Técnicas de los Sistemas Inteligentes

Práctica 1: Planificación de Caminos en Robótica

UNIVERSIDAD DE GRANADA

Realizado por: Daniel Díaz Pareja
Fecha: 14/04/2018

Índice

1. Resumen....3

2. Tarea 1: Obtener el camino más corto y más seguro....3

2.1 Obtener el camino más corto: Implementación de A* básico....3

2.2 Obtener el camino más seguro....4

3. Tarea 2: Mejora del tiempo para encontrar una solución....6

3.1 Reducción del espacio de búsqueda....6

3.2 Ponderación de h(x) dinámicamente....7

4. Tarea 3. Experimentación....8

4.1 Episodio 1....8

4.1 Episodio 2....9

4.3 Episodio 3....9

1. Resumen.

Se ha mejorado la estructura de datos de manera que la lista de abiertos es un set y la de cerrados un unordered map.

Para seguridad, dada una casilla c, se han tenido en cuenta en el valor de h(x) los costes de n casillas adyacentes a c desplazadas en una distancia igual al radio mínimo del robot.

Para mejorar el tiempo par alcanzar una solución se ha modificado el concepto de “vecinos de x”, que son las 8 casillas a modo de “puntos cardinales” desplazadas una distancia r de x. También se ha ponderado dinámicamente h(x).

Se han realizado pruebas comparando tiempos de ejecución, nodos expandidos y longitud del camino encontrado.

2. Tarea 1: Obtener el camino más corto y más seguro.

Esta tarea consiste en extender una implementación de búsqueda en anchura, proporcionada por el profesor, para obtener el camino más corto y más seguro desde la posición actual del robot hasta la posición dada como objetivo a través de Rviz.

2.1 Obtener el camino más corto: Implementación de A* básico.

El algoritmo dado no elige el mejor nodo de la lista de abiertos, tal y como haría A*. Para que adopte este comportamiento, se ha utilizado como implementación de dicha lista un **set**, permitiendo así tener los nodos **ordenados** según su coste (f(x)). Una vez hecho esto, es trivial extraer el mejor nodo pendiente de abiertos (será el primer elemento del **set**).

Respecto a la lista de cerrados, se ha elegido un **unordered map** (tabla hash) para implementarla. Como sus tiempos de búsqueda son lineales, será más rápido reconstruir el camino una vez encontrada la casilla objetivo.

Además, el algoritmo proporcionado no tenía en cuenta el coste de g(x) (el camino recorrido hasta el momento) al calcular f(x). Se ha cambiado de manera que los nodos expandidos incorporen dicha valoración.

2.2 Obtener el camino más seguro.

Para obtener el camino más seguro se ha añadido una valoración a la h(x) del nodo (además de la distancia euclidiana al objetivo) que tiene en cuenta el coste de colocar al robot en una posición. Es decir, utilizamos el **radio mínimo** (r_min**)** del robot para calcular su "**huella**", y dada una posición del mapa, ver la peligrosidad de las casillas contenidas dentro del área determinada por "r_min" desde dicha posición. Finalmente, sumamos a h(x) el coste de las **n casillas** alrededor del robot, obteniendo un valor de seguridad de ruta. Como resultado, el planificador acaba ignorando casillas muy cercanas a las paredes:

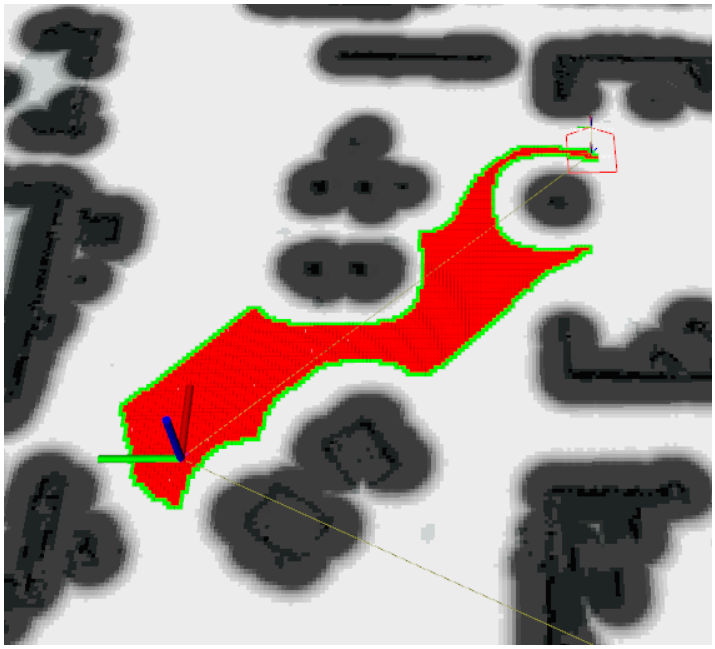
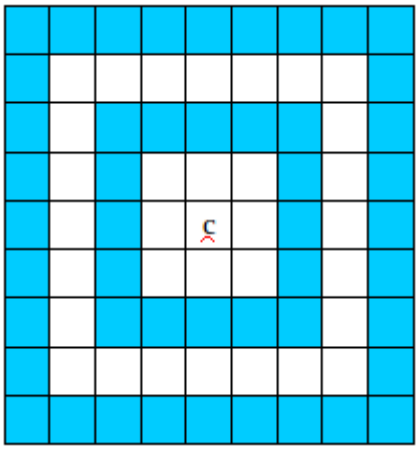


