Application avancée

Autor: Cristopher McIntyre Garcia

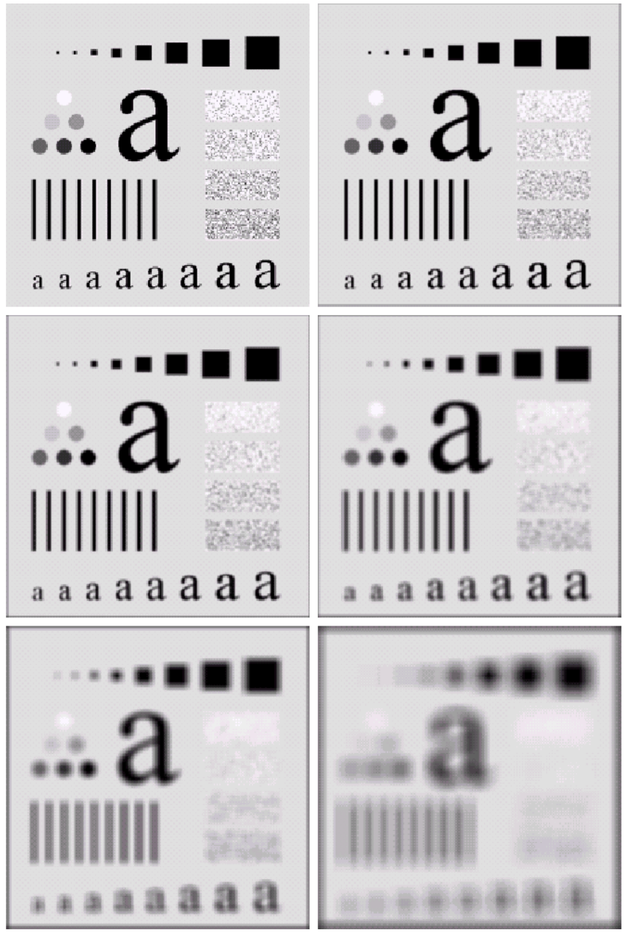
Mini-Cours

- ~~Jour 1: Introduction à l’image digital~~
- ~~Jour 2: Techniques de base de la vision artificiel et au traitement d’image~~
- Jour 3: Filtre et technique traditionnel avancé
- Jour 4: Introduction a l’intelligence artificiel (ML et DL)
- Jour 5: Technique modern - Conversation de nos courent

Filtres spatiaux de lissage

- Filtres linéaires
  - Filtre de lissage nxn
  - Filtres de taille n =3, 5, 9, 15 et 35
- Réduction du bruit
  - Corrompues par du brouillage

1/n	1/n	1/n
1/n	1/n	1/n
1/n	1/n	1/n



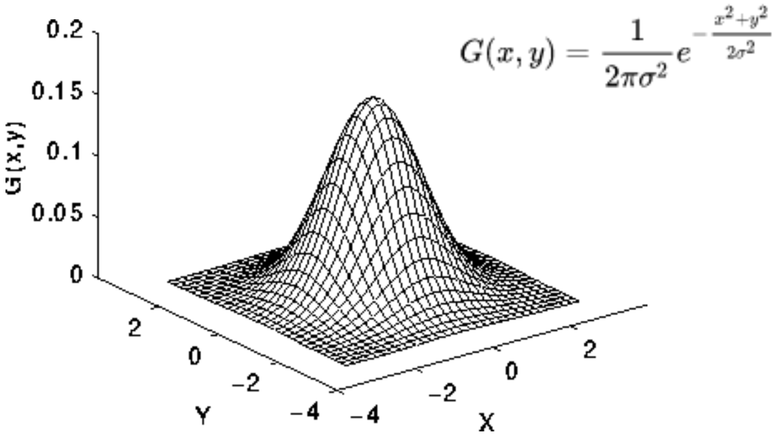
Filtres spatiaux de lissage

- Filtre Gaussien
  - Flou gaussien 3x3
  - Flou gaussien 5x5

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

1/256

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1



Filtres non linéaires

- Filtres de statistiques d’ordre
  - Pixel central est remplacé par le résultat du classement
  - Filtre médian vs moyen
  - Efficace en présence de bruits impulsionnels

Filtre médian 5x5 Filtre moyen 5x5

Filtres spatiaux pour l’affinage

- Mettre en évidence les détails d'une image
- Différenciation numérique
  - Dérivés de premier ordre
  - Dérivées du second ordre
- Améliore les bords et autres discontinuités

