Universidad de Granada

Ingeniería Informática

Computación y Sistemas Inteligentes

Práctica 1: Problema de búsqueda de trayectorias

Autor: José Antonio Ruiz Millán
email: jantonioruiz@correo.ugr.es
Algoritmos: A* y modificaciones
Asignatura: Técnicas de los Sistemas Inteligentes
14 de abril de 2018

Índice

1. Descripción general

Para esta prática los pasos que he seguido son los que se especifica en el guión por lo que he realizado las siguientes tareas.

- En primer lugar, he sustituido el algoritmo de búsqueda en anchura por el **algoritmo A***, para mejorar la calidad de la búsqueda. Para tener en cuenta la seguiridad del camino, la función objetivo que evalúa el A* es la siguiente:

  $$f(n) = g(n) + h(n) \cdot (1 + seguridadCelda/127)$$

  Con esta fórmula conseguimos que si un camino es muy poco seguro, sea hasta 2 veces peor que un camino que es muy seguro, por lo que permitimos caminos el doble de largos si son muy seguros, mientras que penalizamos hasta el doble un camino que seas muy poco seguro.

  También compruebo con el footprint del robot, para las casillas con las que choca por el tamaño, si el valor de esa casilla es segura o no. En caso de no serla, no la cojo como casilla válida.

  La estructura de datos que he utilizado son: Para cerrados he utilizado un mapa sin orden donde el identificador es el índice del nodo y el valor es el nodo en sí. Por otro lado, para la lista de abiertos, he utilizado un multiset para tener los nodos ordenados por la función f. Por otra parte, para mejorar la búsqueda en la lista de abiertos, he utilizado un mapa sin orden donde la clave es el índice del nodo y el valor es un puntero al multiset de abiertos. Con esto conseguimos tener todas las búsquedas tanto en abiertos como en cerrados con *O(1)* y tenemos abiertos ordenados por la función $f(n)$.

  Con la implementación normal del algoritmo A* teniendo en cuenta esta función objetivo, ya tendríamos el primer paso terminado.

- En segundo lugar, para la mejora del algoritmo A* he utlizado algunas modificaciones como **busqueda con saltos**, **pesos dinámicos** y **búsqueda bidireccional**. En las páginas 2, 3 y 3 respectivamente, explico que cambios realizo para cada uno de ellos.

- Para el último apartado, he realizado algunas experimentaciones partiendos desde **3 puntos distintos y llegando a 3 puntos distintos** del mapa. Realizaré un análisis con imágenes y tablas que se muentran en la página 4 donde podremos diferenciar los distintos resultados.

2. Mejoras y alternativas A*

Todas las mejoras se realizan sobre el A* natural, no aplico mejora sobre mejora.

2.1. Búsqueda con saltos

Para esta modificación he realizado una búsqueda híbrida entre el algorimto A* y lo que denomino "saltos". Esta búsqueda consiste en cada vez que exploramos un nodo, exploramos los vecinos del mismo, pero cada x iteraciones de búsqueda, lanzamos una exploracíon en forma de estrella que visitará una línea recta de nodos en 8 direcciones hasta que llegue a un nodo por el que no

pueda pasar. Las direcciones que explora son: *delante, detras, derecha, izquerda, diagonal derecha superior, diagonal derecha inferior, diagonal izquierda superior y diagonal izquierda inferior*. Por eso decimos que explora en forma de estrella. Para ello, he creado unas funciones que va explorando 1 a 1 estos nodos y metiendolos en cerrados hasta llegar al último que lo deja en abiertos. Para cada uno de ellos se ha explorado sus vecinos con normalidad.

Con esto conseguimos aumentar el rango de búsqueda, y si alejandonos conseguimos estar más cerca del nodo, seguimos explorando a partir de ese.

La razón por la que decidí hacerlo híbrido es porque así conseguimos diversificar la búsqueda con los saltos y luego intensificamos la zona con el propio A*.

