Exercices
Semaine 16

Cours Turing

Mini-projet

Quatre images sont à votre disposition :

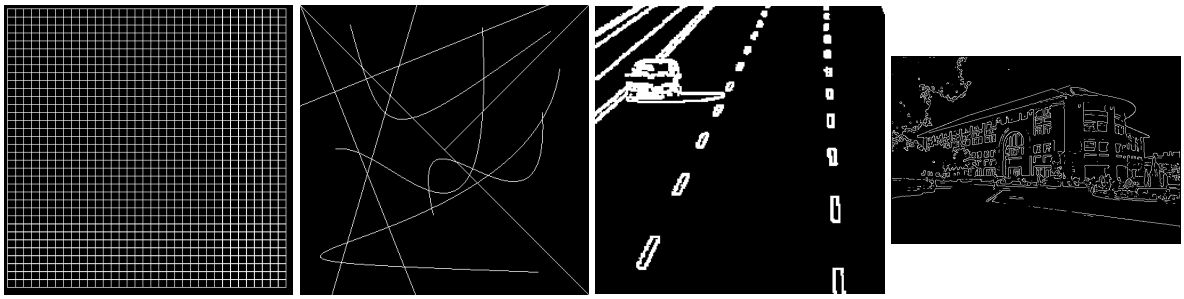

Figure 1: Images de contours à analyser

Questions

1. Dans la première image de la figure 1, quels seront les angles des lignes détectées ? Pourquoi ?
2. Dans la première image de la figure 1, combien de lignes doivent être détectées ? Pourquoi ?
3. Essayer de répondre aux questions précédentes pour les autres images.

Implémentation de l'algorithme de Hough

Implémenter l'algorithme de Hough pour détecter les lignes dans une image.

Pour rappel, L'algorithme proposé est le suivant :

1. Pour chaque pixel (x, y) du contour
   - 1.1 Pour chaque angle $\varphi \in$Liste$_\varphi$
     - 1.1.1 Calculer ρ et l'arrondir à la valeur la plus proche dans $Liste_\rho$
     - 1.1.2 Incrémenter le compteur de la droite associée

où $Liste_\varphi$ est une liste d'angles et $Liste_\rho$ est une liste de distances à définir

Instructions détaillées proposées

Chargement de l'image à analyser et conversion en niveaux de gris
Vous pouvez utiliser la fonction `.convert('L')` sur une image PIL pour la convertir en niveau de gris, avant de la convertir en tableau `numpy`, car elle sera chargée en RGB par défaut.
Alternativement, vous pouvez utiliser une des fonctions de conversion implémentées les semaines précédentes.

Calcul de la longueur de la diagonale de l'image
La longueur de la diagonale de l'image est la distance maximale entre deux points de l'image, ce qui est la valeur maximale de ρ.
Vous pouvez utiliser la fonction `np.shape` pour obtenir la taille de l'image.
Vous pouvez utiliser la fonction `np.sqrt` pour calculer la racine carrée.
Vous pouvez utiliser la fonction `np.ceil` pour arrondir à l'entier supérieur.

Créer les listes de ρ et φ
Rappel : pour la paramétrisation proposée, les valeurs de φ peuvent varier entre $\frac{-\pi}{2}$ et $\frac{\pi}{2}$.
Vous pouvez utiliser la fonction `np.linspace` pour créer une liste de valeurs régulièrement espacées .

Initialisation de l'accumulateur
Vous pouvez utiliser la fonction `np.zeros` pour créer un tableau de zéros.

Analyse des pixels du contour
Vous pouvez utiliser la fonction `np.nonzero` pour obtenir les coordonnées des pixels de contours.

```
ys, xs = np.nonzero(contours)
```

Pour chacun des pixels de contours et chacun des angles, vous devez calculer ρ en utilisant la formule $x \cos \varphi + y \sin \varphi = \rho$.
Ensuite, il faudra trouver l'indice de la valeur de ρ le plus proche dans la liste de ρ et incrémenter le compteur de la droite associée.

Les fonctions `np.argmin` et `np.abs` peuvent vous être utiles.

