Técnicas de los sistemas inteligentes
Práctica 1
A* y mejoras

Guillermo Gómez Trenado | 77820354-S

14 de abril de 2018

Índice

1. Descripción de la práctica

Para el desarrollo de esta práctica tomamos el código de ejemplo, donde viene implementada una búsqueda en anchura y modificamos la selección en la lista de abiertos tomando aquel nodo con mejor $f(n) = g(n) + h(n)$ siendo $g(n)$ la distancia actual desde el origen hasta el nodo y $h(n)$ una heurística de distancia hasta el destino.

Debido a las características del problema, donde creamos un camino que mandamos a *move_base* sin preocuparnos de la orienzación inicial o final del problema, y donde el planificador local interpretará este camino para ir realizando sobre él los ajustes que entienda apropiados, se presenta inútil preocuparnos por obtener rutas óptimas; más aún cuando lo que emulamos es un robot que necesita dar respuestas rápidas a entornos que podrían ser cambiantes y donde el tiempo de cálculo debería ser el mínimo posible.

Por lo anterior, tanto el aumento de la seguridad en el camino, y la posterior implementación de HPA* descartan la búsqueda de caminos óptimos para centrarse en caminos seguros y rápidos.

2. Implementación de A*

Dado que la búsqueda en anchura, la búsqueda de Dijkstra, la búsqueda primero el mejor y el A* se pueden abstraer al mismo problema con $f(n)$ distinta la implementación de A* ha sido sencilla. Para ello sólo he modificado el vector de nodos que funcionaba a modo de cola —FIFO— y lo he sustituido por un multiset de la librería estándar –para permitir nodos con igual $f(n)$—, sobrecargando el operador <para el struct *coupleOfCells*.

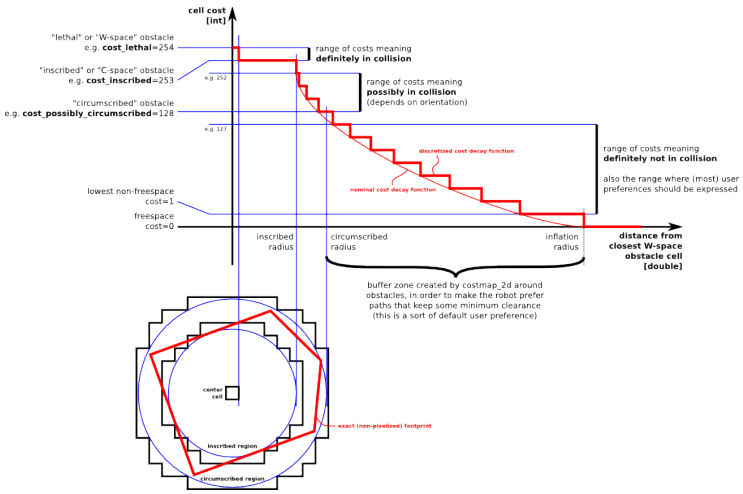
```
multiset<coupleOfCells> openSet;
```

```
bool operator< (coupleOfCells const & a, coupleOfCells const &
    ↪ b){
  return a.fCost < b.fCost;
}
```

A fin de conseguir un mejor rendimiento sustituimos cerrados por un map donde la clave es el valor de la celda, comprobando así en $O(log(n))$ si el nodo se encuentra ya en la lista de cerrados, y para la comprobación de la lista de abiertos además del multiset hemos creado un set que contiene los índices. Por lo demás la implementación es la del A* básico que ya vimos en asignaturas anteriores y no merece la pena describir aquí.

3. Aumentando la seguridad del camino

Para aumentar la seguridad del camino nos apoyamos en las funciones de *costmap*, donde através de la función de *getCost* podemos obtener el coste asociado a posición del mapa.



Como se puede observar en la imagen anterior un valor para una celda menor de 128 implica que el robot cabe en esa casilla —para el *footprint* definido en el lanzamiento— en cualquiera que se el ángulo de acceso. Esto debería ser suficiente para que el robot no chocara en ningún camino donde las celdas por las que transcurre tienen un valor menor a 128, sin embargo, debido a la capa semideliberativa que incluye el planificador local encargado de recibir nuestra ruta, donde se toman decisiones respecto al mejor camino a seguir tomando como guía la ruta recibida por nuestro planificador, es mejor ser de ayuda al planificador y darle rutas que intenten alejarse en la medida de lo posible de zonas de riesgo.

