Curso 2016-2017
Grado en Ingeniería Informática
Universidad de Granada

Práctica 2: Algoritmo A*

Carlos Manuel Sequí Sánchez

8 de mayo de 2017

Índice

Índice de figuras

1. Resumen inicial

Para la resolución de la práctica he realizado todos los puntos que se piden en el guión, incluida la optimización.
Primeramente he realizado el algoritmo A* ordenando los nodos que se meten en la lista de abiertos con un functor, después he terminado de rellenar el método addNeighborCellsToOpenList para que se completasen todos los datos de un nuevo nodo.
Tras esto he procedido a implementar el camino más seguro haciendo uso de la función getCost del costmap.
Más tarde hice la optimización mediante pesos para elegir caminos de forma más rápida y, por último, los experimentos tomando medidas de metros recorridos, tiempo tomado y nodos expandidos.

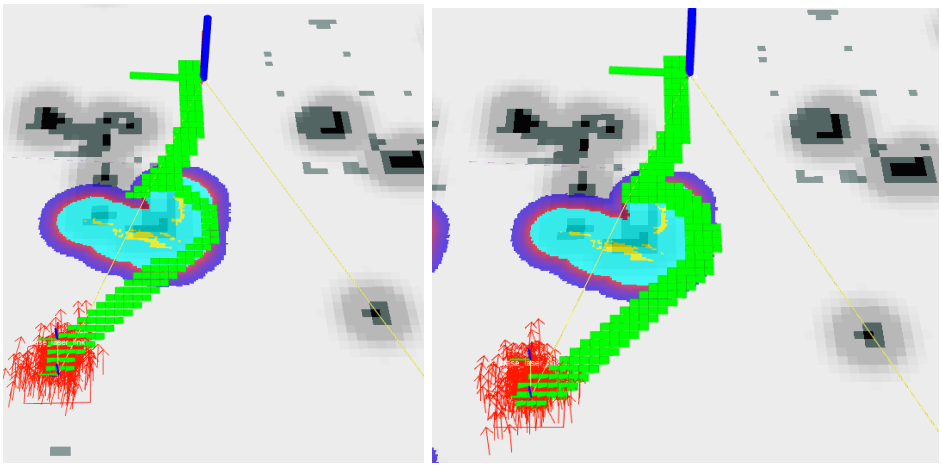
2. Creación del A*

El algoritmo inicial era claramente una búsqueda en anchura, la cual exploraba todos los nodos que rodeaban al robot. Para pasarlo a una búsqueda con algoritmo A* lo único que hice fue cambiar los contenedores "list" por contenedores "set" de la STL y crearme mi propio functor (CMP) para usarlo con el "set" y que a la hora de insertar nuevos nodos en la openList en el método addNeighborCellsToOpenList se ordenasen de forma automática por orden creciente de fCost.
Además del orden de los nodos por fCost, también hubo que terminar de implementar el método addNeighborCellsToOpenList de manera que se completasen todos los campos de los nodos que se iban a añadir a la openList (faltaban por completar los campos gCost, hCost y fCost).
También he tenido en cuenta la conservación de los mejores padres comparando en toda iteración los nodos generados y que además ya están en abiertos con los nodos que hay en abiertos, es decir, una vez generados los nodos hijo, vemos los que ya están en la lista de abiertos y, en caso de que el gCost del padre de estos sea mejor que los de la openList, actualizamos dichos valores en los nodos de la lista de abiertos.

3. Implementación del camino más seguro

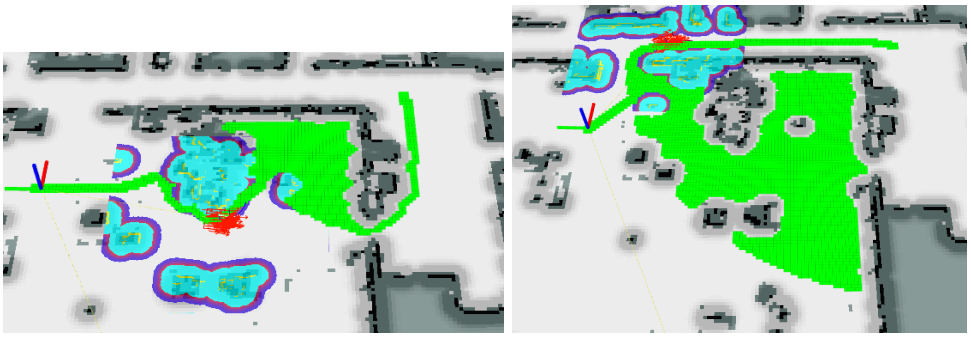
Para dicho propósito traté de utilizar la función footPrint implementada en el código proporcionado en la página web pero, tras comprobar y consultar con el profesor que la variable world_model_ daba problemas de enlazado a la hora de ejecutarse, decidí utilizar la función getCost del costmap, la cuál nos da información acerca de la proximidad que tiene una celda a un obstáculo.
Esta proximidad nos la da en un número del 0 al 255, de manera que cuanto más alto sea el valor devuelto por dicha función, más próxima se encuentra la celda a un obstáculo.
Sabiendo esto es fácil hacer uso de ella, pues nos basta con sumar el valor que devuelve

la función al fCost, de manera que para celdas cercanas a obstáculos, el valor de fCost sea mayor y por tanto, la celda afectada tenga menor probabilidad de ser escogida que otra celda con menor valor de fCost.
Como podemos observar en la siguiente comparación a la izquierda (a) el camino creado está ligeramente más próximo a los obstáculos que el camino de la derecha(b), el cuál ha sido implementado con la función getCost del costmap.



(a) Camino menos seguro (b) Camino más seguro

A continuación un ejemplo aún más claro del uso de esta función. Como vemos en las siguientes figuras, el robot es incluso capaz de decidir si un camino es demasiado peligroso como para tomarlo o no. En la figura (c) podemos ver como pasa por un estrecho, poniéndose en peligro por el hecho de que podría impactar con algún obstáculo. En la figura (d) podemos ver como gracias a la implementación realizada con el getCost el robot no se la juega a pasar por un camino tan estrecho y escoge uno alternativo mucho más seguro.



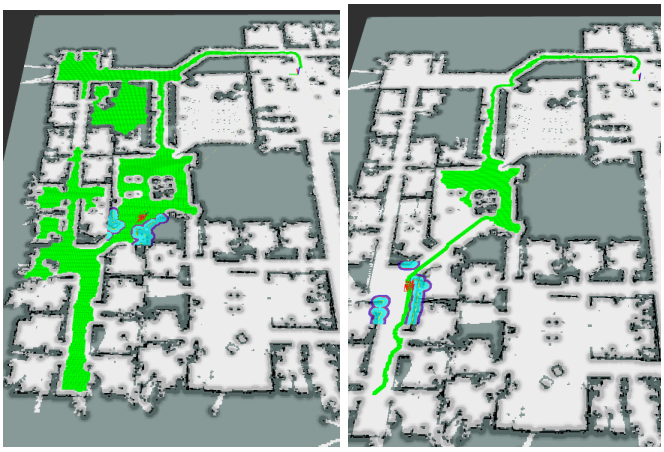
(c) Camino menos seguro (d) Camino más seguro

4. Optimización del algoritmo

Para la optimización del algoritmo he tenido en cuenta el método por pesos explicado en las clases de teoría, el cual se basa en crear una relación (peso) entre la distancia que hay de la posición inicial donde se encuentra el robot y la posición del goal (distanciaInicioGoal) y la distancia que hay entre el nodo que esta siendo evaluado y el goal (distanciaNodoGoal), de manera que nos queda lo siguiente:

$$Peso = \frac{distanciaNodoGoal}{distanciaInicioGoal}$$

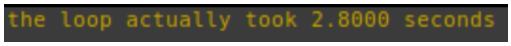
Una vez obtenido dicho peso para cada uno de los nodos evaluados (en cada iteración) simplemente lo multiplicamos por el valor de fCost, de manera que cuanto más lejos se encuentre el nodo evaluado, mayor sera el valor del peso, por lo que también será mayor el valor del fCost y por tanto, tendrá menos prioridad a la hora de ser escogido para seguir expandiendo por ahí. Este método de optimización hace que se encuentren planes mucho antes que sin la mejora como veremos en las experimentaciones a continuación. Este es el ejemplo más claro que he logrado conseguir para mostrar la optimización del algoritmo. En el caso (a) tenemos un camino muy costoso con un tiempo de cálculo de 111 segundos (c) y en el caso (b) tenemos un camino muy poco costoso con un tiempo de cálculo de 2.8 segundos (d), una mejora muy sustancial sin lugar a dudas.



(a) Camino más costoso

(b) Camino menos costoso

(c) Camino más costoso



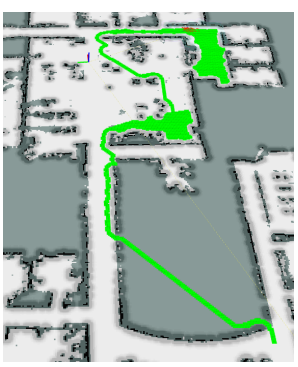
(d) Camino menos costoso

(e) Distancia recorrida: 65 metros

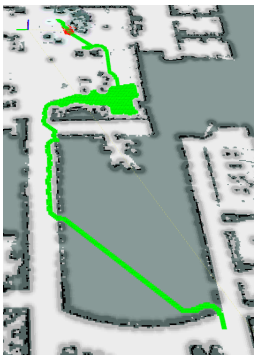
(f) 537 nodos recorridos en el algoritmo optimizado

5. Experimento 1

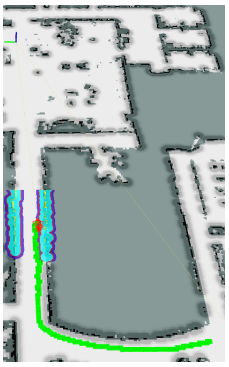
En este primer experimento podemos observar tanto el origen como el destino en la figura (a), cuyo recorrido tardó en ser calculado 5.7 segundos con un costo de 54.77 metros. El robot tuvo que recalcular la ruta 2 veces debido a que se dió cuenta de que no podía pasar por ciertos sitios: la primera vez (figura (b)) tardo 5.9 segundos en recalcularla y la segunda (figura(c)) tardó 1.3 segundos con un costo de 22.7 metros de distancia hasta el destino.
La figura (e) es el tiempo de cálculo que gastó pensando la ruta con el algoritmo optimizado. Esta vez solo tuvo que recalcular una vez la ruta (a la misma altura de donde se encuentra el robot en la figura (c)) y ni tan siquiera apareció el tiempo que tardó en hacerlo por pantalla. Mostramos los nodos recorridos en los dos cálculos realizados con la optimización (figuaras h e i).



(a) Cálculo 1



(b) Cálculo 2



(c) Cálculo 3

```
the loop actually took 5.7000 seconds
the loop actually took 5.9000 seconds
the loop actually took 1.3000 seconds
```

(d) Tiempos de cálculo

```
the loop actually took 4.5000 seconds
```

(e) Tiempo mejorado de cálculo (con pesos)

```
LA DISTANCIA DEL CAMINO ES: 54.772577
```

(f) Distancia 1 = 54.77 metros

```
LA DISTANCIA DEL CAMINO ES: 22.709591
```

(g) Distancia 2 = 22.70 metros

```
CANTIDAD DE NODOS RECORRIDOS EN EL PLAN: 467
```