Affichage de l'accumulateur
Pour visualiser l'accumulateur, actuellement un tableau numpy, il faudra :

1. Transformer les valeurs de l'accumulateur en valeurs entre 0 et 255 en utilisant la formule $Acc_{image} = \frac{Acc - Acc_{min}}{Acc_{max} - Acc_{min}} \cdot 255$
2. Convertir le tableau numpy en tableau d'entiers non signés 8 bits en utilisant la fonction `astype` et `np.uint8`
3. Convertir le tableau numpy en image PIL en utilisant la fonction `Image.fromarray`
4. Sauvegarder l'image PIL en utilisant la fonction `Image.save`

Détection de lignes
Vous devrez dans un premier temps définir votre stratégie pour considérer qu'une droite est détectée. Pour rappel, plusieurs stratégies sont possibles :

1. Toutes les droites qui ont un compteur supérieur à un certain seuil sont considérées comme des droites détectées.
2. Les droites avec les n plus grandes valeurs sont considérées comme des droites détectées.
3. Pour chaque angle, la droite avec la plus grande valeur est considérée comme une droite détectée.
4. ...

Pour l'option 1, vous pouvez utiliser la fonction `np.where` pour obtenir les indices des valeurs supérieures au seuil.

Pour l'option 2, vous pouvez utiliser la fonction `np.argsort` pour obtenir les indices des valeurs triées. Attention néamoins, la liste sera triée dans l'ordre croissant.

Affichage des droites détectées
Pour chaque droite détectée, vous pouvez calculer les coordonnées de deux points de la droite en utilisant la formule $\rho = x \cos \varphi + y \sin \varphi$.

Ensuite, vous pouvez utiliser la librairie `PIL` pour dessiner les droites sur l'image en utilisant la fonction `.draw.line` sur un objet `ImageDraw`.

Le code suivant permet de dessiner une droite entre les points $(50, 100)$ et $(100, 150)$ sur une image `image.png` et de sauvegarder le résultat dans le fichier `image_lignes.png`.

```
from PIL import Image, ImageDraw
img = Image.open('image.png')
draw = ImageDraw.Draw(img)
draw.line((50, 100, 100, 150), fill=128)
img.save('image_lignes.png')
```

Approfondissement

Vous avez vu comment détecter des lignes dans une image en utilisant l'algorithme de Hough.
Nous vous proposons maintenant de réfléchir sur comment adapter cet algorithme pour détecter d'autres formes géométriques.
Ci-dessous, nous vous proposons de détecter des cercles, mais vous pouvez aussi essayer de détecter des ellipses, des rectangles, des triangles, des carrés, des polygones réguliers ou autres en fonction de vos envies.

Détection de cercles

Implémenter l'algorithme de Hough pour détecter les cercles dans une image.

Comment allez-vous paramétriser votre problème ?
Quelles sont les difficultés que vous pouvez rencontrer ?

4 images sont à votre disposition pour tester votre implémentation :

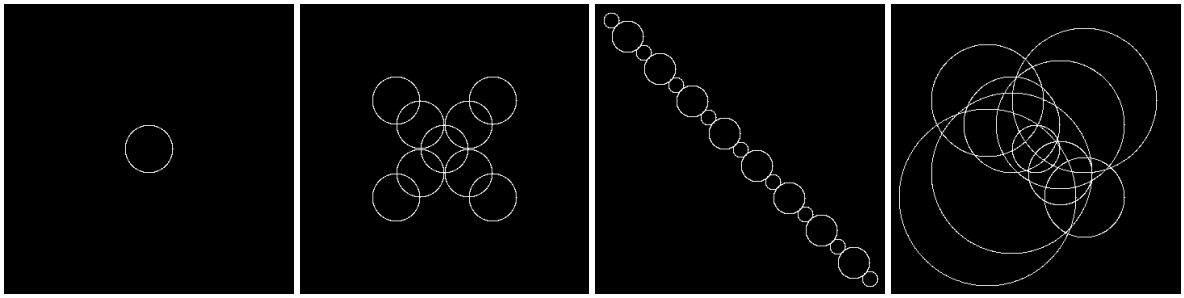

Figure 2: Images de cercles à analyser

Des indications pouvant être utiles :

- Le cercle de la première image a un rayon de 30 pixels
- Les cercles de la deuxième image ont un rayon de 30 pixels
- Les cercles de la troisième image ont des rayons de 10 ou 20 pixels
- Les cercles de la quatrième image ont des rayons de 30, 40, 50, 60, 70, 80, 90, 100 et 110 pixels

Nous vous conseillons de commencer par détecter les cercles de la première image et de la deuxième image, en supposant un rayon de 30 pixels pour votre paramétrisation. Une fois que cela fonctionne, vous pouvez essayer de détecter les cercles des deux autres images.