	Premier ordre	Deuxième ordre
Rampe	Nonzero => Thick edges	Non nul à la frontière => Bords plus fins
Bruit isolé	Plus faible	Plus fort => accentue les détails fins (y compris le bruit) bruit) plus fort
Lignes fines		Passage d'une situation positive à une situation négatif => Une fine ligne double

Dérivées secondes - le Laplacien

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

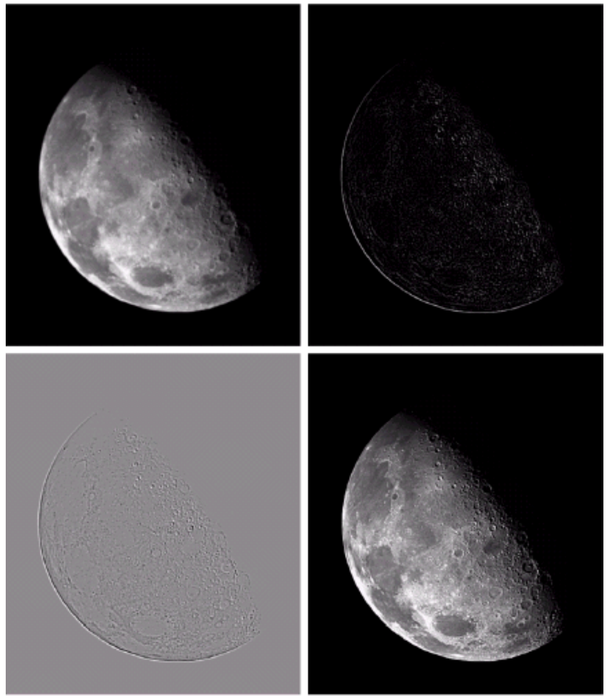
$$\nabla^2 f(x,y) = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

- Filtres isotropes
- Indépendante de la direction des discontinuités
- Invariant par rapport à la rotation
- Plusieurs implantations:

