UNIVERSIDADE D
COIMBRA

MESTRADO EM ENGENHARIA E DE COMPUTADORES

Relatório de Sistema Robóticos Autónomos

Construção de Mapas usando Modelação Bayesiana em Grelhas de Ocupação

Duarte de Sousa Cruz, 2017264087
João Pedro Chaves Castilho, 2017263424

Universidade de Coimbra
08 de Maio de 2022

1 Introdução

Neste trabalho era-nos pedido para incorporarmos nos algoritmos de navegação (VFF/VFH), desenvolvidos no trabalho 2, uma solução de construção de mapas usando modelação bayesiana em grelhas de ocupação. No ambiente de simulação deste trabalho, os algoritmos de navegação funcionam sem qualquer conhecimento prévio do mapa, este é construido ao longo da execução do movimento do robô.

No fim do movimento do robô, é gerado um mapa de ocupação probabilístico. A resolução do mapa, ou seja, quantos centímetros representa cada célula, pode ser definido no início da execução do nosso simulador.

2 Desenvolvimento

2.1 Leitura do Lidar

O Lidar do TurtleBot é simulado no Gazebo, portanto para obtermos uma leitura do sensor Lidar usamos a função `[scan, lddata, ldtimestamp] = tbot.readLidar();`. A variável `lddata` é composta por pelas distâncias observadas pelo Lidar e os ângulos respetivos. Através desta variável conseguimos obter as leituras válidas, ou seja, leituras que efetivamente retornaram uma distância, e leituras que são inválidas, ou seja, que retornam `Inf`. As leituras válidas são usadas para obter as coordenadas xy, no sistema de coordenadas do Lidar, dos obstáculos que o lidar detetou. Por sua vez, as leituras inválidas são usadas para obter os ângulos do Lidar em que essas leituras foram efetuadas.

Uma vez que as coordenadas dos obstáculos obtidas através das leituras do Lidar estão referentes ao sistema de coordenadas do Lidar, é necessário transformar estas coordenadas para o sistema referencial do mundo. Para isso usamos a seguinte matriz de transformada:

$$T_R^W = \begin{bmatrix} \cos\theta & -\sin\theta & x - 0.0305 \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

onde x é o x do robô no referencial do mundo, y é o y do robô no referencial do mundo e θ é a orientação do robô. Esta transformada pode ser usada também para compensar a diferença entre as leituras do Lidar e a leitura da posição do robô. Este passo pode ser relevante quando estas duas leituras têm uma diferença temporal significativa. No nosso caso reparámos que esta diferença podia por vezes ser igual a 0.03 segundos, portanto decidimos fazer esta correção.

A transformada com a correção temporal fica então:

$$T_R^W = \begin{bmatrix} \cos\theta & -\sin\theta & x - 0.0305 - (v_{robot}\tau\cos\theta) \\ \sin\theta & \cos\theta & y - (v_{robot}\tau\sin\theta) \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

onde v_{robot} é a velocidade do robô e τ é a diferença temporal entre as leituras.

Após termos as coordenadas dos obstáculos, usamos a função `world2grid` para obtermos as coordenadas dos obstáculos no mapa de ocupação discretizado.

Todo o código usado para, com as leituras do Lidar obter as coordenadas dos obstáculos detetados e os ângulos livres, pode ser analisado a seguir:

```
[scan, lddata, ldtimestamp] = tbot.readLidar();
[x,y,theta,timestamp] = tbot.readPose();
% compute the time diference between the pose reading and laser scan (in seconds)
timeDiff =  timestamp - ldtimestamp;

% Get in range lidar data indexs (valid data - obstacles).
[vidx] = tbot.getInRangeLidarDataIdx(lddata);
% load valid X/Y cartesian lidar readings
xydata = lddata.Cartesian(vidx,:);

% Get out of range lidar data indexs (too near or too far readings).
[nidx] = tbot.getOutRangeLidarDataIdx(lddata);
% load obstacle free orientations (angles)
freeAng = lddata.Angles(nidx);

[robotX, robotY] = world2grid(x,y,map_cell_size);
```

```
transformMatrix = [cos(theta) -sin(theta) x-0.0305-(v*timeDiff*cos(theta));
                   sin(theta) cos(theta) y-(v*timeDiff*sin(theta));
                   0 0 1];

xyWorld = transformMatrix*[xydata';ones(1,size(xydata,1))];
xyWorld = xyWorld(1:2,:);
[xCell, yCell] = world2grid(xyWorld(1,:),xyWorld(2,:),map_cell_size);
```

2.2 Construção do mapa

Depois de termos as células da grid que correspondem a obstáculos detetados e os ângulos livres do Lidar, conseguimos começar a construir o nosso mapa probabilístico.