Para esto hemos modificado la función $h(n)$ de la siguiente manera:

$$h(n) = \begin{cases} dist(n, goal) * (1 + cost(n))/127), & \text{si } cost(n) < 128 \\ dist(n, goal) * (2 + e^{costMapVal/4-35}), & \text{en otro caso} \end{cases}$$

Permitiendo en la función de generación de hermanos nodos con peor rendimiento —hasta 252— pero penalizándolo con severidad, así, si no hay alternativa válida intentará caminos arriesgados. Para la implementación de HPA* hemos eliminado esta opción y sólo consideramos celdas con coste < 128, y aplicando consecuentemente la primera parte de la anterior $h(n)$ en exclusiva.

3.1. Consideraciones sobre el mapa

Vale la destacar que el mapa que hemos escogido para hacer este ejercicio igual no es el idoneo, ya que el gradiente de grises entre 0 y 255 representa la certeza que se tiene sobre si hay un espacio libre o un obstáculo, y es la representación idonea cuando tenemos sensores que reformulen el conocimiento del mapa cuando nos acercamos a zonas con incertidumbre, sin embargo, como hemos extraido el mapa en una etapa y hayado rutas en otra no tenemos un conocimiento fiable sobre el terreno y puede llevar a disonancias entre lo que interpreta costmap como obstaculos y lo que efectivamente interpreta ros. Así se

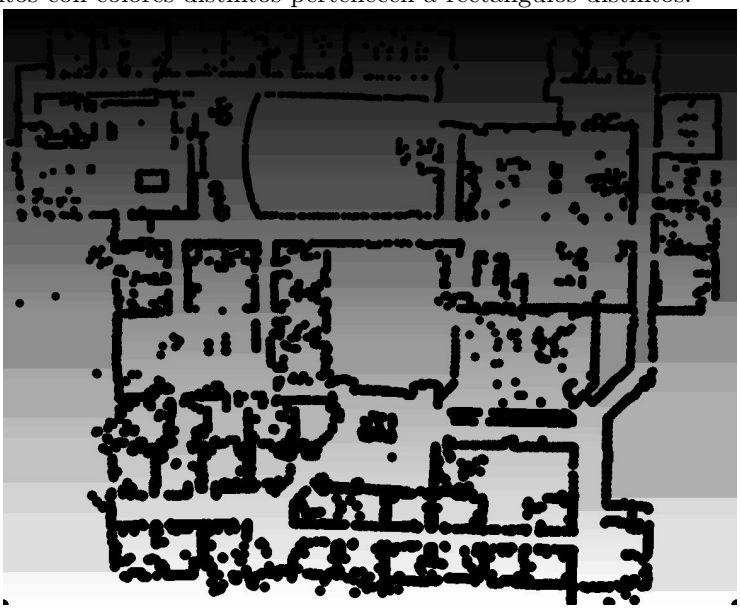
puede dar el caso de que el robot intente calcular una ruta a través de zonas no accesibles porque están definidas como inciertas o que entienda un terreno como transitable pero se produzca un choque al atravesarlo, como sucede a veces con la zona punteada del mapa.

4. Mejora A* (HPA*)

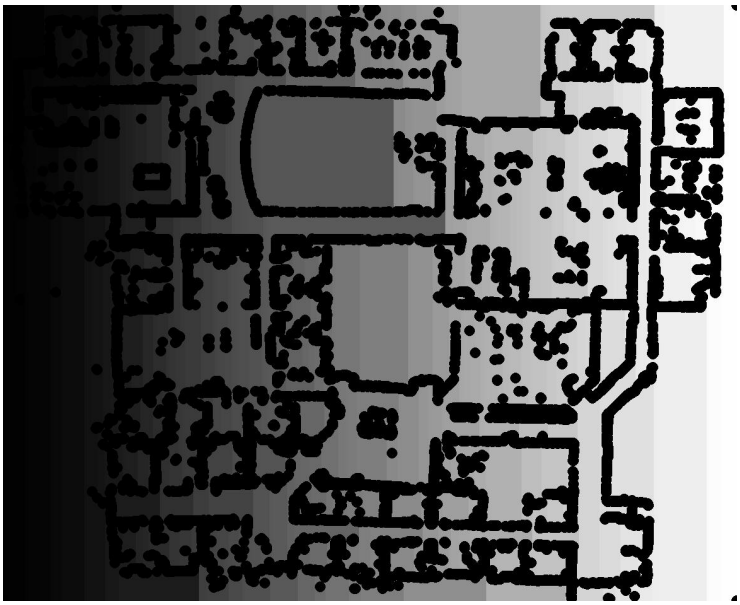
Como mejora he adaptado con algunas modificaciones propias para mejorar el funcionamiento en nuestro entorno el *Near Optimal Hierarchical Path-Finding* o HPA* a nuestro robot, similar a Theta* pero sin la implementación del *any-angle path planning* por cuestión de tiempo. Esta solución precomputa el mapa, buscando las esquinas a fin de crear un representación jerárquica de esta división de grano grueso del mapa —la forma de subdividir el mapa está diseñada por mí a modo de boceto y no pretende ser ni eficiente ni óptima—.

4.1. Creación de la jeraquía

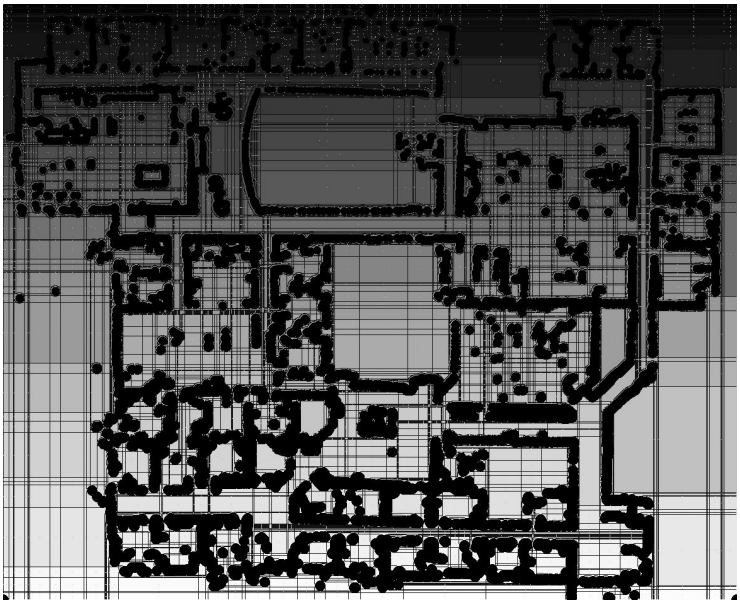
Para crear la jerarquía intento en primer lugar buscar rectángulos donde se premie la movilidad horizontal, como se puede observar en la siguiente imagen. Los puntos con colores distintos pertenecen a rectángulos distintos.



Y hacemos lo mismo para la movilidad vertical.



Finalmente realizo la intersección de los dos mapas y adhiero aquellos rectángulos demasiado pequeños para ser interesantes a su mayor vecino. En la siguiente imagen se observan los rectángulos delimitados por una línea negra y en blanco el punto medio.



Se aprecia perfectamente donde en las zonas más cómodas los rectángulos son grandes y las zonas muy accidentadas el grano es más fino. Durante el proceso hemos analizado también los rectángulos contiguos a otros para crear un grafo que sea la unión de todos los árboles para cada nodo.

Cabe aclarar que los rectángulos no son tal y aquellos próximos a irregularidades absorben a los minibloques próximos a éstas, además hay cierta tolerancia, a la hora de crearlos a que sus filas y sus columnas no tengan exactamente el mismo origen y fin sino uno aproximado, tolerando en este caso diferencias menores a 4. Por otro lado, cada bloque es vecino de todos los bloques que toquen con al menos una celda, incluyendo el movimiento diagonal.

Una vez creada la jerarquía se almacena de forma permanente durante toda la ejecución. Sería posible almacenarla en un archivo una vez creada mientras que las opciones de configuración se mantengan, pero a fin de observar el rendimiento de esta etapa se crea en cada inicio del programa.

