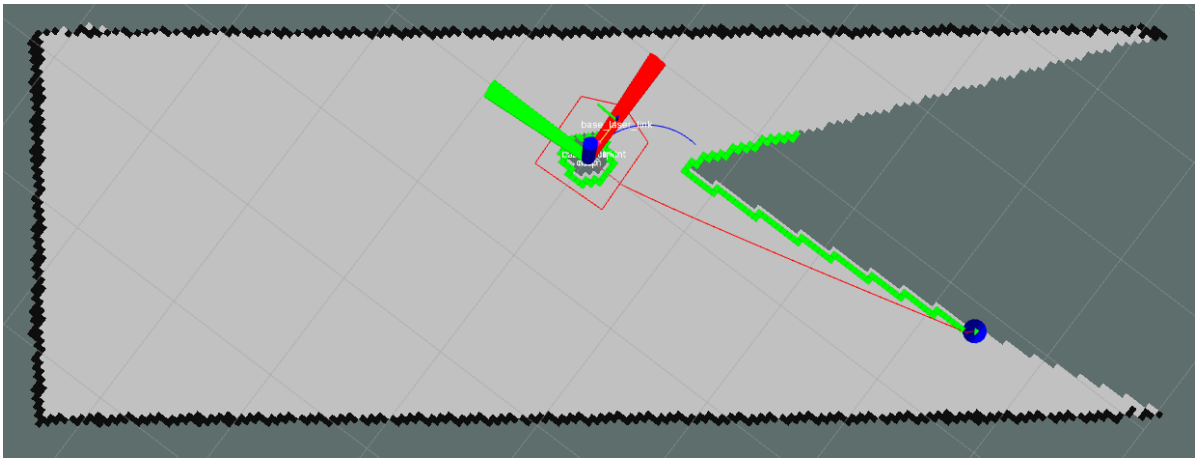
# 1. Introducción.

Esta entrega consiste en desarrollar un robot que permita explorar un mapa usando la técnica de exploración por fronteras. Para usar esta técnica debemos saber que significa el término *frontera*. Diremos que un punto del mapa (también llamado nodo) está en la *frontera* si parte de la vecindad de dicho nodo no ha sido explorado, es decir, nuestro robot ha descubierto la zona cercana a dicho nodo pero no en su totalidad, en la siguiente imagen podemos ver claramente la idea de frontera (representada en color verde):



Los pasos que debe seguir el robot para realizar una exploración del mapa son los siguientes:

1. Girar dos veces 360º grados sobre si mismo.
2. Detectar la frontera.
3. Elegir el nodo de la frontera al cual se va a dirigir.
4. Intentar alcanzar dicho nodo.
5. Adoptar una estrategia si no se alcanza el objetivo.

Para realizar este comportamiento se han realizado tareas descritas a continuación.

# 2. Girar 360º.

Esta tarea sirve para descubrir mapa estando en una posición. La implementación es bien sencilla y está basada en el primer ejercicio que hicimos con ROS, el de mover la tortuga y luego hacerla girar 45º. La idea es jugar con la fórmula de la velocidad. Fijando la velocidad de giro y sabiendo que el espacio que debe girar es 360 grados (en mi caso 720, ya que el doble giro lo hago directamente) sólo debemos calcular el tiempo necesario de giro para girar el espacio deseado.

# 3. Inflación de zonas de obstáculos.

Este comportamiento se ha implementado en la función *getmapCallBack*. La idea es que cada vez que encontremos una celda que represente un obstáculo, es decir, que el valor de la celda sea 100, se rellene con obstáculos el entorno de 1 metro cuadrado centrado en esa celda. Para ello se se hace uso de la función auxiliar *rellenaObstaculos*. El pseudocódigo de ambas funciones es el siguiente:

```
1: function GETMAPCALLBACK(...)
2:     ... (Código proporcionado)
3:     Copiamos el mapa cmGlobal tal cual en theGlobalCm.
4:     for y = 0, y < cmGlobal.info.height, y += 1 do
5:         ini_y = y - 10, fin_y = y + 10
6:         Comprobamos que ini_y > 0 y que fin_y < la altura del mapa.
7:         for x = 0, x < cmGlobal.info.width, x += 1 do
8:             ini_x = x - 10, fin_x = x + 10
9:             Comprobamos que ini_x > 0 y que fin_x < la anchura del mapa.
10:            if cmGlobal.data[currCell] == 100 then
11:                Recorremos el cuadrante definido por ini_y, ini_x, fin_y y fin_x
12:                Llamamos a rellenarObstaculos
13:            end if
14:        end for
15:    end for
16: end function
```

```
1: function RELLENAROBSTACULOS(cell_x, cell_y)
2:     Rellenamos las celdas adyacentes a la celda dada por las coordenadas cell_x, cell_y
       con el valor 100, comprobando que no nos salimos del mapa.
3: end function
```

