Taller 8:
Control a lazo cerrado para el seguimiento de trayectorias

Robótica Móvil

2^{do} cuatrimestre 2025

1 Introducción

El objetivo de este Taller es poner en práctica los conceptos de cinemática y control vistos hasta ahora en la materia. Para ello, se desarrollará un nodo de ROS2 capaz de hacer el seguimiento de trayectorias utilizando técnicas de control a lazo cerrado.
Del campus de la materia pueden descargarse los paquetes necesarios para realizar el taller. Además, son necesarios paquetes del taller anterior, por lo que deben tenerlos terminados y agregados al volumen de Docker.
Para inicializar tanto la simulación en el CoppeliaSim como los nodos de ROS2 deben utilizar la misma escena provista en el taller de lazo abierto, `lazo_abierto.ttt`, junto con el archivo de launch:

```
ros2 launch lazo_cerrado lazo_cerrado.launch.py
```

1.1 Esqueleto y configuración del paquete provisto

El paquete provisto en el Campus se llama `lazo_cerrado` y posee un único nodo a completar denominado `KinematicPositionController`. Específicamente, tienen que trabajar en el archivo: `lazo_cerrado/src/KinematicPositionController.cpp`.

Este nodo permite 3 modos de operación distintos dependiendo del tipo de selección de la pose objetivo (*goal*). Los modos de operación son:

1. **Fixed Goal:** Se envian comandos de velocidad para alcanzar una pose final objetivo predefinida.
2. **Time Based:** Se trata de una trayectoria con requerimientos temporales definidos. En cada momento se define la pose objetivo como aquella "donde debería encontrarse actualmente el robot".
3. **Pursuit Based:** Se trata de una trayectoria con requerimientos de pose (x, y, θ). Se designa un punto vía "cercano" que permita realizar el seguimiento de la trayectoria en base a un algoritmo de selección de pose objetivo (a completar por ustedes).

Es posible configurar el modo de operación modificando los siguientes parámetros en el archivo `/lazo_cerrado/src/launch/lazo_cerrado.launch.py`:

```
parameters=[  {"goal_selection": "FIXED_GOAL"},
              {"fixed_goal_x": float(3.0)},
              {"fixed_goal_y": float(0.0)},
              {"fixed_goal_a": float(-1.570796327)}  ]
```

Los valores permitidos para el parámetro `"goal_selection"` son: `FIXED_GOAL`, `TIME_BASED` y `PURSUIT_BASED`.

Los parámetros `fixed_goal_x`, `fixed_goal_y`, `fixed_goal_a` solo son válidos al configurar el modo de operación `FIXED_GOAL`, y representan una pose objetivo requerida en referencia a la pose inicial que comienza el robot.

Nota1: Deben configurar el ángulo requerido en valores absolutos expresados en radianes.

Nota2: El nodo publica un mensaje de tipo `geometry_msgs/msg/Pose` en el tópico `/goal_pose` para poder visualizar la pose objetivo en RViz2.

1.2 Nodo de registro

Se provee además un nodo de registro, el cual escribe en texto plano las posiciones y orientaciones del robot en cada momento de tiempo.

Se registra la información de odometría, la pose real *ground-truth* publicada por el simulador CoppeliaSim y la pose objetivo seleccionada. Los archivos son generados en la carpeta `/root/ros2_ws/`.

El nombre de los archivos corresponde a:
1. ***timestamp*_poses.log**: poses publicadas por la odometría.
2. ***timestamp*_ground-truth.log**: poses reales del robot en la simulación.
3. ***timestamp*_goals.log**: poses objetivo seleccionadas en cada momento.

El formato de los archivos *.log* es:

```
t x y theta
```

Nota: t corresponde al tiempo en que fue publicada la pose en segundos.

Ejercicio 1: Dirigir el robot hacia una pose objetivo

Configurar al nodo de control en modo **Fixed Goal** y definir una pose objetivo que se desee alcanzar (por defecto el archivo `.launch.py` viene configurado con una interesante). Implementar el algoritmo de control a lazo cerrado visto en clase para un robot diferencial, completando el método:

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

Como resultado del comportamiento, el robot Pioneer deberá converger a la pose objetivo respetando tanto la posición x, y y la orientación θ definidas.