4.2. Búsqueda de caminos

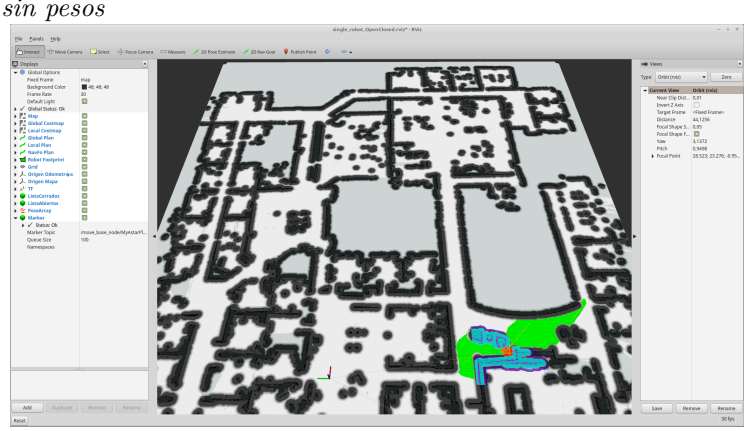
La búsqueda de caminos se realiza en dos etapas, en primer lugar, el camino óptimo dentro de la jerarquía mediante A* sobre la jerarquía, utilizando como distancia entre dos nodos la distancia entre sus representantes —que es el punto medio del rectángulo— modulada por la peligrosidad media del bloque y en segundo lugar la búsqueda A* entre dos bloques contiguo del plan trazado sobre la jerarquía. Es decir, desde el origen, a través de A* multiobjetivo —donde la h corresponde a la mínima distancia entre esa celda y cualquiera de las celdas del bloque objetivo modulada por el coste de la celda; una vez llegado a alguna de éstas el bloque objetivo pasa a ser el siguiente. Aunque sería necesario actualizar la h de los nodos en abiertos cada vez que se cambia el bloque objetivo, por acelerar el cómputo los dejamos así en abiertos aún a riesgo de que se intenten analizar cuando ya no son relevantes o cuando l h debería haber cambiado; en cualquier caso no imposibilita la búsqueda del camino pero sí la ralentiza cuando entre dos bloques hay algún imprevisto que salvar. Queda pendiente esta mejora para próximas implementaciones, aunque para la prueba de concepto es tolerable la ausencia.

La ventaja que tiene mi método es que en el caso de no encontrar un camino por el procedimiento jerárquico la lista de bloques intermedios es 0 y aplica entonces el A* por defecto desde el origen al destinto, que en este caso es el A* con penalización por pesos pero idealmente sería el *any-angle path planning* en alguna de sus variantes.

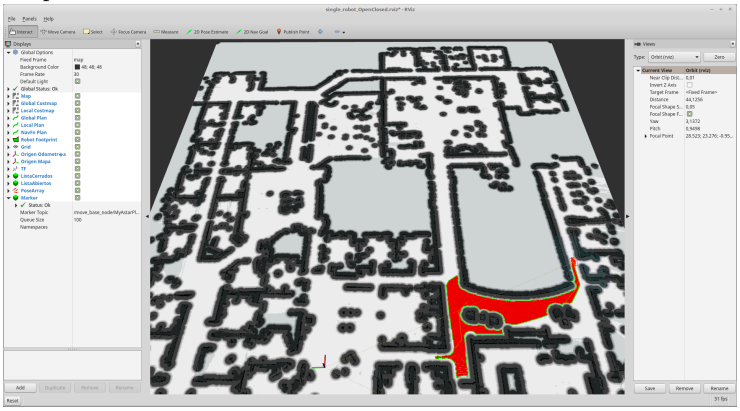
5. Experimento

Las tres versiones sobre las que experimentaremos son, el A* sin modificaciones, el A* con la penalización por el coste y mi versión de HPA*. Para poder compararlos elegimos caminos relativamente fáciles pues si no, las dos versines más simples no puden calcularlo y superan el límite de 10000 nodos explorados. Por quedarnos dentro del máximo de paǵinas sólo ilustramos el experimento 2, siendo el primero llegar al origen del segundo y el tercero volver al origen del anterior. Sin embargo las imágenes son suficientemente explicativas para comprender los pros de cada mejora.

