UNIVERSIDAD DE GRANADA

TÉCNICAS DE LOS SISTEMAS INTELIGENTES

Prácticas Grupo 2, Miércoles 17:30-19:30

Algoritmo A* y mejoras sobre este

Nombre: Adrián Jesús
Apellidos: Peña Rodríguez
Correo: adrianprodri@correo.ugr.es
DNI: 45604927-K
Fecha de entrega: 14/04/2018

1.- Resumen

La temática a tratar en esta práctica es el algoritmo A estrella y su implementación sobre ROS. Lo que se nos pide en la práctica es partiendo de una implementación en anchura del algoritmo A estrella, programar un algoritmo A estrella clásico.

Posteriormente, se nos pide que mejoremos nuestra implementación del A estrella para así conseguir hacer más eficiente dicho algoritmo, al menos en nuestro entorno.

En las siguientes páginas describiré detalladamente el procedimiento que seguí para la elaboración de esta práctica así como una parte final de experimentación con ambas implementaciones.

Por último haré una breve reseña acerca de la mejora conseguida y la eficiencia de esta.

2.- Introducción

En este apartado describiré las primeras mejoras realizadas sobre la búsqueda en anchura inicial:

- Partiendo del esquema que se nos proporciona, observe que la estructura de datos usada era bastante ineficiente ya que se usan listas para las listas abiertas y cerradas. Puesto que tenemos que hacer operaciones como buscar el nodo en el que la función f(x) = g(x) + h(x) sea menor, es decir, debemos minimizar dicha función, necesitaremos sacar en cada iteración el nodo en abiertos con menor f de entre todos lo que hay. Sabiendo esto, usando una lista tenemos que debemos de buscar mediante el uso de iteradores sobre la lista para al final recorrerla entera, esto es tremendamente ineficiente, por esto y por otras razones que no explicaré para no extenderme demasiado, decidí usar como estructura de datos, unordered_map de la STL.
- La g es básicamente la distancia euclidea de la celda actual a la posible expansión.
- Un unordered_map es un mapa no ordenado en donde la key será el index de nuestro nodo y el value será el nodo en sí. Puesto que la key no nos aporta ningún valor acerca de mínimos o prioridad, decidí usar un unordered y no un mapa normal.
- Una vez hecho esto me encontré con diferentes problemas, entre ellos la necesidad de hacer una nueva función para en caso de tener que buscar el nodo con menor f, sea lo más eficiente y rápido posible.
- Como punto final a esta introducción, he de decir que antes de llegar a usar los unordered_map probé otras muchas estructuras de datos como son la cola con prioridad, los mapas convencionales, los sets, etc y todos fueron descartados ya que no eran tan cómodos y rápidos de usar como los unordered_map, exceptuando el mapa convencional que su tratamiento es el mismo pero no necesitamos que nos ordene los nodos mediante el index por lo que es mejor el unordered ya que ahorramos esto.

3.- Implementación sin mejora

Lo primero que hice para hacer este algoritmo fue leer la teoría dada en las clases de teoría y posteriormente buscar información en diversas páginas hasta entender perfectamente su funcionamiento. Una vez hecho esto comencé a implementarlo y las cosas que puedo destacar acerca de la implementación son sobretodo las relacionadas con la estructura de datos descrita anteriormente.

Para empezar necesitaba una nueva función que me retornase el nodo con menor f y que sea lo más rápida posible. Para ello lo que hice fue declarar un vector de pares donde vuelco todos los elementos del mapa que me pasen como argumento y después aplico la función sort de la librería algorithm, esto lo podía haber hecho directamente sobre el mapa pero después de documentarme acerca de esta función leí que era más rápida ejecutarla sobre un vector y puesto que este vector será eliminado después de del retorno de la función, preferí desperdiciar algo de memoria pero ganar en velocidad.
Nota: Para copiar todo el mapa en el vector use la función copy de la librería algorithm la cual tiene una gran eficiencia.