O primeiro passo é transformarmos os ângulos livres do Lidar em células na grid que correspondem a células livres. Para isso, definimos uma distancia limite do feixe do Lidar, no nosso caso definimos 1.5 metros. As células são obtidas através de:

```
freeXY = [1.5*cos(free_angles) 1.5*sin(free_angles)];

xyWorld = transformMatrix*[freeXY';ones(1,size(freeXY,1))];
xyWorld = xyWorld(1:2,:);

[freeAX, freeAY] = world2grid(xyWorld(1,:),xyWorld(2,:),map_size);
```

Para atualizarmos os valores das células de ocupação, temos de passar o mapa probabilístico para um mapa de *log odds*, usando a seguinte fórmula:

$$log\_map = \log\left(\frac{map_{xy}}{1 - map_{xy}}\right) \tag{3}$$

onde map_{xy} representa o mapa probabilístico com probabilidades de ocupação inicializado com probabilidade 0.5 em todas as células deste mapa.

Após termos o mapa representado em *log odds* podemos começar a atualizar as células como livres ou ocupadas. Para obtermos as coordenadas das células que estão entre a célula livre e a célula em que está o robô usamos o algoritmo de bresenham [1]. Este algoritmo não foi implementado por nós, fomos descarregar uma versão do algoritmo disponível na Matlab Exchange `https://www.mathworks.com/matlabcentral/fileexchange/28190-bresenham-optimized-for-matlab`. Para cada célula fornecida pelo algoritmo de bresenham é subtraído 0.65 na célula correspondente.

No caso das células que, através das leituras do Lidar foram detetadas como ocupadas, é novamente usado o algoritmo de bresenham para calcular as células que estão entre a célula ocupada e a célula em que o robô está presente. De seguida, a célula ocupada é marcada no mapa como ocupada, adicionando 0.65 na célula correspondente. As células obtidas com o bresenham são marcadas como livre no mapa, subtraído 0.65 nas células correspondentes.

A lógica de construção do mapa pode ser constatada no bloco de código a seguir:

```
log_map = log(map./(1-map));

% First update cells that are part of the free angles
for i = 1:size(freeAX,2)
    [lineAX, lineAY] = bresenham(robotPose(1),robotPose(2),freeAX(i),freeAY(i));
    lineAX = min(max(lineAX,1),map_size);
    lineAY  = min(max(lineAY,1),map_size);
    for j = 1:size(lineAX,1)
        % Update the free cells
        log_map(lineAY(j),lineAX(j)) = log_map(lineAY(j),lineAX(j)) - 0.65;
    end
end

cellX = min(max(cellX,1),map_size); % Obstacles cells X
cellY  = min(max(cellY,1),map_size);% Obstacles cells Y

% Update obstacle cells
```

```
for i= 1:size(cellX,2)
    % Update occupied cells
    log_map(cellY(i),cellX(i)) = log_map(cellY(i),cellX(i)) + 0.65;
    % Get free cells with bresenham algorithm

    [lineX, lineY] = bresenham(robotPose(1),robotPose(2),cellX(i),cellY(i));

    % Delete the last cell that corresponds to the obstacle cell
    freeCells = [lineX(1:end-1) lineY(1:end-1)];
    freeCells(:,1) = min(max(freeCells(:,1),1),map_size);
    freeCells(:,2)  = min(max(freeCells(:,2),1),map_size);
    for j = 1:size(freeCells,1)
        % Update the free cells
        log_map(freeCells(j,2),freeCells(j,1)) = log_map(freeCells(j,2),freeCells(j,1)) - 0.65;
    end
end
```

Depois de atualizarmos os valores das células livres e ocupadas, adicionamos o valor da probabilidade, em *log odds*, da probabilidade a priori l_0. Esta probabilidade é definida por célula do mapa e representa a nossa certeza inicial da ocupação de dadas células. Por exemplo, podemos definir um valor de $l_0 = 0.2$ nas extremidades do mapa e estas células vão atualizar os seus valores como células ocupadas mesmo sem qualquer leitura de Lidar correspondente.

Por fim na função `updateMap` é convertido o mapa de *log odds* para probabilidades e este mapa é retornado pela função e usado pelos algoritmos de navegação VFF/VFH.

Para efeitos de algoritmos de navegação, são consideradas como células ocupadas, todas as células que tenham uma probabilidade de ocupação superior a 0.75.

3 Resultados e Observações

Para obter os resultados implementados um método de escolher qual a resolução que queremos para o mapa. Nas seguintes figuras vemos o movimento do robô usando os algoritmos VFH e o respetivo mapa de probabilidades.

3.1 Mapa 400x400

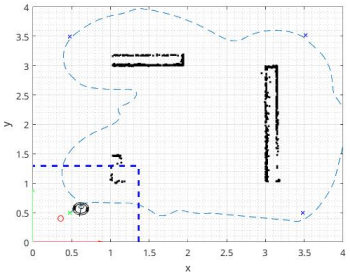

Figure 1: Mapa com obstáculos 400x400

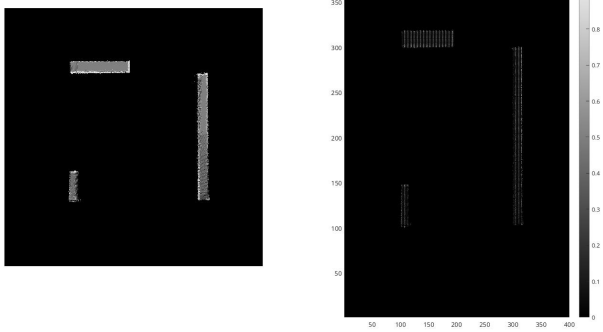

Figure 2: Mapa de probabilidades 400x400

3.2 Mapa 100x100

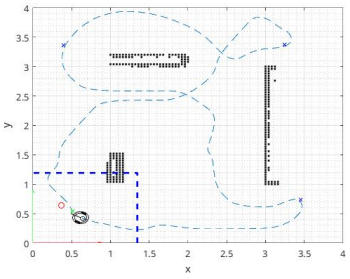

Figure 3: Mapa com obstáculos 100x100

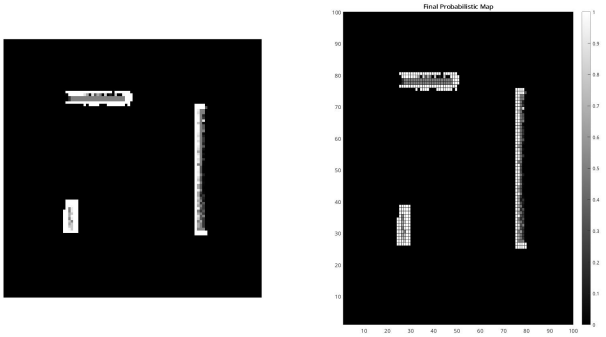

Figure 4: Mapa de probabilidades 100x100

3.3 Mapa 40x40

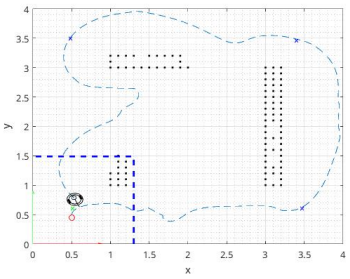

Figure 5: Mapa com obstáculos 40x40

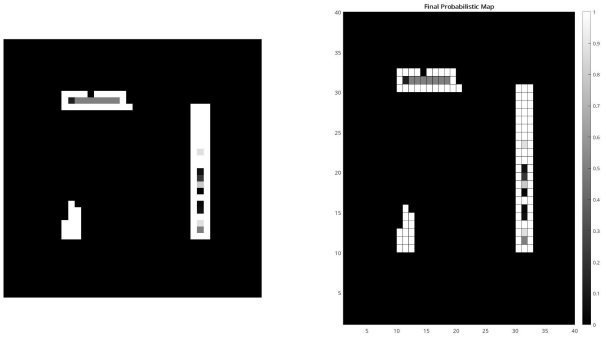

Figure 6: Mapa de probabilidades 40x40

Conseguimos observar que conseguimos implementar bem os modelos do lidar e diminuímos o desfasamento da simulação e o gazebo, por uma boa otimização do código.

References

[1] J. E. Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems Journal* 4.1 (1965), pp. 25–30. DOI: 10.1147/sj.41.0025.