MTRN4010 - 2022

Occupancy Grids

In many applications, we need to generate a map of the environment, for representing certain properties, which are correlated to the geographical position.
Those properties are acquired by the platform, using its onboard sensors. For instance, if a robot detects some obstacle on the terrain, in addition to possibly take some action (e.g., avoiding it if there is a risk of collision) it should be able to add that piece of information to a common map. That map can be used by the robot itself, later, or by other robots which are able to share information with our robot through the common map (usually using a Global Coordinate Frame, GCF).
Some properties are usually massive, e.g. the terrain description is one typical case. In this case the property can be understood as a variable which is function of the position, e.g. V(x,y), where V is the value of the property and (x,y) is the position variable at which the property is measured (or estimated or defined).
In order to represent the property V(x,y) in a numerical and tractable way (for our computers), we apply the concept of discretizing the domain of the variables (x,y), through defining a **grid**. The grid allows us to create a discrete (in space, i.e., (x,y)) version of V(x,y).

Although there are diverse properties which are of interest for the robot, and for the specific task of the robot (in mining, agriculture, factories, etc.), we focus the mapping matter in modeling the terrain for representing the areas that are "traversable" for a moving robot, because those are crucial for the robot to move, and for planning.

We define a property **A(x,y)**, whose value at each point (x,y) indicates if that point is traversable or not. A possible convention would be:

A(x,y)=1	CLEAR	(point (x,y) is traversable - Clear of risk)
A(x,y)=2	OCCUPIED	(the point is not traversable –there is a risk there)
A(x,y)=0	UNKNOWN	(there is no information about the condition at this point)

As we mentioned before, we simplify the representation of A(x,y) by applying an Occupancy Grid (OG). We divide the region of coverage of A(x,y) in a set of *cells*. Suppose we represent the area of coverage $x \in [x_1, x_2), y \in [y_1, y_2)$, divided by a grid of N_x by N_y cells. Each cell will be referred by a couple of indexes *(i,j)*,.

$$\Omega_{j,i} = \left\{ \quad (x,y) \in R^2 \quad / \quad x_1 + (i-1) \cdot c_x \leq x < x_1 + i \cdot c_x, \quad y_1 + (j-1) \cdot c_y \leq y < y_1 + j \cdot c_y \quad \right\}$$

$$1 \leq i \leq N_x \quad , \quad 1 \leq j \leq N_y$$

$$c_x = (x_2 - x_1) / N_x$$

$$c_y = (y_2 - y_1) / N_y$$

Our discrete representation is assumed (or forced) to be constant for all the points that are inside the same cell, i.e. the property of interest, A(x,y), is approximated by a piece-wise constant function.

As our measurements are associated to the continuous variable ***(x,y)*** we project these variables into the grid by applying the following discretization operation, for obtaining the associated discrete indexes ***(i,j)***. Given a point ***(x,y)*** we obtain the associated indexes according to:

$$i = \left\lfloor \left(x - x_1 \right) / c_x \right\rfloor + 1, \quad j = \left\lfloor \left(y - y_1 \right) / c_y \right\rfloor + 1$$

(note: The operator $\lfloor . \rfloor$ is implemented by the function *floor(.)* in C and Matlab)

We apply this operation to any couple (x,y) that belongs to the area of coverage of the grid,

$$\left\{ \left(x, y \right) \quad \backslash \quad x_1 \le x < x_2, \quad y_1 \le y < y_2 \right\}$$

Points that do not belong to the area covered by the OG are ignored, although in certain applications the OG area of coverage needs to be dynamically redefined, in order to contain all the points.

