Curso 2016-2017
Grado en Ingeniería Informática
Universidad de Granada

Documentación práctica 2 TSI: Explorador basado en fronteras.

Carlos Manuel Sequí Sánchez

20 de abril de 2017

Índice

1. Ejercicios planteados en "EjerciciosMapeo1"

1.1. Giro de 360º

Con el fin de reconocer/explorar los alrededores del robot implementamos este método. Su funcionamiento es simple, hacemos, mediante el uso de parámetros o atributos de ros, que la base gire sobre sí misma para reconocer su entorno y rellenar el mapa sobre el que está.

Para obtener mejores resultados nos basta con realizar dos vueltas completas, ya que con una sola vuelta no recoge información suficiente el sensor.

La velocidad angular la he puesto a $2*\pi$.

Pseudocódigo:

```
1: while TiempoDeGiro do
2:     Girar sobre sí mismo
3: end while
```

1.2. Rellenar obstáculos

Método cuyo propósito es hacer que el robot no vaya hacia zonas donde hay obstáculos para evitar impactar contra alguno de estos.
Desde getmapCallBack llamamos a este método para cada una de las casillas que hay en la matriz que representa el mapa sobre el que se sitúa el robot (cmGlobal). Resumiendo un poco, el funcionamiento se basa en recorrer toda la matriz del mapa y, en cuanto encontramos una casilla obstáculo (con valor 100) hacemos que las casillas que se encuentran a su alrededor seas también obstáculos, concretamente 10 casillas hacia cada uno de los 4 lados de la casilla central (realizando cálculos son 10 casillas las que hay que modificar para cumplir ese metro cuadrado que se pide en el guión).

Pseudocódigo:

```
1: for cantidad de filas do
2:     if estamos dentro del límite de anchura then
3:         for cantidad de columnas do
4:             if estamos dentro del límite de altura. then
5:                 theGlobalCm[filaActual][columnaActual] = 100;
6:             end if
7:         end for
8:     end if
9: end for
```

1.3. Etiquetar celdas frontera

Este método se encarga de detectar los nodos frontera en el mapa, básicamente lo explora por completo (exploramos la matriz theGlobalCM) y, para cada una de las casillas de dicha matriz, en caso de que sea una casilla libre y además al menos una de sus casillas vecinas sea desconocida (cosa que sabemos gracias al método someNeighbour is unknown) pues la etiquetamos como casilla frontera, añadiéndola al vector "frontera"para que la base más tarde haga uso del método selectNode para seleccionar el nodo objetivo.

Pseudocódigo:

```
1: vaciarVectorFrontera();
2: for cantidad de filas do
3:     for cantidad de columnas do
4:         if casillaActual es vacía y además tiene vecino desconocido then
5:             calcularCoordenadasReales();
6:             insertarCoordenadasEnVectorFrontera();
7:         end if
8:     end for
9: end for
```

1.4. Seleccionar nodo objetivo

Para este método el criterio por el que he optado es ir escogiendo sucesivamente el nodo de la frontera que más lejano se encuentra a la posición actual de robot, con el fin de que explore la mayor parte del mapa en un tiempo reducido, ya que en cada una de las ïteraciones"(entendiendo como iteración cada una de las veces que llega a un objetivo) se aumenta el tamaño de la distancia a recorrer entre nodos objetivos, cosa que hace que la base recorra cada vez más y más espacio.
En cuanto a código simplemente he ido recorriendo cada uno de los nodos del vector frontera con el fin de calcular sus distancias e ir actualizando el nodo con mayor distancia desde la posición actual de la base.

Pseudocódigo:

```
1: Dist = distancia máxima final
2: for cantidad de nodos en frontera do
3:     dist2 = calcular distancia entre robot y el nodo actual
4:     if dist2 >Dist then
5:         actualizamos el nodo más lejano;
6:         Dist = dist2;
7:     end if
8: end for
```

2. Estrategia en caso de fallo

En caso de fallo por parte del robot a la hora de comprobar si ha podido o no llegar al objetivo ("The base failed for some reason") he utilizado la estrategia de hacer que la base rote 4 veces con el fin de detectar sus alrededores de forma óptima y tras ello escoger un punto intermedio distinto al más lejano, es decir, en caso de fallo, hago la llamada a mi método calculaIntermedio(), el cual calcula el punto intermedio (en cuanto a distancia se refiere) entre el robot y el punto más lejano (¡Ojo! no calcula el punto intermedio en ruta desde la posición actual de la base HACIA el punto de la frontera mas lejano, sino otro punto de otra ruta que esta a la mitad de distancia de lo que estaba el punto más lejano en un principio).
En cuanto a la explicación del método que realiza este cálculo es simple: en un contenedor "set"de la STL de C++ almaceno pares de valores distancia-índice, de modo que el primer valor me indica la distancia entre la posición actual y el punto de la frontera que indica el segundo valor (índice). Como este contenedor se encarga solo de ordenar en orden creciente según el valor del primer atributo de cada par que es introducido, solo he de hacer un recorrido por el vector "frontera"para ir extrayendo su información en forma de pares (pair<distancia,indice>) e ir introduciéndolos en el contenedor set. De tal manera, una vez ordenados, solo he de indicar que el nodo intermedio es el que se encuentra en la posición intermedia del contenedor que he rellenado previamente. Con esto pretendo conseguir que, en caso de que la base no pueda ir hasta el nodo más lejano de la frontera por cualquier razón, intente ir al nodo que tiene a media distancia, incrementando así las posibilidades de acierto.

Pseudocódigo:

```
1: for cantidad de elementos en la frontera do
2:     calcular distancia al robot;
3:     almacenar distancia en pair;
4:     almacenar índice en pair;
5: end for
6: nuevoNodoObjetivo = indice intermedio del set de pairs;
```

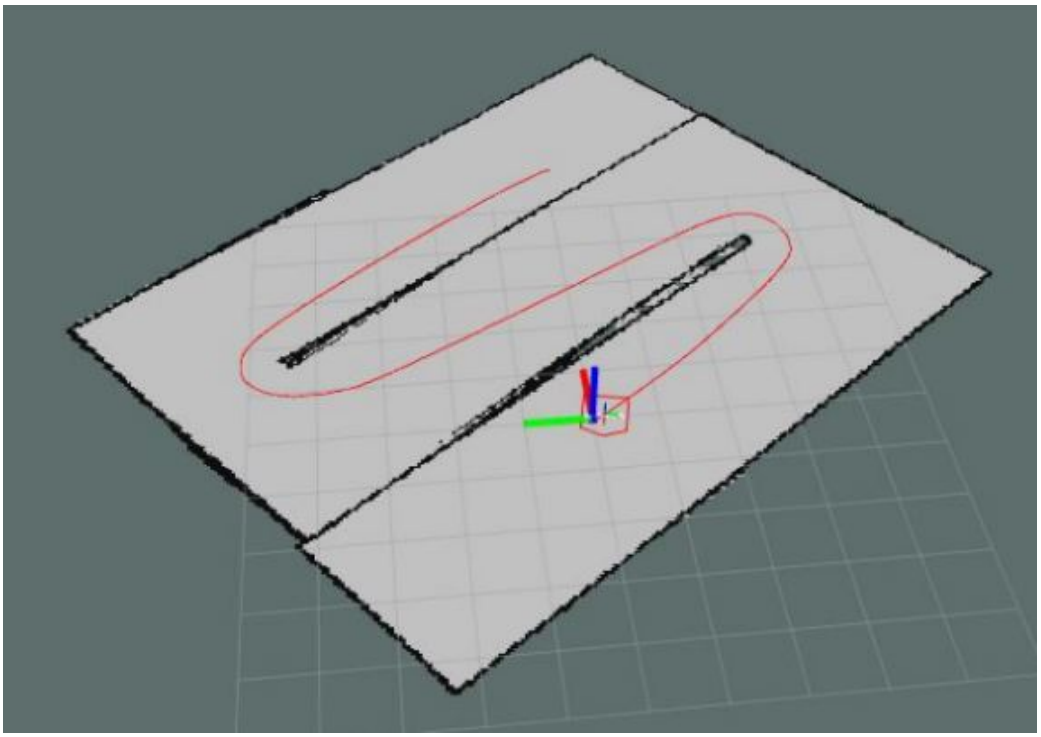
3. Experimentos realizados

Una vez implementados los cuatro ejercicios recomendados y mi propia estrategia me dispuse a realizar los dos experimentos indicados:

3.1. Corridor

Primero el más sencillo, con un tiempo de 446.6 segundos como muestro a continuación, se recorre el mapa por completo sin ningún tipo de problemas y, cuando finaliza, la base vuelve al nodo origen.

Captura del final de la ejecución, donde se muestra el mapa descubierto al completo por la base:



3.2. Willow garage

Por último realizamos el experimento con el mapa complicado. Tras estar ejecutándose unos 30 minutos la base consiguió completar el mapa tal y como lo muestro en la siguiente figura (en azul señalo donde se encuentra el robot en el momento que tomé la captura). He de decir que el robot durante la ejecución se choca varias veces y que además tarda un buen rato en recuperarse y continuar moviéndose por el mapa tras un golpe.