Entrega 2 – Parte 2

TSI

Planificación de caminos en Robótica

Francisco Javier Caracuel Beltrán
Curso 2016/2017
Grupo 3

1. Descripción:

La práctica 2 (parte 2) consiste en finalizar la implementación del algoritmo A* que se ha recibido en parte implementado.
Principalmente, lo que se ha desarrollado ha sido el cálculo de los costes correspondientes a cada nodo. Estos son:
- f(n) se calcula como la suma de g(n) y h(n).
- g(n) es la distancia entre el nodo actual y su padre, más g(n) del padre.
- h(n) es la heurística implementada. En este caso se ha optado por utilizar dos métodos diferentes, correspondientes a la tarea 1 y a la tarea 2, que se explican en los siguientes apartados de este documento.

De esta forma ya se tendría una implementación básica del algoritmo A*.

2. Tarea 1:

Para extender la actual implementación del algoritmo A* es necesario calcular f(n), g(n) y h(n) y con estos datos se puede hacer una buena gestión de los nodos que están en la lista de abiertos y cerrados.

Para calcular g(n) se suma g(n) del nodo padre con la distancia euclídea del nodo padre al nodo actual. Para calcularla se hace uso de la función ya implementada "getMoveCost(nodoPadre, nodoActual)", por lo que la instrucción correspondiente es:

nodoActual.g = nodoPadre.g + getMoveCost(nodoPadre.posicion, nodoActual.posicion)

En todas las prácticas ha habido algún tipo de problema por el que no se podía realizar correctamente y, ésta no ha sido menos. Para calcular la heurística (h(n)), la opción evidente es calcular la distancia euclídea entre el nodo actual y el nodo objetivo. Como bien decía el enunciado de la práctica, hay que tener en cuenta el espacio que ocupa el robot para calcular la ruta.
Para calcular el espacio se podría hacer uso de "world_model_" que dispone del método "footprintCost(x, y, t)", el cual dependiendo del espacio que tenga disponible en una posición, devuelve un valor u otro. Si no hubiera habido ningún problema en utilizar este recurso, se tendría terminada la heurística, pero pese a compilar correctamente, devolvía errores al cargar el "launch". El proceso para utilizar "footprintCost()" ha sido:

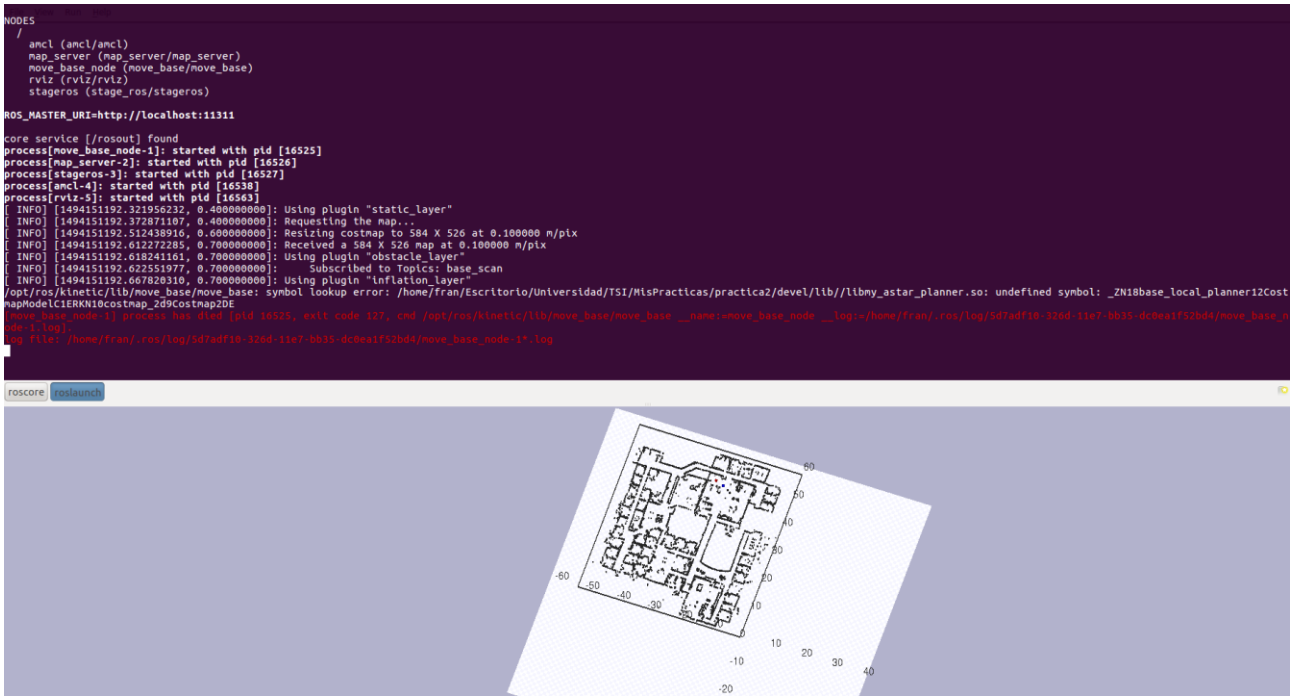
- "world_model_" como atributo de la clase (también se ha probado con "base_local_planner::CostmapModel *world_model_" siguiendo recomendaciones de enlaces de internet):

```
131
132         // Para publicar el plan
133         ros::Publisher plan_pub_;
134
135         base_local_planner::WorldModel* world_model_;
136
```