0	-1	0
-1	4	-1
0	-1	0

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

Les contours renforcés



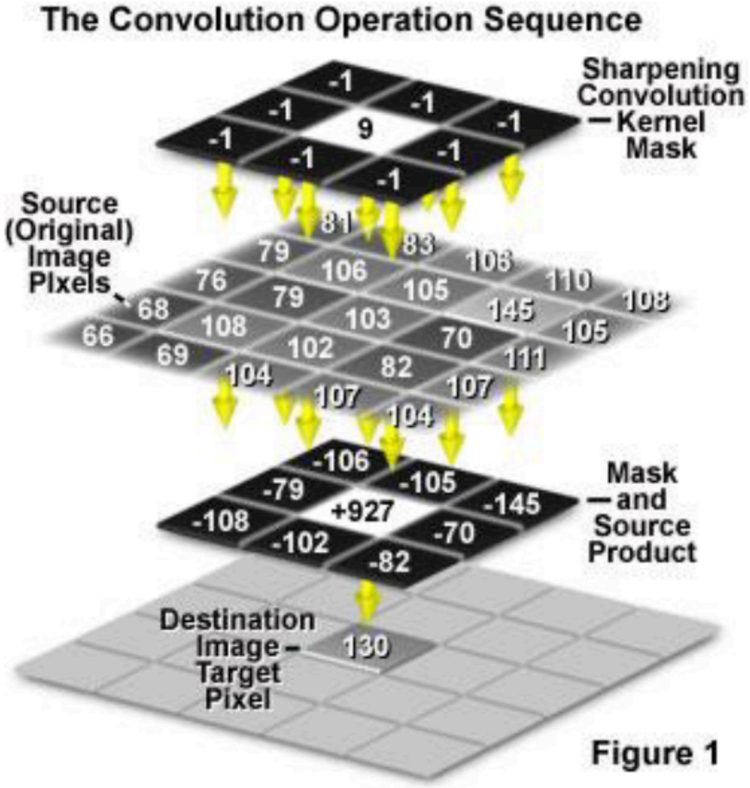


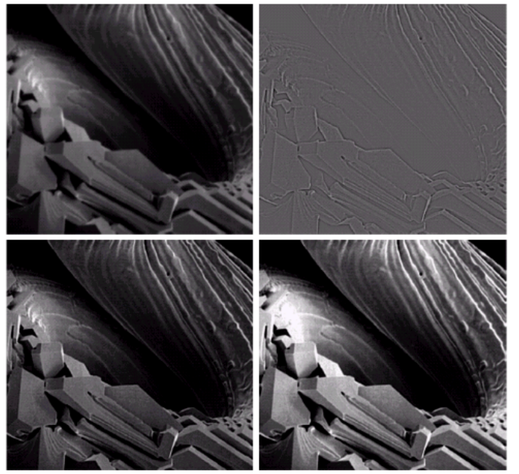
Figure 1

Filtrage High-Boost

$$f_{hb}(x,y) = Af(x,y) - \nabla^2 f(x,y)$$

0	−1	0	−1	−1	−1
−1	$A+4$	−1	−1	$A+8$	−1
0	−1	0	−1	−1	−1

- A=1 Accentuation Laplacienne standard
- A devient plus grand
  - Filtrage devient de moins en moins
  - Si A est suffisamment grand, l'image sera approximativement égale à l'originale multipliée par une constante



Dérivées premières - le gradient

- Le gradient de f est un vecteur
  - Direction de la projection de la normale sur le plan tangent
  - Utilisée pour obtenir la dérivée directionnelle (magnitude)
- Gradient élevées en cas de changement rapide de la luminosité
  - Transition entre une zone sombre et une zone claire

Opérateurs de bord de gradient

- Robert Cross
  - Répondent au maximum aux bords orientés à 45°

+1	0	0	+1
0	-1	-1	0
Gx		Gy	

- Prewitt $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
- Sobel $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
- Kirch $\begin{bmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{bmatrix} \begin{bmatrix} 3 & -5 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & 3 \end{bmatrix}$

Détecteur de bords Canny

1. Lissage

   - par convolution gaussienne.

2. Opérateur de dérivée première 2-D

   - met en évidence les régions de l'image dont les dérivées spatiales premières sont élevées.

3. suppression non maximale (technique d'amincissement des bords)

   - Les bords donnent lieu à des crêtes dans l'image de l'amplitude du gradient.

   - La suppression non maximale est appliquée pour "amincir" le bord. Après le calcul du gradient, l'arête extraite de la valeur du gradient reste assez floue. La suppression du non-maximum peut aider à supprimer toutes les valeurs de gradient jusqu'à 0, à l'exception du maximum local, qui indique l'endroit où le changement de la valeur d'intensité est le plus net.

Détecteur de bords Canny

4. Le processus de suivi

- L'algorithme suit ensuite le sommet de ces crêtes et met à zéro tous les pixels qui ne se trouvent pas sur le sommet de la crête, de manière à obtenir une ligne fine dans la sortie.

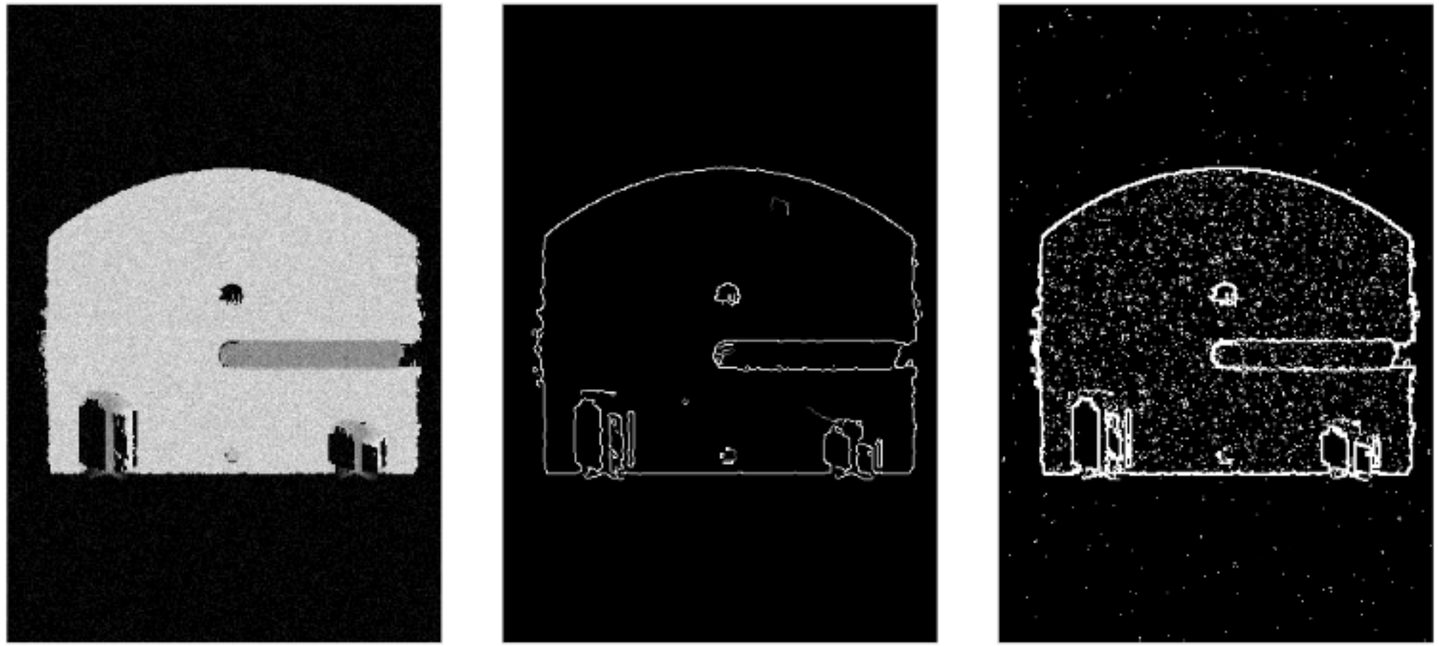
- contrôlée par deux seuils : T_high et T_low avec T_high > T_low

- Le suivi ne peut commencer qu'à un point de la crête supérieur à T_high. Le suivi se poursuit ensuite dans les deux directions à partir de ce point jusqu'à ce que la hauteur de la crête tombe en dessous de T_low.

- Cette hystérésis permet de s'assurer que les arêtes bruyantes ne sont pas divisées en plusieurs fragments d'arêtes.

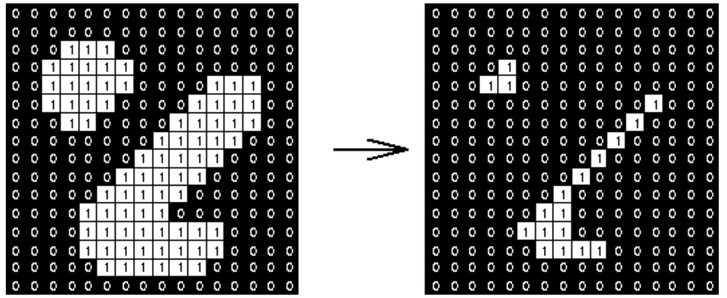
Détecteur de bords Canny

Original vs. Canny vs. Sobel

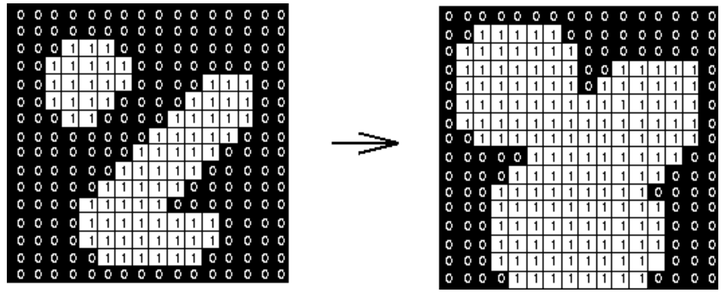


Morphologie mathématique

- Extraire les composants d'image utiles à la représentation d'une région
- Érosion
  - Réduit le masque
  - Si l'un des voisins a une valeur de zéro, le pixel en question est mis à zéro.

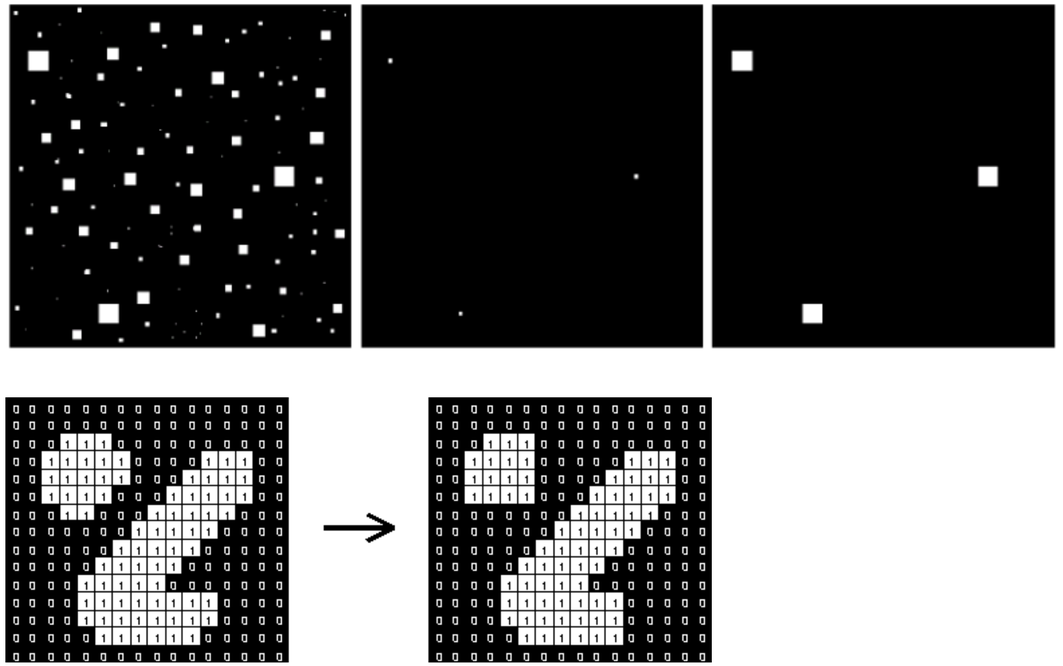


- Dilatation
  - Fait grandir le masque
  - Si un pixel voisin a la valeur 1, le pixel en question est mis à 1



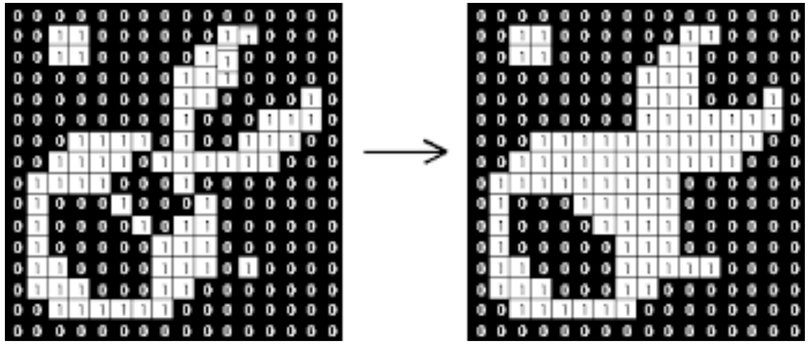
Érosion et dilatation

- Ouverture
- L'érosion suivie de la dilatation
- Lisse les contours d’un objet



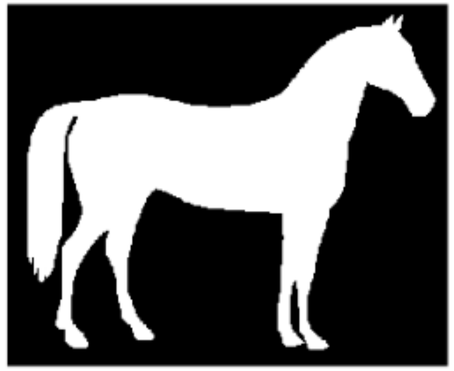
Érosion et dilatation

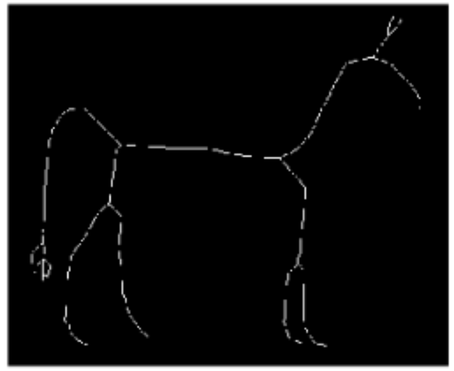
- Fermeture
- La dilatation suivie de l’érosion
- Lisser les sections de contours
- Fusionne généralement les ruptures étroites
- Élimine les petits trous



Squelette

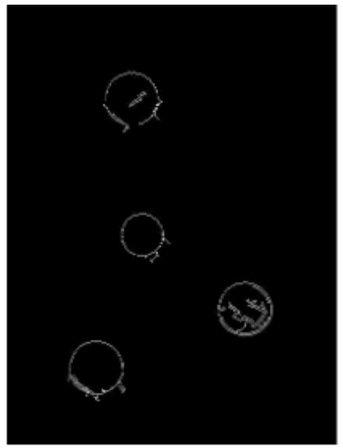
- Représenter la forme structurelle d'une région plane dans un graphique
- Exprimée en termes d'érosions et d'ouverture

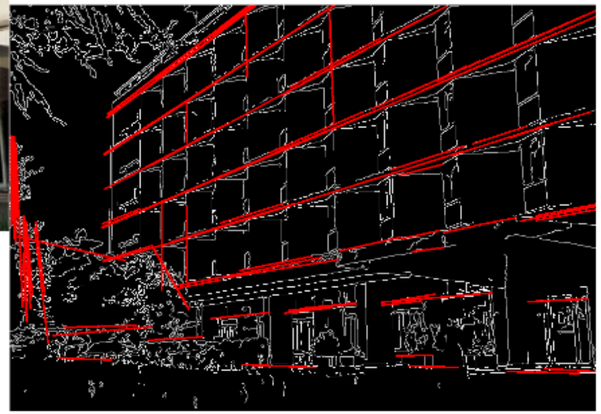




Détection de ligne

- Transformée de Hough





Python + OpenCV

- GitHub
  - https://github.com/cmcin019/mini-cours
- Google Colab

Filtres linéaires

```
gaussian_blur = cv2.GaussianBlur(input_image, (5, 5), 0)  # (5, 5) is the kernel size, 0 is sigma
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
```

```
# Apply Sobel Filter
sobelx = cv2.Sobel(input_image, cv2.CV_64F, 1, 0, ksize=5)  # Sobel X
plt.imshow(cv2.cvtColor(np.uint8(np.absolute(sobelx)), cv2.COLOR_BGR2RGB))

sobely = cv2.Sobel(input_image, cv2.CV_64F, 0, 1, ksize=5)  # Sobel Y
plt.imshow(cv2.cvtColor(np.uint8(np.absolute(sobely)), cv2.COLOR_BGR2RGB))
```

Filtres non linéaires

```
mean_blur = cv2.blur(input_image, (5, 5))  # The (5, 5) is the kernel size specifying a 5x5 window
plt.imshow(cv2.cvtColor(mean_blur, cv2.COLOR_BGR2RGB))
```

```
median_blur = cv2.medianBlur(input_image, 5)  # 5 is the kernel size
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
```

Filtre Laplacien

```python
# Apply Laplacian filter
laplacian = cv2.Laplacian(input_image, cv2.CV_64F)  # Use cv2.CV_64F to capture both negative and positive edges

# Convert back to uint8
laplacian = np.uint8(np.absolute(laplacian))

plt.imshow(cv2.cvtColor(laplacian, cv2.COLOR_BGR2RGB))
```

```python
gray_image = cv2.imread('noidea.jpg', cv2.IMREAD_GRAYSCALE)  # Load as grayscale for simplicity

# Sharpening filter: Create a sharpening kernel
sharpening_kernel = np.array([[-1, -1, -1],
                              [-1,  9, -1],
                              [-1, -1, -1]])

# Apply the sharpening kernel
sharpened_image = cv2.filter2D(gray_image, -1, sharpening_kernel)

# Apply Laplacian filter
laplacian = cv2.Laplacian(sharpened_image, cv2.CV_64F)
# Convert back to uint8
laplacian = np.uint8(np.absolute(laplacian))

plt.imshow(laplacian)
```

Morphologie Mathématique

```
# Define the kernel size
kernel = np.ones((5,5), np.uint8)  # You can change the kernel size as needed

# Apply erosion
erosion = cv2.erode(input_image, kernel, iterations=1)  # Increase iterations for more erosion

plt.imshow(cv2.cvtColor(erosion, cv2.COLOR_BGR2RGB))
```

```
# Define the kernel size
kernel = np.ones((5,5), np.uint8)  # You can change the kernel size as needed

# Apply dilation
dilation = cv2.dilate(input_image, kernel, iterations=1)  # Increase iterations for more dilation

plt.imshow(cv2.cvtColor(dilation, cv2.COLOR_BGR2RGB))
```

Transformation de Hough

```python
h_image=cv2.imread('noidea.jpg')

gray = cv2.cvtColor(h_image, cv2.COLOR_BGR2GRAY)

# Apply edge detection (e.g., using the Canny edge detector)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)

# Apply Hough Line Transform
lines = cv2.HoughLines(edges, 1, np.pi/180, 150)  # rho accuracy of 1 pixel, theta accuracy of 1 degree, threshold 150

# Draw the lines
for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.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))

    cv2.line(h_image, (x1, y1), (x2, y2), (0, 0, 255), 2)

plt.imshow(cv2.cvtColor(h_image, cv2.COLOR_BGR2RGB))
```

Fin

- Merci