TSI

Entrega 2

Parte 1

Francisco Javier Caracuel Beltrán

Curso 2016/2017

Grupo 3

Índice

1. Descripción del algoritmo

Para recorrer todo el mapa, el explorador de fronteras comienza realizando dos giros de 360º, a continuación, detecta las fronteras, elige el punto más lejano de éstas y se dirige hacia él. Cuando ya ha descubierto el mapa al completo, termina la exploración.

La clase *FrontierExplorer* se encarga de gestionar todo el proceso.
Además de la funcionalidad que ya viene incluida, se han añadido los siguientes métodos:

- *void processActions();*
- *bool someNeighbourIsFree(int cell_x, int cell_y);*
- *void selectNearestNode(nodeOfFrontier &selectedNode);*
- *void selectFarthestNode(nodeOfFrontier &selectedNode);*
- *unsigned int getDifferenceAbsNodes(nodeOfFrontier node1, nodeOfFrontier node2);*
- *bool fullMap();*
- *void setTimeout(float t);*

En el ciclo principal del programa, se llama al método *processActions()*. En este método se gestiona los movimientos u operaciones que el robot debe realizar. Tiene implementadas 5 operaciones: terminar; girar dos veces 360º; girar una vez 360º; detectar y visualizar las fronteras; seleccionar y enviar el robot al objetivo.

A través de un entero, se le indica cuál de las operaciones anteriores es la que debe realizar, de este modo se puede guiar por el proceso que se debe realizar durante la exploración.

El robot comienza girando dos veces 360º, se le activa la acción de buscar y visualizar fronteras y las muestra.

Para detectar la frontera, se utiliza el callback *getmapCallBack()* y en él, se rellenan las posiciones del mapa con el método *rellenaObstaculos(x, y)* en el caso de encontrar uno. Este método actualiza con el valor 100 todas las casillas que se encuentran junto con la que se recibe por parámetro.

En cada ciclo, el mapa es actualizado, por lo que cuando ya se ha activado la opción de detectar fronteras, se puede recorrer la matriz que contiene todos los datos.

Para detectar las fronteras, se borra el vector que contiene todos los nodos frontera antiguos, se recorre todo el mapa y se comprueba si cada casilla tiene un valor diferente a -1, tiene algún vecino desconocido y tiene algún vecino libre. Cuando cumple esas tres condiciones, se añade al vector *frontera*.

Una vez relleno el vector *frontera*, para seleccionar el nodo objetivo se recorren todos los nodos frontera y se asigna el que mayor distancia tiene con la posición actual, usando el método *getDifferenceAbsNodes()*.

Si no consigue el objetivo enviado, se cambia el criterio de selección y asigna el nodo más cercano a la posición actual. En cuanto consigue llegar a este nodo, vuelve a utilizarse el criterio del nodo más lejano.

