UNIVERSIDAD DE GRANADA

PRÁCTICA 1:

Planificación de caminos en Robótica

Realizada por:

Juan Emilio García Martínez

Granada, 11 de Abril de 2018.

ÍNDICE

1. RESUMEN

El primer paso que se realiza es modificar el código proporcionado, la implementación búsqueda en anchura simple, para implementar el algoritmo A* básico. El objetivo: encontrar el camino más **corto** posible de manera eficiente. Dicha implementación se realiza basándose en el código proporcionado. Una vez implementado se añade la técnica para que el camino encontrado sea seguro.

Una vez tenemos A* con la técnica de rutas seguras, se han hecho algunas modificaciones en el algoritmo: cambios en las estructuras de datos proporcionadas, se ha incluido la mejora "Dynamic weighting"[1] y se han añadido otras heurísticas para una posterior experimentación.

La experimentación se ha realizado en el mapa "willow_garage".

2. TAREAS

2.1. TAREA 1: EXTENSIÓN Y SEGURIDAD

El objetivo de la tarea 1 es extender la implementación de búsqueda en anchura para obtener el camino más corto y más seguro desde la posición actual hasta un objetivo dado a través de Rviz. Que un camino sea seguro quiere decir que garantiza en todo paso que la distancia del robot a los obstáculos es la más amplia posible.

El **extender** la implementación de búsqueda en anchura para obtener una implementación básica de A* se ha realizado introduciendo los siguientes cambios en el código proporcionado:

- se elimina *el mejor* nodo de ABIERTOS (currentNode)
- se introduce en CERRADOS
- una vez obtenidos los vecinos de currentNode, se ignoran los que están en CERRADOS y se le establece a todos como padre currentNode
    - Si no están en ABIERTOS, se añaden, calculando su correspondiente g (coste desde meta a dicho vecino pasando por currentNode), h (coste estimado hasta meta) y f (g+h).
    - Si están ya en abiertos
        - Si el coste g del vecino en análisis (pasando por currentNode) es menor que el coste g del nodo ya en abiertos, se actualiza dicho nodo de abiertos con la información del vecino.

[1] Técnica escogida de *http://theory.stanford.edu* para mejorar el rendimiento de A*, haciendo que se expandan menos nodos.

Una vez realizada la implementación de un A* normal (análisis posterior de heurísticas y estructuras de datos), añadimos la técnica de encontrar **los caminos seguros**.

Como vemos en el código proporcionado, se usa como representación las celdas de un global costmap de ROS, por tanto, extraemos de aquí la información para implementar dicha técnica.

Con la ayuda de la función *footprintCost(double x, double y, double tehta)* (proporcionada) tenemos en cuenta la huella del robot para calcular el coste correspondiente de que el robot invada una casilla en la que hay un obstáculo (o haya alguno muy cercano).

En esta función se llama al método *base_local_planner::CostmapModel footprintCost(…)* que hace esto automáticamente. Para poder utilizar dicho método se ha tenido que crear una clase propia que hereda de la clase base_local_planner::CostmapModel, la cual se inicializa con un Costmap2D, y modificar el CmakeList que incorpore la carga de las librerías necesarias para cargar dicha clase.

Una vez se pudo utilizar dicho método, primero calculamos las coordenadas del robot (pentágono) y, utilizando dichas coordenadas, se calcula el coste. En la generación de vecinos, se comprueba que el coste obtenido sea menor que un valor representativo de un obstáculo (entre 230 y 254), es decir, no se generarán aquellas casillas en las que cuando pase el robot, incida con obstáculos o haya alguno muy cercano.

2.2. TAREA 2: MEJORAS, ESTRUCTURAS DE DATOS Y HEURÍSTICAS

El objetivo de la tarea 2 es tratar de reducir el tiempo en que tarda en encontrar una solución ya que la implementación está integrada en un arquitectura para la navegación de un robot en tiempo real.

Mejora

La principal **mejora** que realizamos es introducir la variante de A* “Dynamic weighting”: dicha mejora consiste en asumir que al principio de la búsqueda es más importante buscar rápido cualquier ruta y que, a medida que se va profundizando en la búsqueda, es más importante llegar al objetivo.

Dicha mejora se realiza introduciendo un peso w tal que w>=1 asociado a la heurística. Dicho peso al principio será grande (le damos más importancia a la heurísitca) y a medida que nos vamos acercando al objetivo, vamos decrementando el peso, es decir, le vamos dando cada vez más importancia al costo real de la ruta.