Example in Matlab

```
% Short example about how to implement a deterministic OG
% Jose Guivant. For MTRN4010 students.

% ..............................................................
function main( )
% define the area of coverage and the number of cells we want to spend
MyContext.x1 = -20 ;MyContext.x2 =  20 ;
MyContext.y1 = 0 ;MyContext.y2 =  20 ;
MyContext.Nx = 100 ;
MyContext.Ny = 50 ;
%  The area of coverage is: x values between -20m to 20m (suppose I am working
% in meters) and y values are between 0m to 20m
% I also want to represent the OG by using a discrete grid of 50 x 100  (Ny
% x Nx ) cells.

% Create OG (I do it, inside this function) which I am calling.
DefineOG() ;

% ....................
% Some figure, to get points from user (because in this example I use points
% specified by the user)
figure(1) ; clf ; plot(0,0) ;
axis([MyContext.x1,MyContext.x2,MyContext.y1,MyContext.y2]) ;

% Get some points, provided by the user
disp('input (mouse click) 10 points in figure 1') ;
pp =ginput(10) ;
% ....................

% Set the cells that own the points (see the code of the function "PopulateOG")
PopulateOG(pp(:,1),pp(:,2)) ;

% show OG, as an image
figure(2); clf ; imagesc(MyContext.M) ; colormap gray ;
% in the image: the cells =1 will look white, and the rest (=0) will be black


set(gca,'ydir','normal','xdir','normal') ;
% just to tell Matlab how to show the image

return ;    % done
% ..........................................................

function DefineOG( )
```

```
    % Define a structure, with parameters useful for processing an Occupancy
    % Grid.

    % here I define a Matrix for storing the values of the OG (I call it "M")
    MyContext.M = zeros(MyContext.Ny,MyContext.Nx,'uint8') ;

    % These constants are useful for scaling (x,y) points to cells' indexes.
    MyContext.Cx = MyContext.Nx/(MyContext.x2-MyContext.x1) ;
    MyContext.Cy = MyContext.Ny/(MyContext.y2-MyContext.y1) ;
    return;
end
% ..........................................................

% This function transforms points (x,y), for obtaining their equivalent cells' indexes.
% and then it uses them to set the related cells
function PopulateOG(x,y)
% Firstly, we filter out those points that are outside our ROI (Region of Interest).
% We keep the valid ones.
ii = find((x>=MyContext.x1)&(x<MyContext.x2)&(y>=MyContext.y1)&(y<MyContext.y2));
    x=x(ii) ;    y=y(ii) ;
% just consider the points that are inside the OG's coverage

    % convert to indexes
    ix =  floor((x-MyContext.x1)*MyContext.Cx)+1 ;
    iy =  floor((y-MyContext.y1)*MyContext.Cy)+1 ;

    % Convert 2D indexes to linear indexes
    ixy = sub2ind(size(MyContext.M),iy,ix) ;

    % Set the associated OG's cells, to contain value =1
    MyContext.M(ixy) =1 ;
 return;
end
% ..........................................................
 end            % end of this program

m
```

Add and Erase Operations

Suppose we implement an OG for representing a basic property such as a map for describing obstacles and for clear (of obstacles) zones. As the **mapping process** obtains evidence about the state of certain cells, then it is able to set their values.

In our projects (not in 2022), you are requested to implement an OG based on the information provided by the on-board front 2D LiDAR. The sensor scans in a horizontal plane, so we work in a 2D context. Each range measurement reports a point, i.e. due to the reflection on a surface (part of an obstacle). This means that that point should be assumed to be an obstacle and, consequently, its associated cell in the grid should be labeled as ***occupied***.

There is more information, implicitly included in an individual LiDAR range measurement. The space between the LiDAR sensor and the reflecting point must be clear of obstacles in order to allow the beam to reach the measured reflecting point. We can assume that all the intermediate points, which belong to the segment between the LiDAR's laser emitter and the measured point, are ***clear***. This assumption allows us to perform an ***erase*** operation on all the points that belong to that segment. This concept is easy in its definition; however certain considerations must be taken into account when it is implemented:

1) The sensor's beam is not a line: it is an angular sector (e.g. laser beam is 0.5 degrees wide (*)).
2) The sensor's beam may invade certain cells just partially.
3) Position and orientation of sensor (e.g. robot pose) are usually polluted by uncertainty. The measurement of the laser is also polluted by noise.

Due to issue #3 the implementations of OG of this type are usually defined in a stochastic fashion. However, stochastic processes are not part of MTRN4010, so we operate in a deterministic way, but working in a conservative way (to be discussed in class), for reducing the probability of having "False negatives".
We assume the localization is accurate enough, and that the scanning sensor is accurate enough as well.

(*) This beam's width is for the usual mode of operation of the laser scanners we use in our UGV). However, in these values may change, depending on the horizontal pixel resolution we use.

