Master Oficial en Sistemas Ingeligentes y Aplicaciones Numéricas en la Ingeniería

Universidad de las Palmas de Gran Canaria

Trabajo para la asignatura: Sistemas autónomos inteligentes

Planificación y localización de un sistema autónomo en un ambiente bidimensional

Autor: Sergio Marrero Marrero

Tutor de la asignatura:
José Isern Gonzáles

Índice

1. Introducción

El contexto de este documento es la asignatura Sistemas Autónomos Inteligentes del Máster Oficial de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería (SIANI)(año académico 2015/2016). En este documento se llevan a cabo dos implementaciones algorítmicas utilizando la herramienta Matlab. La primera de ellas consiste en un algoritmo de planificación y la segunda un algoritmo de locaclización. El ambiente en el que se encuentra el robot autónomo es bidimensional.

1.1. Aspectos de interés

La carpeta que se presenta contiene el archivo *planningAndLocalization.m*, desde el cual se deberá ejecutar los distintos programas que se han implementado.

```
1 -  clear
2 -  clc
3    %1) Cargamos mapa
4 -  load mapWorld1
5 -  map=mapWorld1;
6    %% 2) Elegimos metodo
7 -  option=input('random(r) or manual with mouse(m) or manual for only Localization(ml)','s');
8    % Esta funcion genera START y GOAl
9 -  [start,goal] =startAndGoal(map,option);
10   %% 3) PLANNING %%
11 -  [findWay] = waveFrontPlan( mapWorld1,start,goal );
12
13   %% 4) LOCALIZATION %%
14 -  [whereTheRobotIs] = localizationFunction( map,start);
15
16   %% 5) INTEGRATION %%
17 -  [whereTheRobotIs] = localizationFunction( map,start);
18 -  [findWay] = waveFrontPlan( map,whereTheRobotIs,goal );
19
```

Figura 1: Archivo principal: planningAndLocalization.m

En la imagen 1 se puede ver el archivo, el cual está separado en distintos bloques. Para facilitar un poco la tarea se explica a continuación:

- En el primer bloque se carga el mapa sobre el que se desea trabajar
- En el segundo bloque se elige el método: 1) random(r): Se escoge START y GOAL de forma aleatoria 2) manual with mouse (m) : Se escoge START y GOAL de manera manual con el ratón 3) manual for only Localization : Se escoge solamente START. Esta tercera opción está pensada para si sólo se quiere comprobar la parte de localización
- En el tercer bloque se ejecuta el planeador implementado.
- En el cuarto bloque se ejecuta el localizador implementado.
- En el quinto bloque se ejecutan ambos algoritmos conectados entre sí, de tal forma que se comienza con una etapa de localización seguida de una etapa de planificación.

En las siguientes páginas se describirá brevemente el contenido del trabajo realizado. **Cabe advertir que la numeración que aparece a la derecha de las distintas líneas de los pseudocódigos desarrollados en esta pequeña memoriatienen su equivalente en los código originales en Matlab. De tal forma que cada número hace referencia a un lugar concreto de cada función original.**

2. Planificación

2.1. Objetivo

El objetivo será encontrar uno de los posibles caminos mas cortos entre dos puntos. El punto de partida se denominará START y el de llegada GOAL. El método que se utilizará será un método de planificación no informado. Este método se denomina : Planificador de frente de onda.

2.2. Descripción de las funciones principales

Las funciones que se han escrito para esta tarea son:

- Funciones de apoyo
  - startAndGoal(...): Permite seleccionar un punto de START y un punto de GOAL. Se podrá hacer de manera manual (usando el ratón) o de manera aleatoria.
  - plotStartGoal(...): Esta función permite plotear de manera cómoda y eficaz.
- Funciones fundamentales
  - updateMapandWaveFront(...): Permite expandir la onda a través de todo el mapa. Esta función aumenta una unidad (en valor) en las fronteras del frente de onda. Esta detallada en el algoritmo 1.
  - findingWay(...): Encuentra el camino entre GOAL y START siguiendo el descenso provocado por el método de expansión de la onda. Está detallada en el algoritmo 2
  - waveFrontPlan(...): Esta es la funcion principal y tiene la finalidad de articular a las demás. Está detallada en el algoritmo 3.