Para realizar este cambio en nuestra implementación, nos basamos en la siguiente definición:

$$f(n) = g(n) + (1 + \varepsilon w(n))h(n), \text{ where } w(n) = \begin{cases} 1 - \frac{d(n)}{N} & d(n) \leq N \\ 0 & \text{otherwise} \end{cases},$$

Donde d(n) es la estimación de ir desde el vecino en análisis al objetivo (hCost) y N es la estimación del camino desde la celda de inicio al objetivo. Garantizamos que la solución obtenida no es peor que veces la solución óptima que $1 + \varepsilon$. Por eso se llama que el algoritmo es ε - admisible. En la tarea 3 se experimentará con esta mejora.

Estructuras de datos

En esta parte se especifican los cambios respecto a las estructuras de datos que se nos proporcionan y aprovechando los cambios en las estructuras, la **remodelación de la mayor parte del código proporcionado**.

Principalmente se ha utilizado una cola de prioridad propia para la gestión de ABIERTOS, la cual incorpora como nodos (elementos de la cola) un *std::pair<double, int>*, es decir, las celdas quedan definidas con ID(int) y un coste f(double). Por tanto, **prescindimos** de la estructura de datos CoupleOfCells proporcionada. Veremos que esto nos ahorra bastante código para al final hacer lo mismo. Esta cola de prioridad tiene las operaciones básicas necesarias (vacia, inserción, borrado y clear) de manera eficiente ya que están implementadas con un Binary Heap, ordenación de cola al insertar en O(logn).

El manejo de nodos en CERRADOS se ha hecho con **dos** *std::unordered_map,* uno que almacena los padres de los nodos y otro que almacena el coste g de los nodos, esto reduce considerablemente la complejidad de varias tareas necesarias en A*, comprobación de nodos YA en abiertos y su modificación si es necesario(O(1)) y la creación del plan obteniendo los padres de cada nodo, O(1), aunque también es cierto que se utiliza algo más de memoria (de ahí la utilización de enteros y un par de dos elementos para representar la información de cada nodo, para reducir uso de memoria) y que dicha complejidad puede aumentar dependiendo de la función hash.

Como podemos ver en el código fuente, se prescinde de muchas funciones proporcionadas, quedándonos solo con las necesarias (calculateHCost, footprintCost, findFreeNeighborCell y getMoveCost) para el funcionamiento de A*. Esto hace que tengamos un total de 693 lineas que incluyen los comentarios esenciales, include y cpp frente a las 750 SIN NADA IMPLEMENTADO que se nos proporcionan.

Considerable reducción de código fuente para tener más claridad en su comprensión a parte de la rapidez de las estructuras de datos utilizadas.

Se ha de mencionar en esta sección, ya que tiene que ver con estructuras de datos, que previo a modificar las estructuras de datos proporcionadas, se implementó A* utilizando también estas (listas simples), pero se hizo para que funcionase de manera “rápida” aunque no óptima.

Se utilizó una cola de prioridad simulada con las listas proporcionadas. El porqué de la simulación de la cola de prioridad es porque dicha estructura de datos no permite la modificación directa de un nodo (actualización si el nodo estaba ya en abiertos).

En cada iteración se ordena dicha lista para siempre tener el elemento de menor f en el tope de la cola y el elemento de menor f al final. Las operaciones usadas son pop y push (como en una cola) y las listas nos permiten comparar mediante iteradores elementos de la lista para su modificación si fuera necesario.

Si nos damos cuenta, la cola es LIFO, es decir se comporta como búsqueda en profundidad en rutas de igual costo (**evitando explorar más de una solución igualmente óptima**) y esto nos ahorra muchas expansiones de nodos debiendo de pagar el coste de que la ruta no será la óptima siempre.

Por este anterior párrafo indico también el uso de estas estructuras de esta manera, porque veremos en la experimentación el comportamiento peculiar que tiene sin hacer mejora alguna (inclusión de pesos, por ejemplo).

Heurísticas

Se han utilizado 4 heurísticas distintas para estimar el coste hacia el objetivo. La primera de ella es la que se proporciona, *distancia euclídea:*

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

- La segunda utilizada es *la distancia euclídea al cuadrado (sobreestimación).*

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx * dx + dy * dy)
```

- La tercera es la *distancia octil.*

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
```

  Donde D=1 y D2=$\sqrt{2}$

- La cuarta es *la distancia Chebyshev*

  Igual que la distancia octil pero D=1 y D2=1.

