

Étude de la lecture des codes-barres

KAMARA Mohamed

Objectif et Problématique

- **Objectif** : *Mettre en place un programme informatique capable de lire un code-barre à partir d'une image.*
- **Problèmes potentiels** : contraste faible, rotation, zone des barres inconnues



Fig1 : Exemple

→
pré-traitement



Fig2 : Modèle

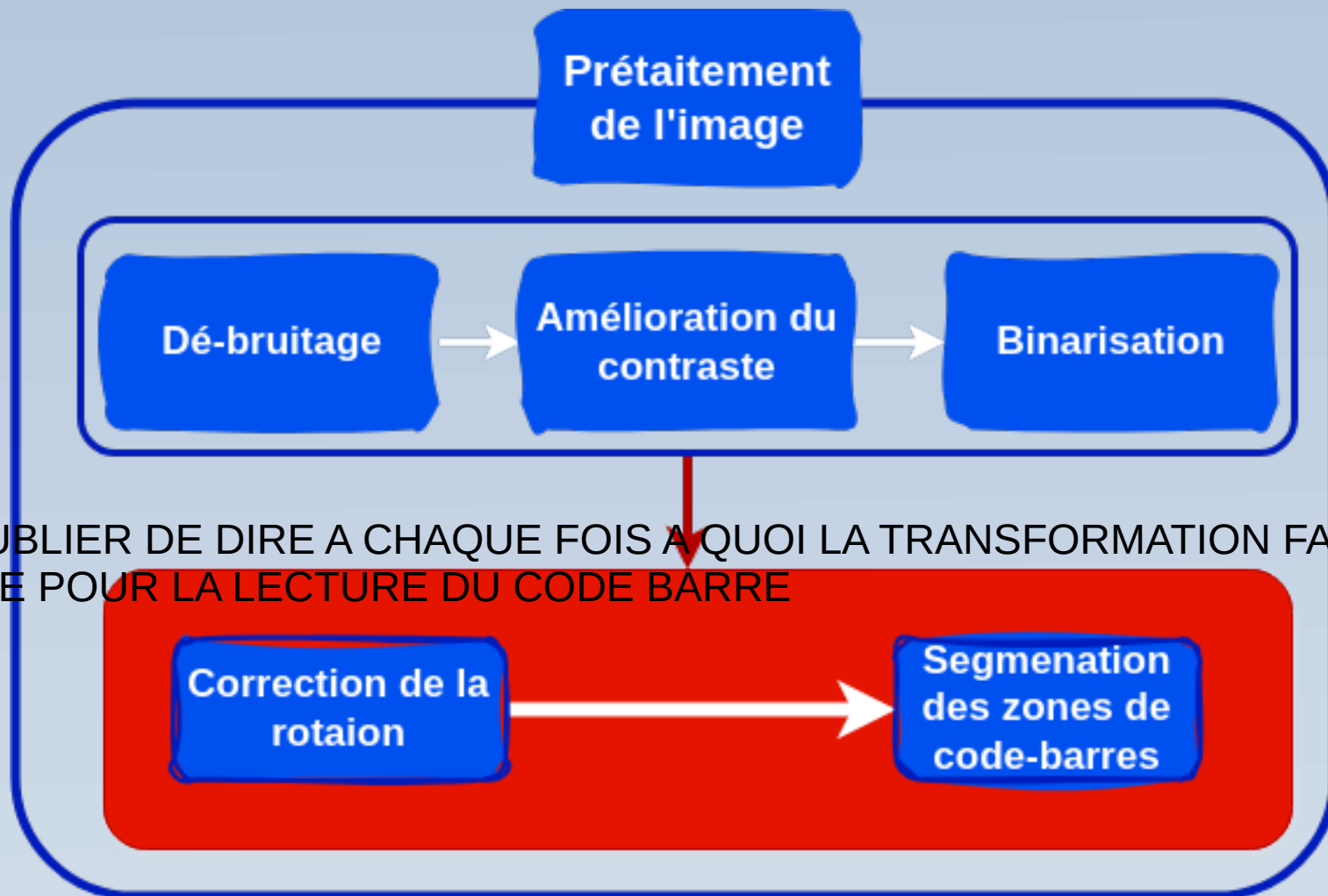
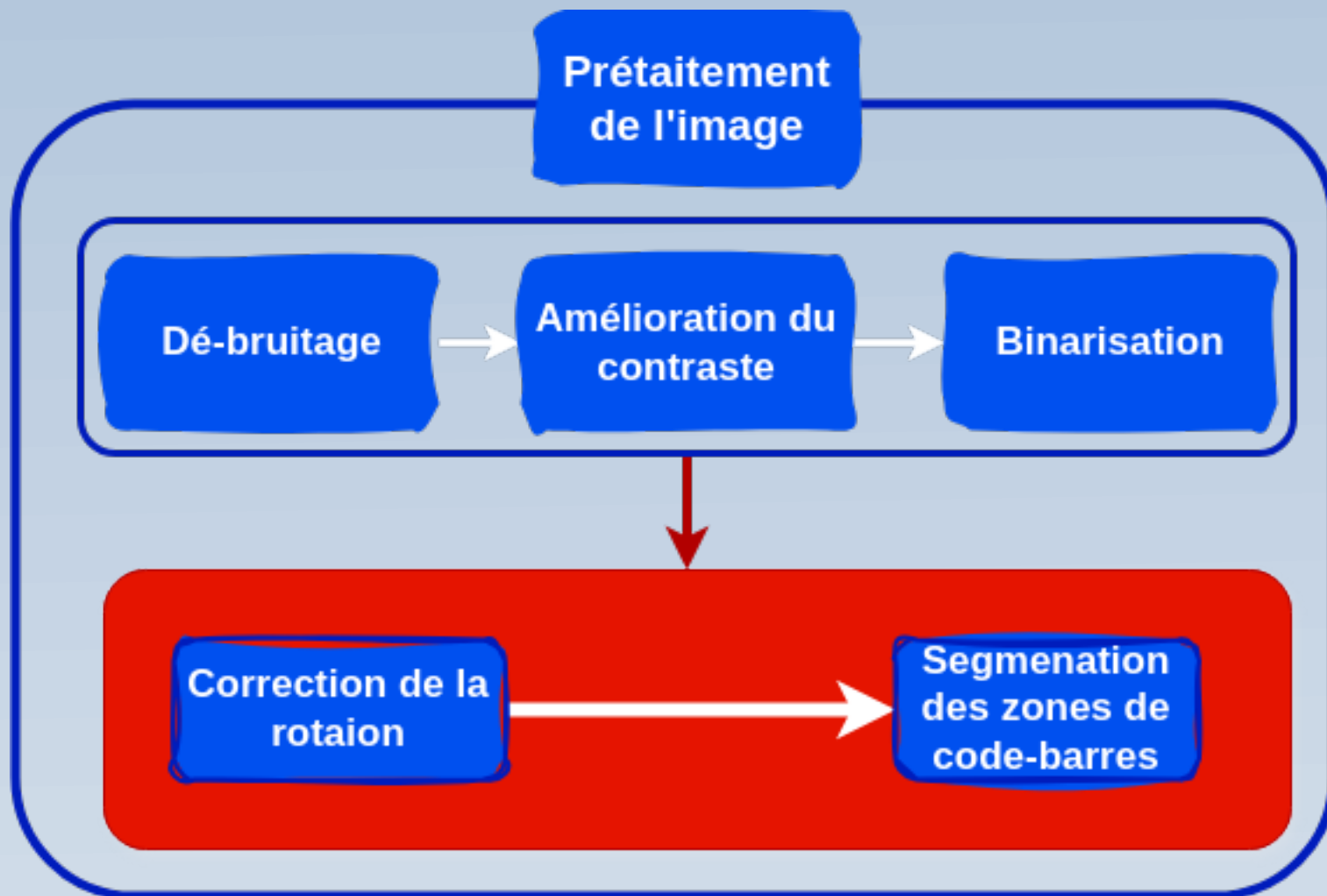