Una vez creada la función y modificando algunas cosas más que no resultaron difíciles conseguí tener el algoritmo funcionando y empecé a experimentar con diferentes heurísticas. Intente con la distancia Manhattan, posteriormente visite la página web sobre el A estrella que se nos recomienda en el seminario y descubrí la distancia diagonal y la distancia octile pero después de probar con todas la distancia euclídea me proporcionaba mejores resultados.

Por último, le añadí a la heurística una componente que era la de seguridad de la celda a la que va, de este modo también se tiene en cuenta la seguridad al elegir casillas.

Una vez hecho esto me dí cuenta de que el robot parecía no ser muy consciente de su propio tamaño por esto decidí ser más específico con las casillas que se consideran vecinos, es decir, redefiní el concepto de vecindad. Para ello lo primero que hice es modificar la función footprintCost ya que la que viene por defecto no está bien implementada. En este función lo que hago es hallar el radio del robot (radio máximo hasta el punto más alejado del punto origen del robot) para posteriormente usar este para describir una circunferencia alrededor del robot y de este modo evaluar la seguridad de las casillas por las que pasa la circunferencia y obtener la media para de este modo decidir si el nodo es viable para expandir o supone demasiado riesgo. La circunferencia se describe centrando de manera virtual el robot en la casilla a expandir y ejecutando este procedimiento a raíz de esta casilla.

4.- Implementación con mejora

Después de muchas pruebas me di cuenta de varios problemas:

- Gran desperdicio de memoria a costa de ninguna mejora de velocidad.
- Heurística con mal comportamiento en gran parte de las situaciones.
- Lentitud a la hora de buscar caminos.
- Reinicios de la búsqueda debido a la lentitud del algoritmo.
- Poca velocidad y suficiente memoria libre como para poder desperdiciar un poco a costa de una ganancia de velocidad.
- Casillas a explorar muy pequeñas como para ir una a una.

El primero de los problemas, el uso de memoria elevado para la velocidad que se obtiene lo solvente añadiendo una mejora llamada “beam search”, dicha mejora la saqué de la página que se nos proporcionó en los seminarios. Básicamente consiste en eliminar de la lista de abiertos los nodos con menos posibilidades de expandirse una vez se ha alcanzado un número máximo de nodos en la lista (en este caso el límite lo sitúe a 180). Una vez alcanzado el límite se eliminan lo x nodos con menos posibilidades de expansión (80).
Esto se ejecuta cuando se va a coger el próximo nodo a expandir.

Para el segundo problema decidí colocar ponderaciones a la g y a la h de tal modo que se va a priorizar una de ellas o las dos según el número de iteraciones que llevemos hechas. Con esto conseguí una pequeña mejora sobretodo en determinadas situaciones en las que el robot está muy cerca de una pared y un posible camino.

El tercero, cuarto y quinto los solvente aplicando una técnica llamada bidirectional search la cual consiste en no solo buscar desde el robot hasta el nodo goal si no en buscar desde el nodo start al goal y desde el goal al start ya que es mejor tener dos listas pequeñas de nodos que una sola mucha más grande. Para implementar esto, tuve que añadir otros dos unordered_map (abiertosF y cerradosF) para poder buscar desde el nodo goal también. Una vez añadidas tuve que modificar la estructura básica a otra en la cual se tenga en cuenta también la otra dirección a buscar.
Me encontré con numerosos problemas a la hora de implementar esta mejora pero sin duda alguna el problema más notorio y que me llevo una tarde completa arreglarlo fue el de la modificación de la parte del algoritmo que genera el plan ya que con la búsqueda clásica del A estrella el plan se genera al revés y se le da la vuelta pero con esta implementación no es así y a parte de este problema tuve otros en relación con que el plan generaba errores que provocaron la finalización prematura del programa. Después de una tarde de probar mil cosas hallé la solución y funcionó. Después de ver el resultado, pensé que quizás sería buena idea que en vez de que cada parte intentase buscar su punto a llegar, después de un número de iteraciones se intentasen unir entre ellas, pero al probar esto obtuve como resultado una gran velocidad pero caminos que están muy distantes de ser óptimos por lo que decidí quitar esto último y dejar la bidirectional search convencional.