Nota: Recordar hacer la conversión necesaria a las variables de manera de establecer el marco de referencia del objetivo como el marco inercial (revisar las cuentas vistas en clase).

a) ¿Qué sucede cuando el robot se acerca al objetivo? En caso de haber algún conflicto, proponer alguna estrategia para solucionarlo.

b) Probar distintos valores de K_ρ, K_α y K_β ¿Cómo modifican el comportamiento estos parámetros? ¿Qué afectan cada uno de ellos?

c) Opcional: Graficar la trayectoria realizada por el robot. Puede hacer esto para distintos valores de K_ρ, K_α y K_β. Comparar y sacar conclusiones. Puede utilizar los *logs* generados para conseguir los valores a graficar.

Ejercicio 2: Seguir una trayectoria con requerimiento temporal

Configurar al nodo de control en modo **Time Based** y definir una trayectoria con un requerimiento temporal "exigente" en el archivo `.launch.py` (por defecto ya viene una configurada).

Utilizar los parámetros K_ρ, K_α y K_β que mejor resultado dieron en el ejercicio anterior.

a) ¿El método de control es capaz de seguir la trayectoria a la perfección? Si no lo logra, ¿por qué creen que sucede esto?

b) Buscar los mejores parámetros K_ρ, K_α y K_β que permitan seguir lo mejor posible la trayectoria.

c) Opcional: Graficar y comparar la trayectoria del robot con respecto a la trayectoria definida por las poses objetivos seleccionadas.

Ejercicio 3: Algoritmo de seguimiento de persecución

Dado que cumplir requerimientos temporales resulta muy restrictivo, vamos a plantear un algoritmo de seguimiento que permita seguir una trayectoria de puntos vía (waypoints) definidos por poses (**x, y,** θ). Para esto seleccionaremos puntos vía cercanos pero que mantengan una determinada distancia del de la posición actual del robot (ver Fig. 1).

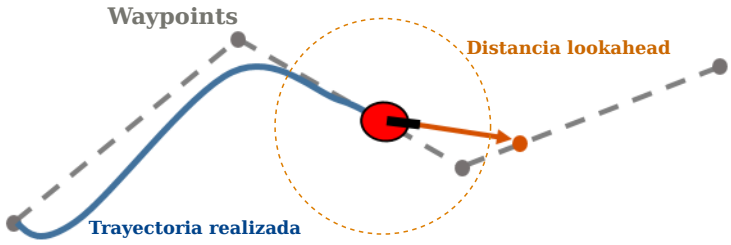

Figure 1: Circuito de simulación del robot

Se desea completar la trayectoria de "manera segura", por lo que deberán implementar un algoritmo de selección de pose objetivo completando el método:

```
bool getPursuitBasedGoal(const rclcpp::Time& t,
                         double& x, double& y, double& a)
```

Este método debe revisar la trayectoria requerida y determinar una "pose objetivo adecuada". Para esto pueden suponer que la trayectoria posee una gran cantidad de waypoints definidos y curvas "suficientemente suaves" con ángulos de giro abiertos.
Para probar el algoritmo de seguimiento deberán configurar el nodo de control utilizando el modo de operación **Pursuit Based**.

Nota: Se recomienda encontrar el waypoint de la trayectoria más cercano al robot (en términos de x, y) y luego buscar el primer waypoint que se encuentre a una distancia predefinida de lookahead en x, y.

a) Explicar detalladamente el método propuesto. ¿Qué ventajas presenta en comparación a la selección basada en la restricción temporal?

b) Utilizar los mismos parámetros K_ρ, K_α y K_β hallados en el ejercicio anterior. Realizar el seguimiento de la misma trayectoria seleccionada. Comparar los resultados obtenidos.

c) Opcional: Graficar y comparar la trayectoria del robot y la trayectoria definida por las poses objetivos seleccionadas.