Taller 13: Planificación de caminos utilizando RRT

Introducción a la Robótica Movil

June 5, 2018

1 Introducción

En este trabajo práctico resolveremos el problema de la planificación de caminos utilizando el algoritmo RRT. A diferencia de otros métodos, el algoritmo RRT resulta muy versátil y permite aplicar ciertas restricciones a la planificación de manera de obtener una **trayectoria realizable**.

1.1 Esqueleto del código e implementación del algoritmo

Se trabajará sobre paquete **rrt_pioneer_planning** provisto y deberán completar el esqueleto del código presente en el archivo **/rrt_pioneer_planning/src/PioneerRRTPlanner.cpp**.

Dada la flexibilidad del algoritmo RRT, este puede adaptarse un gran número de aplicaciones y robots. De esta manera lo que se debe hacer es definir **el espacio de configuraciones del problema a tratar** y especificar la manera de operar entre estos.

En términos generales el algoritmo RRT se regirá por el siguiente código:

```
uint i = 0;
while(i < max_iterations_)
{
  if(isGoalAchieve())
  { path_found = true; break; }

  rand_config_ = generateRandomConfig();
  near_config_ = nearest();
  new_config_ = steer();

  if(isFree())
  {
    graph_[near_config_].push_back(new_config_);
    if(graph_.count(new_config_) == 0)
      graph_[new_config_] = std::list<SpaceConfiguration>();
  }

  i++;
}
```

De esta manera **solo deberán completar** las funciones `isGoalAchieve()`, `generateRandomConfig()`, `nearest()`, `steer()` y `isFree()`

Disponen de la estructura auxiliar:

```
struct SpaceConfiguration
{
  std::vector<double> config;

  SpaceConfiguration();

  SpaceConfiguration(std::initializer_list<double> l);

  double get(size_t n) const;

  void set(size_t n, const double& toSet);
};
```

Esta estructura permite trabajar con configuraciones de espacio **de dimensión arbitraria**. Pero al trabajar con nuestro robot diferencial Pioneer, el espacio de configuraciones tendrá **solo 3 componentes** (x, y, θ). Y podrán construir una configuración de la siguiente manera:

```
SpaceConfiguration c( {0.2, 0.5, M_PI} ); // NOTAR LOS { }

double c_x = c.get(0);
double c_y = c.get(1);
double c_theta = c.get(2);
```

NOTA: Deben utilizar '{' '}' dentro del constructor por estándar de C++11.

El árbol de configuraciones construido por RRT se define, entonces, como:

```
map<SpaceConfiguration, list<SpaceConfiguration> > graph_;
```

1.2 Experimento y configuración del .launch

Tienen disponibles dos escenas de vrep:
/rrt_pionner_planning/vrep/rrt_pioneer_planning.ttt
/rrt_pionner_planning/vrep/rrt_pioneer_planning_dificil.ttt

Podrán visualizar la experiencia en RViz utilizando:
/rrt_pioneer_planning/launch/rrt_pioneer_planning.rviz

Para ejecutar los nodos de ROS deberán utilizar:
/rrt_pioneer_planning/launch/rrt_pioneer_planning.launch

En este archivo de configuración podrán además calibrar los parámetros que gobiernan el comportamiento general del algoritmo **y deberán tenerlos en cuenta** al completar la implementación:

```
<param name="goal_bias" type="double" value="0.2"/>
<param name="max_iterations" type="int" value="20000"/>
<param name="linear_velocity_stepping" type="double" value="0.05"/>
<param name="angular_velocity_stepping" type="double" value="0.025"/>
```

- `goal_bias` ($\in [0, 1]$) determina el porcentaje de veces que la configuración random "cae cerca del goal".
- `max_iterations` (> 0) determina la cantidad de iteraciones que ejecutará el algoritmo RRT antes de darse por vencido en la búsqueda de un camino factible al goal.
- `linear_velocity_stepping` determina la velocidad lineal aplicada a la configuración near para la determinación de una nueva configuración factible en cada iteración del algoritmo.
- `angular_velocity_stepping` determina la velocidad angular aplicada a la configuración near para la determinación de una nueva configuración factible en cada iteración del algoritmo.

