Robótica Espacial
Aula prática nº 12

Planeamento Global de Caminhos em Robôs Móveis
Decomposição Poligonal e Grelhas de Ocupação

Vitor Santos

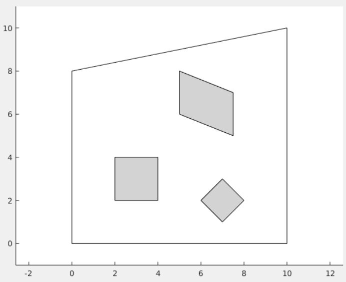
Universidade de Aveiro

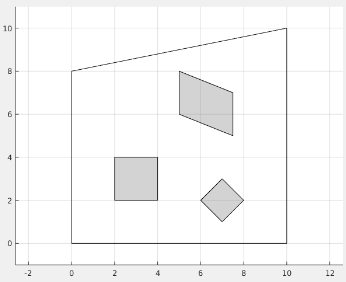
22 maio 2025

Sumário

Exercício 1 - Criar um mapa com obstáculos

- Criar um mapa limitado por um polígono
- Criar 3 obstáculos
- Usar a função `polyshape` que permite criar regiões mais complexas
- Completar o código abaixo (***) para representar a figura

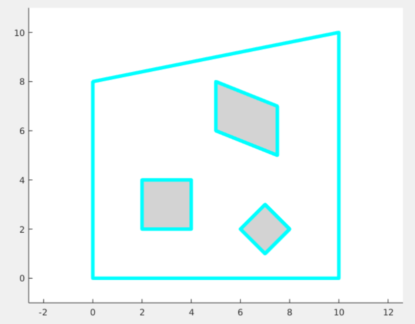





```
workspace = polyshape([0 10 10 0], [0 0 10 8]);
obs(1)=polyshape([2 4 4 2],     [2 2 4 4]);
obs(2)=polyshape([5 7.5 7.5 5], [6 5 7 8]);
obs(3)=polyshape([*** *** *** ***], [*** *** *** ***]);
obstacles = polyshape(); % initial empty list
for k=1:numel(obs)
    obstacles = union(obstacles,obs(k)); %join all obs
end
plot(workspace, 'FaceColor','none')
axis equal, hold on
plot(obstacles, 'FaceColor', [0.5 0.5 0.5])
```

Exercício 2 - Criar e representar o mapa do espaço livre

- O espaço livre é o espaço total menos as zonas ocupadas por obstáculos
- Pode ser construído pela subtração de todos os obstáculos ao espaço total
- A operação `subtract()` permite criar essas regiões poligonais mais complexas com "buracos".
- Completar o código abaixo (***)



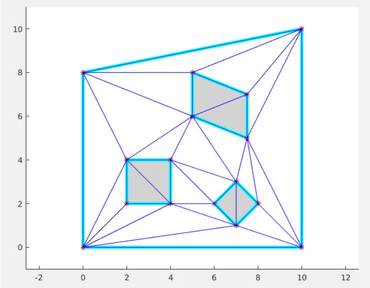

```
free_space = workspace;
for i = 1:length(***)
    free_space = subtract(free_space, obstacles(***));
end
plot(free_space,'FaceColor','none','EdgeColor', 'c','LineWidth',4)
```

Exercício 3 - Decomposição do espaço livre em triângulos

Triangulação de Delaunay

- Obter os vértices da fronteira do espaço livre (inclui os vértices dos buracos)
  - Cada sub-região poligonal está separada por uma linha de `NaN` na matriz dos vértices da região completa
- Calcular a triangulação de Delaunay e ilustrar os triângulos e os vértices

```
verts=free_space.Vertices;                 %all vertices
verts=verts(all(isfinite(verts),2),:);%removes NaN
verts=unique(verts, 'rows');               %no duplicates
plot(verts(***), verts(***),'*r')          %optional plot
dt = delaunayTriangulation(verts);         %the Delaunay
hDT=triplot(dt);                           %display it
```

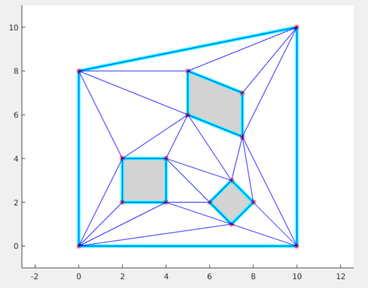


A estrutura dt tem dois campos:

```
           Points: O mesmo que os vértices
ConnectivityList: Lista triângulos com
                  3 vértices cada
```