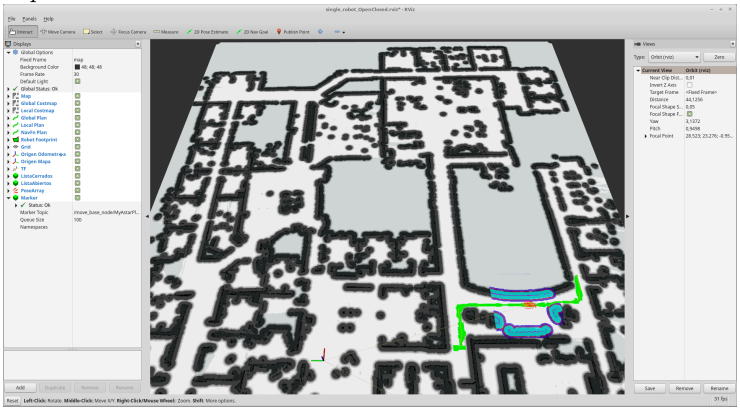
A sin pesos*



A con pesos*



HPA personalizado*



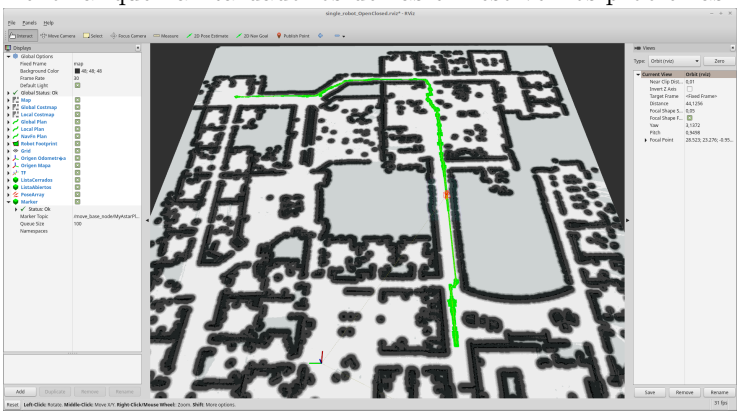
Cuadro 1: Tabla de pesos

		A* sin pesos	A* pesos	HPA*
Ruta 1	**Abiertos**	6898	5208	1740
	Exporados	6493	4438	1120
	Nodos camino	218	219	248
	Tiempo	14.41601	8.000061	1.648703
Ruta 2	**Abiertos**	7596	11322	2434
	Exporados	7327	9944	1559
	Nodos camino	243	337	338
	Tiempo	18.180438	31.8325	2.211028
Ruta 3	**Abiertos**	11609	9193	2309
	Exporados	11286	7857	1475
	Nodos camino	311	325	321
	Tiempo	39.20305	20.897144	2.013914

Como se esperaba, A* sin pesos obtiene por regla general mejores caminos que el resto, pero a riesgo de quedarse atascado como de hecho ha sucedido dos veces durante el experimento, aunque la capa reactiva ha podido sacarlo al final, tardando al final en recorrer el camino más que los otros dos. Pero por el contrario explora más nodos que A* con pesos, ya que no aprovecha el efecto pared —ir cerca a la pared para reducir el camino sin explorar el lado contiguo— que A* con pesos sí aprovecha.

Por descontado, HPA* consigue un resultado considerablemente mejor aunque no podemos apreciar su verdadero valor ya que no podemos hacerlos competir en caminos que supongan un desafío real.

Probemos ahora con un problema que los dos primeros no pueden resolver en un tiempo razonable. Este mapa se ha recorrido en 14 segundos, abriendo **8991** nodos, explorando **5977** de ellos y obteniendo un camino de **1219** nodos. Un tiempo menor al que han tardado los demás en resolver los problemas fáciles.



6. Reflexiones y mejoras posibles

La estrategia de la búsqueda jerárquica funciona de maravilla como hemos comprobado y tiene la ventaja de ser compatible con el resto de variaciones de A* pues utiliza a éstas en un capa inferior. Además aprovecha los movimientos óptimos que son los verticales, horizontales y diagonales. Por esto mismo, habría sido interesante implementar el *any-angle path planning* que seguro que en combinación éste obtendría unos resultados espectaculares, lanzándose de un bloque a otro en un tiempo mínimo. Lamentablemente el objetivo que me he propuesto ha sido demasiado ambicioso y me ha penalizado en otras tareas complementarias como ésta mencionada, que habría sido relativamente fácil de implementar y que por cuestión de tiempo me he quedado sin el gusto de implementar y analizar.

7. Bibliografía

1. A.Botea, M. Müller, J. Schaeffer. Near Optimal Hierarchical Path-Finding, `https://webdocs.cs.ualberta.ca/~mmueller/ps/hpastar.pdf`, Accedido el 14 de Abril del 2018.

2. K. Daniel, A. Nash, S. Koenig and A. Felner. Theta*: Any-Angle Path Planning on Grids. Journal of Artificial Intelligence Research, 39, 533-579, 2010. `http://idm-lab.org/bib/abstracts/papers/jair10b.pdf`. Accedido el 14 de Abril de 2018.