Image Initiale

→
Dé-bruitage



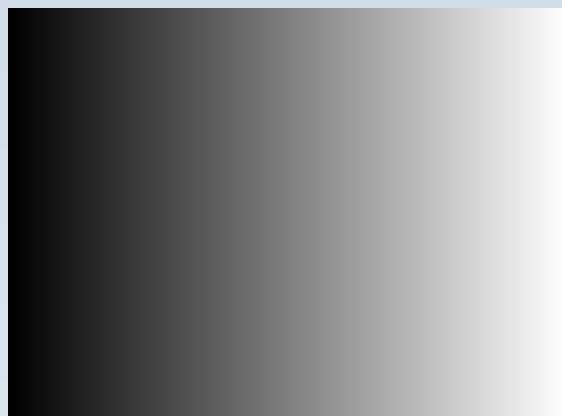
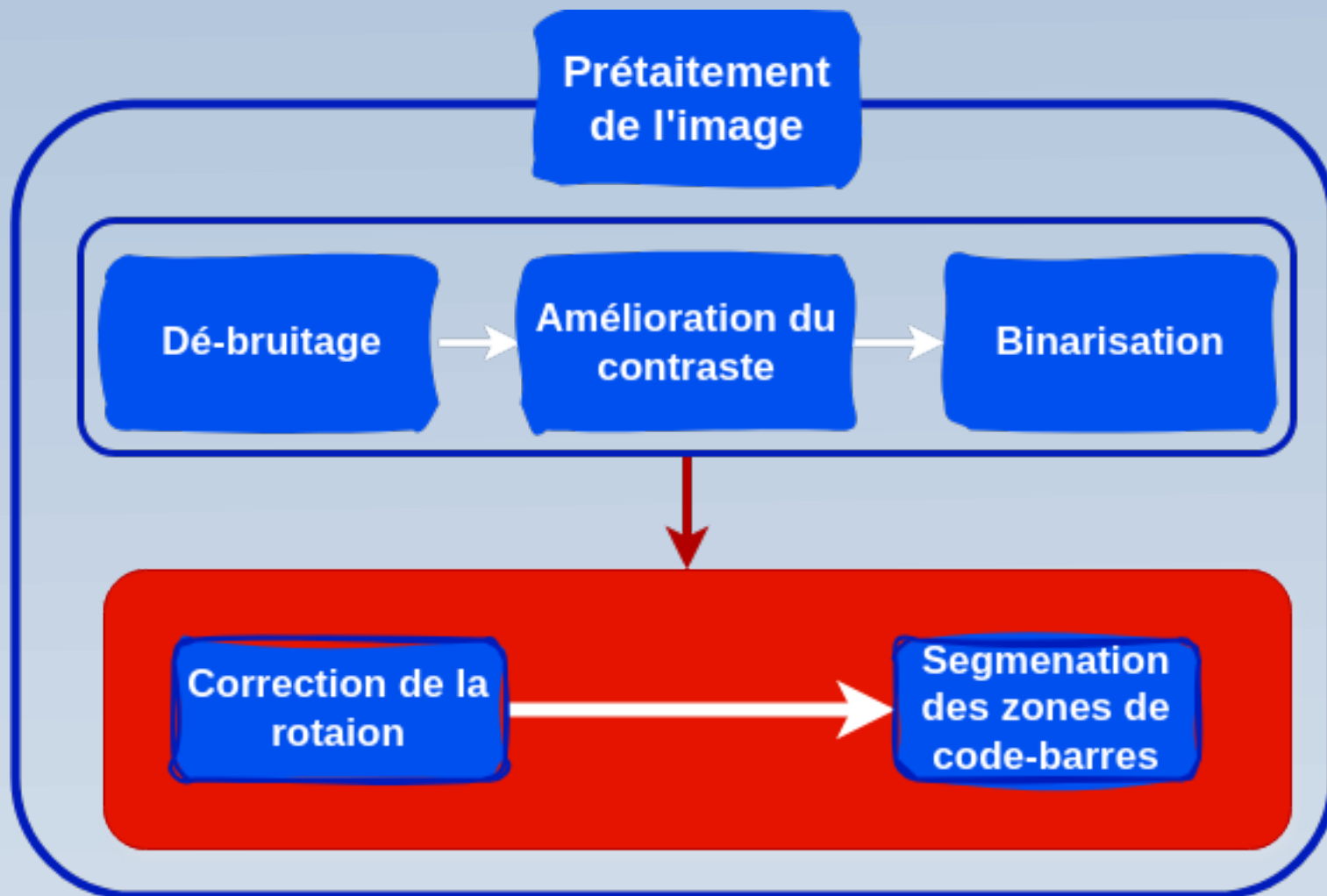
Image Obtenue