Figura 1: Seguridad del camino

Nota: las casillas contenidas dentro del área determinada por "r_min" desde una casilla "c" se tienen en cuenta de "2 en 2", para incrementar el rendimiento del cálculo de h(x). Podemos ilustarlo así



Donde **c** es la casilla cuya seguridad se quiere comprobar y las casillas azules son las que se tienen en cuenta para el cálculo.

Destacar que se ha disminuido el valor de **inflación** de los obstáculos (de 0.55 a 0.25), ya que ahora la propia heurística los está teniendo en cuenta. Si no hacemos esto, los obstáculos se verán demasiado grandes y habrá zonas que el planificador considere no seguras cuando sí lo son (como vemos en la Figura 1, que se consideran casillas demasiado alejadas del obstáculo del mapa real). Como resultado obtenemos algo así:

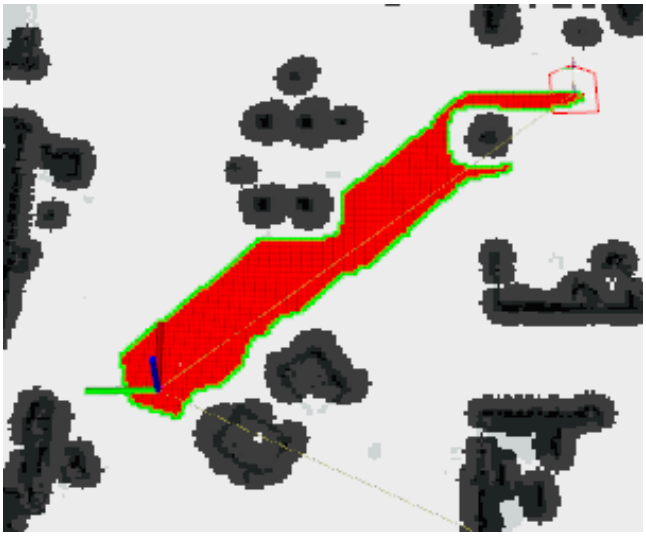


Figura 2: Inflación reducida

Como alternativa a destacar, se intentó utilizar el **radio máximo** del robot (sabemos que el robot es un pentágono irregular) en vez del radio mínimo para ver las casillas adyacentes a una posición concreta.

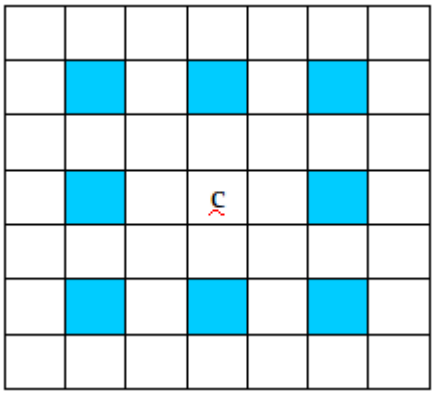
Haciéndolo así, había demasiados sitios por los que el robot, en realidad, sí que podía pasar, pero la heurística les daba un valor malo porque consideraba demasiadas casillas alrededor de dicha posición dada.

Por tanto, se llegaban a ver muchas “casillas peligrosas” que realmente no lo eran. Lo serían si el robot diese una vuelta de 360 grados en ese punto, ya que podría llegar a chocar, pero en este caso sería capaz de reorientarse, mediante su mecanismo de recuperación, para pasar correctamente. Por esto se ha acabado optando por elegir el radio mínimo.

3. Tarea 2: Mejora del tiempo para encontrar una solución.

3.1 Reducción del espacio de búsqueda.

El objetivo en este apartado es reducir el número de vecinos que se generan por cada casilla, o lo que es lo mismo, reducir el número de nodos a generar por el algoritmo. Para ello, en vez de considerar como vecinos de una casilla “c” las casillas directamente adyacentes a la misma, vamos a considerar 8 casillas distanciadas “r” posiciones de “c”, como si fuesen los puntos cardinales Norte, Sur, Este, Oeste, Noreste, Noroeste, Sureste y Suroeste. Gráficamente sería algo así:



Donde, en este ejemplo, r sería igual a 2.