# 4. Etiquetar celdas frontera.

El pseudocódigo para rellenar la frontera es el siguiente:

```
1: function LABELFRONTIERNODES
2:     frontera.resize(0)
3:     for i = 0, i < theGlobalCm.size(), i += 1 do
4:         for j = 0, j < theGlobalCm[i].size(), j += 1 do
5:             if someNeighbourIsUnknown(j, i) y theGlobalCm[i][j] == 0 then
6:                 n.x = j * cmGlobal.info.resolution + cmGlobal.info.origin.position.x
7:                 n.y = i * cmGlobal.info.resolution + cmGlobal.info.origin.position.y
8:                 frontera.push_back(n)
9:             end if
10:        end for
11:    end for
12: end function
```

Como se ve en el pseudocódigo debemos comprobar que algún vecino es desconocido y que dicha celda esté vacía, para que no sea un obstáculo y no sea desconocida. Una vez se ha rellenado la frontera, podemos pasar a la siguiente tarea, elegir un nodo e ir hacia el.

# 5. Seleccionar un nodo (celda) de la frontera.

Yo he seleccionado el nodo más lejano de la frontera, con la idea de descubrir el máximo mapa posible en el intento de alcanzar ese nodo. El pseudocódigo es trivial:

```
1: function selectNode
2:     d_actual = 0, d = -1
3:     for i = 0, i < frontera.size(), i += 1 do
4:         d_actual = distancia(nodoPosicionRobot, frontera[i])
5:         if d_actual > d y d_actual > 0.15 then
6:             d = d_actual
7:             indice = i
8:         end if
9:     end for
10: end function
```

Le he exigido que la distancia sea mayor a *0.15* para evitar que seleccione nodos demasiado cercanos. A la hora de seleccionar el nodo más lejano no influye, pero en la estrategia implementada en caso de que el robot no llegue al nodo seleccionado si es necesario.

# 6. Enviar el robot al nodo seleccionado.

Para enviar el robot al nodo elegido se ha usado el código que se nos proporciono en el paquete *mi_send_goals*. Este código levanta un cliente *MoveBase* y le pasa el punto a donde se debe mover el robot y ya el planificador de ROS se encarga de llevar al robot a la posición. Debemos tener en cuenta que puede darse el caso de que el robot no llegue al objetivo y que entre en un ciclo infinito. Para evitar esto se ha introducido un parámetro que determina un *timeout* durante el cual el robot intentará alcanzar el objetivo. Para los casos en los que el robot no llegue al objetivo debemos adoptar una estrategia, esto lo podemos ver en la siguiente sección.

# 7. Que hacer si el robot no llega al nodo seleccionado.

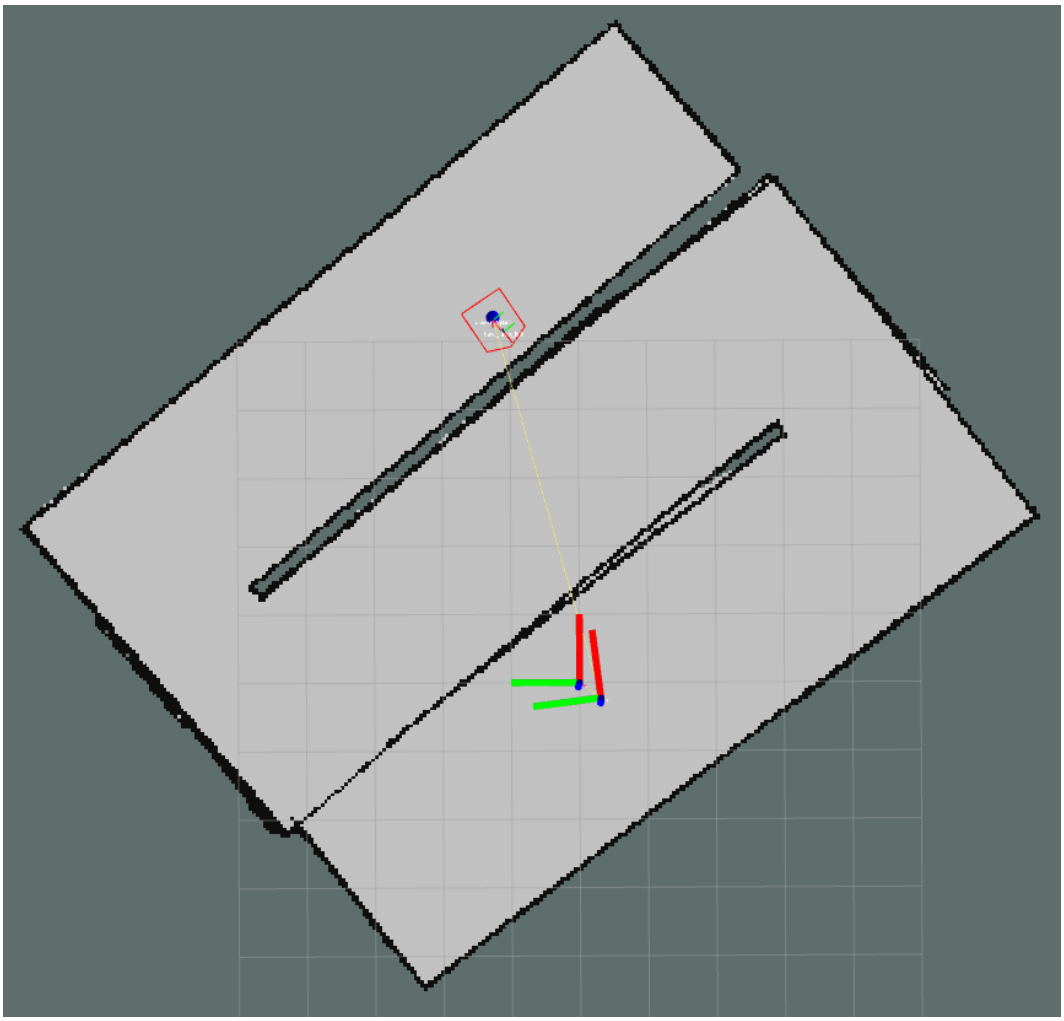
La estrategia desarrollada es simple, si no se llega al objetivo el robot intentará volver a la posición anterior en la que estaba pero con un *timeout* igual a la mitad del *timeout* original para no retroceder demasiado. Pero, ¿qué pasa si tampoco lo consigue? ¿Entraría en un ciclo infinito? Para evitar esto lo que he hecho es que intente volver a la posición

original 3 veces, en el caso de que no lo consiga, se cambiará la manera de elegir el nodo y se pasará a elegir el nodo el nodo más cercano (teniendo en cuenta que la distancia sea mayor que *0.15*). Escogiendo el nodo más cercano es más probable que el robot llegue a el y se reinicie todo el proceso.

# 8. Experimentos.

Se han realizado dos experimentos, uno en el mapa *corridor* y otro en el mapa *willow_garage*. Para ver el comportamiento de nuestro robot debemos ejecutar *roslaunch explorador_fronteras explorador_fronteras_corridor.launch* para lanzar el robot en el mapa *corridor* y *roslaunch explorador_fronteras explorador_fronteras_willow.launch* para lanzar el robot en el mapa *willow garage*. A continuación, en una nueva terminal, ejecutamos *rosrun explorador_fronteras explorador_fronteras <timeout>*, donde *timeout* indica el valor del *timeout* que se usa en el código explicado anteriormente.

El mapa descubierto en el caso del mapa *corridor* es el que podemos ver a continuación:



Como podemos ver se descubre el mapa entero. Este proceso se ha realizado en 4 minutos y 53 segundos.

En el caso del mapa *willow garage* tardaríamos mucho en descubrirlo entero por el tamaño del mismo, así que he ejecutado mi robot durante diez minutos. El mapa del cubierto es el siguiente: