

Analyse d'Image

Transformée de Hough

Axel Bruyere, Dorian Fabregue
4ETI-IMI

7 mai 2021

- Objectifs –**
- Implémentation de la transformée de Hough pour la détection de droites sur une image.
 - Utilisation de la transformée de Hough généralisée pour la détection de caractères brailles sur un document scanné.

1 Contexte

Le but de ce TP est d'implémenter sous Matlab des méthodes de reconnaissance de droites et de cercles utilisant la transformée de Hough. L'objet de ce rapport sera donc des algorithmes les mettant en oeuvre.

2 Détection de droites

2.1 Introduction

La détection de droites est une méthode de segmentation particulière. Contrairement aux méthodes de segmentation vues jusqu'à maintenant (LPE et Snakes), la détection de droites ne cherche pas à identifier les contours des objets de l'image mais de réussir à en déterminer les droites les plus caractéristiques.

Cette méthode sera implémentée sur l'image *buildings.png* afin d'identifier ses structures principales.

2.2 Méthode

Image de contours

Afin de déterminer les droites représentatives des bords des objets de l'image *buildings.png*, il est nécessaire de réduire la quantité d'informations de l'image afin de ne prendre en compte que les contours de ces objets. Il est possible pour cela d'utiliser le gradient de l'image (détection des variations d'intensité) et d'ensuite le binariser pour ne garder que les pixels des contours.

Afin d'obtenir un résultat satisfaisant, nous utiliserons simplement la fonction `edge` sous Matlab.

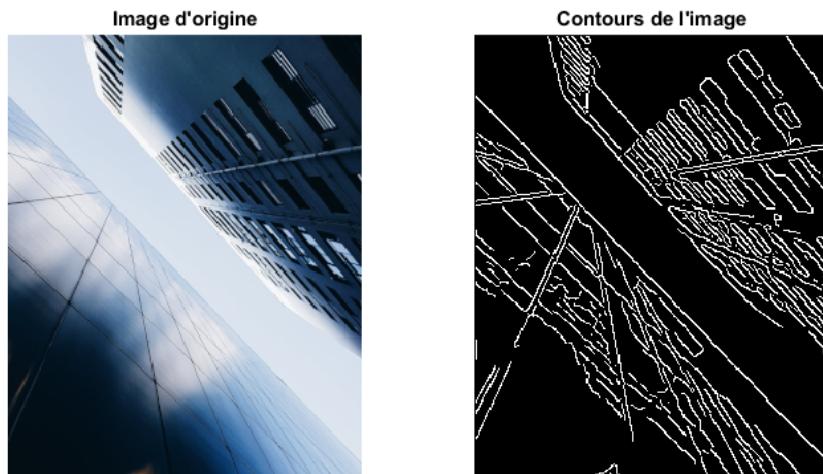


FIGURE 1: Image *buildings.png* et l'image binaire de ses contours

Changement de repère

La transformée de Hough repose sur un changement de repère : toutes les droites passant par un point du plan image $\mathcal{P}(x, y)$ peuvent être représentées dans une unique droite du plan de paramètres

$$\mathcal{H}(\theta, \rho) \text{ tels que : } \begin{cases} x = \rho \cos \theta \\ y = \rho \sin \theta \end{cases}$$

Dans le plan $\mathcal{H}(\theta, \rho)$, toutes les droites passant par un point $P_i(x_i, y_i)$ sont représentées par une courbe $C_i : \rho = x_i \cos \theta + y_i \sin \theta$

Algorithme de la transformée de Hough

L'algorithme de la transformée de Hough génère une matrice d'accumulation H qui s'incrémente au passage de chaque courbe C_i des points blancs $P_i(x_i, y_i)$ de l'image de contours.

Dans le cas d'une image des contours de taille $N_1 \times N_2$, nous définissons les paramètres $\theta \in [0, \pi[$ et $\rho \in [-\rho_{max}, \rho_{max}]$ (avec $\rho_{max} = \sqrt{(N_1 - 1)^2 + (N_2 - 1)^2}$) discrétilisés arbitrairement par leurs intervalles respectifs $d\theta$ et $d\rho$.

Une fois la matrice H obtenue, nous cherchons à extraire les maxima locaux afin de pouvoir déterminer les droites correspondantes de coordonnées $(\hat{\theta}, \hat{\rho})$.

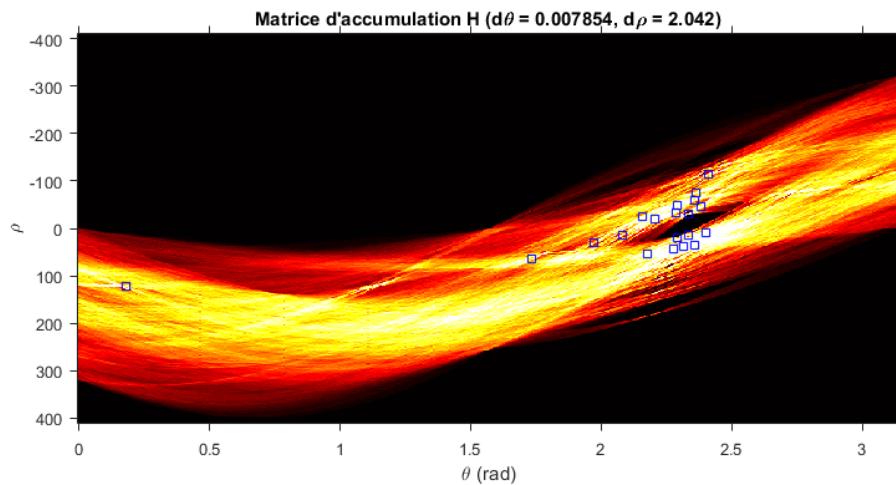


FIGURE 2: Matrice d'accumulation H et positions de 20 maxima locaux

Choix des intervalles $d\theta$ et $d\rho$

Chaque cellule de la matrice H couvre un intervalle $d\rho \times d\theta$. Si $(d\rho, d\theta)$ sont trop élevés, alors H perd en précision. Cependant si $(d\rho, d\theta)$ sont trop faibles, les vecteurs `theta` et `rho` sont de taille trop importante par rapport aux dimensions de l'image `buildings.png`. La précision de H n'augmente pas et son temps de calcul oui. Nous choisirons

Détermination des maxima locaux

Plusieurs méthodes sont possibles : le seuillage de H et l'identification des n plus grandes valeurs de H ne permettent pas de déterminer précisément ces maxima locaux. En utilisant un seuillage relativement faible, nous allons utiliser la fonction `islocalmax()` pour déterminer plus précisément ces maxima locaux.

Calcul des points extrémaux de chaque droite

Connaissant désormais les différentes couples de données $(\hat{\theta}, \hat{\rho})$, nous calculons les point extrémaux $P_1(x_1, y_1)$ et $P_2(x_2, y_2)$ de chaque droite correspondante :

$$\text{si } x_1 \neq x_2, \text{ alors } \begin{cases} x_1 = 0 \\ y_1 = \frac{\hat{\rho}}{\sin \hat{\theta}} \\ x_2 = N_2 \\ y_2 = \frac{\hat{\rho} - N_2 \cos \hat{\theta}}{\sin \hat{\theta}} \end{cases}, \quad \text{si } x_1 = x_2, \text{ alors } \begin{cases} x_1 = \frac{\hat{\rho}}{\cos \hat{\theta}} \\ y_1 = 0 \\ x_2 = \frac{\hat{\rho}}{\cos \hat{\theta}} \\ y_2 = N_1 \end{cases}$$

3 Détection de caractères braille

3.1 Introduction

Dans cette partie, notre objectif sera d'implémenter un pré-traitement d'images sur lesquelles sont visibles des caractères braille. Les images n'étant pas de très bonne qualité, il s'agit alors de mettre en oeuvre différents outils, notamment de morphologie mathématique, afin de faire ressortir les points qui nous intéressent. Cette opération se fait dans une optique de pouvoir par la suite lire les caractères, c'est pourquoi il est tout d'abord nécessaire de les marquer. Voici les deux images sur lesquelles nous allons travailler : Pour la première image, nous pouvons voir que les

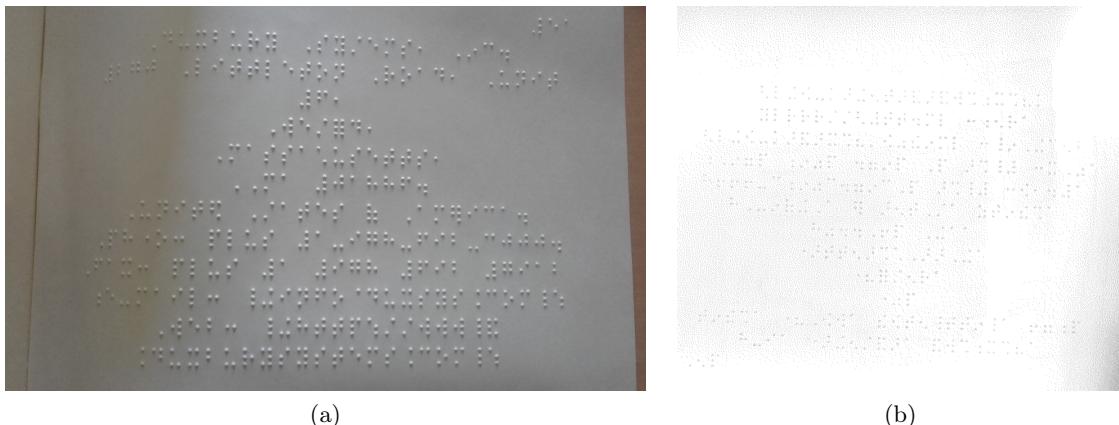


FIGURE 3: Images à traiter

difficultés majeures résident dans un léger manque de contraste, ainsi que dans l'ombre à gauche de la photo. Pour la seconde, la mauvaise qualité de l'image, ainsi qu'un gros manque de contraste nous compliqueront la tâche.

3.2 Méthode

Vous pouvez retrouver les codes complets en annexes

3.2.1 Première image

Nous commencerons naturellement par récupérer notre image et la passer en niveaux de gris. Il s'agit maintenant de la traiter, afin d'isoler les éléments qui nous intéressent, et diminuer l'impact de l'ombre, à gauche de l'image. Pour cela, nous procéderons à une transformation *top-hat*, puis seuillons l'image obtenue afin de ne garder que les pixels d'intérêt. Il ne nous reste donc plus qu'à chercher les cercles, à l'aide de la transformée de *Hough*, en utilisant la fonction fournie par Matlab une fois cela fait, nous pouvons afficher notre image et les points calculés.

3.2.2 Seconde image

Pour cette image, la tâche est plus complexe : sa mauvaise qualité entrave grandement le calcul. Il faut alors y appliquer un traitement plus lourd. Nous commençons par filtrer notre image, à l'aide d'un filtre gaussien, afin d'éliminer une partie du bruit. Ensuite, nous la seuillons à deux reprises afin de séparer clairement nos points du fond. Le problème est alors qu'une partie du bruit restant a été conservée lors du seuillage. Nous appliquons alors ensuite deux ouvertures successives, avec deux éléments structurants différents, puis enfin une dilatation pour rendre les points plus visibles. Il suffit alors de calculer la transformée de *Hough* de notre résultat.

3.3 Résultats

3.3.1 Première image

Lors de l'exécution de notre programme nous obtenons la figure suivante :

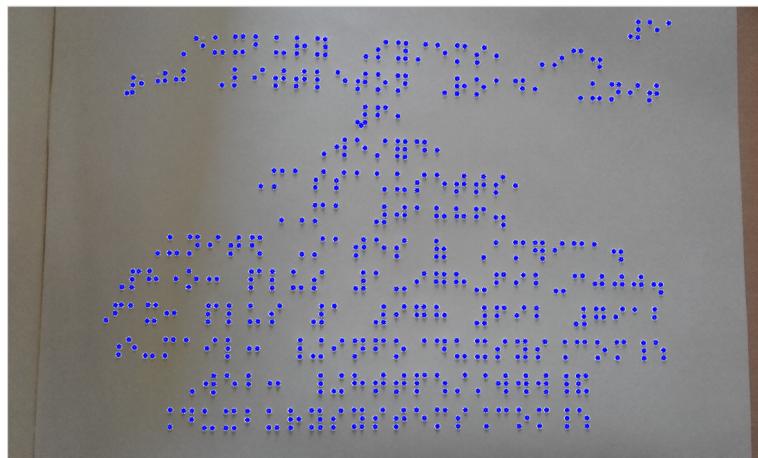


FIGURE 4: Résultat de la transformée de *Hough* sur la première image

Nous voyons sur cette figure que le traitement est plutôt très bon. Nous ne relèvons qu'une petite surségmentation en haut au centre de l'image. Mis à part ce détail que nous n'avons pas su corriger, les points sont bien identifiés, et aucun autre point absent du texte en Braille n'est identifié à tort. Après de multiples tests, il s'avère que les paramètres les plus importants de notre programme sont la taille du disque servant d'élément structurant à la transformation *top-hat*, le seuil choisi pour retirer le bruit, et enfin l'intervalle choisi pour la recherche des cercles. D'après notre expérience, le meilleur des compromis se trouve entre un disque de taille 8, un seuil à 9 et un intervalle de [4;40]. Afin de solutionner ce problème de susegmentation, nous pensons qu'il est pertinent de procéder à plus d'essais du côté de ces paramètres, mais le manque de temps ne nous permet pas de mener à bien ces essais.

3.3.2 Seconde image

Pour cette seconde image, nous obtenons le résultat suivant : Nous voyons sur cette figure que le

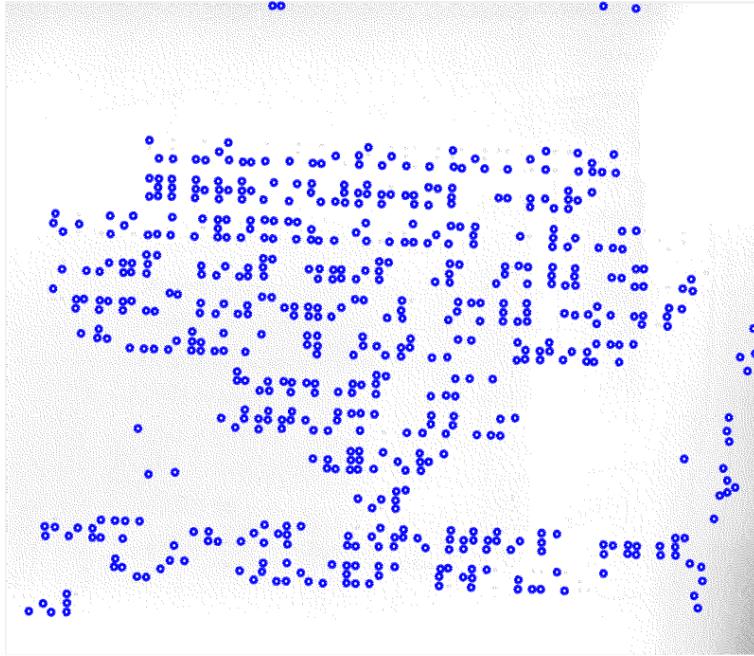


FIGURE 5: Résultat de la transformée de *Hough* sur la première image

résultat est moins satisfaisant. Nous n'avons ici pas de problème de sursegmentation, mais des pixels de bruits identifiés comme des points d'intérêt, en plus d'un temps de calcul plus conséquent. Cette image était très bruitée au départ, et c'est ce qui a rendu son traitement bien plus difficile. De plus, le bruit n'était pas uniforme au niveau de sa couleur. En bas à droite de l'image se trouvait un bruit plus foncé que sur le reste de l'image, et même plus foncé que les caractères que nous cherchions à isoler. Par ailleurs, le contraste n'était pas très bon, et c'est pourquoi nous avons dû faire un compromis, en laissant des points du haut de l'image être identifiés à tort, afin de garder le maximum de caractères Braille. Ainsi, les points du bord haut de l'image, ceux en bas à droite de celle-ci, ainsi que ceux se trouvant dans le creux à gauche ne sont pas des points de caractères. En dépit d'une tentative de débruitage par l'algorithme de *Perona-Malik*, nous n'avons pas su trouver le moyen de supprimer efficacement tous ces points. En revanche, il est à noter que tous les points des caractères ont été identifiés, ce qui est un résultat positif. Il s'agit maintenant de trouver quel algorithme ou quelle combinaison d'opérations de morphologie mathématique conviendrait mieux afin de ne garder que nos caractères.

3.4 Conclusion

Notre objectif était de détecter des cercles sur une images, cercles qui étaient des points de caractère Braille. Ceci se faisait en pré-traitement, dans l'optique de pouvoir les lire par la suite. Nous avons constaté que cette méthode peut s'avérer très efficace sur une image peu bruitée, mais qu'elle peut devenir complexe et lourde en temps de calcul. En revanche, le résultat est tout de même assez bon, au vu de la qualité des images de départ. Le problème qui se pose est alors de devoir trouver un traitement presque unique à chaque nouvelle image. En effet, cela rend une automatisation de la tâche plus complexe.

4 Annexes

4.1 Détection de droites

```
1 clear; close all; clc;
2
3 %% Détection de droites
4
5 %% Lecture de l'image
6 I = im2double(imread('buildings.png'));
7
8 %% Paramètres de l'algorithme
9 [N1,N2] = size(rgb2gray(I));
10 %rho
11 rho_max = sqrt((N1-1)^2+(N2-1)^2);
12 d_rho = 1/round(max([N1,N2])) * rho_max;
13 rho = -rho_max : d_rho : rho_max ;
14 %theta
15 d_theta = 1/round(max([N1,N2])) * pi;
16 theta_max = pi - d_theta;
17 theta = 0 : d_theta : theta_max ;
18
19 %% Image des contours
20 Icont = edge(rgb2gray(I), 'Canny');
21
22 %% Calcul de la matrice d'accumulation H
23 H = zeros(length(rho),length(theta));
24 R = zeros(size(H));
25 for x=1:N1
26     for y=1:N2
27         if (Icont(x,y) == 1)
28             rau_j = x*cos(theta) + y*sin(theta);
29             rau_j = (rau_j + rho_max) * (length(rho)/(2*rho_max));
30             rau_j = round(rau_j);
31             for i=1:length(theta)
32                 H(rau_j(i),i) = H(rau_j(i),i) + 1;
33             end
34         end
35     end
36 end
37
38 %% Extraction des maxima locaux
39 %% Seuillage de H
40 seuil = 0.42 * max(H(:));
41 H_max = (H >= seuil).*H;
42 % H_max = islocalmax(H_max,2); %Matrice des maxima locaux
43 %                                         %(necessite version récente de matlab)
44 max_ind = find(H_max);
45 n = length(max_ind);
46 [Rau,Theta] = ind2sub(size(H),max_ind);
```

```

47 Theta = Theta * d_theta ;
48 Rau = Rau * d_rho - rho_max;
49
50 %%Calcul des points extremaux de chaque droite
51 P1=zeros(2,n);
52 P2=zeros(2,n);
53 for k=1:n
54     if Theta(k) == 0 %si pente infinie
55         P1(1,k) = Rau(k)/cos(Theta(k));
56         P1(2,k) = 1;
57         P2(1,k) = Rau(k)/cos(Theta(k));
58         P2(2,k) = N1;
59     else %cas general
60         P1(1,k) = 1;
61         P1(2,k) = Rau(k)/sin(Theta(k));
62         P2(1,k) = N2;
63         P2(2,k) = (Rau(k)-N2*cos(Theta(k)))/sin(Theta(k));
64     end
65 end
66
67 %%Affichage de la matrice H et de ses maxima locaux
68 figure(1)
69 imshow(imadjust(H/(max(max(H)))),[],'XData',theta, ...
70          'YData',rho,'InitialMagnification','fit');
71 xlabel('\theta (rad)')
72 ylabel('\rho')
73 axis on
74 axis normal
75 hold on
76 colormap(gca,hot)
77 title(['Matrice d''accumulation H (d\theta = ',num2str(d_theta), ...
78        ', d\rho = ', num2str(d_rho), ')']);
79 P = houghpeaks(H,10,'threshold',ceil(0.2*max(H(:))));
80 x = theta(P(:,2));
81 y = rho(P(:,1));
82 plot(x,y,'s','color','blue');

83 %%Affichage des droites detectees sur l'image d'origine
84 figure(2)
85 imshow(I);
86 title(['Droites detectees avec ',num2str(n),' maxima']);
87 hold on;
88 for k=1:n
89     plot([P1(1,k),P2(1,k)],[P1(2,k),P2(2,k)]);
90 end

```

4.2 Détection de cercles - première image

```

1 clear all; close all;clc;
2
3 %%Recup ration de l'image

```

```

4 I = im2double(imread('braille1.png'));
5 I_rgb = rgb2gray(I);
6
7 %%Debruitage
8 %Top hat avec un disque de 8 pixels comme lment structurant
9 se = strel('disk',8);
10 I_th = imtophat(I_rgb,se).*255;
11 %%Seuillage
12 th_seuil = (I_th>9).*I_th;
13
14 %%Recherche des cercles
15 [c1,r1] = imfindcircles(th_seuil,[4 40]);
16
17 %%Affichage
18 figure(1)
19 hold on;
20 imshow(I);
21 viscircles(c1,max(r1)*ones(size(r1))-2,'EdgeColor','b');
22 sgtitle('Image d''origine et mise en vidence des points')

```

4.3 Détection de cercles - seconde image

```

1 clear all; close all; clc;
2
3 %%Rcup ration de l'image
4 I = im2double(imread('braille2.png'));
5 I_rgb = rgb2gray(I);
6
7
8
9 %%Debruitage
10 filtre = fspecial('gaussian',[50,50],0.1);
11 I_filtre = (1-imfilter(I_rgb,filtre)).*255;
12 %%Seuillage
13 I_seuil1 = (I_filtre>17).*I_filtre;
14 I_seuil2 = (I_seuil1>18).*255;
15 %%Morphologie math matique
16 S = [0 0 0; 0 1 0; 0 1 0];
17 I_ouv1 = imopen(I_seuil2,S);
18 S2 = [1,1];
19 I_ouv2 = imopen(I_ouv1,S2);
20 I_dil = imdilate(I_ouv2,ones(4));
21
22
23 %%On trouve les cercles avec la tranform e de Hough
24 [c1,r1] = imfindcircles(I_dil,[1 40]);
25
26 %%Affichage
27 figure(8)
28 hold on;
29 imshow(I);

```

```
30 viscircles(c1,max(r1)*ones(size(r1))-2,'EdgeColor','b');  
31 sgttitle('Image d''origine et mise en vidence des points')
```