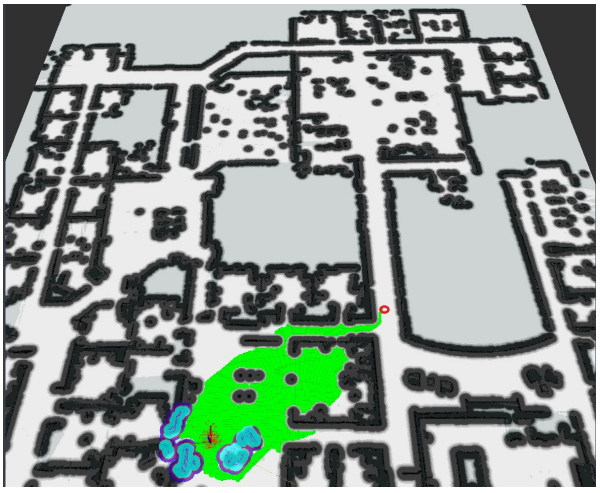
El último de los problemas fue el más sencillo de solventar, tan solo modifiqué el algoritmo de búsqueda de casillas libres y le indique que en vez de buscar de una casilla de distancia en una buscasé de tres en tres. También probe con 5, 6, 2, etc pero lo que me daba mejores resultados en el mapa de 5cm fue 3.

Estás son todas las mejoras realizadas, voy a pasar a mostrar la mejora que provocaron con respecto al A estrella convencional.

5.- Experimentos SIN MEJORA VS MEJORADO

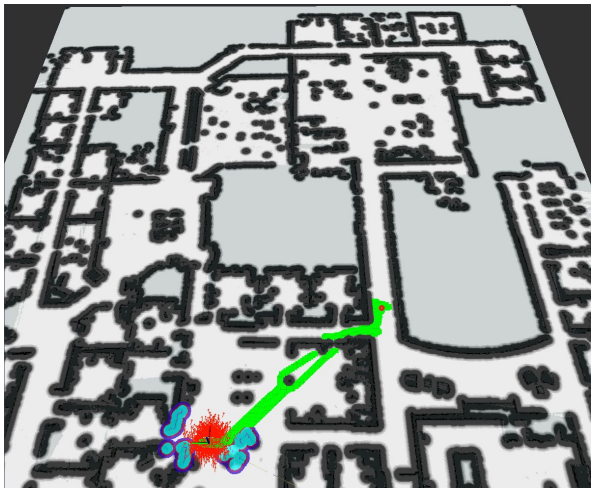
Caso 1:

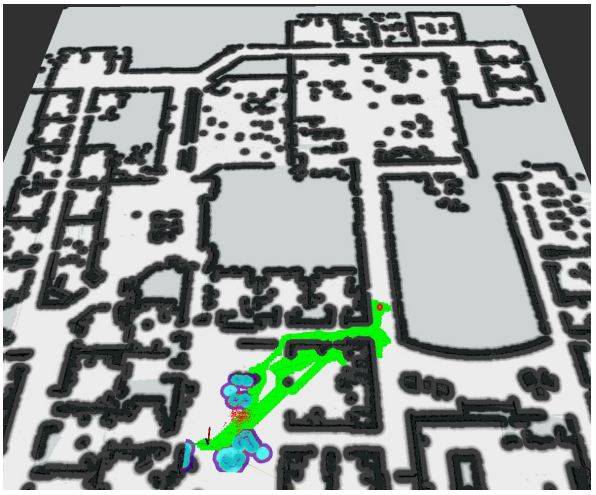
- Sin mejoras:



```
*****************************
Time taken: 61.50s
Longitud del camino: 292
Nodos en abiertos: 2104
Nodos en cerrados: 16063****
***************[ WARN] [1523
```

- Con mejoras:



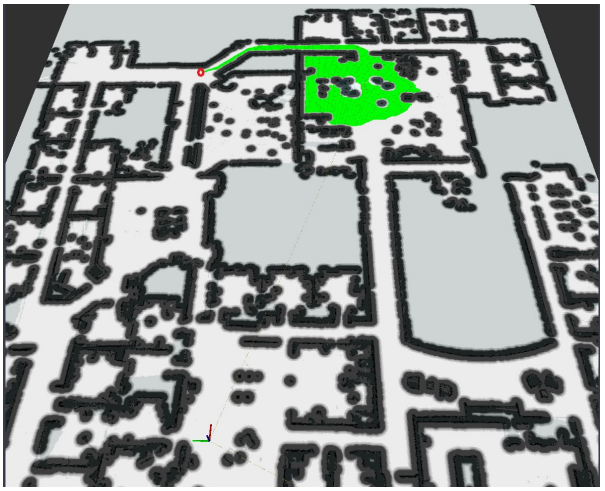


```
*****************************
Time taken: 4.64s

Longitud del camino: 184
Nodos en abiertos: 169
Nodos en abiertosF: 142
Nodos en cerrados: 1111
Nodos en cerradosF: 1115
*************************
```

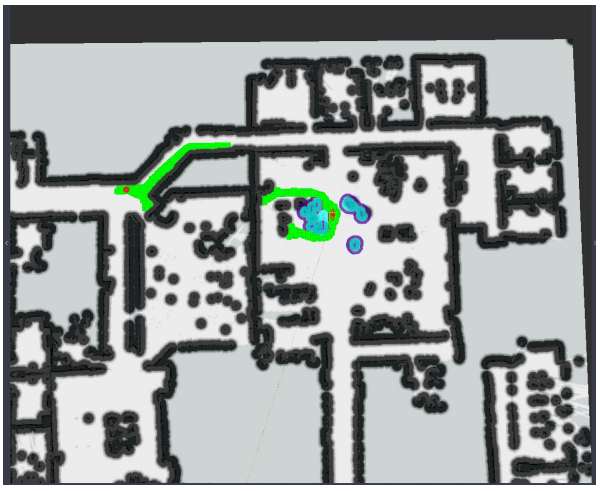
Caso 2:

- Sin mejoras:



```
**************************
Time taken: 123.96s
Longitud del camino: 455
Nodos en abiertos: 2765
Nodos en cerrados: 23158
**************************
```

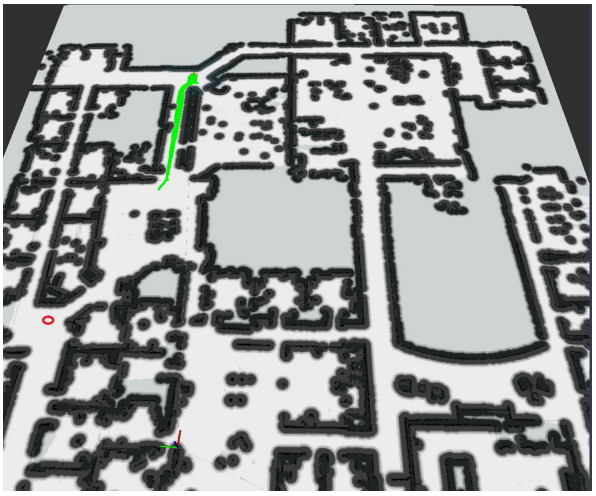
- Con mejoras:

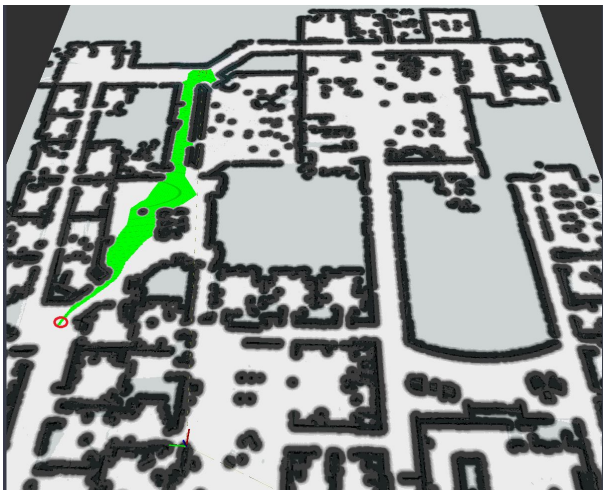


```
**************************
Time taken: 6.30s

Longitud del camino: 247
Nodos en abiertos: 161
Nodos en abiertosF: 146
Nodos en cerrados: 1387
Nodos en cerradosF: 1396
**************************
```