Las funciones updateMapandWaveFront(...), findingWay(...) y plotStartGoal(...) están autocontenidas en la función waveFrontPlan(...).

2.3. Análisis y conclusiones

Este planificador de ruta es eficiente en cuanto a que encuentra un camino mínimo entre el START y GOAL. Sin embargo no es eficiente en cuanto al modo en el que realiza la búsqueda, ya que expande la búsqueda de la solución hacia todas las direcciones. Los métodos informados resolverían el problema de una forma bastante más eficiente. Debido a que solo se añadiría una serie de cálculos de la distancia para informar acerca de qué direcciones provocan un alejamiento del GOAL, no aumentaría demasiado la carga computacional, con lo cual estaría completamente justificado preferir un planificador informado a uno no informado. Cuanto mayor es el número de celdas (o bien porque el mapa es mas grande, o bien porque el tamaño de las celdas es menor) mayor se vería reforzado el interés en implementar un método informado en lugar de uno no informado.

Algorithm 1 Propagacion de la onda

```
1: function updateMapandWaveFront(Arguments Out,Arguments In)
2:     Arguments Out: map, frontWave, finish
3:     Arguments In: map, goal, frontWave
4:
5:     newfrontWave=[ ];          ▷ 1) Esta variable contiendrá el frente de onda actualizado
6:     for cada casilla del frente de onda do                                        ▷ 2)
7:         for cada dirección posible do      ▷ Arriba, abajo, izquierda, derecha    ▷ 3)
8:             if es posible moverte then                                            ▷ 4)
9:                 Esta posición pertenecerá al nuevo frente de ondas
10:                Actualiza: map          ▷ 5) Adicionamos una unidad mas en esta casilla
11:                Actualiza: newfrontWave                ▷ 6) Guardamos esta posición
12:            end if
13:        end for
14:    end for
15:    frontWave=newfrontWave;                ▷ 7) Asignamos el nuevo frente de onda.
16:    if frontWave contiene a goal then                                             ▷ 8
17:        fin del algoritmo: finish=1;
18:        return: Arguments Out
19:    end if
20: end function
```

Algorithm 2 Buscando el camino

```
1: function findingWay(Arguments Out,Arguments In)
2:     Arguments Out: way
3:     Arguments In: map, start, goal
4:
5:     way=goal                                  ▷ 1) Iniciamos el camino en goal
6:     while no llegues a start do                                                   ▷ 2)
7:         Calcula el gradiente en todas las direcciones y almacénalo                ▷ 3)
8:         Elige una dirección de avance. Para esta dirección hacemos:               ▷ 4)
9:         if no hay nada y el gradiente es > 0 then                                 ▷ 5)
10:            Esta casilla pertenece al camino: Actualizamos way                    ▷ 6)
11:        end if
12:    end while
13:    return: Arguments Out
14: end function
```

Algorithm 3 Función principal

```
1: function waveFrontPlan(Arguments Out,Arguments In)
2:     Arguments Out: findWay
3:     Arguments In: map, start, goal
4:
5:     while finish=0 do                          ▷ 1) Mientras no llegues a GOAL
6:         [finish]=waveFrontPlan(...)
7:     end while
8:     [findWay] = findingWay(map,start,goal)                                        ▷ 2)
9:     return: findWay
10: end function
```

3. Localización

3.1. Objetivo

El objetivo será generar un algoritmo que permita al robot encontrar cual es su posición, conocido el mapa completo. Para ello se valdrá de lo que sus sensores captan de sus entorno. En este problema el entorno se idealizará con el conocimiento de qué hay en sus celdas inmediatamente contiguas. En resumidas cuentas, se colocará al robot en alguna posición del mapa y este deberá averiguar su posición.

3.2. Descripción de las funciones principales

Las funciones que se han escrito para esta tarea son:

- Funciones de apoyo
  - plotMap(...): Esta función permite plotear de manera cómoda y eficaz.
  - startAndGoal: Inicia al robot en algún punto del mapa.
  - selectDirection2Move(...):Elige una dirección dentro de las posibilidades.
  - putPoints(...): Ayuda en labores de ploteo y representación.
- Funciones fundamentales
  - robotMove(...): El robot se mueve en la posición indicada y actualiza algunas variables importantes
  - localizationFunction(...): Es la función principal. Articula al resto de funciones. Se describe en el algoritmo 6.
  - whatRobotSee(...): Genera con una representación matricial(3x3) del entorno del robot.Está detallado en el algoritmo 4.
  - matchingMapEnviromentMatrix(...): Inspecciona el mapa con la matriz obtenida en la función *whatRobotSee* comprobando que puntos del mapa contienen este entorno. Está detallado en el algoritmo 5.

Las funciones whatRobotSee(...), matchingMapEnviromentMatrix(...), selectDirection2Move(...) , robotMove(...) y putPoints(...) están autocontenidas en la funcion localizationFunction(...).

3.3. Análisis y conclusiones

El algoritmo que se ha implementado es una simplificación muy sencilla de los casos que se podrían presentar en la realidad. Entre otras cosas, el espacio probabilístico con el que se determina la probabilidad de que haya algo sobre alguna casilla se está reduciendo a dos valores, TRUE o FALSE. Por otro lado, se está presuponiendo que el robot es capaz de orientarse en el espacio, y por esta razón es capaz de distinguir la orientación de las paredes con las que se encuentra.

El algoritmo que se utiliza para avanzar es el más sencillo posible, ya que camina en la dirección que puede. Sería conveniente diseñar un algoritmo que determinara qué dirección sería la que añadiría mayor información al robot, de esta forma se optimizaría el número pasos a dar hasta encontrar su posición.

Algorithm 4 Reconociendo el entorno actual

```
1: function whatRobotSee(Arguments Out,Arguments In)
2:     Arguments Out: enviromentMatrix
3:     Arguments In: map, pos
4:     for cada vecino contiguo de pos  do                                  ▷ 1)
5:         comprobar lo que hay y guardarlo en la matriz enviromentMatrix   ▷ 2)
6:     end for
7:     return: enviromentMatrix
8: end function
```

Algorithm 5 Comprobando el entorno del mapa

```
1: function matchingMapEnviromentMatrix(Arguments Out,Arguments In)
2:     Arguments Out: pointsWhereRobotCouldBe
3:     Arguments In: map, enviromentMatrix,candidates
4:     for cada candidato del vector candidates  do                         ▷ 1)
5:         comprobar si la matriz enviromentMatrix coincide                 ▷ 2)
6:         if coincide con enviromentMatrix then                            ▷ 3)
7:             incuir esta posicion en la lista pointsWhereRobotCouldBe
8:         end if
9:     end for
10:    return: pointsWhereRobotCouldBe
11: end function
```

Algorithm 6 Busqueda del mapa

```
1: function localizationFunction(Arguments Out,Arguments In)
2:     Arguments Out: whereTheRobotIs
3:     Arguments In: map,posStart
4:     Se declaran y modifican algunos parametros                           ▷ 1)
5:     pathThatRobotHaveDone=start                                          ▷ 2)
6:     refresh(map), plot(map)                  ▷ 3) Se actualiza mapa y se plotea
7:     [currentEnvMatrix]=whatRobotSee(map,posStart) ▷ 4) Se obtiene la matriz del enorno
8:     Se calcula la variable whereTheRobotCouldBe   ▷ 5) Se quitan los bordes del mapa
9:     [pointsWhereRobotCouldBe]=matchingMapEnviromentMatrix(...)           ▷ 6)
10:    refresh(map),plot(map)                   ▷ 7) Se visualizan estos puntos
11:    while length(pointsWhereRobotCouldBe)¿1 do   ▷ 8) Iniciamos bucle de búsqueda.
12:        Movemos el robot y actualizamos pathThatRobotHaveDone            ▷ 8)
13:        refresh(map), plot(map)                                          ▷ 9)
14:        calculamos nueva currentEnvMatrix                                ▷ 10)
15:        actualizamos pointsWhereRobotCouldBe                             ▷ 10
16:    end while
17:    return: whereTheRobotIs
18: end function
```