Amélioration du contraste

A blue arrow pointing from the low-contrast barcode to the high-contrast barcode, indicating the direction of the contrast improvement process.





→
Binarisation



Correction de la rotation

INCOMPLET

Principe :

- Appliquée un **filtre canny** pour detecter les bords,
- Appliquée une **transformée de hough** pour detecter les lignes, les droites
- A partir de là calculer l'angle moyen d'inclinaison des lignes
- Redresser l'image à partir de cet angle moyen en appliquant une matrice de convolution

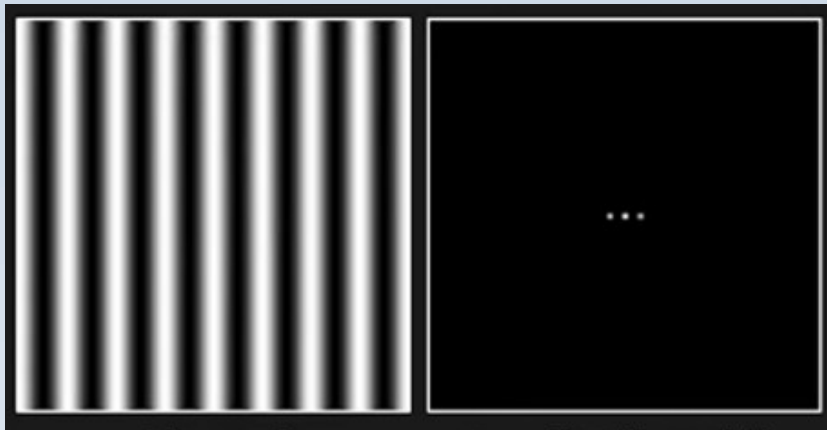
Segmentation

Idée : On cherche dans l'image les zones quasi-périodiques, présence d'un motif répétitif

