# Taller 6: Control para el seguimiento de trayectorias

Introducción a la Robótica Movil

April 23, 2018

## 1 Introdución

El objetivo de este taller es explorar la generación y el seguimiento de trayectorias complejas. 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 Catkin** que serán utilizados durante este taller.

**NOTA:** El paquete robmovil_msgs, provisto anteriormente, deberá ser "reemplazado" dado que se agregaron mensajes al mismo.

### 1.1 roslaunch y configuración de parámetros

La herramienta **roslaunch** permite dinamizar la configuración de parámetros propios de un nodo. Es posible llevar a cabo esto modificando un simple **archivo de texto** de manera de facilitar la experimentación con diferentes parámetros.

En el archivo de configuración **/lazo_abierto/lazo_abierto.launch** se especifican los nodos que deben ser ejecutados para el taller junto con un **apartado de configuración**:

```
<param name="trajectory_type" type="str" value="sin"/>
<param name="stepping" type="double" value="0.1"/>
<param name="total_time" type="double" value="20"/>
<param name="amplitude" type="double" value="1"/>
<param name="cycles" type="double" value="1"/>
```

**stepping** 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.

#### 1.1.1 Trayectoria Sinusoidal

Modificando los valores (cambiando el campo "value") es posible controlar la generación de la trayectoria sinusoidal que producirá el nodo **trajectory_generator**. El cual provee de la trayectoria a recorrer al nodo **trajectory_follower**.

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 en el eje **Y**.
4. **cycles** controla la cantidad de ciclos del seno deben efectuarse.

### 1.1.2 Trayectoria Generada con Splines

Para el caso de splines es necesario completar los siguientes parametros:

1. **stepping** controla la granularidad de muestreo de la función seno.
2. **spline_waypoints** puntos y tiempos utilizados para generar la trayectoria.

## 1.2 Flujo de control

Los comandos de control de velocidad lineal y angular son publicados por el nodo **trajectory_follower** en base a la trayectoria notificada por el nodo **trajectory_generator**.

Independientemente de la calidad de **muestreo** de la trayectoria notificada, **trajectory_follower** genera comandos de velocidad por cada **0.01s**. Para lograr esto inicializa un temporizador (ros::Timer) el cual ejecuta el método:

```
bool control(const ros::Time& t, double& v, double& w)
```

Las variables **v** y **w** serán, entonces, los comandos de velocidad lineal y angular que se deban notificar en el momento actual. El cálculo de estos valores se deben realizar en base a la información que provea la trayectoria en los **puntos temporalmente próximos**.

**NOTA:** Leer atentamente los comentarios en el código.

# Ejercicio 1: Enviar comandos de control

Se requiere realizar una **interpolación lineal** entre los comandos de velocidad notificados 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**

**NOTA:** Necesitaran operar sobre tipos **ros::Time**, es posible utilizar la función .toSec() para recuperar resultados en unidad de segundos. Por ejemplo, para obtener la diferencia de tiempo entre dos momentos ros::Time utilizar:
**(t1 - t0).toSec()**

a) Inicializar el entorno de simulación con **V-Rep** utilizando la escena provista en **lazo_abierto/vrep/lazo_abierto.ttt** (**revisar el primer taller de ROS en caso de tener dudas**). **No olvidar darle "PLAY" a la simulación**

b) Inicializar el visualizador **RViz**, lanzar los nodos requeridos utilizando el comando **roslaunch lazo_abierto lazo_abierto.launch** y comprobar que la trayectoria realizada por el pioneer es la adecuada.

En el visualizador RViz, es posible **visualizar la trayectoria requerida** agregando la visualización de los mensajes **nav_msgs/Path**. Para esto realizar:
**Add → By topic → Seleccionar "Path" dentro de target_path**

## Ejercicio 2: Limitaciones de lazo abierto

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

a) Modificar el parámetro de "total_time" en el archivo de configuración **lazo_abierto.launch** de manera de requerir el total de la trayectoria en tan solo **20** segundos. Realizar el experimento y visualizar los resultados en **RViz**. ¿Como se compara con el desempeño anterior?

b) Prueben haciendo cambios en los parámetros de amplitud y cantidad de ciclos. ¿Como podría mejorar el comportamiento obtenido al utilizar lazo cerrado?

## Ejercicio 3: Generalización de la generación de trayectorias (Splines)

a) En el archivo de configuración **lazo_abierto.launch** modificar el parámetro de "trajectory_type" de manera de utilizar un spline. Además, completar el parámetro "spline_waypoints" con los puntos deseados. ¿El robot es capaz de seguir cualquier secuencia de puntos? ¿Por qué?

   **NOTA:** Prestar atención y respete la forma de completar los waypoints.

b) Elija adecuadamente los puntos para que el robot siga una trayectoria con forma de pan casero.