- Asignación de "costmap" a "world_model_" para poder hacer uso de "footprintCost()" (ya estaba implementado y solo era necesario descomentar el código):

```
82          private_nh.param("min_dist_from_robot", min_dist_from_robot_, 0.10);
83
84          world_model_ = new base_local_planner::CostmapModel(*costmap_);
85
86          // El plan se va a publicar en el topic "planTotal"
87          plan_pub_ = private_nh.advertise<nav_msgs::Path>("planTotal", 1);
88
```

- Error al ejecutar el launch como consecuencia de lo anterior:



Como alternativa a esta forma de calcular la heurística, "costmap" dispone de un método llamado "getCost()", que devuelve el costo de una celda en el mapa. Teniendo esto en cuenta, la heurística se compone finalmente de la distancia euclídea entre el nodo actual y el nodo objetivo, más el valor que devuelve "getCost()" en la celda actual, más el valor que devuelve "getCost()" en todas las celdas que se encuentran alrededor de la actual.

Esto permite penalizar más las celdas que tienen un grupo grande de costo en cada casilla. Pese a esto, la penalización es excesiva y no calcula rutas si pasan por huecos pequeños, pero en los que el robot sí entraría, por lo que se divide su valor entre 50 y se obtiene un buen rendimiento.

Una vez que se tiene calculada g(n) y h(n), solo hay que sumarlas para obtener f(n).

Cuando ya se tiene f(n), al inicio de cada iteración se ordenan todos los nodos que se encuentran en la lista de abiertos en base a f(n) (de menor a mayor), por lo que solo es necesario coger el primero de ellos y añadirlo a cerrados.

3. Tarea 2:

Para mejorar el funcionamiento del algoritmo A* se ha optado por el uso de pesos en h(n).

El coste g(n) se mantiene el mismo, por lo que la diferencia en f(n) depende exclusivamente de h(n).

Para modificar h(n) y ajustar mejor el camino se ha optado por incrementar considerablemente la distancia euclídea entre el nodo actual y el nodo objetivo (concretamente se aumenta 50 veces dicha distancia) y se ha suprimido la comprobación del costo de cada celda adyancente a la actual. Solo se suma el costo de la celda actual, pero con el doble de su valor.
Para terminar de ajustar la heurística, se hace uso de una adaptación del método "footprintCost()" que no funcionaba correctamente. Para solucionar (en parte) el problema, se actualiza en cada llamada a este método la variable "world_model_" en base al "costmap" actual.
Realizando esto, tampoco funcionaba el método como debería (según su implementación), por lo que se devuelven los valores sin comprobar el tamaño mínimo del "footprint" que se calcula en éste.