(h) 467 nodos recorridos en el cálculo 1

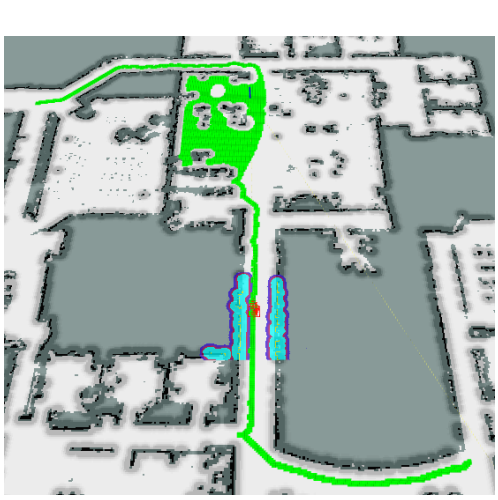
```
CANTIDAD DE NODOS RECORRIDOS EN EL PLAN: 189
```

(i) 189 nodos recorridos en el cálculo 2

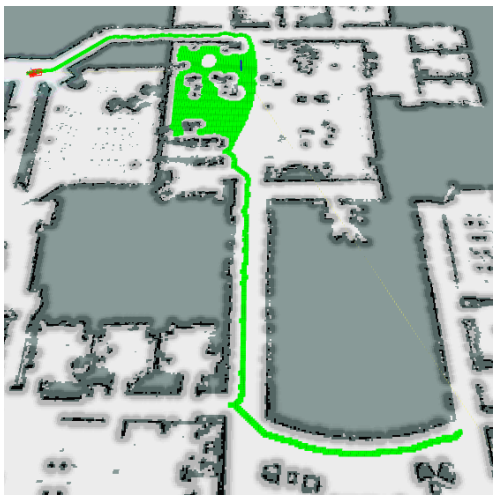
6. Experimento 2

En este experimento no obtuve mejoras ni en el tiempo de cálculo del recorrido ni en el propio recorrido, así que simplemente expongo el experimento con la figura (h) mostrando como el robot avanza hacia el objetivo y con la figura (i) mostrando como ha llegado ya al destino (arriba a la izquierda el punto rojo).

El tiempo de cálculo usado para este caso ha sido un total de 6 segundos y, la distancia recorrida en metros es de 65.9.



(a) Cálculo 1



(b) Cálculo 2

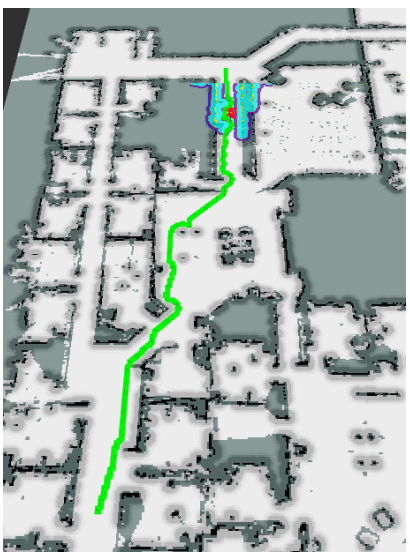
(c) Tiempo de cálculo = 6 segundos

(d) Distancia = 65.93 metros

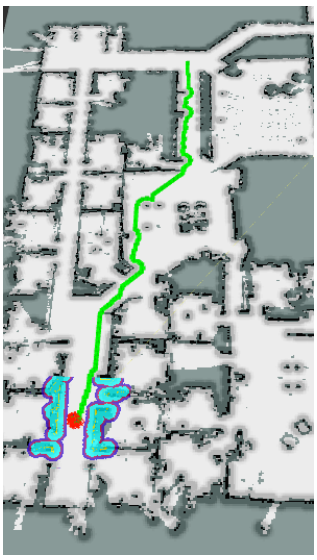
(e) 537 nodos recorridos

7. Experimento 3

Este es el último experimento realizado. En las siguientes figuras muestro tanto el inicio (figura(l)) como el final (figura (m)) del recorrido calculado por el robot en la zona izquierda del mapa completo.
No pongo ningún tiempo porque el cálculo del recorrido ha sido instantáneo, es decir, al indicar el recorrido que quería que hiciese el robot, no ha aparecido el tiempo en pantalla ni tan siquiera.
La distancia recorrida es de 41 metros



(a) Inicio del recorrido



(b) Final del recorrido

(c) Distancia = 41 metros

(d) 345 nodos recorridos