Exercício 4 - Eliminar os triângulos inválidos

- Como a triangulação de Delaunay fez também a decomposição da região obstáculos ...
- ... isso deve ser retirado e assim obter a lista de triângulos válidos.
- Obtém-se uma lista lógica (0 ou 1) para todos os centros e os que caírem fora do espaço livre são a triângulos a eliminar na representação




```
% Obtain the full list of triangle centers
triCenters = dt.incenter;
% Obtain the list of the overlaping triangles
% 1-list of logical values whether inside or outside free_space
inside = isinterior(free_space, triCenters(***), triCenters(***));
% 2-With the list extract the valid triangles only
validTriangles = dt.ConnectivityList(***, :);
pause %wait for key press before deleting previous graph
hDT.delete    %delete previous plot to show next plot
%and plot the valid triangles only
triplot(validTriangles, dt.Points(:,1), dt.Points(:,2));
```

Exercício 5 - Criar a lista de adjacências de triângulos

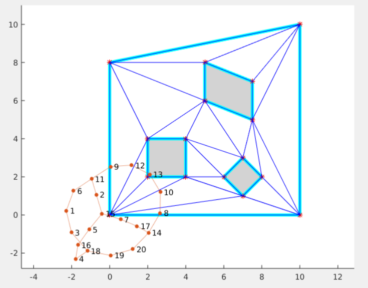
- Para cada triângulo válido testar com todos os outros triângulos válidos para verificar se têm um lado em comum.
- Cada triângulo é definido por três vértices.
- Basta verificar se a intersecção dos conjuntos de vértices têm exatamente dois em comum.
- Usar a função `intersect()` entre triângulos para esse efeito.
- Se o resultado tiver 2 elementos, atualizar a matriz de adjacências (nas duas direções)

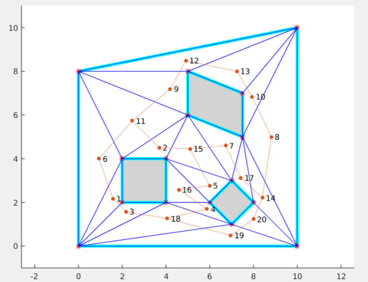
```
numTri = size(validTriangles,1);     % number of triangles
adj = zeros(numTri);                 % init adjacency matrix
% Two triangles are adjacent if they share an edge, i.e. 2 points
for i = 1:numTri
    for j = i+1:numTri
        shared = intersect(validTriangles(***,:), validTriangles(***,:));
        if numel(shared) == 2
            adj(i,j) = 1;            % from i to j
            adj(***) = 1;            % from j to i
        end
    end
end
```

Exercício 6 - Grafo da matriz de adjacência

- Obter o grafo da matriz de adjacência e representá-lo.
- Porém, antes é necessário obter os centróides dos triângulos válidos para representar o grafo corretamente sobre o mapa.
  - Sem isso, grafo não se alinha com o mapa, como se observa na figura de cima.

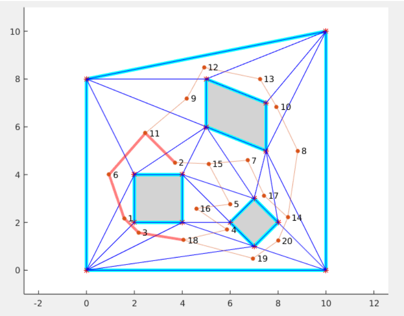
```
centroids = triCenters(inside, :);
g = graph(adj);
hG = plot(g, 'XData', ***, 'YData', ***);
```





Exercício 7 - Caminho com menos troços entre dois nós

- A matriz de adjacências só tem zeros e uns, ou seja, não reflete ainda o custo da distância real.
- O algoritmo de procura (Dijsktra) devolve o caminho com menos troços porque todos os troços têm custo 1.
- Em matlab a função designa-se `shortestpath`.
- Retorna o caminho (vetor de nós) e o seu comprimento.
- Sugere-se procurar o caminho entre o nó 2 e o nó 18, mas quaisquer outros são possíveis!
- Pode-se usar a função `highlight` para destacar o caminho mais curto encontrado.




```
startNode=2;
endNode=18;
[path1, pathLength] = shortestpath(g, ***, ***)
highlight(hG,path1,'EdgeColor', ***,'LineWidth', ***)
```