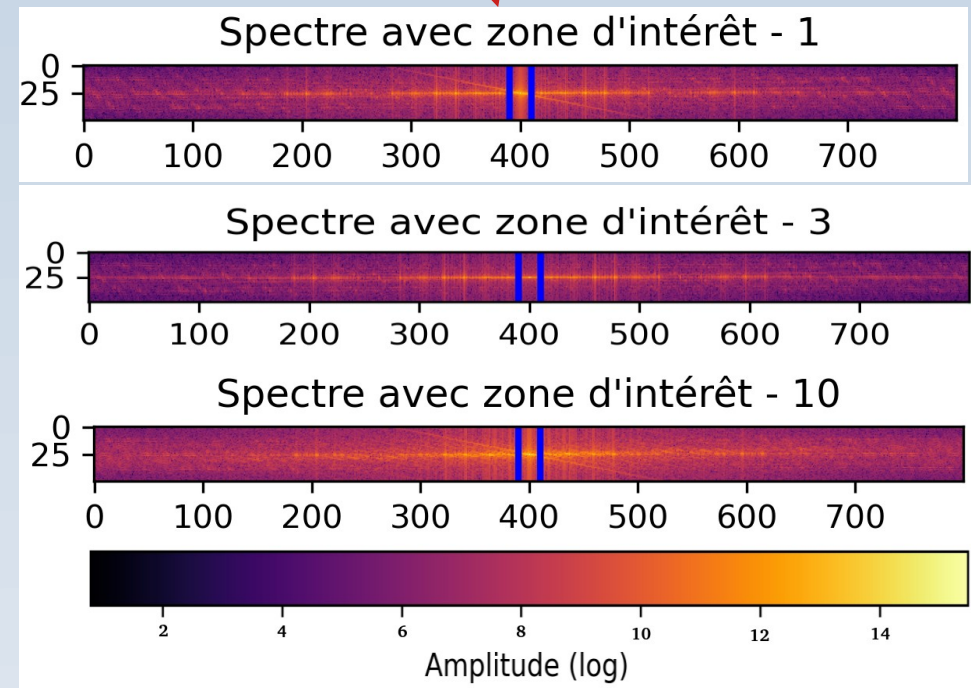


Fenêtrage

Application de la Transformée De Fourier discrète 2D



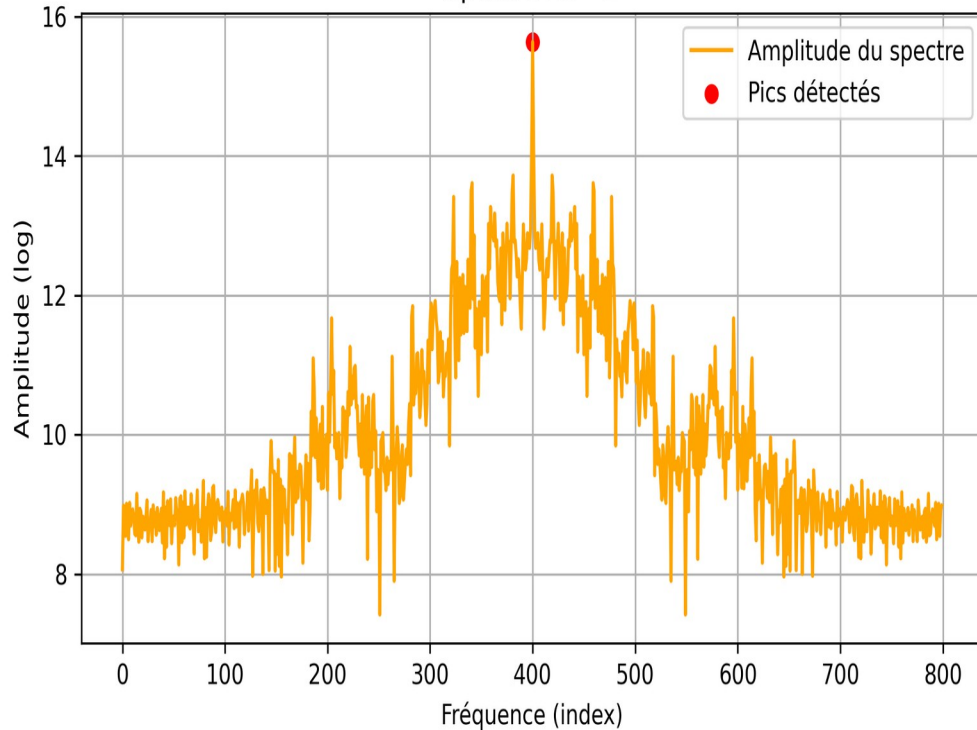
Exemple de la DFT-2D de la représentation de la fonction cosinus



