Table des matières Python

Extraction de lignes par transformation de Hough

1. Introduction

Lorsque des bords ont été détectés dans une image, il peut être utile d'extraire les lignes droites. Ce document présente la méthode de détection de lignes de Hough ([1]).

2. Transformation de Hough

Considérons l'équation d'une droite dans un plan, avec deux paramètres *(a,b)* :

$$ax + by + 1 = 0 \qquad (1)$$

Toute droite traversant une image peut être mise sous cette forme. Chacune de ces droites peut être représentée par un point dans le plan *(a,b)*. Soit *M(x,y)* un point quelconque de l'image. L'ensemble des droites passant par ce point est représenté par une droite dans le plan *(a,b)*.

La méthode de Hough repose sur l'utilisation d'un accumulateur dans le plan *(a,b)*. L'accumulateur est une matrice qui correspond à un domaine rectangulaire du plan *(a,b)*. Pour chaque pixel *M(x,y)* de l'image, la droite d'équation *ax+by+1=0* est tracée sur l'accumulateur : plus précisément, chaque pixel rencontré est incrémenté d'une unité. Lorsque tous les points de l'image sont traités, les éléments de l'accumulateur les plus peuplés correspondent à des droites détectées.

Le paramétrage des droites par les coefficients *(a,b)* n'est toutefois pas adapté à cette méthode, car les valeurs possibles de *a* et *b* ne sont pas bornées. Il est pratiquement impossible, avec un échantillonnage périodique, de couvrir à la fois les petites, les moyennes et les grandes valeurs de ces coefficients. Pour cette raison, Hough a utilisé le paramétrage suivant des droites, utilisant les coordonnées polaires :

$$\rho = x \cos\ \theta + y \sin\ \theta \qquad (2)$$

La signification géométrique de ρ et θ est montrée sur la figure suivante :

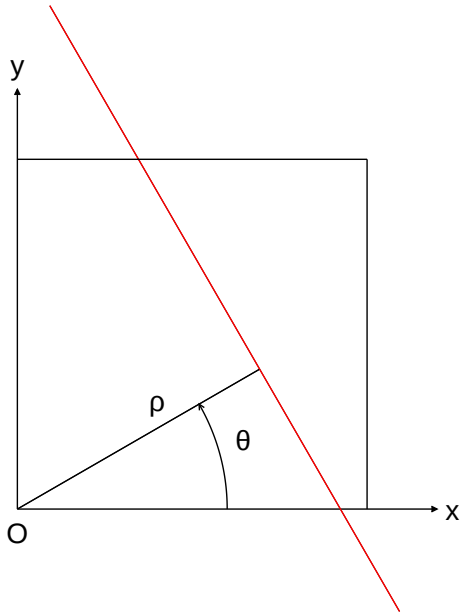


Figure pleine page

Les limites de l'image ont aussi été représentées. Le point *O* est en bas à gauche de l'image. Les deux paramètres sont bornés. Si on note N_x et N_y les nombres de pixels horizontalement et verticalement, les domaines sont :

$$\rho \in \left[0, \sqrt{N_x^2 + N_y^2}\right] \qquad (3)$$

$$\theta \in [0, \pi] \qquad (4)$$

L'accumulateur est une matrice qui correspond à une discrétisation de ce domaine. On note N_θ et N_ρ les nombres de points de l'accumulateur dans ses deux dimensions. Initialement, l'accumulateur est vide. L'image à traiter doit être binaire (image noir et blanc). Pour chaque pixel *M(x,y)* de l'image, on incrémente dans l'accumulateur les points de la courbe d'équation

$$\rho = x \cos\ \theta + y \sin\ \theta \qquad (5)$$

pour θ variant de 0 à 180 degrés. Il s'agit d'une sinusoïde. Il faut bien sûr échantillonner cette courbe de manière à remplir l'accumulateur de manière assez dense.

En fonction des indices *(i,j)* des pixels (l'indice *j* correspond à la ligne), l'équation est :

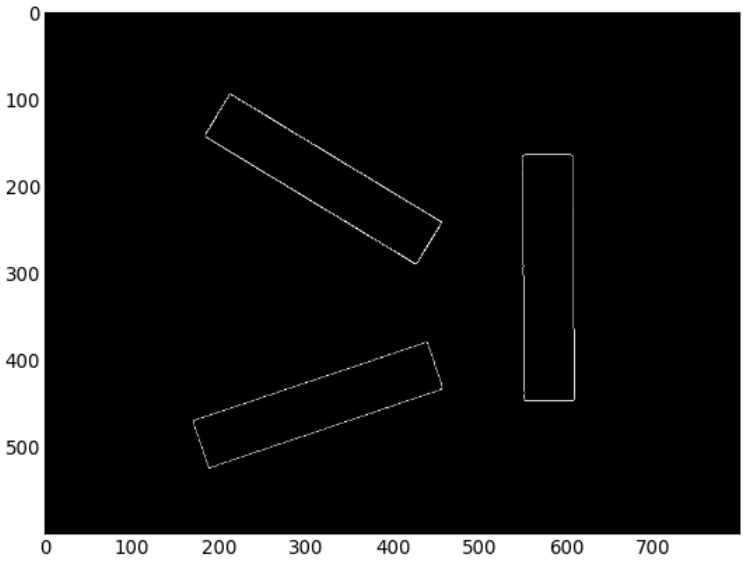
$$\rho = i \cos\ \theta + (N_y - j) \sin\ \theta \qquad (6)$$

Lorsque tous les points de l'image ont été traités, on sélectionne les éléments de l'accumulateur remplis au dessus d'un certain seuil, qui sont alors interprétés comme les paramètres de droites correspondant à des lignes droites sur l'image. Le nombre d'éléments de l'accumulateur doit être choisi en fonction de la précision souhaitée. Par exemple, si l'on veut une précision de l'ordre du degré sur l'angle, il faut $N_\theta=180$ degrés. Cette précision a bien sûr un coût : plus le nombre de points est élevé, plus la sinusoïde doit être échantillonnée finement.

3. Implémentation

L'image utilisée comme exemple a été obtenue par détection de bords sur une photographie comportant trois objets rectangulaires clairs sur un fond noir.

```
import numpy
import math
import cmath
from  matplotlib.pyplot import *
from PIL import Image
img = Image.open("../../../../simul/image/bords-kapla.jpg")
array=numpy.array(img)*1.0
figure(figsize=(8,6))
imshow(array,cmap=cm.gray)
```



figA.pdf

On commence par définir l'accumulateur. `rho` et `theta` sont les résolutions, respectivement en pixel et en degré.

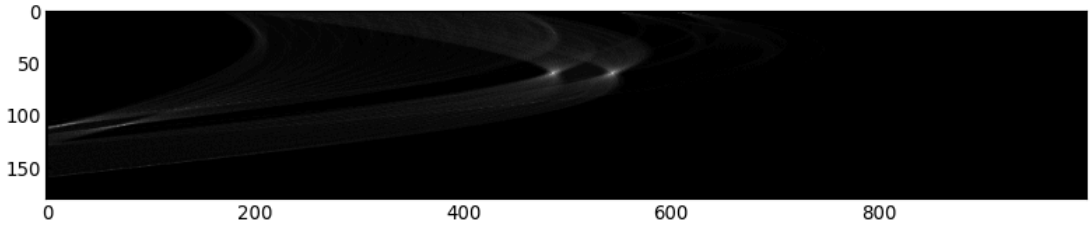
```
(Ny,Nx) = array.shape
rho= 1.0
theta=1.0
Ntheta = int(180.0/theta)
Nrho = int(math.floor(math.sqrt(Nx*Nx+Ny*Ny))/rho)
dtheta = math.pi/Ntheta
drho = math.floor(math.sqrt(Nx*Nx+Ny*Ny))/Nrho
accum = numpy.zeros((Ntheta,Nrho))
```

Pour chaque pixel blanc de l'image, on incrémente les points de l'accumulateur obtenus en échantillonnant la sinusoïde :

```
for j in range(Ny):
    for i in range(Nx):
        if array[j][i]!=0:
            for i_theta in range(Ntheta):
                theta = i_theta*dtheta
                rho = i*math.cos(theta)+(Ny-j)*math.sin(theta)
                i_rho = int(rho/drho)
                if (i_rho>0) and (i_rho<Nrho):
                    accum[i_theta][i_rho] += 1
```

L'accumulateur peut être représenté comme une image :

```
figure(figsize=(12,6))
imshow(accum,cmap=cm.gray)
```

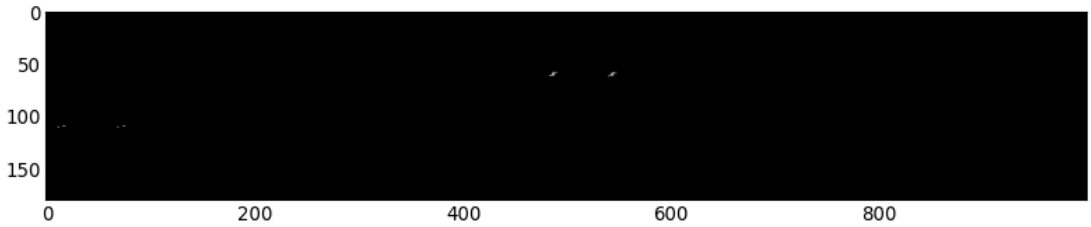


figB.pdf

On retient les valeurs dans l'accumulateur au dessus d'un seuil :

```
seuil=130
accum_seuil = accum.copy()
for i_theta in range(Ntheta):
    for i_rho in range(Nrho):
        if accum[i_theta][i_rho]<seuil:
            accum_seuil[i_theta][i_rho] = 0

figure(figsize=(12,6))
imshow(accum_seuil,cmap=cm.gray)
```



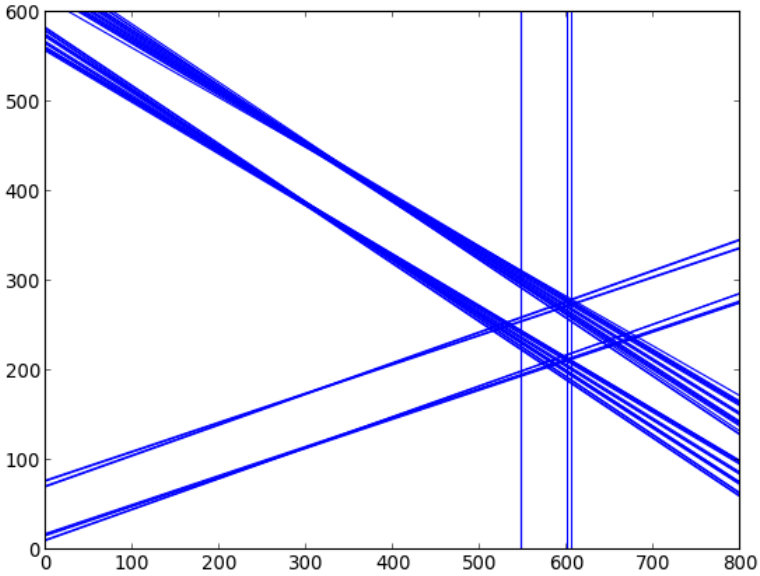
figC.pdf

On constate qu'il y a une forte accumulation autour de certains points, alors que d'autres sont à peine visibles. Pour bien voir les droites obtenues, nous allons placer les couples (ρ,θ) dans une liste :

```
lignes = []
for i_theta in range(Ntheta):
    for i_rho in range(Nrho):
        if accum_seuil[i_theta][i_rho]!=0:
            lignes.append((i_rho*drho,i_theta*dtheta))
```

Enfin on trace ces droites :

```
figure(figsize=(8,6))
axis([0,Nx,0,Ny])
for rho,theta in lignes:
    a = math.cos(theta)
    b = math.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    plot([x1,x2],[y1,y2],color="b")
```



figD.pdf

Avec ce seuil, seuls les grands côtés des rectangles sont détectés. On remarque que plusieurs droites sont obtenues pour le rectangle le plus incliné. Pour le rectangle vertical, la détection est beaucoup plus fine.

4. Recherche des maxima dans l'accumulateur

Pour réduire le nombre de droites obtenues tout en détectant le maximum de segments de l'image, il faut détecter les maxima locaux dans l'accumulateur. Pour cela, on peut faire une recherche de régions de pixels connexes, en suivant l'algorithme expliqué dans Extraction de régions.

L'exploration d'une région de pixels se fait par une fonction de coloriage qui utilise une pile de voisins.

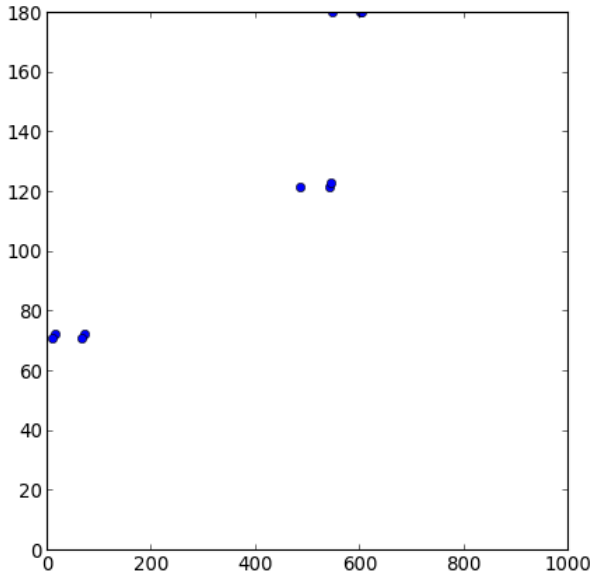
```
def coloriage_pixel_pile(image,result,shape,seuil,valeur,pile,i,j):
    image[j][i] = valeur
    result[j][i] = 255
    voisins = [(i+1,j),(i-1,j),(i,j-1),(i,j+1)]
    for pixel in voisins:
        (k,l) = pixel
        if k>=0 and k<shape[1] and l>=0 and l<shape[0]:
            if image[l][k]>seuil:
                image[l][k] = valeur
                pile.append(pixel)
```

Voici la fonction d'analyse qui calcule le barycentre de chaque région. Les points obtenus sont classés par nombre de pixels décroissants.

```
def analyse(image,seuil):
    shape = image.shape
    Nx = shape[1]
    Ny = shape[0]
    compteur = 0
    positions = []
    result = numpy.zeros((Ny,Nx),dtype=numpy.uint8)
    for y in range(Ny):
        for x in range(Nx):
            if image[y][x] > seuil:
                compteur += 1
                pile = [(x,y)]
                si = 0
                sj = 0
                npix = 0
                while len(pile)>0:
                    (i,j) = pile.pop()
                    si += i
                    sj += j
                    npix += 1
                    coloriage_pixel_pile(image,result,shape,seuil,0,pile,i,j)
                xb = si*1.0/npix
                yb = sj*1.0/npix
                positions.append((xb,yb,npix))
    print("Nombre de taches : %d"%compteur)
    positions = sorted(positions, key=lambda positions:-positions[2])
    for p in positions:
        print("x=%f, y=%f, npix=%d"%(p[0],p[1],p[2]))
    return (positions,result)
```

Voici les points obtenus pour l'accumulateur précédent :

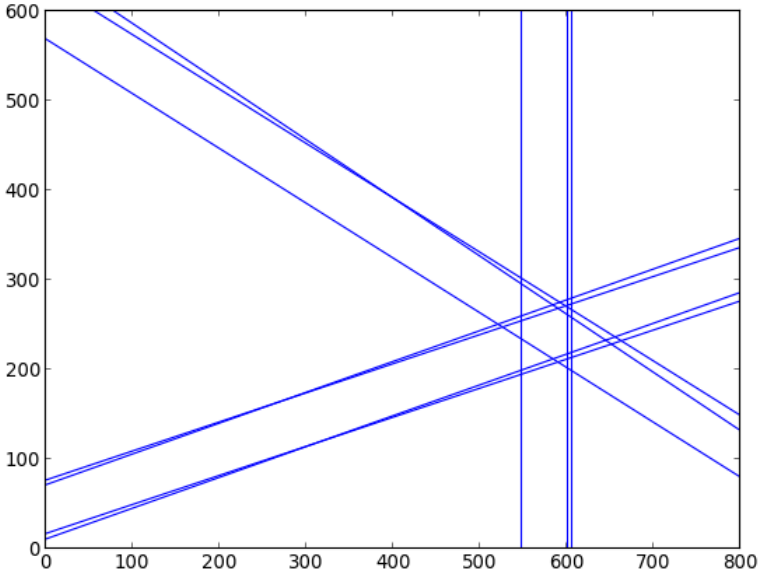
```
(positions,result) = analyse(accum,130)
shape = accum.shape
figure(figsize=(6,6))
for p in positions:
    plot([p[0]],[shape[0]-p[1]],'ob')
axis([0,shape[1],0,shape[0]])
```



figE.pdf

et enfin les droites détectées :

```
figure(figsize=(8,6))
axis([0,Nx,0,Ny])
for point in positions:
    i_rho = point[0]
    i_theta = point[1]
    rho = i_rho*drho
    theta = i_theta*dtheta
    a = math.cos(theta)
    b = math.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    plot([x1,x2],[y1,y2],color="b")
```

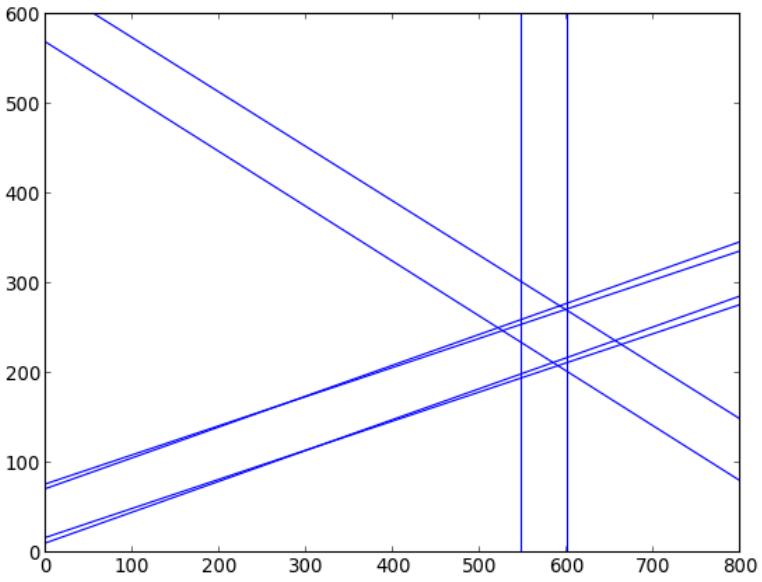


figF.pdf

Il y a beaucoup moins de droites qu'avec un simple seuillage, bien qu'il reste encore des paires pour certains segments de l'image.

Parmis les 10 droites détectées, voyons les 7 qui correspondent aux plus petites taches dans l'accumulateur (celles qui ont au moins deux pixels) :

```
figure(figsize=(8,6))
axis([0,Nx,0,Ny])
for point in positions[0:8]:
    i_rho = point[0]
    i_theta = point[1]
    rho = i_rho*drho
    theta = i_theta*dtheta
    a = math.cos(theta)
    b = math.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    plot([x1,x2],[y1,y2],color="b")
```



figG.pdf

Références

[1] M. Nixon, A. Aguado, *Feature extraction and image processing,* (Academic Press, 2008)