2.2. Pesos dinámicos

Para esta mejora, se utiliza un peso que va modificando al $h(n)$ de la siguiente forma:

$$f(n) = g(n) + w(n) \cdot h(n)$$

Teniendo en cuenta nuestra función $f(n)$ tenemos que:

$$f(n) = g(n) + w(n) \cdot h(n) \cdot (1 + seguridadCelda/127)$$

En mi caso, defino $w(n)$ como un valor que es alto si $h(n)$ (distancia hasta el destino) es alto, y va decreciendo conforme decrece el valor de $h(n)$. Con esto conseguimos que cuando un nodo está cerca de llegar, se centre y sea mucho mas problable que ese nodo siga expandiendose respecto otro nodo que está más lejos del destino. Por lo que tenemos que finalmente nuestra funcíon $f(n)$ es la siguiente:

$$distanciaIniFin \leftarrow calcularH(ini, fin)$$

$$f(n) = g(n) + (1 + nodo.valorH/distanciaIniFin) \cdot h(n) \cdot (1 + seguridadCelda/127)$$

Donde $calcuarH(ini, fin)$ devuelve la distancia euclídea entre ini y fin.

De esta forma, ya tendríamos el algoritmo A* con la mejora de pesos dinámicos funcionando.

2.3. Búsqueda bidireccional

Para esta mejora, lo que he realizado ha sido una **dupicación de la lista de abiertos y de la lista de cerrados**, utilizando una para ir desde el origen al destino, y otra para ir desde el destino al origen. La preferencia a la hora de escoger un nodo u otro es **exactamente igual que en algoritmo A***, por lo que para cada iteración, escogemos un nodo de la lista que va desde el origen al destino y otro nodo de la lista que va desde el destino al origen teniendo en cuenta el orden utilizando la función $f(n)$.

Una vez seleccionados los nodos, los pasos a seguir son iguales que en el A*, con la diferencia de que **la condición de parada** es cuando se encuentra un nodo que ya está en la lista opuesta a la que ívamos a explorar. Es decir, vamos a expandir un nodo de la listaA que ya está en listaB o viciversa. Una vez se cumple esta condición, paramos la busqueda y creamos el plan.

Una vez realizadas estas modificaciones, ya tendríamos el algorimto A* con la mejora de búsqueda bidireccional funcionando.

3. Experimentación

Antes de nada, mostraré **una tabla con los resultados** de todos los algoritmos para cada una de las pruebas. Para en los apartados posteriores llevar una ligera idea de como han funcionado cada uno de ellos.

Tabla 1: Tabla resumen experimentos

	Camino 1						**Camino 2**						**Camino 3**					
	E	**E%**	**P**	**P%**	**T (s)**	**T%**	**E**	**E%**	**P**	**P%**	**T**	**T%**	**E**	**E%**	**P**	**P%**	**T (s)**	**T%**
A*	24030	202.6	376	0.5	5.48	412.15	13586	1608.93	676	0.9	2.99	1658.82	185728	77.58	1313	**0**	42.8	183.44
Saltos	8744	10.11	430	14.97	1.07	**0**	1112	39.87	715	6.71	0.17	**0**	122376	17.01	1515	15.38	15.1	**0**
Pesos	7941	**0**	374	**0**	1.86	73.83	795	**0**	670	**0**	0.24	41.18	105108	0.5	1366	4.04	23.10	52.98
Bidireccional	44992	466.58	375	0.26	9.02	743	10292	1194.59	672	0.3	2.02	1088.24	104590	**0**	1345	2.44	20.74	37.35
Media%		226.43		5.24		409.66		947.8		2.64		929.41		31.7		7.29		91.26

La estructura de la tabla es, por cada fila tenemos un algoritmo distinto, por cada columna, para cada ruta, tenemos los datos de "nodos explorados", "tamaño del plan" y "tiempo de calculo para obtener el plan".