2.3. TAREA 3: EXPERIMENTACIÓN

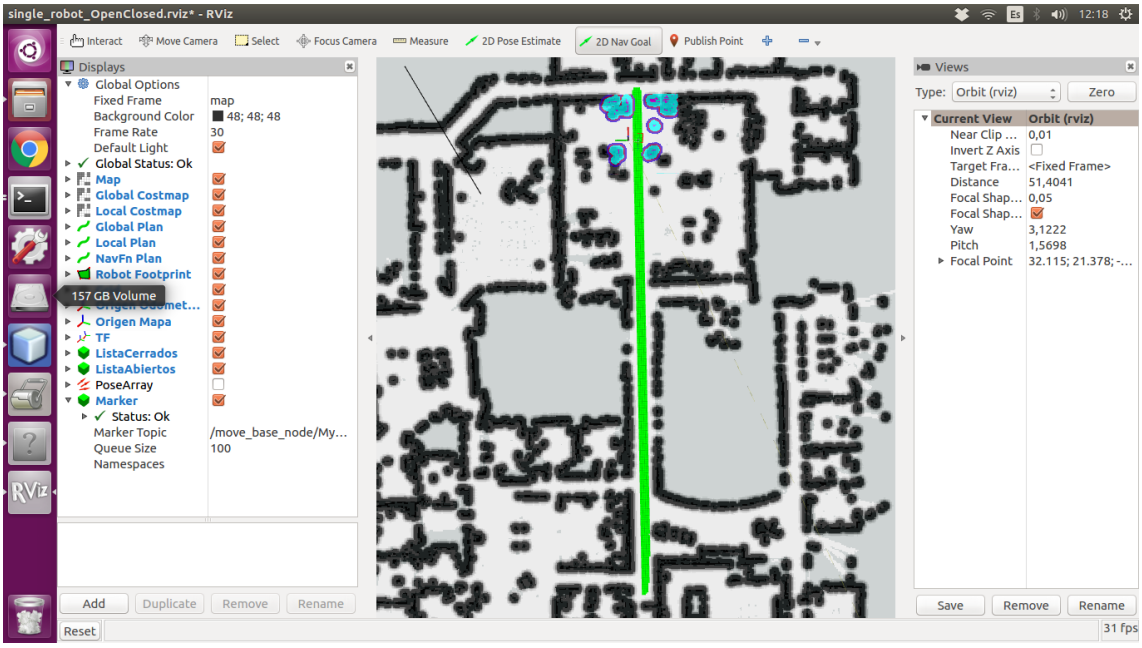
Para la experimentación, que mejor manera de hacerlo visualmente. Se van a proponer **tres** episodios de navegación distintos en los cuales vamos a medir el comportamiento de A* introduciendo la mejora, cambiando heurísticas y cambiando estructuras de datos.

Se medirá el tiempo de ejecución del plan, nodos expandidos y longitud del camino (en metros y en nodos).

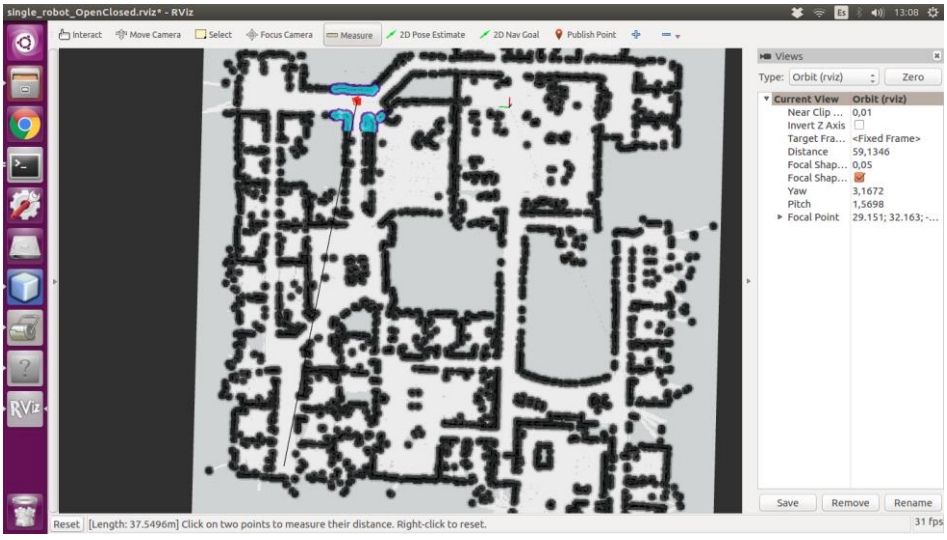
Episodios de navegación

Presentamos los episodios, sin medir todavía nada (posteriormente en tabla).

- EPISODIO 1: contemplamos una línea recta sin obstáculo alguno (la mayor del mapa *willow*) de 40 metros:



- EPISODIO 2: contemplamos una distancia larga en línea recta (40m) pero esta vez con obstáculos en medio.



- EPISODIO 3: contemplamos ahora una distancia pequeña en línea recta (15m), pero en otra habitación distinta, es decir, con muchos obstáculos en medio (principalmente la esquina).

Basándonos en estos episodios, construiremos nuestras tres tablas de mediciones: la tabla 1 para las mediciones sin la mejora, la tabla 2 con dicha mejora, dichas tablas utilizan las estructuras de datos que se han incluido. Construiremos otra tabla extra (tabla 3) para mostrar el funcionamiento usando las estructuras de listas mencionadas anteriormente (ya proporcionadas con modificaciones).

