EL7009 - Robótica Móvil
Profesores: Javier Ruiz del Solar
Felipe Inostroza
Auxiliar: Francisco Leiva

EL7009 - Tarea 4: Planificación

0) Configuración del espacio de trabajo

Esta tarea no requiere un espacio de trabajo de ROS para poder ser desarrollada. La tarea se programa en Python, y las únicas dependencias que necesita son `numpy` y `matplotlib`.

1) Planificación global (40%)

En esta parte de la tarea implementará y evaluará una versión simple del algoritmo *Rapidly Exploring Random Trees* (RRT) aplicado a problemas de navegación 2D para un robot diferencial. Los mapas de entrada para el algoritmo son imágenes binarias con un sistema de coordenadas cuyo origen coincide con el origen de las posiciones `(x, y)` (números reales) que considera la versión de RRT que usted implementará.

1.1) Implementación

Para implementar RRT debe completar varias funciones según las instrucciones proporcionadas en el código fuente base:

- Complete el constructor de la clase `Node` del código base. Esta clase caracteriza a un nodo del árbol que se construye en RRT.
- Complete el constructor de la clase `RRTPlanner` y el método `reset_tree`.
- Complete la función `sample_random_position`.
- Complete la función `get_nearest_neighbour`.
- Complete la función `get_new_position`.
- Complete la función `check_for_collision`.
- Complete la función `generate_rrt`.
- Complete la función `recover_plan`.
- Complete la función `plot_rrt` (para visualización).

1.2) Evaluación

Considere los mapas binarios predeterminados cuya especificación se encuentra en el código base (`exps.py`). Cada experimento señalado a continuación debe ser ejecutado en cada uno de estos mapas (método `rrt_exps`). Los mapas se especifican en tamaño por (`alto x ancho`), pero las coordenadas `x` e `y` van en horizontal (➡) y vertical (⬆), respectivamente.

- Para mapas libres de obstáculos de `64 x 64` (`height x width`), y `max_attempts=1`:
  - Para `traverse_distance = 2.0`:
    - Reporte el resultado de aplicar RRT con `nb_iterations ∈ {10, 100, 500}`.[1]
  - Para `nb_iterations = 200`:
    - Reporte el resultado de aplicar RRT con `traverse_distance ∈ {1, 2, 4}`.
- Repita el experimento anterior para un mapa de `64 x 128` (`height x width`), pero agregue estos resultados en un anexo de su informe (sólo las imágenes obtenidas).

- Para mapas de `64 x 64` (`height x width`) y `max_attempts=1`:
  - Agregue los obstáculos especificados en `add_map_obstacles_1`.
  - Especifique una posición objetivo en `(x, y) = (55, 32)`.
  - Reporte los resultados obtenidos para `traverse_distance = 2.0` y `nb_iterations ∈ {500, 1000, 2000, 4000}`.

- Repita el experimento anterior con las siguientes modificaciones:
  - Agregue los obstáculos especificados en `add_map_obstacles_2`, posición inicial en `(2, 60)` y posición objetivo en `(x, y) = (60, 12)`.

- Repita el experimento anterior con las siguientes modificaciones:
  - Agregue los obstáculos especificados en `add_map_obstacles_3`, posición inicial en `(2, 60)` y posición objetivo en `(x, y) = (2, 2)`, y `nb_iterations ∈ {5000, 10000, 20000}`.

Realice un análisis a partir de los resultados obtenidos en términos de performance, costo computacional y calidad de los planes resultantes y conecte esto al algoritmo mismo y su implementación. Responda además las siguientes preguntas:

¿Cuál es el cuello de botella del algoritmo en la versión implementada? ¿Cómo podría hacer que el algoritmo sea más eficiente desde el punto de vista computacional?

[1] Esta notación implica que debe hacer un experimento por cada elemento del conjunto.

2) Planificación local (40%)

En esta parte de la tarea implementará el algoritmo *Pure Pursuit*, nuevamente considerando un robot diferencial en un ambiente 2D.

1.1) Implementación

Debe completar dos clases en función de las instrucciones proporcionadas en el código base: la clase `DifferentialRobot`, que sirve como un simulador extremadamente simplificado para un robot diferencial circular en 2D, y la clase `PurePursuitPlanner`, que sirve como planificador local de las instancias de `DifferentialRobot`.

- Clase `DifferentialRobot`:
  - Complete la función `set_pose` y `set_map`.
  - Complete la función `step`.
  - Para visualizar al robot moviéndose, revise las funciones `visualize` y `update_frame` (no es necesario que las edite).

- Clase `PurePursuitPlanner`:
  - Revise el constructor de la clase.
  - Complete la función `get_waypoint`.
  - Complete la función `get_local_pose`.
  - Complete la función `get_ctrl_cmd`.