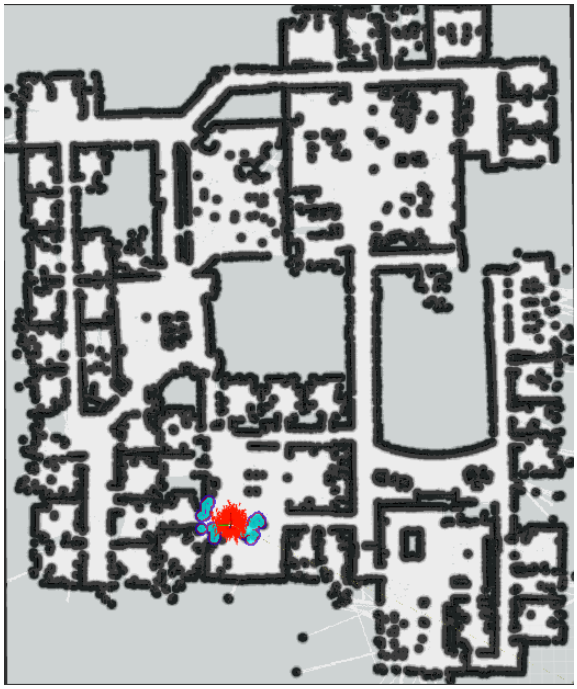
En esta tabla tenemos los siguientes elementos:

1. **E** → Nos dice el numero de nodos expandidos.
2. **P** → Nos dice el tamaño del plan.
3. **T (s)** → Nos dice el tiempo que ha tardado en calcular el plan.
4. **E/P/T%** → Nos dice el porcentaje en el que este elemento es peor respecto al mejor algoritmo para una característica determinada y un camino determinado.

En la tabla podemos sacar nuestras propias conclusiones, que podremos visualizar en el apartado siguiente. Vemos como los algoritmos utilizados mejoran en todos los casos (excepto en el camino 1 para el bidireccional) al algoritmo original A*.

Los caminos que yo he utilizado para los experimentos son los siguientes: **El origen está determinado por la posición del robot y el destino marcado en verde**.

- **Camino 1:**



Escogido por **cercanía pero poco seguro**.

- **Camino 2:**

Escogido por **lejanía pero fácil y seguro**.

- **Camino 3:**

Escogido por **lejanía y complejo**.

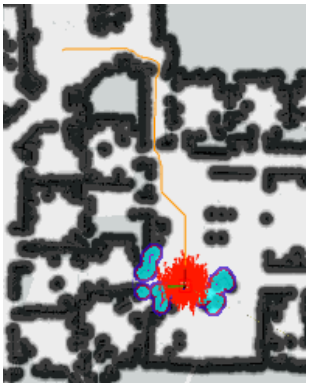
He de comentar que algunas posiciones del mapa no están bien representadas, es por ello que a la hora de expandir nodos, expande nodos que realmente no deberían existir porque el mapa no permite llegar a ellos pero la representación del mapa que tenemos nosotros si que lo permite.

3.1. Algorimto A*

3.1.1. Camino 1:

- Nodos expandidos:

- Camino:

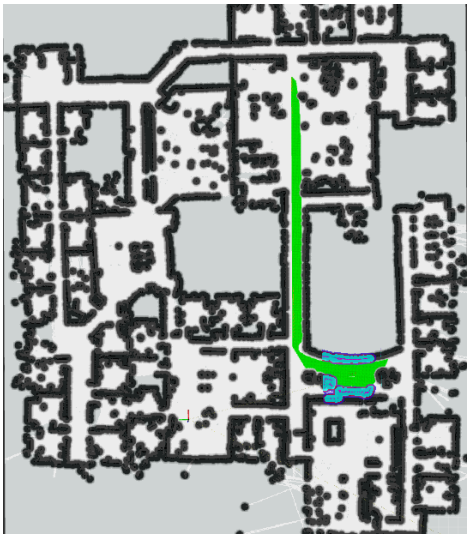


- Resultados

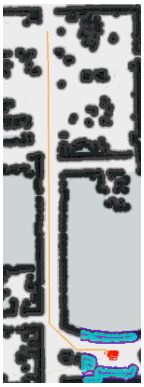
  Se pueden ver en la tabla 1

3.1.2. Camino 2:

- Nodos expandidos:



- Camino:

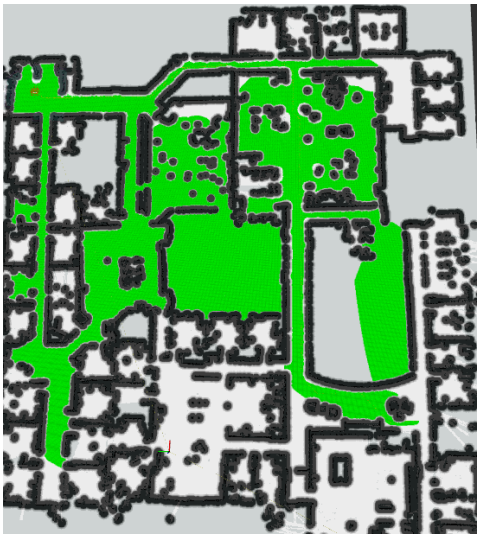


- Resultados

  Se pueden ver en la tabla 1

3.1.3. Camino 3:

- Nodos expandidos:



- Camino:

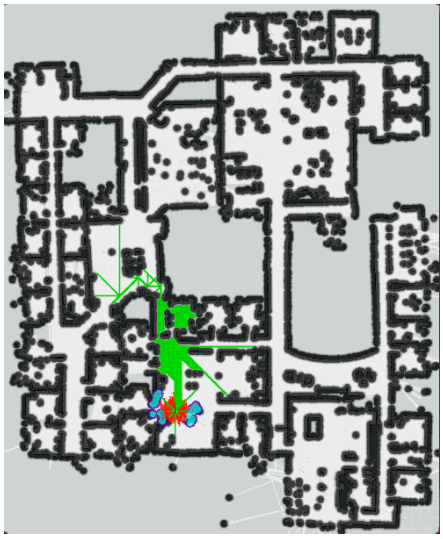
- Resultados

  Se pueden ver en la tabla 1

3.2. Modificacion de salto

3.2.1. Camino 1:

- Nodos expandidos:



- Camino:

- Resultados

  Se pueden ver en la tabla 1

3.2.2. Camino 2:

- Nodos expandidos:

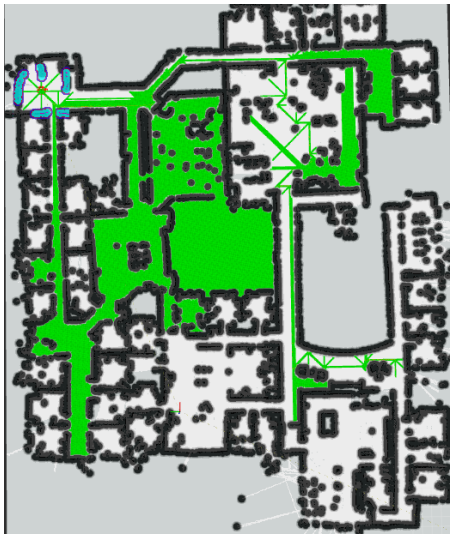
- Camino:

- Resultados

  Se pueden ver en la tabla 1

3.2.3. Camino 3:

- Nodos expandidos:



- Camino:

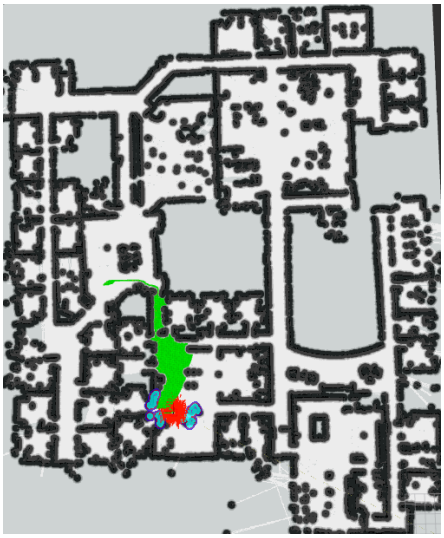
- Resultados

  Se pueden ver en la tabla 1

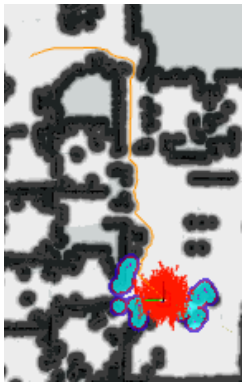
3.3. Modificacion de pesos dinámicos

3.3.1. Camino 1:

- Nodos expandidos:



- Camino:



