El problema de la exploración basada en fronteras

La mayoría de las aplicaciones de robots móviles requieren la capacidad de navegar. Mientras que muchos robots pueden navegar usando mapas, y algunos pueden hacer un mapa de lo que pueden ver, pocos pueden explorar autónomamente más allá de su entorno inmediato. Por lo general, un ser humano debe mapear el territorio por adelantado, proporcionando la localización exacta de los obstáculos (para los mapas métricos) o un gráfico que representa la conectividad entre las regiones abiertas (para los mapas topológicos). Como resultado, la mayoría de los robots de navegación se vuelven inútiles cuando se colocan en entornos desconocidos.

La exploración tiene el potencial de liberar a los robots de esta limitación. Definimos la exploración como el acto de moverse a través de un entorno desconocido, construyendo un mapa que puede usarse para la navegación posterior. Una buena estrategia de exploración es aquella que genera un mapa completo o casi completo en un tiempo razonable.

Nuestro objetivo es desarrollar estrategias de exploración para entornos complejos típicamente encontrados en edificios de oficinas reales. Nuestro enfoque se basa en la detección de una frontera, un conjunto de puntos del mapa del entorno que tienen en su vecindad espacio inexplorado.

Desde cualquier frontera, el robot puede ver en el espacio inexplorado y añadir las nuevas observaciones a su mapa. Desde cada nuevo punto de vista, el robot puede ver nuevas fronteras situadas en el borde de su percepción. Explorando cada frontera, o determinando que la frontera sea inaccesible, el robot puede construir un mapa de cada localización accesible en el ambiente.

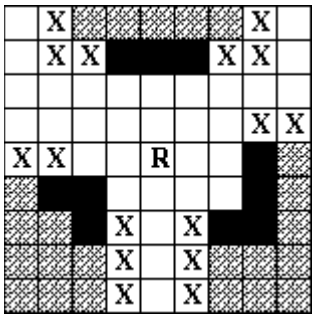
Vamos a implementar la exploración de fronteras con un robot equipado con un sensor de rango láser con un ángulo de visión de 270º y un rango de aproximadamente 3 metros.

Detección de Fronteras

Utilizaremos el concepto de Occupancy Grid, explicado en clase, como nuestra representación espacial. Una Occupancy Grid es una matriz de celdas y cada celda almacena la probabilidad de que la región correspondiente en el espacio esté ocupada. El paquete gmapping de ROS se encarga de obtener lecturas del sensor láser y actualizar la Occupancy Grid publicándola en el topic /map. Mediante la implementación de una función callback suscrita al topic /map podremos obtener una matriz en la que el valor de cada celda es alguna de estas posibilidades:

- 0: representa espacio libre
- 100: representa un obstáculo
- -1: representa espacio desconocido (inexplorado).

Vamos a llevar a cabo un proceso análogo (aunque simplificado) a la detección de fronteras en visión por computadora para encontrar los límites entre el espacio abierto y el espacio desconocido. Cualquier celda libre adyacente a una celda desconocida la identificamos como una celda de frontera, y estas celdas identificadas se insertan en una lista de nodos frontera.



La figura anterior muestra un ejemplo simplificado de detección de frontera. La ubicación del robot está marcada con una R. Las celdas que se sabe que están ocupadas son negras; Las celdas que se sabe que están desocupadas son blancas; Y las celdas cuya ocupación es desconocida están sombreadas en gris. Las celdas (libres) marcadas con X son las celdas del borde fronterizo.

Una vez que la frontera ha sido detectada, haremos que el robot intente navegar a un punto de la frontera que ha tenido que ser seleccionado siguiendo algún criterio, a elección del programador.
Puede ser el punto más cercano a la posición actual del robot, el más lejano, el que tenga una mayor área desconocida a su alrededor, o cualquier otro criterio razonable para mejorar la eficiencia en el reconocimiento del entorno. Para determinar la posición del robot tenemos que implementar una funcion callback suscrita al topic /odom, en el que se publica la pose del robot.

La navegación hacia el punto escogido se llevará a cabo comunicándose con el paquete move_base de ROS, como se explica en clase. Este paquete implementa un algoritmo de planificación de rutas (path planning) que encuentra un camino entre la posición actual del robot y el punto de la frontera seleccionado.

Descripción del código fuente

El código fuente que se describe a continuación contiene las funciones básicas para que el alumno pueda extenderlo e implementar un explorador basado en fronteras. Los ficheros fuente a los que se hace referencia en esta sección están en el directorio src del paquete mi_mapeo_stage que puede descargarse desde PRADO, tal y como se explica en el documento HowTo_mi_mapeo_stage.

El fichero ***frontier_explorer_lite.h*** contiene la implementación de la clase FrontierExplorer, donde se encuentran declarados las variables y métodos necesarios para implementar un explorador de fronteras simple, tal y como se ha descrito arriba. **Ver el código fuente comentado** de este fichero para entender cada una de las variables de clase y de los metódos. La clase también contiene información sobre visualización de objetos en rviz cuya funcionalidad se describe brevemente al final de este documento.

Es importante tener en cuenta que en la declaración de la clase están definidas las cabeceras de métodos **que no están implementados y que deberán implementarse siguiendo la descripcón de los ejercicions de la relación de problemas.** En concreto, las funciones no implementadas y declaradas en la clase son las siguientes:

```
void gira360(); //Método para que el robot gire 360º
bool someNeighbourIsObstacle(int cell_x, int cell_y); //Devuelve true si algún vecino de la celda (x,y), en coordenadas cartesianas, es un obstáculot
void labelFrontierNodes(); //Método que detecta nodos (celdas) frontera y las almacena en la lista frontera
void selectNode(nodeOfFrontier &selectedNode);//Método que selecciona un nodo de la frontera para establecer el objetivo al que debe navegar el robot.
void rellenaObstaculos(int cell_x, int cell_y); //Método que modifica el mapa rellenando obstáculos alrededor de la celda (x,y), en coordenadas cartesianas
```

El fichero ***frontier_explorer_lite.cpp*** contiene la implementación de algunos métodos del explorador de fronteras y una función main en la que se implementa un bucle sencillo en el que se llevan a cabo las etapas de escritura del mapa capturado en un fichero y se muestra por pantalla la posición del robot. Las funciones implementadas son las siguientes:

Constructor de la clase FrontierExplorer. Inicializa las variables de clase y, fundamental, inicializa el publisher de las velocidades del robot, en el topic /**cmd_vel** y los subscribers al topic **/odom** y al topic **/map**. Adicionalmente, incializa los publishers necesarios para visualizar información en rviz.

```
FrontierExplorer::FrontierExplorer()
{
  //Inicializa el explorador de fronteras.
  mapIsEmpty = true; //informa si el mapa recibido solo tiene valores -1
  posGoal.x = posGoal.y = 0;  //Posición del objetivo
  nodoPosicionRobot.x = nodoPosicionRobot.y = 0;      //Posición actual
  yaw =0;     //Angulo (en radianes) de orientación del robot
  v_angular = v_lineal = 0.0; //velocidad angular
   // Advertise a new publisher for the simulated robot's velocity command topic
   commandPub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);
  //Publicador del objetivo en rviz.
  marker_pubobjetivo = node.advertise<visualization_msgs::Marker>("visualizacion_objetivo", 1);
 //Publicador de los nodos de frontera para visuoalizarlos en rviz
  marker_pub = node.advertise<visualization_msgs::Marker>("visualization_marker", 1000);
     //Se subscribe a la posición publicada por la odometría
  odomSub =  node.subscribe("odom", 100, &FrontierExplorer::odomCallBack, this);
  //Se subscribe al mapa publicado por gmapping
  mapSub = node.subscribe("map", 100, &FrontierExplorer::getmapCallBack, this);
}
```

Función CallBack para obtener el mapa publicado por gmapping. Recibe el mapa en el argumento **msg**, copia sus metadatos (dimensiones, resolución, etc) en la variable de clase **cmGlobal** y copia el mapa del mensaje en la variable de clase **theGlobalCm.**

```
void FrontierExplorer::getmapCallBack(const nav_msgs::OccupancyGrid::ConstPtr& msg){
   //callback para obtener el mapa.
   mapIsEmpty = true; //asumimos que el mapa está vacío antes de recibirlo
   cmGlobal = *msg;   //guardamos en la variable cmGlobal el mensaje recibido para
                      // disponer del mensaje en nuestro programa
   //redimensiona el  mapa segun las dimensiones del mapa recibido
   theGlobalCm.resize(cmGlobal.info.height);//resize al numero de filas
   for (int y = 0; y < cmGlobal.info.height; y++)
     theGlobalCm[y].resize(cmGlobal.info.width);

   //copia el mapa del mensaje en el mapa del explorador de fronteras
   //informa en mapEmpty si hay celdas con información.
   int currCell = 0;
   for (int y=0; y < cmGlobal.info.height; y ++)
     for (int x = 0; x < cmGlobal.info.width; x++){

         theGlobalCm[y][x] =  cmGlobal.data[currCell];
      if (theGlobalCm[y][x] != -1) mapIsEmpty = false;

      currCell++;
      }
```

Función CallBack para obtener la posición del robot. Captura la posición del robot en la variable de clase **nodoPosicionRobot.**

```
void FrontierExplorer::odomCallBack(const nav_msgs::Odometry::ConstPtr& msg)
{
   //obtengo la posición. y la guardo en el dato miembro del objeto FrontierExplorer
   nodoPosicionRobot.x=msg->pose.pose.position.x;
   nodoPosicionRobot.y=msg->pose.pose.position.y;
   //también la guardo en FrontierExplorer::odometria
   odometria.pose.pose.position.x = nodoPosicionRobot.x;
   odometria.pose.pose.position.y = nodoPosicionRobot.y;
   //get Quaternion anglular information
   double x=msg->pose.pose.orientation.x;
   double y=msg->pose.pose.orientation.y;
   double z=msg->pose.pose.orientation.z;
   double w=msg->pose.pose.orientation.w;
   //la guardo en la odometría
   odometria.pose.pose.orientation.x = x;
   odometria.pose.pose.orientation.y = y;
   odometria.pose.pose.orientation.z = z;
   odometria.pose.pose.orientation.w = w;

   //convertir to pitch,roll, yaw
   //consultado en http://answers.ros.org/question/50113/transform-quaternion/
   //guardo el yaw en FrontierExplorer::yaw
   yaw=atan2(2*(y*x+w*z),w*w+x*x-y*y-z*z);
};
```

Función para escribir el mapa en un fichero.

```
void FrontierExplorer::printMapToFile () {

  //Escribe el mapa en el fichero "grid.txt"
   std::ofstream gridFile;
   gridFile.open("grid.txt");

    for (int x=0 ;x < theGlobalCm[0].size(); x++) {
        for (int y= theGlobalCm.size()-1 ; y >=0 ;y--){
           gridFile << (int) theGlobalCm[y][x] << " ";
         }
         gridFile << std::endl;
      }
      gridFile.close();
}
```

La función main inicializa el nodo ros (llamado frontier_explored_node), declara e inicializa la variable **explorador** de tipo ***FrontierExplorer*****,** inicializa los Markers para visualizar información en rviz. Después entra en un bucle de llamadas a **spinOnce()** para gestionar las llamadas a las callbacks hasta que se obtenga un mapa no vacío. Finalmente, implementa un bucle sin fin (con una frecuencia de 0.2 Hz, es decir, con un período de 5 segs) en el que continuamente se escribe el mapa en un fichero y se muestra la posición del robot.

```
 int main(int argc, char** argv) {

   ros::init(argc,argv, "frontier_explorer_node");
  //inicializa el explorador
   FrontierExplorer explorador;
   //inicializa la visualización de la frontera
   explorador.inicializaMarkers("frontera", 0,0.0f,1.0f,0.0f);
   //inicializa la visualización del objetivo seleccionado.
   explorador.inicializaMarkerSphere("objetivo",0, 0.0, 0.0, 1.0);
   //hacemos que se gestionen las callbacks hasta que el mapa tenga información
   ROS_INFO("Esperando a que mapa REinicie y tenga información");
   while (explorador.mapIsEmpty)
     ros::spinOnce();
   //ROS_INFO("YA SE QUE EL MAPA NO ESTÁ VACÍO %d filas, %d columnas",
  explorador.cmGlobal.info.height, explorador.cmGlobal.info.width);
   ros::Rate frecuencia(0.2);

   while (ros::ok()) {
      //printMapa(explorador.theGlobalCm);
      ROS_INFO("Escribiendo mapa en fichero de texto.");
      explorador.printMapToFile();
      ROS_INFO("Posicion actual del robot (%f %f %f)",
            explorador.nodoPosicionRobot.x,
            explorador.nodoPosicionRobot.y,
                              explorador.yaw);

      ROS_INFO("Esperando a finalizar el bucle");
      frecuencia.sleep();
      ros::spinOnce();

     }
   // shutdown the node and join the thread back before exiting
   ros::shutdown();
   return 0;
```

Visualización de información en rviz.

Para facilitar la depuración y resultado final de la implementación se proporcionan funciones para poder visualizar información en rviz, en concreto se puede mostrar en rviz:

- La lista de nodos que forman la frontera
- La celda de la frontera seleccionada como punto destino de la navegación.

Un objeto se puede visualizar en rviz si se define un **objeto marker** de tipo visualization_msgs::Marker y se habilitan las funciones necesarias para inicializarlo, visualizarlo y borrarlo de rviz.

Para visualizar un marker en rviz hay que publicarlo en un t**opic al que se subscribe rviz** y es necesario declarar el publisher de ese topic. Esto se hace en el constructor de la clase FrontierExplorer, donde se definen los topics visualización_objetivo (para mostrar el objetivo) y visualization_marker (para mostrar la lista de puntos de la frontera).

```
FrontierExplorer::FrontierExplorer()
{
   …..
   //Publicador del objetivo en rviz.
   marker_pubobjetivo = node.advertise<visualization_msgs::Marker>("visualizacion_objetivo", 1);
  //Publicador de los nodos de frontera para visuoalizarlos en rviz
   marker_pub = node.advertise<visualization_msgs::Marker>("visualization_marker", 1000);
   ……
}
```

Las funciones de inicialización
void FrontierExplorer::inicializaMarkerSphere(std::string ns, int id, float r, float g, float b)
void FrontierExplorer::inicializaMarkers(std::string ns, int id, float r, float g, float b)

declaran la forma de los objetos a visualizar: una esfera azul para el objetivo de navegación) y una lista de cuadrados de color verde para las celdas de la frontera.

L**as funciones para visualizar son las siguientes:**

void FrontierExplorer::visualizaObjetivo(double x, double y)

Que muestra en rviz una esfera azul en el punto del mundo real (x,y).

void FrontierExplorer::visualizaLista(TipoFrontera lista)

Que muestra en rviz la lista de puntos de la frontera, pasándole una lista de TipoFrontera, como un conjunto de cuadrados de color verde.

Estas funciones deben ser llamadas cuando el programador considere que debe visualizarse en rviz su objeto correspondiente.

Las funciones para borrar (limpiar) un objeto de la visualización son

void FrontierExplorer::limpiaMarkerObjetivo()
que borra de rviz la esfera donde esté visualizándose el punto objetivo de navegación.

void FrontierExplorer::limpiaMarkers()

que borra de rviz todos los puntos de la frontera.