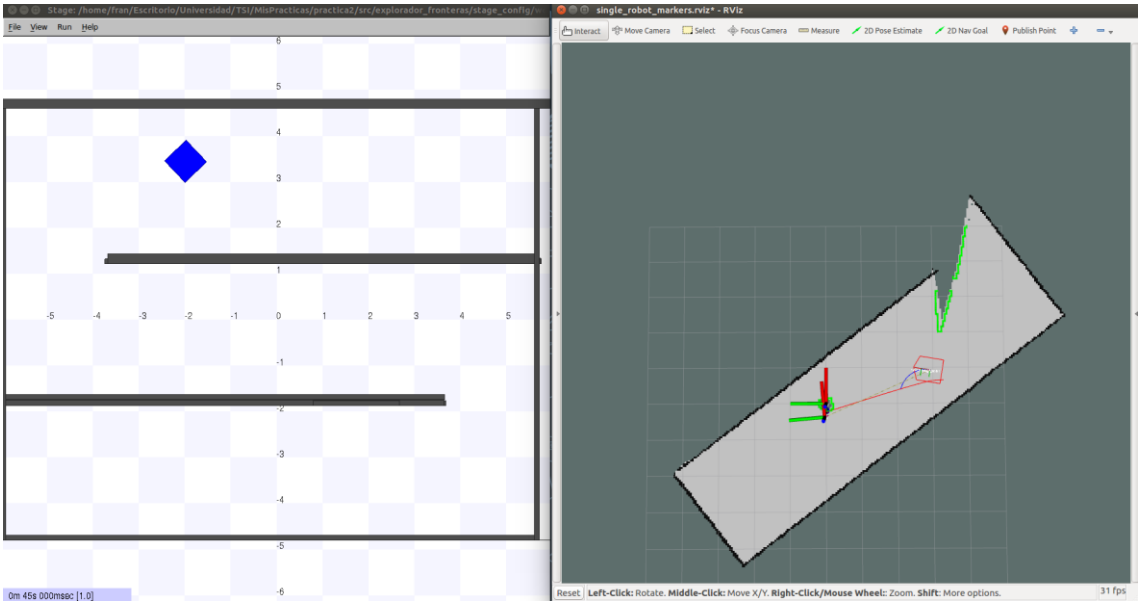
2. Experimentación

Se ha experimentado con dos mapas:

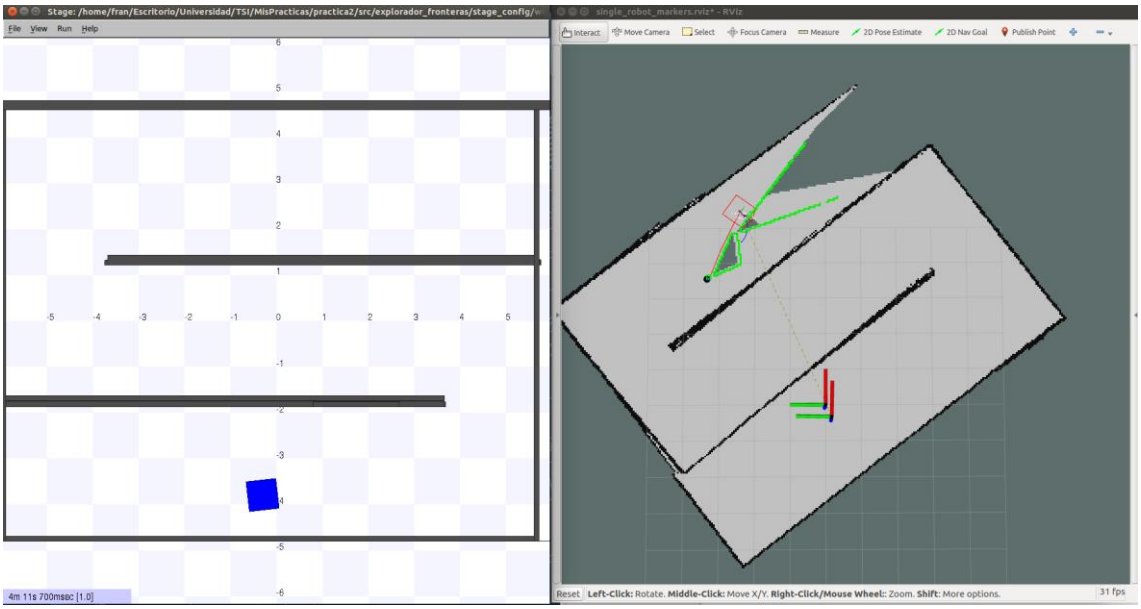
- *Corridor:*

  Ha conseguido explorar todo el mapa en un tiempo de 5 minutos y 23 segundos.

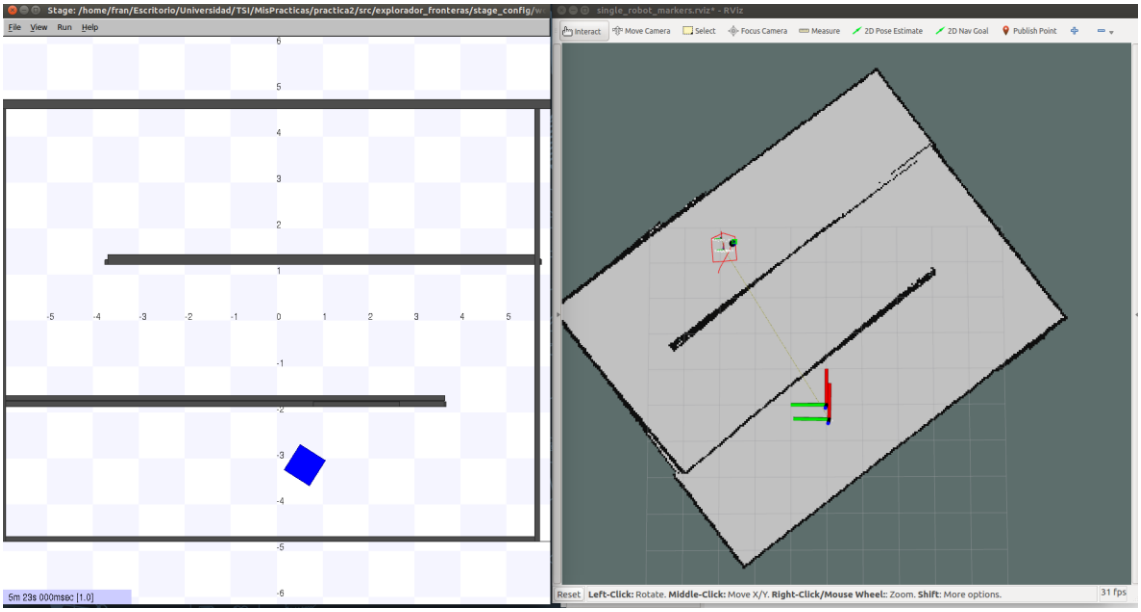
a) Comienza girando dos veces 360º, descubre el punto más lejano y lo explora. Como la zona donde se encontraba no la ha reconocido, vuelve al punto de inicio para explorarlo (ya que se ha convertido en el más lejano inexplorado):



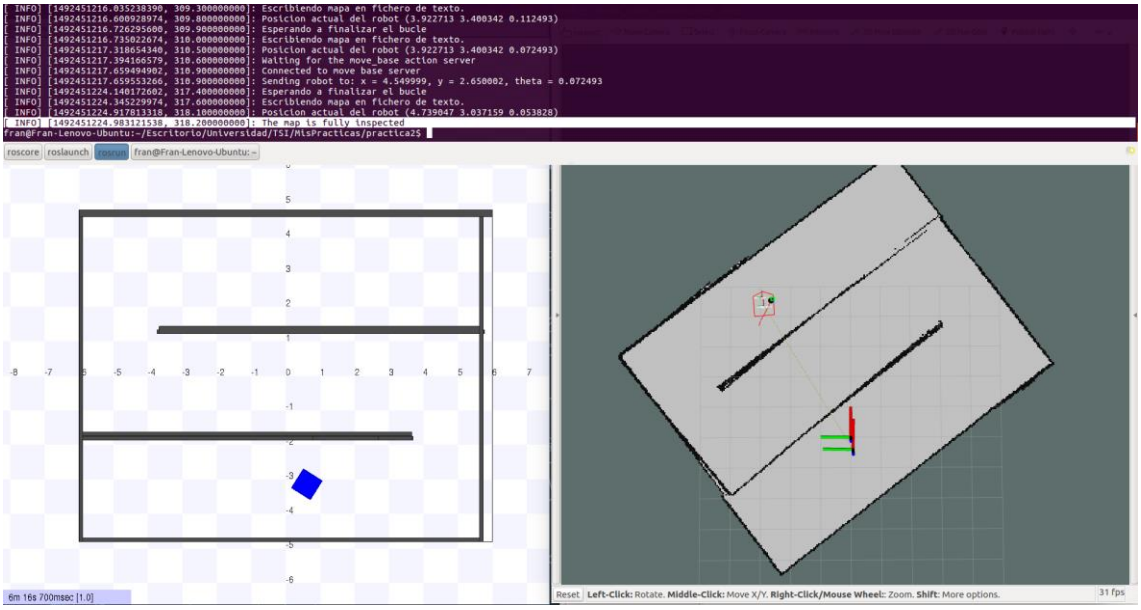
b) Consigue recorrer sin problema los pasillos. A veces, si el objetivo está cerca de un obstáculo, el planificador no consigue marcar una ruta y selecciona la frontera más cercana:



c) Finalmente, consigue explorar el mapa completo:



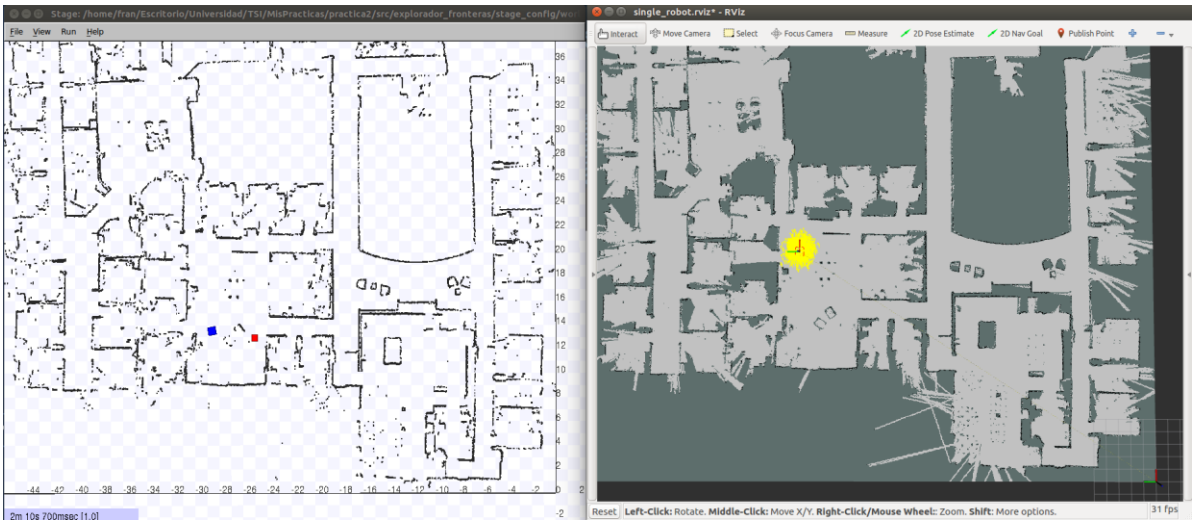
d) Cuando termina nos informa de que lo ha explorado al completo:



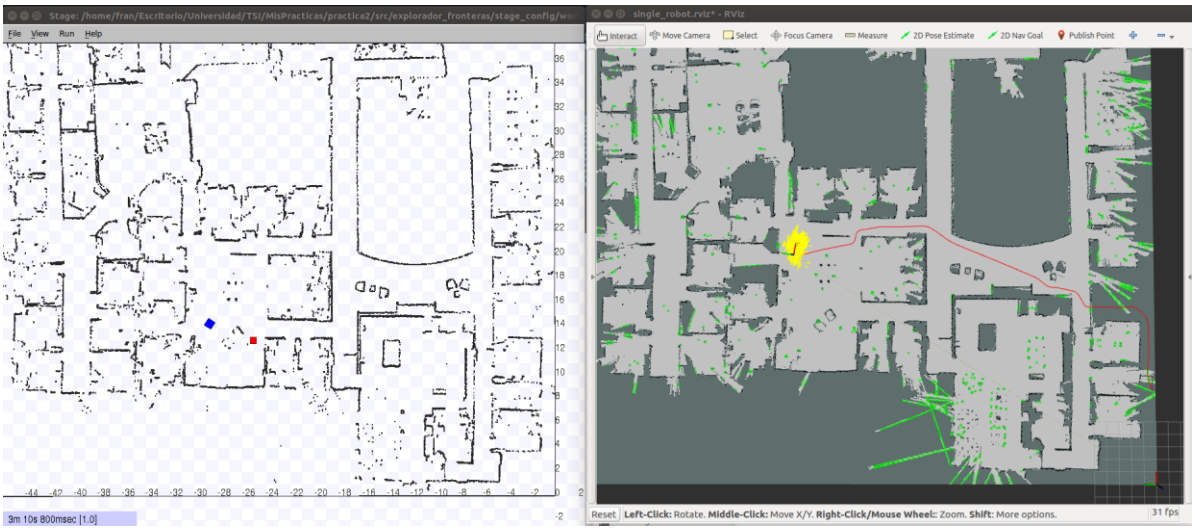
- *Willow_garage:*

  La experiencia con *willow_garage* no ha sido buena.