- Resultados

  Se pueden ver en la tabla 1

3.3.2. Camino 2:

- Nodos expandidos:

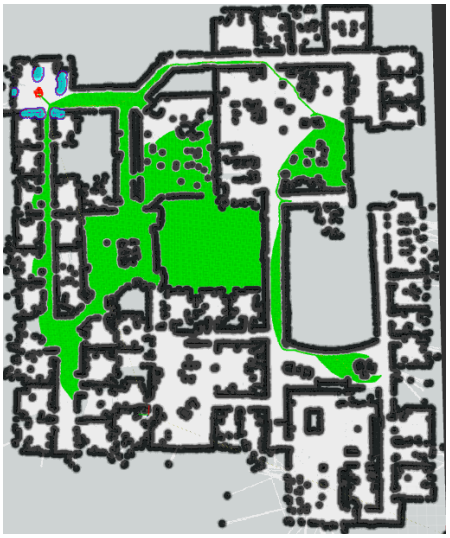
- Camino:

- Resultados

  Se pueden ver en la tabla 1

3.3.3. Camino 3:

- Nodos expandidos:



- Camino:

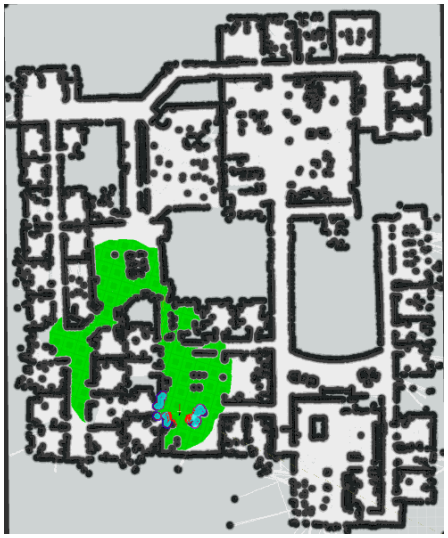
- Resultados

  Se pueden ver en la tabla 1

3.4. Modificacion de búsqueda bidireccional

3.4.1. Camino 1:

- Nodos expandidos:



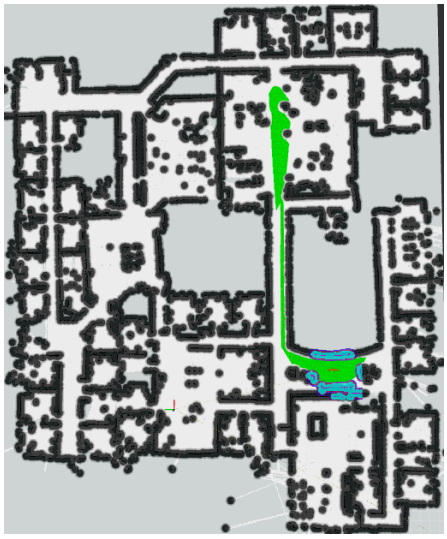
- Camino:

- Resultados

  Se pueden ver en la tabla 1

3.4.2. Camino 2:

- Nodos expandidos:



- Camino:

- Resultados

  Se pueden ver en la tabla 1

3.4.3. Camino 3:

- Nodos expandidos:

- Camino:

- Resultados

  Se pueden ver en la tabla 1

4. Conclusión

Finalmente podemos ver en la tabla 1 que aunque el algoritmo de la página 3 que modifica los pesos nos da mejores resultados, vemos en las imágenes que sacrificar tanto la distancia y el querer llegar tan rápido al destino nos crea unos caminos bastante menos seguros. Tambien vemos que el algoritmo de la página 2 que nos permite saltar a nodos más lejanos rápidamente nos tarda menos en tiempo, pero tambien obtenemos un camino que no es el más correcto. Por lo que a la hora de escoger un algoritmo, debemos de tener el cuenta que es lo que queremos y si debemos sacrificar el mejor camino posible por bajar el tiempo o no es necesario. En este caso tenemos el algoritmo de búsqueda bidireccional que no mejora tanto en tiempo y en nodos cargados como los demás pero consigue en un tiempo mejor que el A* dadnos un camino más seguro que el resto de algoritmos.