1.3 Visualización del árbol RRT resultante

Cada vez que se ejecuta el nodo de planificación se vuelca el árbol construido por el algoritmo en la ruta:
/.ros/rrt_graph.log

Se provee, además, de un script permite visualizar las configuraciones exploradas por el algoritmo:
/rrt_pioneer_planning/scripts/graph_rrt.py

Ejercicio 1: Especificación del problema

Establecer configuración aleatoria

Completar la función **generateRandomConfig()** de manera que retorne una configuración aleatoria **dentro del espacio de búsqueda** teniendo en cuenta:

- El `goal_bias_` porcentaje de las veces, la configuración aleatoria generada debe estar en un **"área cercana"** del goal.
- Las dimensiones de la grilla determinan el espacio de búsqueda, deberán utilizar funciones auxiliares **vistas en talleres anteriores** para establecer los límites de los valores de la configuración a generar.

Explicar como definieron el "área cercana al goal". ¿Tomaron en cuenta el angulo θ? ¿Cómo?.

Establecer configuración más cercana a la aleatoria

Completar la función **nearest()** de manera que retorne una configuración presente en el árbol, que tenga menor distancia para con la configuración aleatoria previamente calculada. Teniendo en cuenta:

- Deberán establecer una definición de distancia entre configuraciones.
- Cada configuración presente en el árbol tiene **un número limitado de "hijos"**, deberán comprobar que el nodo seleccionado tenga al menos una "relación libre".

Explicar que definición de distancia utilizaron. ¿Cómo integran el angulo θ?.

Establecer la nueva configuración

Completar la función **steer()** de manera que retorne una nueva configuración a partir de la "más cercana" previamente calculada. Esta nueva configuración debe obtenerse **de aplicar las velocidades** lineales y angulares configuradas (en el .launch). Deben tener en cuenta:

- Deberán establecer un **número finito** de posibilidades aplicando las velocidades establecidas **de diferentes formas**. Para esto deben tener en cuenta las cuentas necesarias para convertir de **polares a cartesianas**.
- Deberán filtrar las posibles configuraciones establecidas de manera de asegurar que estas **no existan ya como hijas**.
- Dentro de las posibles configuraciones seleccionar la más cercana a la configuración aleatoria.

Explicar como "discretizaron" el espacio de posibilidades a partir de la "configuración más cercana".

Comprobar la disponibilidad de la nueva configuración

Completar la función **isFree()** de manera que valide si la nueva configuración calculada **no produce colisiones** con objetos en el entorno. Para esto deben utilizar la función auxiliar provista en talleres anteriores **isPositionOccupy**. Deben tener en cuenta:

- No basta con comprobar que la configuración esta libre, el robot ocupa un espacio y todo este debe estar libre.
- **Todos** los objetos del entorno son cuadrados de `grid_->info.resolution` de lado.

Explicar como resolvieron esta comprobación.

Definir cuando se llego al goal

Completar la función **isGoalAchieve()** de manera que valide si la nueva configuración calculada y validada ya esta lo suficientemente cerca del goal. Deben tener en cuenta:

- La configuración final debe terminar **a menos de 0.1 metros** del goal. Y su orientación puede variar **únicamente en $\frac{\pi}{2}$ radianes**.

Ejercicio 2: Evidenciar fortalezas y debilidades del método

Para cada una de las escenas de vrep provistas plantear (configurando el archivo .launch):

- Dos ejecuciones, uno con `goal_bias` "bajo" y uno "alto".

- Dos conjuntos de valores para el stepping de velocidades. Uno con "velocidades muy demandantes" y uno con "velocidades poco demandantes".

Presentar un gráfico con el árbol generado por el algoritmo RRT en cada situación. ¿En que casos se pudo encontrar solución?, ¿Como resulto el seguimiento de la trayectoria a lazo abierto?. En los casos en que se pudo encontrar un camino, **presentar** una captura de pantalla con la posición final del pioneer en el **RViz**

¿Como se podría mejorar el algoritmo RRT de manera de "guíarlo" de manera más eficiente hacia el goal?. Contemplar el algoritmo A*.