Estos pasos permiten penalizar mucho a las celdas que se alejan más del nodo objetivo, controlando la distancia mínima que debe tener en cuenta el robot para poder pasar por un punto.

Se ha conseguido una ruta que consume muchos menos nodos y necesito menos tiempo de cálculo.

4. Experimentación:

* En los experimentos con la mejora de los pesos no he conseguido que aparezcan coloreados los nodos en abiertos y los nodos en cerrados. Aparecen todos de color verde.

a) Recorrido básico:

a) Tarea 1:

Como se puede ver en la imagen, en cada iteración necesita explorar nodos del camino, lo que hace que explore todo el pasillo que se encuentra desde la salida al destino.

El total de tiempo que ha necesitado es: 7'3 segundos.
La cantidad de nodos expandidos es: 4.677.
La longitud en nodos es: 309.
La longitud en metros es 31'54 m.

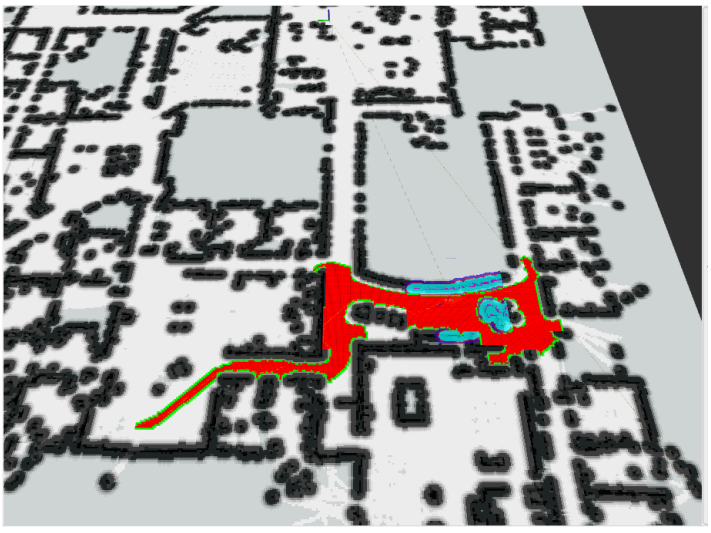
b) Tarea 2:

Ahora se tiene muy en cuenta la distancia euclídea por lo que se elegirá el nodo que esté más cerca del destino.

El total de tiempo que ha necesitado es: 0'29 segundos.
La cantidad de nodos expandidos es: 310.
La longitud en nodos es: 309.
La longitud en metros es 34'53 m.

b) Recorrido básico (se prueba un recorrido un poco más complejo):

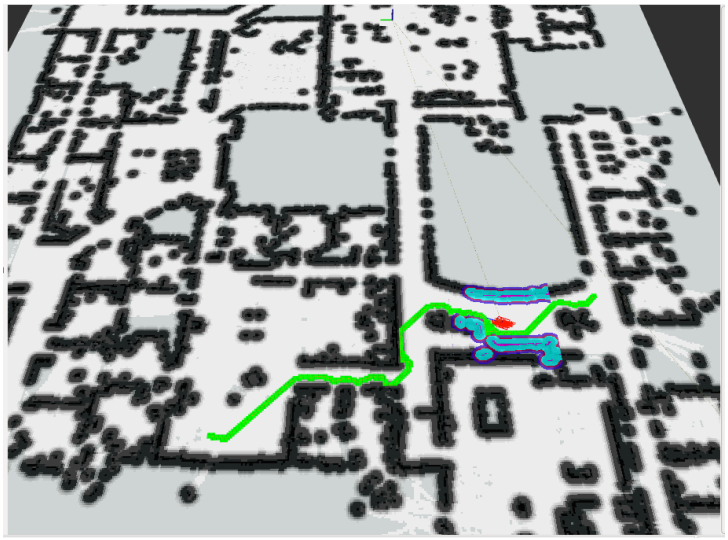
a) Tarea 1:



Cada vez que tiene un cambio de dirección comienza a explorar casi como la búsqueda en anchura, hasta llegar a un camino que de nuevo no ofrezca muchos obstáculos.