Exercício 8 - Associar distâncias ao grafo e recalcular trajeto

- Obter as distâncias euclidianas entre todos os nós
- Criar uma matriz simétrica com esses valores
- Isso pode ser feito com ciclos como estes:

```
D=zeros(height(centroids));
for i=1:height(centroids)
    for j=1:height(centroids)
        D(i,j)=norm(centroids(i,:)-centroids(j,:));
    end
end
```

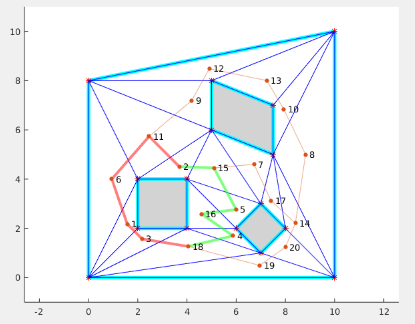
- Mas há soluções mais compactas

```
% D(i,j) Euclidean distance between node i and j
D = squareform(pdist(centroids));
```

- Depois basta combinar as matrizes desta forma:

```
% Keep distances only for connected node pairs
adjc = D .* adj; g2=graph(adjc);
```

- O novo cálculo do exercício anterior resulta num caminho diferente. É o mesmo numero de segmentos mas este é ligeiramente mais curto: 8.19 vs. 8.72!

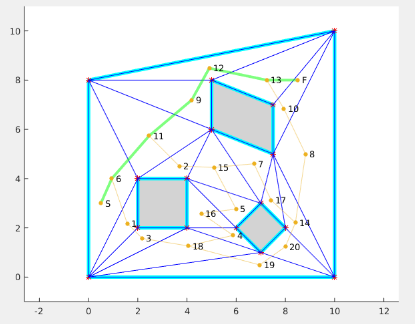


O caminho a verde resulta da aplicação do mesmo algoritmo mas agora com uma matriz de adjacências com custos de distância.

```
Caminho Inicial:2   11    6    1    3   18
Caminho Novo:    2   15    5   16    4   18
```

Exercício 9 - Adicionar pontos de partida e chegada no grafo

- Considerar os pontos de partida e chegada como:
  - SS=[0.5 3]; FF=[8.5 8];
- Dar nomes aos nós existentes de 1 até ao último
- Adicionar estes novos nós (addnode)
- Adicionar novos arcos (edges) que ligam estes novos nós aos nós mais próximos existentes.




```
g2.Nodes.Name = string(1:numnodes(g2))';
g2 = addnode(g2, 'S');
g2 = addnode(g2, 'F');
*** % calculate IDs of closest nodes to 'S' and to 'F'
g2 = addedge(g2, ""+nearStartID, 'S', norm(centroids(nearStartID,:) - SS));
g2 = addedge(g2, ""+nearEndID, 'F', norm(centroids(nearEndID,:) - FF));
hG2=plot(g2, ***)
newStartNode='S'; newEndNode='F';
[path3, pathLength] = shortestpath(g2, newStartNode, newEndNode)
```

Exercício 10 - Criar uma grelha de ocupação para o mesmo espaço

- Usar o mesmo `workspace` e os mesmos obstáculos
- Criar uma grelha até aos limite de `workspace` e definir uma resolução para essa grelha

```
%workspace =                  %defined earlier
%obstacles =                  %defined earlier

res = 0.2;                    %grid resolution
ext=max(workspace.Vertices);%maximum limits of workspace
x = 0:res:ext(1);             %span of x coordinate
y = 0:res:ext(2);             %span of y coordinate
rows = numel(y);              %number of rows of grid
cols = numel(x);              %number of columns of grid
occGrid = false(***, ***);    %initial empty occupancy grid (0=free, 1=occupied)
```

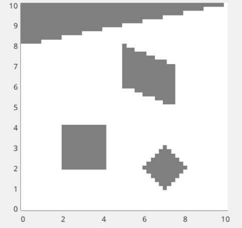
Exercício 11 - Preencher a grelha de ocupação

- Testar todas as células (pelas suas coordenadas métricas) se estão no `workspace` ou se são obstáculos

```
[X, Y] = meshgrid(x, y);          %All combinations of x and y coordinates
for i = 1:rows                    %row index
    for j = 1:cols                %column index
        pt = [X(i,j), Y(i,j)];    %actual metric coordinates of grid cell (i,j)
        if isinterior(workspace, pt) && ~isinterior(obstacles, pt)
            occGrid(i,j) = ***;   %empty (free space)
        else
            occGrid(i,j) = ***;   %occupied (workspace limits, obstacles)
        end
    end
end
```

