VISIÓ PER COMPUTADOR: EXERCICI 5

Preàmbul: No he aconseguit completar les funcionalitats esperades del segon punt de la pràctica. Ho he intentat fer utilitzant només operacions morfològiques, però no he estat capaç de resoldre-ho per més hores que li hagi posat (moltes més que en pràctiques anteriors...).

En aquest exercici dissenyarem un programa que a partir d'un laberint amb dos possibles punts d'inici, detecti explícitament mitjançant operacions morfològiques quin dels dos punts inicials es troba més a prop del centre del laberint, tot mostrant el camí en qüestió en color groc i coneixent el nombre de píxels del camí.

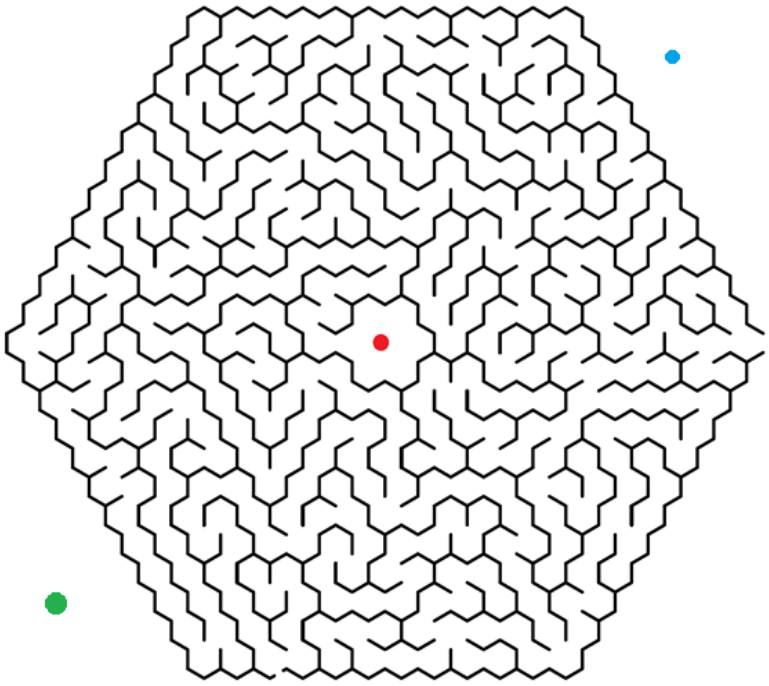

Figura 1: Imatge del laberint a partir del qual s'ha dissenyat el programa.

En primer lloc, llegim la imatge proporcionada i la fragmentem en tres de diferents, cadascuna per un canal RGB:

```
I = imread('Laberint.png');
Red = I(:,:,1);
Green = I(:,:,2);
Blue = I(:,:,3);
```

Seguidament, creem diverses màscares per identificar cadascun dels tres punts (els dos inicials i el de destí), a més dels murs del laberint com a tal:

```
% Punt al centre del laberint:
DetectaVermell = (Red > 200) & (Green < 50) & (Blue < 50);

% Punts d'inici:
DetectaVerd = (Red < 50) & (Green > 150) & (Blue < 150);
DetectaBlau = (Red < 50) & (Green < 200) & (Blue > 150);

% Murs del laberint:
Laberint = (Red < 100) & (Green < 100) & (Blue < 100);
```

Figura 2: Imatges dels tres punts: a l'esquerra de tot el punt central i els dos següents els inicials.

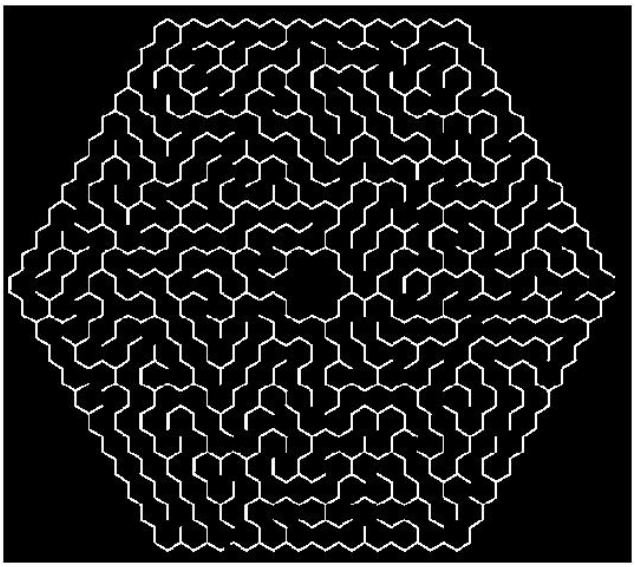

Figura 3: Imatge dels murs del laberint.