Segmentation

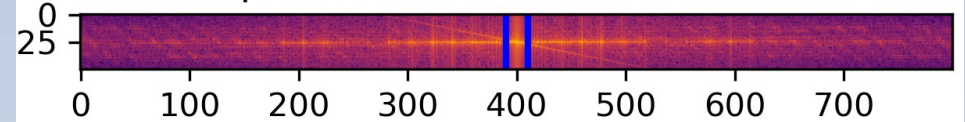
Filtre Passe Haut

Spectre 1D - 3

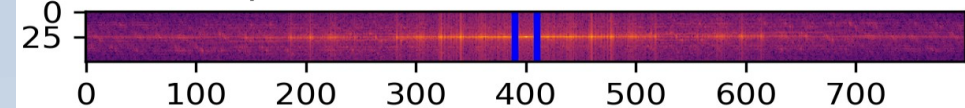


Application de la Transformée De Fourier discrète 2D

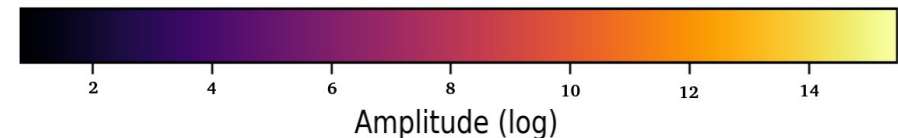
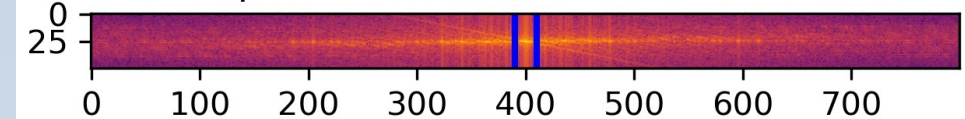
Spectre avec zone d'intérêt - 1



Spectre avec zone d'intérêt - 3



Spectre avec zone d'intérêt - 10



D'après la DFT-2D :

Basses fréquences → Régions uniformes (zones sans variation brutale).

Hautes fréquences → Changements rapides, comme des motifs alternés noir/blanc.

On sélectionne les fenêtres contenant des pics d'amplitude élevée par rapport à un seuil défini.
qui sont potentiellement des zones contenant des codes barres.

Plan de l'annexe

- Dé-bruitage (p7)
- Amélioration du contraste(p8)
- Binarisation (p9)
- Correction de la rotation (p10)
- Segmentation (p12)
- Détection de la texture des code-barres (p13)

Dé-bruitage



```
def denoising(gray_img):  
    print(" ----- DENOSING -----")  
  
    # Filtre bilatéral pour un débruitage initial sans perte de contours  
    bilateral_denoised = cv2.bilateralFilter(gray_img, d=15, sigmaColor=75, sigmaSpace=75)  
  
    # Filtre médian pour finaliser le débruitage  
    final_denoised = cv2.medianBlur(bilateral_denoised, ksize=3)  
  
    return final_denoised
```

Amélioration du contraste



```
def contrast_improve(gray_img):  
    """  
        Par la technique de normalisation de l'histogramme  
    """  
    print(" ----- CONTRAST IMPROVE -----")  
  
    normalized_img = cv2.equalizeHist(gray_img)  
  
    return normalized_img
```

Binarisation



```
def binarise(img):  
    """ Applique une conversion en noir et blanc et une binarisation de l'image """  
    print(" ----- BINARISATION -----")  
    _, black_and_white_img = cv2.threshold(gray_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  
  
    return black_and_white_img
```

Correction de la rotation

```
def correction_rotation(gray_img):  
    """  
    NOTION CLE : filtre CANNY et TRANSFORMÉE DE HOUGH  
    Correction de la rotation de l'image  
    - filtre canny  
    - detection de ligne avec hough  
    - calcul de l'angle de rotation  
    - appliquer la correction a l'image originale  
    """  
    print(" ----- CORRECTION ROTATION -----")  
  
    # Filtre Canny  
    canny_img = cv2.Canny(gray_img, 100, 200)  
  
    # Detection avec HOUGH  
    lines = cv2.HoughLines(canny_img, 1, np.pi / 180, 200)  
  
    if lines is None:  
        # Si aucune ligne n'est détectée, retourne l'image d'origine  
        return gray_img  
  
    # Calculer l'angle moyen des lignes détectées  
    angle_sum = 0  
    count = 0  
    for line in lines:  
        # rho : La distance entre l'origine de l'image (le point (0,0)) et la ligne.  
        # theta : L'angle entre l'axe horizontal (l'axe des x) et la ligne.  
        for rho, theta in line:  
            angle = (theta * 180 / np.pi) # Convertir l'angle en degrés  
            # Ignorer les lignes quasi-verticales pour éviter de fausser l'angle moyen  
            if -85 < angle < 85:  
                angle_sum += angle  
                count += 1
```

Correction de la rotation(suite)

```
if count > 0:
    average_angle = angle_sum / count
else:
    # Si toutes les lignes détectées étaient quasi-verticales, ignorer la rotation
    average_angle = 0

# Redresser l'image en fonction de l'angle moyen
(h, w) = gray_img.shape[:2]
center = (w // 2, h // 2)

# Création de la matrice de rotation
rotation_matrix = cv2.getRotationMatrix2D(center, average_angle, 1.0)

# Application de la convolution (rotation)
straightened_img = cv2.warpAffine(gray_img, rotation_matrix, (w, h))

return straightened_img
```


Segmentation

```
def segmentation(straightened_img):  
    print(" ----- SEGMENTATION -----")  
  
    bars_zones_images = []  
    # Paramètres de la fenêtre  
    window_height = 50 # Hauteur de la fenêtre mobile  
  
    # Parametres #-2 1 30 30 10 10  
    sign = -20  
    factor = 1  
    offset_1 = 30  
    offset_2 = 30 # offset_2 doit etre >= offset_1  
    offset_3 = 10  
    offset_4 = 10  
  
    # Obtenir les dimensions de l'image  
    image_height, image_width = straightened_img.shape[:2]  
  
    # Balayage vertical de l'image  
    for y in range(0, image_height - window_height, window_height):  
        # Définir la fenêtre de balayage  
        window = straightened_img[y:y + window_height, 0:image_width]  
  
        if detect_barcode_texture(window, sign, factor, offset_1, offset_2, offset_3, offset_4):  
            print(f"Zone contenant des barres détectée à partir de y = {y}")  
            bars_zones_images.append(window)  
  
    return bars_zones_images
```

Détecte code-barres texture

```
def detect_barcode_texture(gray_img, sign, factor, offset_1, offset_2, offset_3, offset_4):

    # Appliquer la transformée de Fourier
    f = np.fft.fft2(gray_img)
    fshift = np.fft.fftshift(f)

    # Calculer le spectre de fréquence
    magnitude_spectrum = sign * np.log(np.abs(fshift)) # Application d'un filtre passe haut

    # Calculer la moyenne du spectre de fréquence
    mean_magnitude = np.mean(magnitude_spectrum)

    # Définir le seuil comme un multiple de la moyenne
    threshold = mean_magnitude * factor # Ajuste le facteur selon les résultats observés

    # Localiser des pics de fréquence caractéristiques des codes-barres
    # (paramètres à ajuster selon les caractéristiques de l'image)
    rows, cols = magnitude_spectrum.shape
    center_row, center_col = rows // 2, cols // 2
    freq_zone = magnitude_spectrum[center_row-offset_1:center_row+offset_2, center_col-
offset_3:center_col+offset_4]

    # Vérification si les pics sont présents dans la zone d'intérêt
    if np.mean(freq_zone) > threshold: # Définir un seuil approprié
        return True
    else:
        print("Aucune zone de code-barres détectée")
        return False
```