TABLA 1 (SIN MEJORA)				
	OBJETIVO	N. EXPLORADOS	TIEMPO	TAM PLAN
H0(Euc)	40(sin obs)	3725	0.1	380
	40(con obs)	22563	24.3	364
	15(con obs)	13975	9.7	185
H1(Euc SQ)	40(sin obs)	SE COMPORTA FATAL EN ESTE TIPO DE MAPAS		
	40(con obs)			
	15(con obs)			
H2(octile)	40(sin obs)	1503	0.2	380
	40(con obs)	2879	0.6	368
	15(con obs)	3273	0.8	371
H3(cheb)	40(sin obs)	4549	1.6	382
	40(con obs)	30000(TOPE)	45.5	
	15(con obs)	16171	13	186

Analizando dicha tabla vemos como la H0 se comporta normal, H1 se comporta de manera desastrosa, imposible de obtener una ruta, lo que quiere decir, que no funciona para este tipo de mapas, H2 es la que mejor se comporta en estos casos (se podría seguir su estudio más a fondo para conseguir una heurística rápida y óptima en todas las situaciones del mapa) y H3 se

comporta de mala manera también ya que está pensada para mapas donde no hay movimiento diagonal (se ve en la generación de nodos como intentar ir siempre en línea recta).

TABLA 2 (CON MEJORA)				
	OBJETIVO	N. EXPLORADOS	TIEMPO	TAM PLAN
H0(Euc)	40(sin obs)	785	0.1	383
	40(con obs)	1793	0.2	384
	15(con obs)	9729	4.8	207
H1(Euc SQ)	40(sin obs)	NO ES CAPAZ DE ENCONTRAR RUTA ALGUNA, SALTA DEMASIADO DE UNA CASILLA A OTRA		
	40(con obs)			
	15(con obs)			
H2(octile)	40(sin obs)	763	0.001	381
	40(con obs)	1940	0.3	455
	15(con obs)	15541	11.9	185
H3(cheb)	40(sin obs)	780	0.001	382
	40(con obs)	26515	35	374
	15(con obs)	16947	14.3	183

Podemos ver como para cada H la mejora afecta (en algunos casos más significativamente que otros). Se ha de decir, que según el ajuste de pesos que se haga en el valor E, hay una variación en el comportamiento de A* ya que, para algunos valores de este E (sobretodo grandes) la gestión de los empates con nuestras estructuras de datos se hace de una manera o de otra dependiendo de dicho parámetro, es decir, el algoritmo a veces expande muchos más nodos porque escoge mal los nodos con empate, esto se arregla reajustando los pesos.

Vamos a ver la última tabla en la que precisamente, por cómo se trata la gestión de dichos nodos de ABIERTOS con empate, cambia el comportamiento del A*. Esto viene dado por la utilización las estructuras de datos proporcionadas (listas) y su gestión.

TABLA 3 (listas para ABIERTOS Y CERRADOS usando struct CoupleOfCells)

TABLA 3 (ESTRUCTURAS PROPORCIONADAS)				
	OBJETIVO	N. EXPLORADOS	TIEMPO	TAM PLAN
H0(Euc)	40(sin obs)	767	0.1	383
	40(con obs)	841	0.2	403
	15(con obs)	5685	2.4	237
H1(Euc SQ)	40(sin obs)	795	0.1	382
	40(con obs)	1537	0.5	449
	15(con obs)	6131	3	214
H2(octile)	40(sin obs)	765	0.2	382
	40(con obs)	833	0.2	401
	15(con obs)	5631	2.5	201
H3(cheb)	40(sin obs)	767	0.2	383
	40(con obs)	1067	0.2	461
	15(con obs)	15793	19.3	445

Asombrosamente podemos ver como, por el tipo de mapa (en otros mapas podría ser un auténtico desastre) las rutas no son siempre las óptimas pero para los 3 episodios, para todas las heurísticas utilizadas, encuentra las rutas relativamente rápido, lo que quiere decir que las estructuras de datos utilizadas y su gestión en este caso han influido significativamente.

Las imágenes con el cálculo de rutas en Rviz están en una carpeta aparte.

Conclusión: para este tipo de mapa curiosamente se comporta mejor, con cualquier heurística, la implementación de A* usando listas simples con dicha gestión de los empates del fCost a la hora de extraer nodos de ABIERTOS, ya que se expanden muchos menos nodos, lo que hace que podamos no obtener la ruta óptima. Esta implementación también asegura que nunca se modificarán los nodos de ABIERTOS, ya que el que hay siempre es mejor. Sin embargo, para otros mapas la implementación “correcta” sería la primera, en concreto utilizando la heurística de la distancia octil sin mejora (se comporta de muy buena manera cuando hay obstáculos).