El total de tiempo que ha necesitado es: 7'98 segundos.
La cantidad de nodos expandidos es: 4232.
La longitud en nodos es: 238.
La longitud en metros es 27,88 m.

b) Tarea 2:

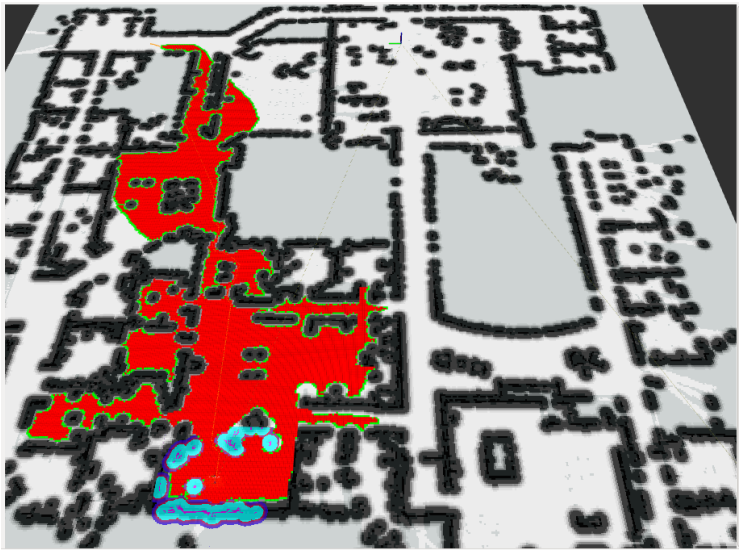


Se puede ver como ajusta la ruta más rápido sin perder tiempo en expandir zonas que no debe y como cuando detecta una celda en la que el robot no entra, bordea el obstáculo.
Aun así, se puede mejorar un poco el recorrido que realiza con los nodos de la lista de cerrados.

El total de tiempo que ha necesitado es: 0'11 segundos.
La cantidad de nodos expandidos es: 148.
La longitud en nodos es: 145.
La longitud en metros es 17,47 m.

c) Recorrido con muchos obstáculos:

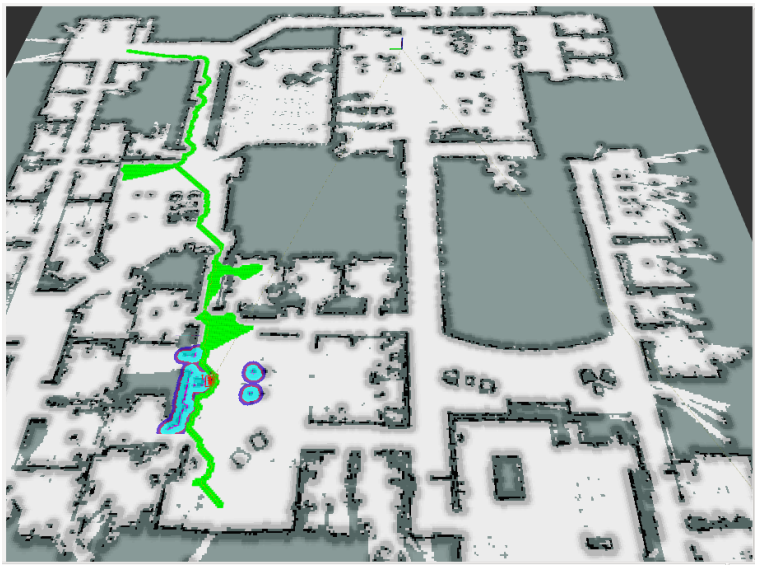
a) Tarea 1:



Como no penaliza mucho la distancia euclídea, en cuanto tiene una zona esquinada, en diagonal y con obstáculos, comienza a hacer una búsqueda más extensa.