Probabilistic Approach

The concept of Occupancy Grid is well implemented if the values of the cells are interpreted and modeled as values of the probability of a cell about being occupied. In the Robotics research community this probabilistic approach is usually applied. The reason of its popularity is due to the uncertainty which is usually present in the sources of information, i.e. noise in measurements and uncertainty in the assumed models for the systems we control and use. Although the probabilistic approach is not covered in MTRN4010, we will apply some deterministic approximation in order to deal with the uncertainty present in our sensors.

Sources of Error in our Mapping Process

We synthesize a map by fusion of information, which is obtained from the robot's sensors. Those are measurements taken at different times and from different places (i.e. position and orientation of the robot). Those measurements are polluted by noise. If we assume that the measurements are perfect (free of noise) we may produce inconsistent results (e.g. wrong maps); consequently, the robot would be exposed to take the wrong decisions.
The onboard scanner introduces errors in its range measurements. You can assume that these errors are bounded, e.g. to be less than ~3cm (MTRN401-2022). The error in the angle can be considered to be < 0.5 degrees.
An additional source of error, which is relevant as well, is the uncertainty in the provided estimates of the platform's pose (position and attitude). The error we expect in our indoor localization system will be in order of cm (5cm) and the error in the heading of about 2 degrees.

Playing Safe: Accepting False Positives

When we try to map the terrain's risks (obstacles), in order to perform a safe operation of the robot, we prefer being conservative than being overconfident. I.e. it is safer if we avoid obstacles that do not actually exist than colliding with obstacles whose existence is ignored by us.
Due to errors and noise, the ***erase*** operation in the OG can be overconfident in erasing cells which are actually "occupied". This can be the consequence of uncertainty, which is present in the measurements, what may make us to clear cells in regions where we should not be clearing.
In order to deal with this issue, and to reduce its effects, we can use the following erasing policy.
When we infer that a cell is occupied, we give that cell a value (asap!). This value would be interpreted as a count (so, each cell is described by a "counter"). If the cell is later involved in an *erase* operation its value will then be decremented (partially). After a sequence of erasing operations, it will eventually reach the value equivalent to "clear". This means that we require many measurements to confirm that some previously occupied cell would be finally assumed to be clear (we are "playing safe").

Assume the following convention:

Value = 0 :	Unknown
Value = 1 :	Clear
Value > 1 :	Occupied

In a ***set*** operation we set the cell value to be, for instance, =20., ***Value(i,j) = 20***

In an erase action we perform the following operation:

```
if Value( i,j ) >1,        Value( i,j )= Value( i,j )-1.
else
        if Value( i , j )==0,  Value( i , j )=1 ;  end
end
```

or equivalently

```
if Value( i,j ) >1,
        Value( i,j )= Value( i,j )-1.
else,
        Value( i , j )=1
end
```

or

Value(i,j) = max (Value(i,j)-1 , 1) ;

It means, in this example, that after 19 ***erase*** operations the previously occupied cell will become to be classified as ***clear***. This policy means that we need many instances of observation to finally accept that a previously detected obstacle is considered non-existent anymore.
Note the particular case in which the cell was previously considered to be ***unknown***; in such a case we immediately assume the cell to be ***clear***.
This policy would eventually produce some **False Positives**, because even for an obstacle that does really disappear (e.g. being physically removed), its image would still be in the map for certain time until it is finally removed (from our belief).

Note: The approach, discussed in this document, covers the case where the robot's scanning device is a laser scanner or similar sensor, where the plane of scanning is horizontal.
Under this approach (using a horizontal scanner), we are assuming that all the obstacles are "prismatic" (vertical prisms) objects, e.g. their sections are identical for any intersecting horizontal plane (such as the cases of vertical poles, trunks, walls, etc.) We also assume that the platform (and its scanning device) does operate in a perfect 2D context (pitch(t) = roll(t) = 0 for all t, z(t) = constant).
Alternatively, we can use the full 3D imagery provided by our robots' sensors. In such a case we would need to process the 3D images for inferring terrain depressions, obstacles, etc. We will discuss about this concept in the lecture. Under this full 3D approach, our method for inferring if a cell is clear or occupied would be different. Note that even if the robot does operate in a context that is not perfectly 2D, we still use a 2D occupancy grid, in order to simplify the mapping and planning problems.
We end this lecture, running a basic mapping process using an OG.