Hay que tener en cuenta que ahora, como condición de A* para saber si estamos en el objetivo, habrá que comprobar si la casilla objetivo es una sub-casilla del conjunto formado por las casillas a una distancia r/2 de los nodos investigados.

En nuestro mapa, el valor de r es el radio mínimo del robot, de forma que se comprueban estas 8 casillas en los extremos del mismo.

Este nuevo proceso disminuye drásticamente el espacio de búsqueda, como podemos ver en la siguiente imagen:

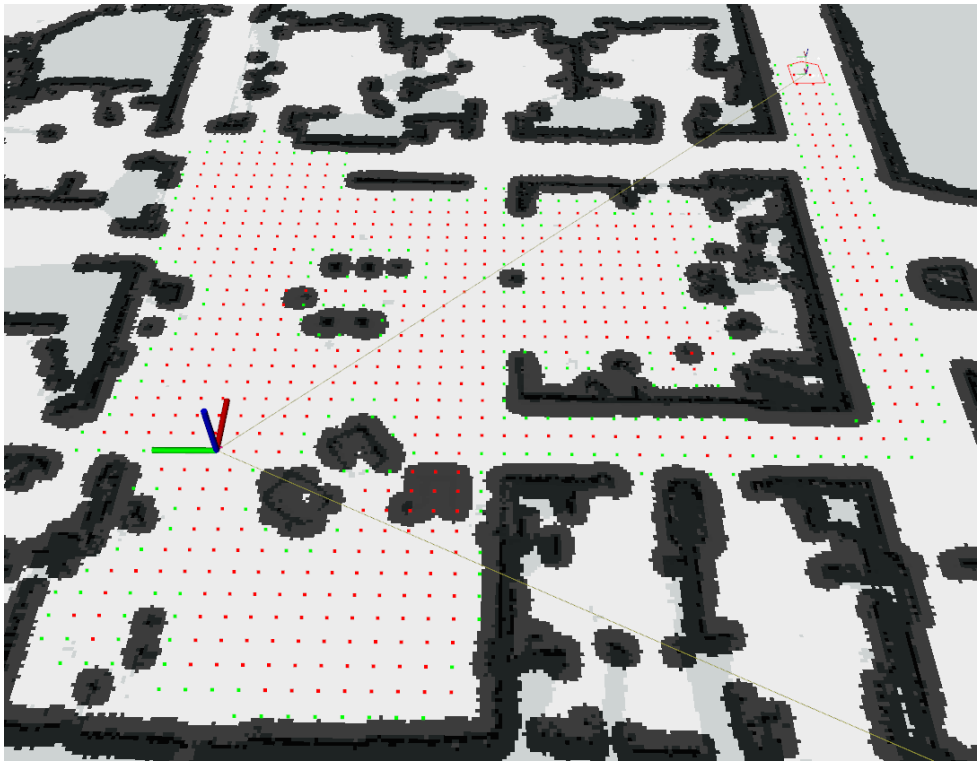


Figura 3: Reducción del espacio de búsqueda

No obstante, hay que tener en cuenta que para un valor de “r” muy elevado, podemos perder posibles rutas por las que llegar al objetivo (caminos que sean muy estrechos), ya que en definitiva estamos perdiendo precisión en cuanto a la representación interna del mapa.

En nuestro caso, al generar nodos a una distancia del radio mínimo del robot, nos aseguramos que el algoritmo investiga todas las zonas por las que este puede pasar.

3.2 Ponderación de h(x) dinámicamente.

Como última mejora se ha optado por multiplicar h(x) por un valor w, comprendido entre 0 y 1 y calculado como la distancia del nodo actual al objetivo dividido entre la distancia del nodo inicial al objetivo.

Esto permitirá que el algoritmo tenga más en cuenta la heurística si el objetivo está muy lejos, y tenga más en cuenta la longitud del camino final si nos encontramos cerca del objetivo.

En definitiva, es una manera de “orientar” al algoritmo para que al principio intente acercarse rápidamente a la zona del objetivo (como lo haría un algoritmo Greedy) y después tenga más prioridad el funcionamiento típico de A*, es decir, encontrar el camino óptimo.