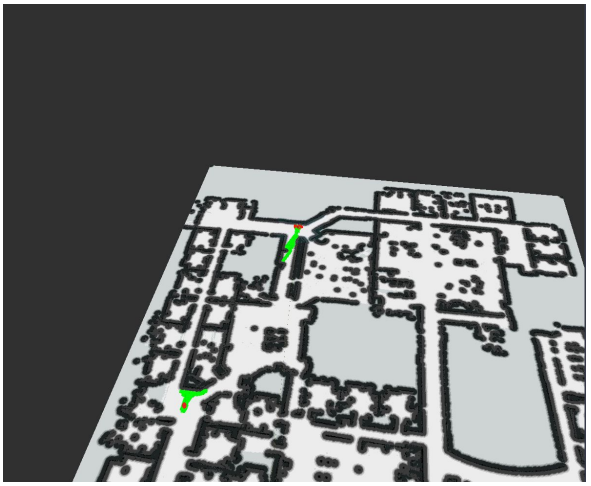
- Sin mejoras:

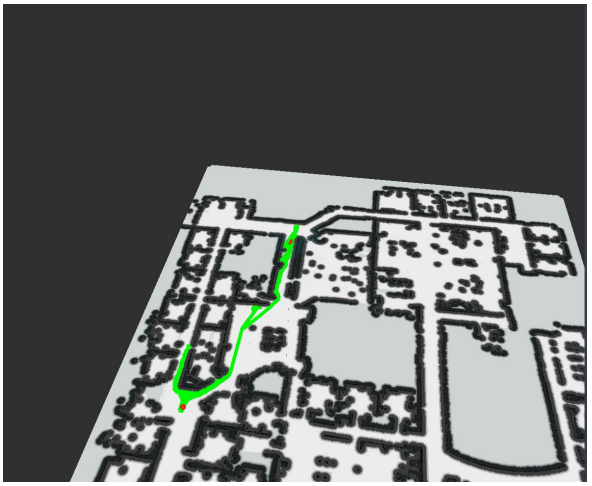




```
**************************
Time taken: 38.72s
Longitud del camino: 525
Nodos en abiertos: 1888
Nodos en cerrados: 12337
**************************
```

- Con mejoras:





```
**************************
Time taken: 4.41s

Longitud del camino: 380
Nodos en abiertos: 108
Nodos en abiertosF: 152
Nodos en cerrados: 1069
Nodos en cerradosF: 1068
**************************
```

6.- Conclusiones

Podemos apreciar la gran mejora tanto en nodos como en velocidad.
Para el caso de los mejorados tenemos que sumar ambas listas tanto de abiertos como cerrados para tener el equivalente al modelo con solo dos una para abiertos y otra para cerrados.

Podemos observar como una de las mejoras principales la provoca la mejora beam search la cual provoca que los nodos que mantenemos en abiertos siempre sean menores que el umbral que he establecido (anteriormente descrito).

Luego tenemos que la otra gran mejora es sin duda la búsqueda bidireccional la cual provoca que la velocidad de búsqueda de camino sea mucho más rápida. La mejora en cuanto a nodos expandidos también es muy sustancial ya que en cuanto ambas búsquedas se unen se consigue un camino con muchos menos nodos evaluados (cerrados).

Por último, tenemos que la mejora de búsqueda de tres en tres por si sola mejoraría la implementación en un orden de tres veces menos aproximadamente lo cual junto a las mejoras anteriores se consigue que la mejora del algoritmo convencional A estrella sea muy elevada.

En cuanto a las ponderaciones de g y h tenemos que mejoran la implementación en casos muy específicos en los que sin esta mejora se tardaría en encontrar el camino bastante más, a costa de empeorar un poco en orden de milésimas la búsqueda de caminos en otros casos.