajes `nav_msgs/msg/Path`. Para esto realizar: **Add** → **By topic** → **target_path** → **Path**.

Nota: Necesitarán operar sobre variables de tipo `rclcpp::Time`; es posible utilizar la función `.seconds()` para obtener los resultados de las operaciones sobre estas variables en unidades de segundos. Por ejemplo, para obtener la diferencia de tiempo entre dos momentos `t0` y `t1` de tipo `rclcpp::Time`, utilizar: `(t1 - t0).seconds()`.

Ejercicio 2: Limitaciones de lazo abierto

El seguimiento de trayectorias a lazo abierto requiere de poder generar trayectorias continuas con la certeza de que el robot será capaz de llevarlas a cabo de manera precisa.

a) Modificar el parámetro de `total_time` en `lazo_abierto.launch.py` de manera de requerir el total de la trayectoria en tan solo 20 segundos. Replicar el experimento del ejercicio 1 y visualizar los resultados en RViz2. ¿Cómo se compara con el desempeño anterior? ¿Qué limitaciones presenta este método de seguimiento de trayectorias?

b) Probar modificar los parámetros de amplitud y cantidad de ciclos de la función senoidal. ¿Cómo podría mejorar el comportamiento obtenido al utilizar lazo cerrado?

Ejercicio 3: Generalización de la generación de trayectorias con Splines

a) Completar en el archivo `src/trajectory_generator_node` la forma de calcular los parámetros de los polinomios utilizados para calcular los Splines, en el método `build_spline_trajectory`.

b) Modificar el parámetro `trajectory_type` de manera de utilizar splines, y completar el parámetro `spline_waypoints` con puntos de paso. Sorprender al docente eligiendo una trayectoria con una forma deseada por ustedes. ¿El robot es capaz de seguir cualquier secuencia de puntos? ¿Por qué?