El total de tiempo que ha necesitado es: 98'98 segundos.
La cantidad de nodos expandidos es: 15.387.
La longitud en nodos es: 397.
La longitud en metros es 44'32 m.

b) Tarea 2:

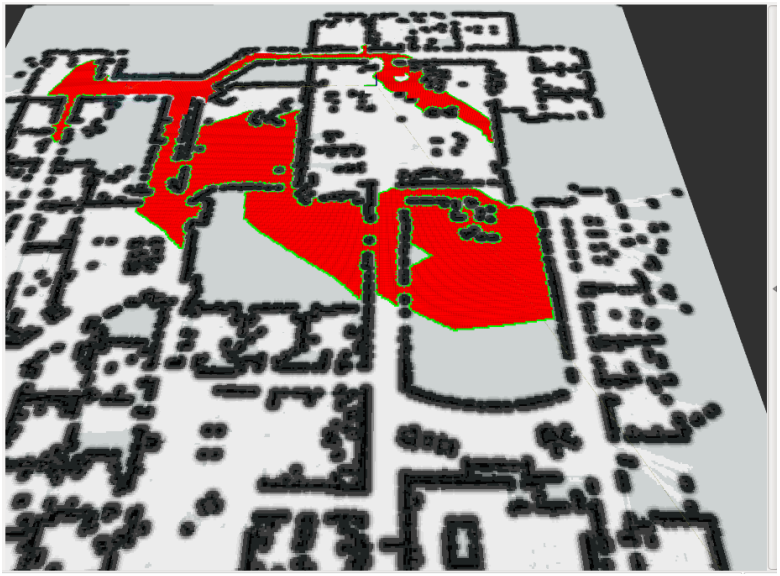


El uso de nodos sigue siendo mínimo y la velocidad muy rápida, aunque cuando tiene que buscar un recorrido más esquinado, hasta que no se alejan los nodos del objetivo y penalizan más que los nodos con obstáculos, comienza a concentrarse en una zona en concreto:

El total de tiempo que ha necesitado es: 0'44 segundos.
La cantidad de nodos expandidos es: 533.
La longitud en nodos es: 262.
La longitud en metros es 31'65 m.

d) Recorrido de lado a lado:

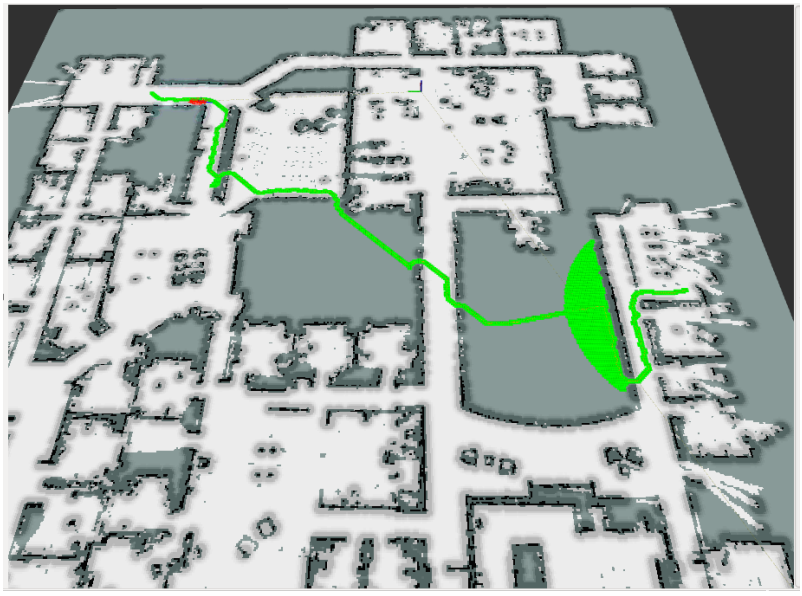
a) Tarea 1:



No termina de calcular la ruta de tantos nodos que tiene disponibles, por lo que antes se consumiría el tiempo que tiene máximo de cálculo.

Pese a que el recorrido sea más lejano, sigue haciéndolo rápido. Aquí entra en juego el mapa que piensa que tiene el robot. Se puede ver cómo cree que existe más hueco del que realmente hay, y planifica una ruta por lugares por los que no puede pasar.

b) Tarea 2:



Cuando llega a esos lugares que realmente no puede pasar, actualiza el mapa y recalcula la ruta, hasta que llegue al destino: