Práctica 4
Planeación de rutas utilizando A* y celdas de ocupación

Laboratorio de Bio-Robótica

Robots Móviles y Agentes Inteligentes

Objetivos

- Familiarizar al alumno con el uso de mapas de celdas de ocupación.
- Calcular una ruta a partir de un mapa de celdas de ocupación utilizando el algoritmo A*.
- Publicar la ruta en un tópico y desplegarla en el visualizador `rviz`.

1. Introducción: el algoritmo A*

La planeación de rutas consiste en la obtención de un movimiento continuo libre de colisiones que conecte una configuración inicial, con una final. Se asume que se dispone de una representación del ambiente con información sobre el espacio navegable y el ocupado por los obstáculos. En esta práctica se considera que el robot sólo se mueve sobre un plano y que se tiene una representación que consiste en un mapa de celdas de ocupación (obtenido en la práctica 3).

Una posible solución es aplicar un algoritmo de búsqueda en grafos. En el caso de las celdas de ocupación, cada celda representa un nodo en el grafo y se considera que está conectada únicamente con aquellas celdas vecinas que pertenezcan al espacio libre. Para determinar los nodos vecinos se puede utilizar conectividad cuatro u ocho. En esta práctica se utilizará la conectivad cuatro.

A* es un algoritmo de búsqueda que explora la ruta con el menor costo esperado. Para un nodo n, el costo esperado $f(n)$ se calcula como

$$f(n) = g(n) + h(n)$$

donde $g(n)$ es el costo de la ruta desde el nodo origen hasta el nodo n y $h(n)$ es una heurística que determina *un* costo que se esperaría tener desde el mismo nodo n hasta el nodo objetivo. Este costo esperado de hecho subestima el valor real, es decir, se debe cumplir que $h(n) \leq g(n) \quad \forall\, n \in \ Grafo$.

En la búsqueda por A* se manejan dos conjuntos principales: la *lista abierta* y la *lista cerrada*. La lista abierta contiene todos los nodos que han sido visitados pero no expandidos y la cerrada, aquellos que han sido visitados *y* expandidos (también llamados nodos conocidos). El algoritmo 1 muestra los pasos en pseudocódigo para implementar A*.

El valor g asociado a cada nodo n representa el costo para llegar a n desde el nodo inicial. Como se puede observar en el algoritmo 1, dicho costo se calcula a partir de la distancia $d(n_1, n_2)$ entre dos nodos. Como primera aproximación se puede utilizar la distancia euclidiana, sin embargo,

Algoritmo 1: Búsqueda con A*

Datos: Grafo, nodo inicial, nodo meta
Resultado: Ruta óptima expresada como una secuencia de nodos
Cerrado $\leftarrow \emptyset$
Abierto $\leftarrow$ {nodo_inicial}
previo(nodo_inicial) $\leftarrow \varnothing$
mientras *Abierto* $\neq \emptyset$ **hacer**
  nodo_actual $\leftarrow$ nodo con el menor valor f del conjunto *Abierto*
  Abierto $\leftarrow$ Abierto - {nodo_actual}
  Cerrado $\leftarrow$ Cerrado $\cup$ {nodo_actual}
  si *nodo_actual es nodo_meta* **entonces**
    Anunciar éxito y salir de este ciclo
  fin
  para cada *nodo_vecino de nodo_actual* **hacer**
    //Los nodos vecinos se obtienen con conectividad cuatro
    si *nodo_vecino* $\in$ *Cerrado* **entonces**
      Continuar con el siguiente nodo_vecino
    fin
    si *nodo_vecino* $\in$ *Abierto* **entonces**
      costo_temporal $\leftarrow g$(nodo_actual) + d(nodo_actual, nodo_vecino)
      si *costo_temporal* $<$ *g(nodo_vecino)* **entonces**
        g(nodo_vecino) $\leftarrow$ costo_temporal
        f(nodo_vecino) $\leftarrow$ costo_temporal + heurística(nodo_vecino, nodo_meta)
        previo(nodo_vecino) $\leftarrow$ nodo_actual
      fin
    en otro caso
      g(nodo_vecino) $\leftarrow g$(nodo_actual) + d(nodo_actual, nodo_vecino)
      f(nodo_vecino) $\leftarrow g$nodo_vecino) + heurística(nodo_vecino, nodo_meta)
      previo(nodo_vecino) $\leftarrow$ nodo_actual
      Abierto $\leftarrow$ Abierto $\cup$ {nodo_vecino}
    fin
  fin
fin
si *nodo_actual* $\neq$ *nodo_meta* **entonces**
  Anunciar falla
en otro caso
  RutaOptima $\leftarrow \emptyset$
  mientras *nodo_actual* $\neq \varnothing$ **hacer**
    //El nodo actual se inserta al principio de la ruta
    RutaÓptima $\leftarrow$ {nodo_actual} $\cup$ RutaÓptima
    nodo_actual $\leftarrow$ previo(nodo_actual)
  fin
  Regresar RutaÓptima
fin

dado que se está trabajando con celdas de ocupación y se emplea conectividad cuatro, la distancia se puede calcular como un entero que incrementa su valor en uno con respecto al nodo anterior.