1.2) Evaluación

Ejecute el método `pure_pursuit_exps` para que el robot diferencial siga un camino geométrico predefinido. Usando este método realice los siguientes experimentos:

- Para `kx=1` y `look_ahead_distance = 5.0`:
  - Instancie el robot en posición `(x,y) = (30, 10)` y orientación $\theta = 0$ (en dirección horizontal, ➡). Ponga una imágen del resultado obtenido una vez que el robot termina la trayectoria.
  - Repita para posición inicial `(x,y) = (20, 30)`
- Repita los experimentos anteriores para `look_ahead_distance = 1.0` (i.e., 2 experimentos) y luego para `look_ahead_distance = 10.0` (i.e., 2 experimentos).
- Repita los experimentos anteriores para límites de velocidad angular `[-0.1, 0.1]` para el robot diferencial (i.e., 6 experimentos adicionales en total).

Analice los resultados obtenidos en términos de los parámetros que varió. ¿Cómo se relacionan estos resultados con la implementación de *pure pursuit*? ¿Qué *trade-off* observa respecto a precisión de *tracking* y el parámetro `look_ahead_distance`? ¿Cómo afecta a lo anterior los límites de velocidad del robot?

3) Planificación global + local (20%)

En esta parte de la tarea el enfoque es en la integración de los algoritmos de planificación implementados en las partes 1) y 2). Para ello debe usar la función `integration_exps` y realizar los siguientes experimentos:

- Configure RRT con parámetros `nb_iterations = 2000` y `traverse_distance = 2.0`, y *PurePursuit* con parámetros `kx = 1` y `look_ahead_distance = 5.0`:
  - Para un mapa de `64 x 64` con obstáculos dados por `add_map_obstacles_1` ejecute `integration_exps` para generar un plan global y que el planificador local lo siga. Reporte el resultado obtenido. Si el robot colisiona, la simulación no parará, pero deberá reportar este último estado (el robot quedará atascado).
  - Repita el experimento anterior con un mapa de `64 x 64` con obstáculos dados por `add_map_obstacles_2` y luego con `add_map_obstacles_3`.
- Repita los experimentos anteriores con `PurePursuit` con parámetros `kx = 1` y `look_ahead_distance = 1.0` y luego con `look_ahead_distance = 10.0`.

Analice los resultados obtenidos conectando sus observaciones con los análisis previamente realizados para `RRT` y `PurePursuit`.

Indique cuales son las principales debilidades del sistema y seleccionando una de ellas, plantee una solución hipotética detalladamente.

Informe y código

- Escriba un informe con sus derivaciones, análisis, resultados y respuestas a preguntas teóricas.
- El informe no requiere introducción ni marco teórico.
- Adjunte el código utilizado para resolver los problemas (obligatorio). Puede crear funciones auxiliares y miembros de clase adicionales si así lo desea (pero no es necesario para completar la tarea).
- Para estandarizar las entregas, se solicita no modificar los colores ni las opciones de visualización de los objetos que se grafican en el código base. En la Figura 1 se muestran ejemplos de los gráficos esperados tras la realización de la tarea. En el caso de *path tracking* e integración, usted observará animaciones, pero debe reportar el resultado terminal del experimento (ya sea por éxito en la tarea o fracaso por colisión).

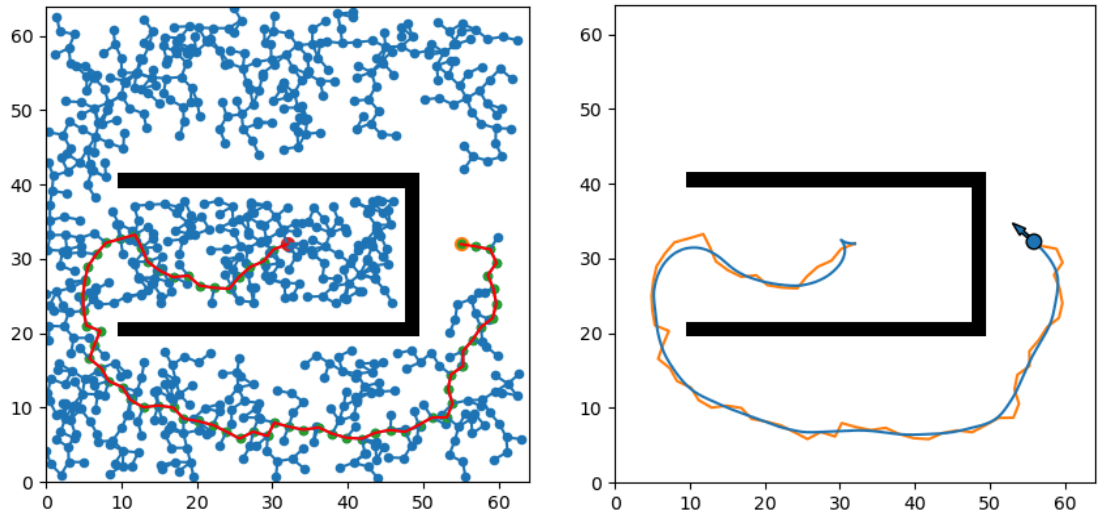

Figura 1. Izquierda: Ejemplo de resultado esperado para RRT. Derecha: ejemplo de resultado para integración RRT + *Pure Pursui*t (*snapshot,* en naranja el plan de la imagen de la izquierda, y en azul la trayectoria ejecutada por el robot diferencial).