4. Tarea 3. Experimentación.

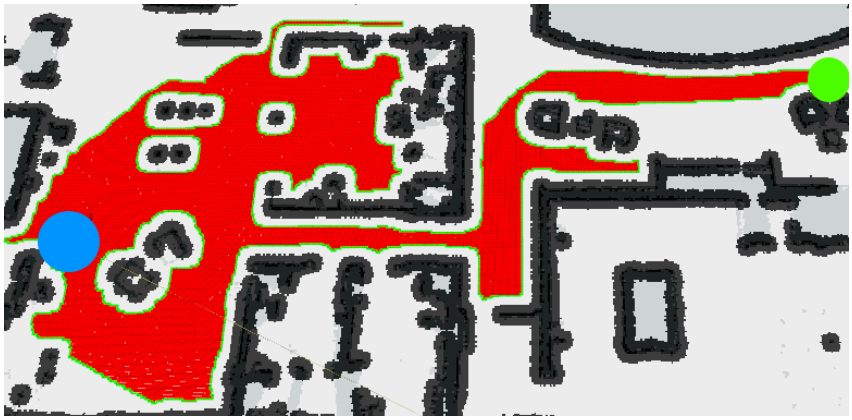
Se han realizado tres episodios de navegación con ambos algoritmos, para distintos puntos de inicio (círculo azul) y objetivo (círculo verde), el mapa de willow garaje (5cm). Se detallan a continuación con imágenes y con datos acerca del tiempo de ejecución, los nodos explorados y la longitud del camino encontrado en nodos.

Procesador utilizado: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz

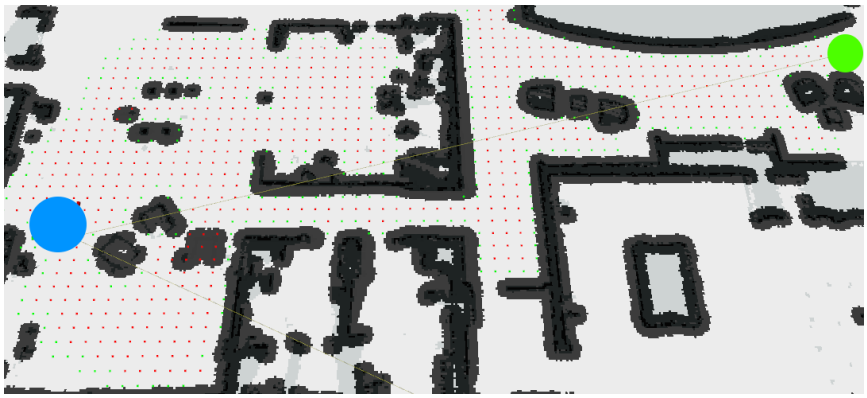
4.1 Episodio 1.

Algoritmo	T. ejecución (s)	Nodos explorados	Long. Camino (nodos)
Básico	72,458	24586	479
Mejorado	0,000813	620	71

Algoritmo básico:



Algoritmo mejorado:

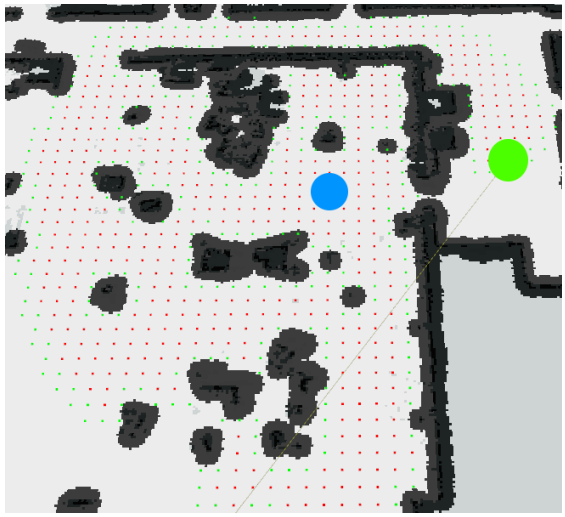


4.1 Episodio 2.

Algoritmo	T. ejecución (s)	Nodos explorados	Long. Camino (nodos)
Básico	101,632	29128	401
Mejorado	0,000617	475	58

Algoritmo básico

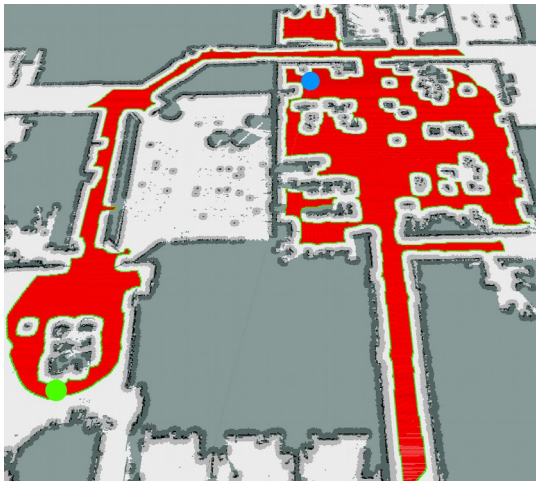
Algoritmo mejorado



4.3 Episodio 3.

Algoritmo	T. ejecución (s)	Nodos explorados	Long. Camino (nodos)
Básico	678,49	167591	1861
Mejorado	0,272413	1273	120

Algoritmo básico



Algoritmo mejorado