La heurística $h(nodo_{vecino}, nodo_{meta})$ también puede calcularse mediante distancia euclidiana, sin embargo, dado que se usa conectividad cuatro, el costo computacional se reduce si se emplea *Distancia de Manhattan* y dicha distancia se expresa en número de celdas.

2. Desarrollo

2.1. Prerrequisitos

Antes de continuar, actualice el repositorio y recompile:

```
cd ~/RoboticsCourses
git pull origin master
cd catkin_ws
catkin_make
```

Con el objetivo de facilitar las pruebas, se ha incluido un paquete llamado `navig_msgs` que contiene un servicio llamado `CalculatePath`. El archivo se encuentra en `navig_msgs/srv` y tiene el siguinte contenido:

```
geometry_msgs/Pose start_pose
geometry_msgs/Pose goal_pose
nav_msgs/OccupancyGrid map
---
nav_msgs/Path path
```

Como se muestra en el algoritmo 1, los datos de entrada para A* constan de un grafo, un nodo inicial y uno final. Como resultado se obtiene una secuencia de nodos que representan la ruta óptima. El servicio `CalculatePath` contiene todos los datos necesarios para ejecutar A*, sólo es necesario traducir las posiciones en coordenadas métricas a índices en el conjunto de celdas de ocupación y viceversa.

El paquete `map_server` contiene un nodo del mismo nombre que publica periódicamente un mapa de celdas de ocupación y atiende un servicio con el que se puede obtener dicho mapa. Este nodo recibe por parámetro el nombre del archivo `.yaml` que contiene la información sobre el mapa a utilizar, en este caso, `biorobotics_lab.yaml` (o bien el nombre del mapa construido en la práctica 3).

Para facilitar el llamado del servicio `navig_msgs/CalculatePath` se ha incluido una interfaz gráfica que se encuentra en `catkin_ws/src/hri/justina_simple_gui`. La figura 1 muestra una captura de pantalla de ella. Cuando se presiona `enter` en alguno de los campos del cuadro *Mobile base and navigation*, este programa llama al servicio de nombre `/navigation/a_star` de tipo `navig_msgs/CalculatePath` con los datos correspondientes en la parte de la petición (`start_pose`, `goal_pose` y `map`).

Para correr todos los nodos necesarios (`map_server, justina_simple_gui`, etc.) ejecute el comando

```
roslaunch bring_up hardware_simul.launch
```

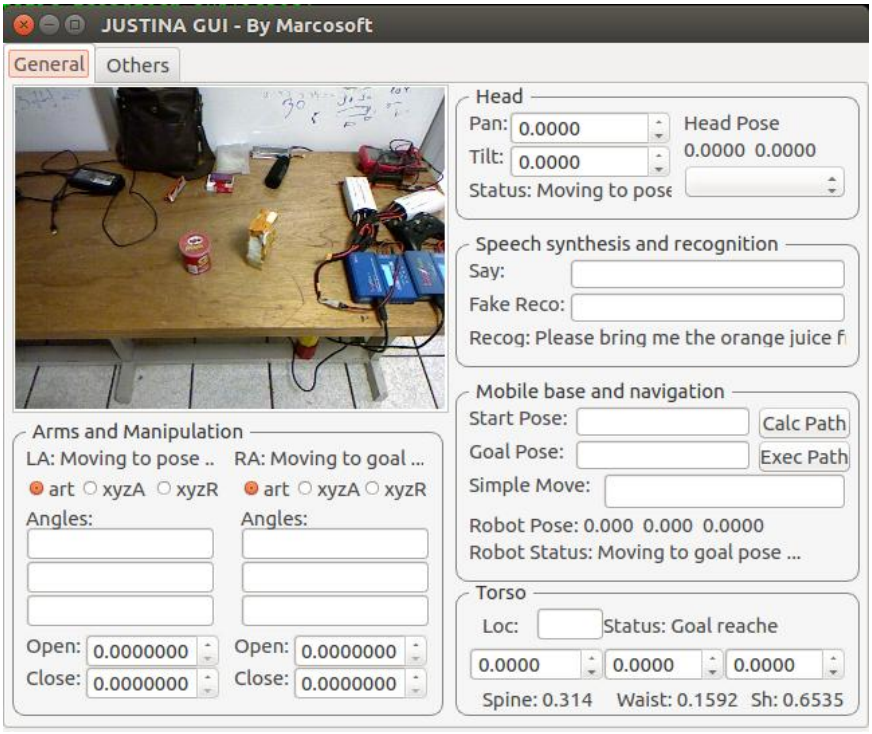

Figura 1: Interfaz gráfica

2.2. Nodo que calcula y publica la ruta

Cree un nuevo paquete de ROS con el nombre `path_calculator` que tenga las siguientes características:

- Atender un servicio con el nombre `/navigation/a_star`, de tipo `navig_msgs/CalculatePath`, que calcule una ruta a partir de un mapa y posiciones inicial y final.
- El valor de retorno del servicio (*true, false*) debe corresponder al éxito al calcular la ruta resultante y ésta debe asignarse a la parte de la respuesta (*CalculatePath::Response*) del servicio.
- El nodo debe publicar de manera periódica la última ruta calculada. El nombre del tópico debe ser `/navigation/a_star_path` de tipo `/nav_msgs/Path`. Esto con el objetivo de poder desplegar la ruta calculada en el visualizador *rviz*.
- Todos los algoritmos deben estar contenidos en funciones o métodos bien definidos. No debe haber *código espagueti*.

La figura 2 muestra un ejemplo del resultado esperado en `rviz`.

3. Evaluación

- El cálculo debe ser rápido (retardo no perceptible para un humano).
- El programa debe verificar que inicio y meta NO estén en el espacio ocupado.

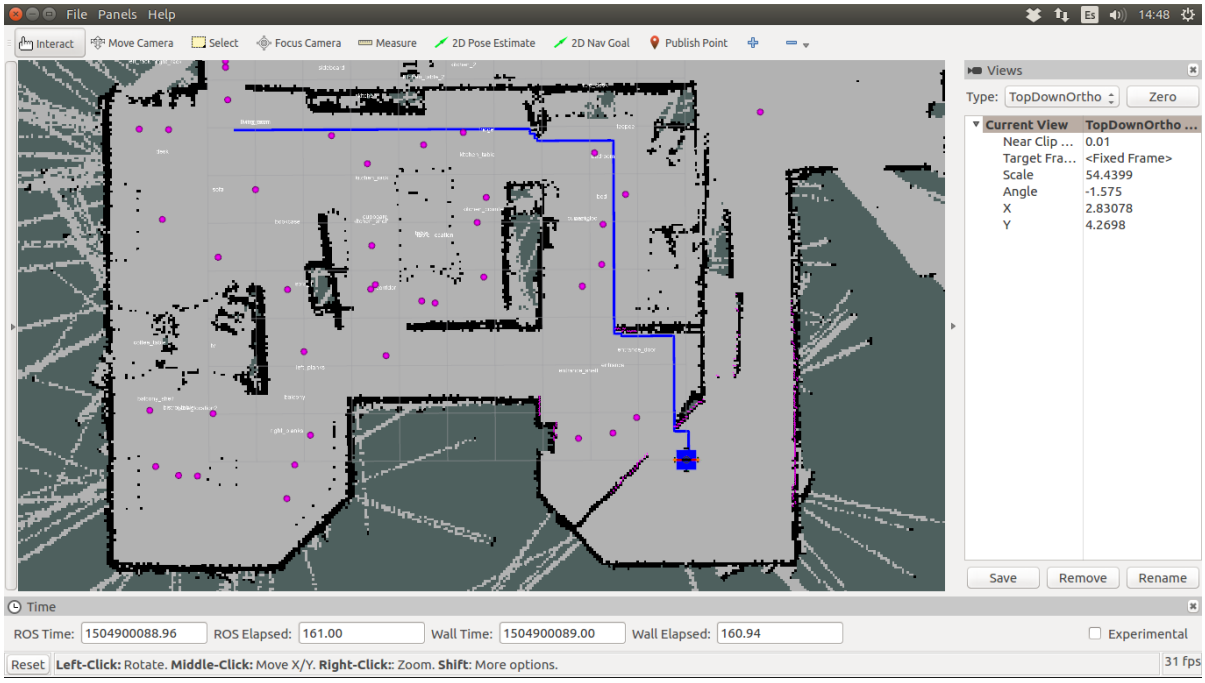

Figura 2: Ejemplo de ruta calculada

- Se probabará con varios pares de posiciones iniciales y finales aleatorias.
- Las posiciones iniciales y finales se deben poder cambiar en tiempo de ejecución y se fijarán haciendo uso de la GUI.
- El código debe estar ordenado.
- **Importante:** Si el alumno no conoce su código, NO se contará la práctica.