Introducción a la Robótica Móvil

Primer cuatrimestre de 2018

Departamento de Computación - FCEyN - UBA

Planeamiento de caminos con RRT - clase 13

Algoritmo general de RRT

Algorithm 1 RRT

```
1:  i ← 0
2:  while (i < max_iterations) do
3:      if (isGoalAchieve()) then
4:          path_found = true;
5:          break
6:      end if
7:      rand_config_ = generateRandomConfig();
8:      near_config_ = nearest();
9:      new_config_ = steer();
10:     if isFree() then
11:         graph_[near_config_].push_back(new_config_);
12:         if graph_.count(new_config_) == 0 then
13:             graph_[new_config_] = std :: list < SpaceConfiguration > ();
14:         end if
15:     end if
16: end while
```

generateRandomConfig()

- El espacio de configuración para el Pioneer es (x, y, θ). Utilizamos una estructura de datos auxiliar: SpaceConfiguration.

Algorithm 2 RRT

```
1: structSpaceConfiguration
2: {
3: std :: vector < double > config;
4: SpaceConfiguration();
5: SpaceConfiguration(std :: initializer_list < double > l);
6: double get(size_tn) const;
7: void set(size_tn, const double& toSet);
8: };
```

Algoritmo general de RRT

Vamos a ver en detalle que hacen las funciones en azul:

Algorithm 3 RRT

```
1:  i ← 0
2:  while (i < max_iterations) do
3:      if (isGoalAchieve()) then
4:          path_found = true;
5:          break
6:      end if
7:      rand_config_ = generateRandomConfig();
8:      near_config_ = nearest();
9:      new_config_ = steer();
10:     if isFree() then
11:         graph_[near_config_].push_back(new_config_);
12:         if graph_.count(new_config_) == 0 then
13:             graph_[new_config_] = std :: list < SpaceConfiguration > ();
14:         end if
15:     end if
16: end while
```

generateRandomConfig()

- Genero un nuevo estado (x, y, θ) random en el mapa.
- Limites en el espacion cartesiano son:

$$orig_x = grid\_ \rightarrow info.origin.position.x$$
$$orig_y = grid\_ \rightarrow info.origin.position.y$$

- Usamos la función *getOriginOfCell* para obtener las coordenadas cartesianas, de la primera y ultima celda.
- Para la última celda tengo que sumar el tamaño de la misma (info.resolution).

generateRandomConfig()

¿Qué problema tiene esto?

generateRandomConfig()

¿Qué problema tiene esto?
El arbol va a crecer en todas las direcciones y voy a tardar mucho en converger.

Solución:

Defino un cuadrado alrededor del goal y pido que un porcentaje (*goal_bias_*) de las configuraciones random, caigan dentro de ese cuadrado. De esta forma, controlo hacia donde se expande el arbol.

nearest()

- Tengo que encontrar el nodo del árbol mas cercano a la configuración random que cree.

  ¿Que definición de distancia uso?

  $dist_{config} = ||(x_1, y_1), (x_2, y_2)|| + 0{,}5(\theta_2 - \theta_1)$

- Para recorrer el arbol recorro todas las claves del graph:
  $for(config : graph\_)$
  donde: $config =< spaceConfig, list < spaceConfig >>$
- Además, pido que cada nodo tenga como máximo 3 hijos.

steer()

- Defino las tres configuraciones nuevas (nodos) a las que el robot puede ir, dada la configuración del árbol (x_i, y_i, θ_i) elegida en nearest().
  $\theta_0 = \theta_i + wz\_step$, $\theta_1 = \theta_i$ y $\theta_2 = \theta_i - wz\_step$
  $x_0 = x_i + cos(\theta_0) Vx\_step$
  $y_0 = y_i + sin(\theta_0) Vx\_step$
  igual para x_1, y_1, x_2 y y_2
- Tengo que chequear que no existes esos nodos ya creados. De los que no existen agrego el mas cercano a $rand\_config\_$.

isFree()

- Chequeo que no este ocupada la pose elegida ni las poses alrededor de la misma.
- Puedo usar la función isPositionOccupy() para la configuración elegida, y para todas las poses alrededor de la pose elegida que corresponden a las celdas vecinas.

isGoalAchieve()

- Una forma de implementar esto es definir un área alrededor del goal y pedir que el robot esté dentro de esa área, y además pedir que el error en la orientación sea pequeño:
  $||(x_i, y_i), (x_g, y_g)|| < 1$
  $(\theta_i - \theta_g) < \pi/2$
- ¿Qué repercusiones tiene elegir un área y/o error de orientación muy pequeño?