- Representar a grelha numa nova janela usando branco e cinzento a 50% para o espaço livre e ocupado respetivamente

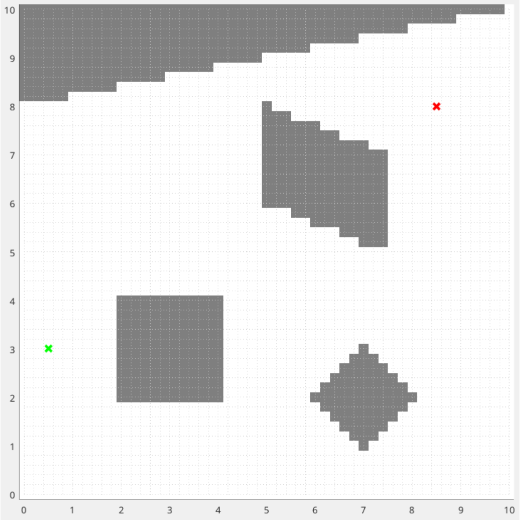
```
figure
hold on; axis equal tight;
colormap([1 1 1; 0.5 0.5 0.5]);%white and gray
imagesc(x, y, occGrid);         %display occGrid as an image
```



Exercício 12 - Linhas na grelha e pontos de partida e chegada

- Definir pontos de partida e chegada

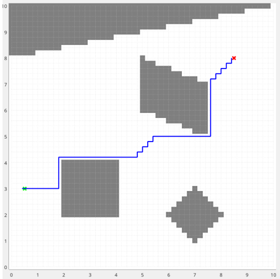
```
startPt=[0.5 3]; % start position (=SS)
goalPt=[8.5 8];  % finish position (=FF)
%cell IDs in grid for start and goal
[~,startRow]=min(abs(y-startPt(2)));
[~,startCol]=min(abs(x-startPt(1)));
[~,goalRow] =min(abs(y-goalPt(2)));
[~,goalCol] =min(abs(x-goalPt(1)));
% Add grid lines for better cell separation
for xi = x
    line([xi xi], [min(y) max(y)], ...
    'Color', [0.8 0.8 0.8], 'LineStyle', ':');
end
for yi = y
    line([min(x) max(x)], [yi yi],...
    'Color', [0.8 0.8 0.8], 'LineStyle', ':');
end
plot(***,***,'gx',...
    'MarkerSize',8,'LineWidth',2);
plot(***, ***, 'rx',...
    'MarkerSize',8,'LineWidth',2);
```

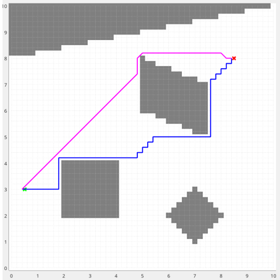


Exercício 13 - Calcular e representar os caminhos com A*

- Usar as funções fornecidas `a_star_search()` e `a_star_search_8conn()` para calcular os caminhos entre a chegada e a partida.
- O primeiro não planeia ao logo da diagonal gerando por isso caminhos mais longos do que o segundo.

```
path4 = a_star_search(occGrid, ...
        [startRow, startCol], [goalRow, goalCol]);
path8 = a_star_search_8conn(occGrid, ...
        [startRow, startCol], [goalRow, goalCol]);
% Draw paths
pathXY = [x(path4(:,2))', y(path4(:,1))'];
plot(pathXY(:,1), pathXY(:,2),...
    'b-', 'LineWidth', 2);
pathXY = [x(path8(:,2))', y(path8(:,1))'];
plot(pathXY(:,1), pathXY(:,2),...
    'm-', 'LineWidth', 2);
```





Exercício 14 - Comparação dos comprimentos dos caminhos

- Verificar que os comprimentos dos caminhos pelos 3 métodos para este exemplo em particular são dados por:
  - Decomposição triangular: 10.74
  - A* sem diagonais: 13
  - A* com diagonais: 10.59
- Rwcorde-se que os deslocamentos elementares na diagonal medem $\sqrt{2} \approx 1.4$ e os horizontais e verticais medem 1 unidade.
- Uma solução expedita em matlab para o cálculo pode passar por obter as diferenças entre pontos sucessivos (vetores) e somar a norma de todos esses vetores.