Tot seguit, definim la funció *troba_distància*:

```
function [dist, punt_init, cami] = troba_distancia (punt_init, punt_fi,
laberint)

      % Per conèixer la mida de la matriu:
      tam = size(punt_init);

      % Matriu que contindrà el camí seguit per la dilatació de forma expansiva:
      cami = zeros(tam(1), tam(2), 'uint16');

      % Saber si coincideixen els píxels del punt inicial i final:
      coincidencia = (punt_init == 1) & (punt_fi == 1);

      % Saber nombre de píxels entre els punts final i inicial:
      dist = 0;

      % SE per ampliar el punt inicial de manera circular, un píxel per
      % iteració:
      SED = strel('disk', 1);

      % Si tota la màscara de coincidències (sum) és zero, vol dir que encara no
      % han coincidit els punts final i inicial:
      while sum(coincidencia, 'all') == 0

            % Ampliació del punt inicial, dilatant-lo circularment:
            punt_init = imdilate(punt_init, SED);

            % Eliminant del punt inicial ampliat les parets dels laberint per
```

```
            % evitar que segueixi creixent a través d'elles, fent una resta
            % lògica, ja que així en futurs passos permet utilitzar certes
            % funcions de modificacions morfològiques:
            punt_init = xor(punt_init, laberint) & punt_init;

            % Incrementar la distància un píxel:
            dist = dist + 1;

            % Actualitzar la matriu d'expansió:
            cami = cami + uint16(punt_init);

            % Actualitzar la màscara de coincidència:
            coincidencia = (punt_init == 1) & (punt_fi == 1);
      end
end
```

A continuació, les dues crides a la funció exposada per trobar els camins de tots dos punts inicials fins al centre del laberint:

```
[dist_blau,    DetectaBlau,    cami_blau]    =    troba_distancia(DetectaBlau,
DetectaVermell, Laberint);
[dist_verd, DetectaVerd, cami_verd] = troba_distancia(DetectaVerd,
DetectaVermell, Laberint);
```

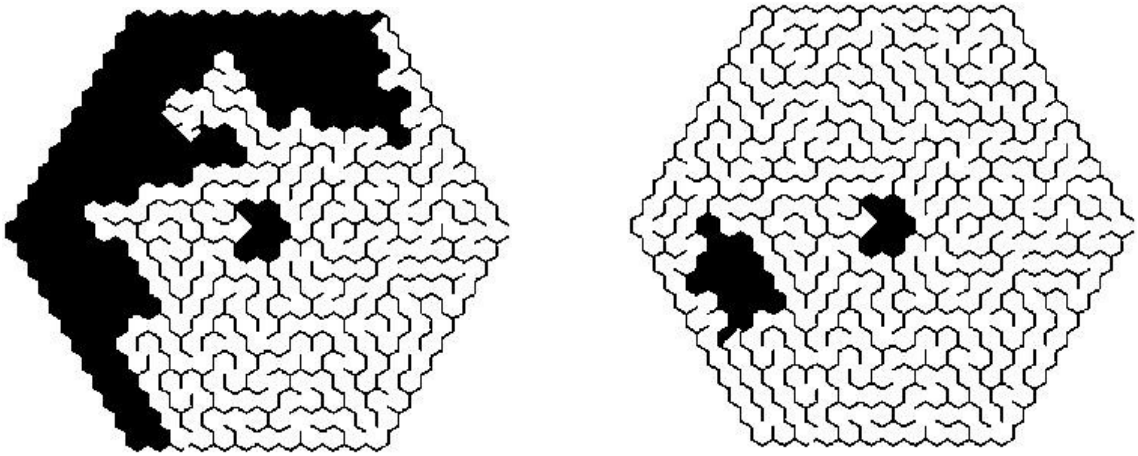

Figura 4: A l'esquerra el resultat de *troba_distància* del punt verd, i a la dreta del punt blau.

La distància del punt blau al vermell (centre) és de 1846 píxels mentre que la del punt verd al vermell és d'un total de 1464 píxels. D'aquesta manera deduïm que el punt verd és més a prop del centre que el blau.

Posteriorment, volem reduir tots els possibles camins a una única línia de píxels:

```
skel_verd = DetectaVerd;
skel_verd = bwskel(skel_verd);

skel_blau = DetectaBlau;
skel_blau = bwskel(skel_blau);
```

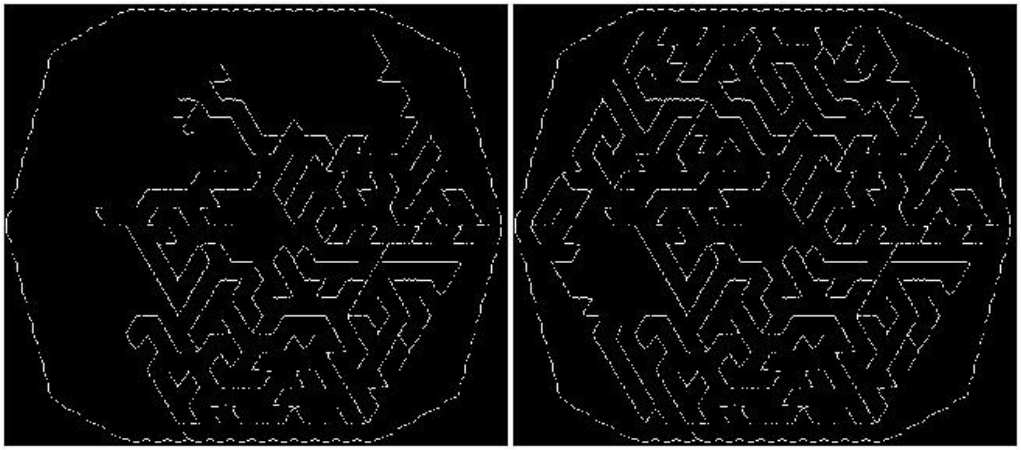

Figura 5: A l'esquerra el resultat d'esqueletitzar els possibles camins del punt verd, i a la dreta el resultat fer fer-ho pel punt blau.

Potseriorment, superposem el camí a l'esquelet de manera que només continguin valors vàlids (els de la matriu d'expansió) els punts de l'esquelet. Els punts resultants que valen zero es passen a que valguin infinit per tal de trobar el primer píxel del camí de tornada del punt vermell al punt exterior en qüestió.

La idea és trobar aquest primer píxel mitjançant el còmput del mínim valor. El procés següent és anàleg pel punt blau:

```
cami_skel_verd = cami_verd .* uint16(skel_verd);
cami_skel_verd(cami_skel_verd == 0) = inf;
[min_tmp, min_x] = min(cami_skel_verd);
[min_tmp, min_y] = min(min_tmp);
min_x = min_x(min_y);
```

En no ser capaç de trobar correctament el punt d'origen del camí de retorn del punt vermell cap al punt exterior, l'algoritme de tornada plantejat a continuació no troba el camí desitjat:

```
tamany = size(I);
cami_curt = zeros(tamany(1), tamany(2), 'uint16');

i = min_x;
j = min_y;
cami_curt(i, j) = 1;

while cami_curt(i,j) & punt_verd == 0
      if isequal(cami_skel_verd(i + 1, j), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i + 1, j), (cami_skel_verd(i,j))) % Baix
      i = i + 1;
      elseif isequal(cami_skel_verd(i+1, j-1), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i+1, j-1), (cami_skel_verd(i,j))) % Baix-esq
            i = i + 1;
            j = j - 1;
      elseif isequal(cami_skel_verd(i, j-1), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i, j-1), (cami_skel_verd(i,j))) % Esq
            j = j - 1;
      elseif isequal(cami_skel_verd(i-1, j-1), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i-1, j-1), (cami_skel_verd(i,j))) % Dalt-esq
            i = i - 1;
            j = j - 1;
      elseif isequal(cami_skel_verd(i-1, j), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i-1, j), (cami_skel_verd(i,j))) % Dalt
            i = i - 1;
      elseif isequal(cami_skel_verd(i-1, j+1), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i-1, j+1), (cami_skel_verd(i,j))) % Dreta-dalt
            i = i - 1;
            j = j + 1;
```

```
      elseif isequal(cami_skel_verd(i+1, j+1), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i+1, j+1), (cami_skel_verd(i,j) ))% Baix-dreta
            i = i + 1;
            j = j + 1;
      elseif isequal(cami_skel_verd(i, j+1), (cami_skel_verd(i,j) + 1)) ||
      isequal(cami_skel_verd(i, j+1), (cami_skel_verd(i,j))) % Dreta
            j = j + 1;
      end
      cami_curt(i, j) = 1;
end
```

En primer lloc, es crea una matriu de zeros per anotar amb 1s el camí de tornada. En segon lloc, s'inicialitzen les variables *i* i *j,* amb el punt d'origen del camí de tornada, i la matriu amb un 1 en aquell punt. Després la condició de parada del bucle és que el camí arribi a qualsevol dels píxels del punt exterior. En aquest bucle, es navega comprovant el valor dels veïns del punt, es comproven totes vuit direccions limítrofes a la recerca d'un punt que tingui el mateix valor o una unitat superior a l'actual.