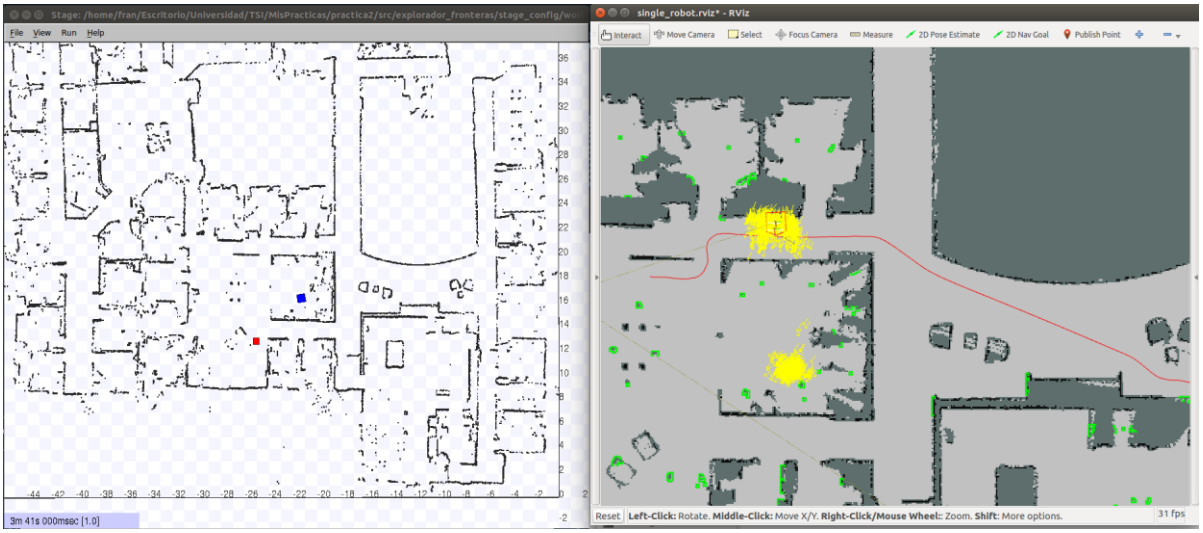
1. Comienza el proceso normalmente:



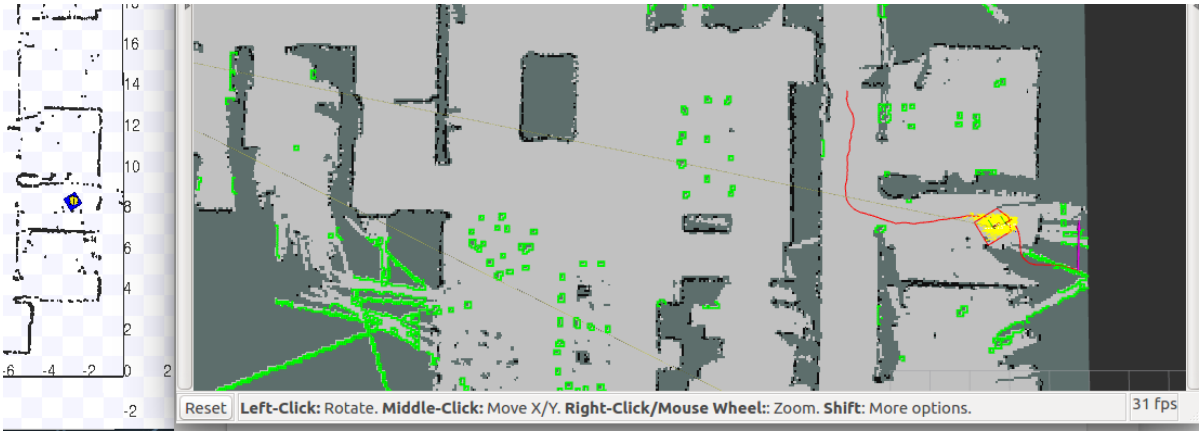
2. Obtiene las fronteras y establece un objetivo:



3. Pero en cuanto se mueve hacia él no consigue ubicarse en el mapa y se pierde:



4. El mapa es muy pequeño en algunas zonas y hay que moverlo manualmente porque choca:



Durante todo el proceso no consigue llegar a los destinos con normalidad y se queda sin poder desplazarse, por lo que, al menos de momento, no consigue una buena exploración en este mapa.

3. Uso

Para poder ejecutar esta práctica, se ofrecen dos *launch*:

- *mi_amcl.launch*
- *mi_gmapping.launch*

El nodo es capaz de recibir por parámetro el *timeout*, por lo que su ejecución puede ser:

- *rosrun explorador_fronteras explorador_fronteras_node <timeout>*