Détection des codes-barres sur une image

KAMARA Mohamed
14793

Objectif et Problématique

- **Objectif** : Reconnaissance de code-barres par traitement d'image.

- **Problèmes potentiels** :faible contraste, rotation, zone des barres inconnue.

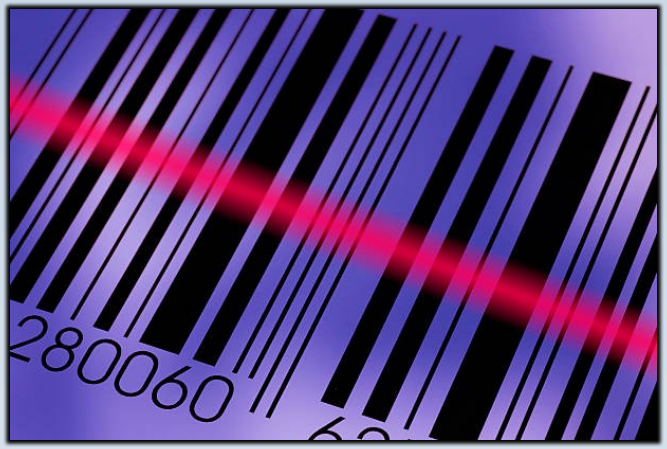
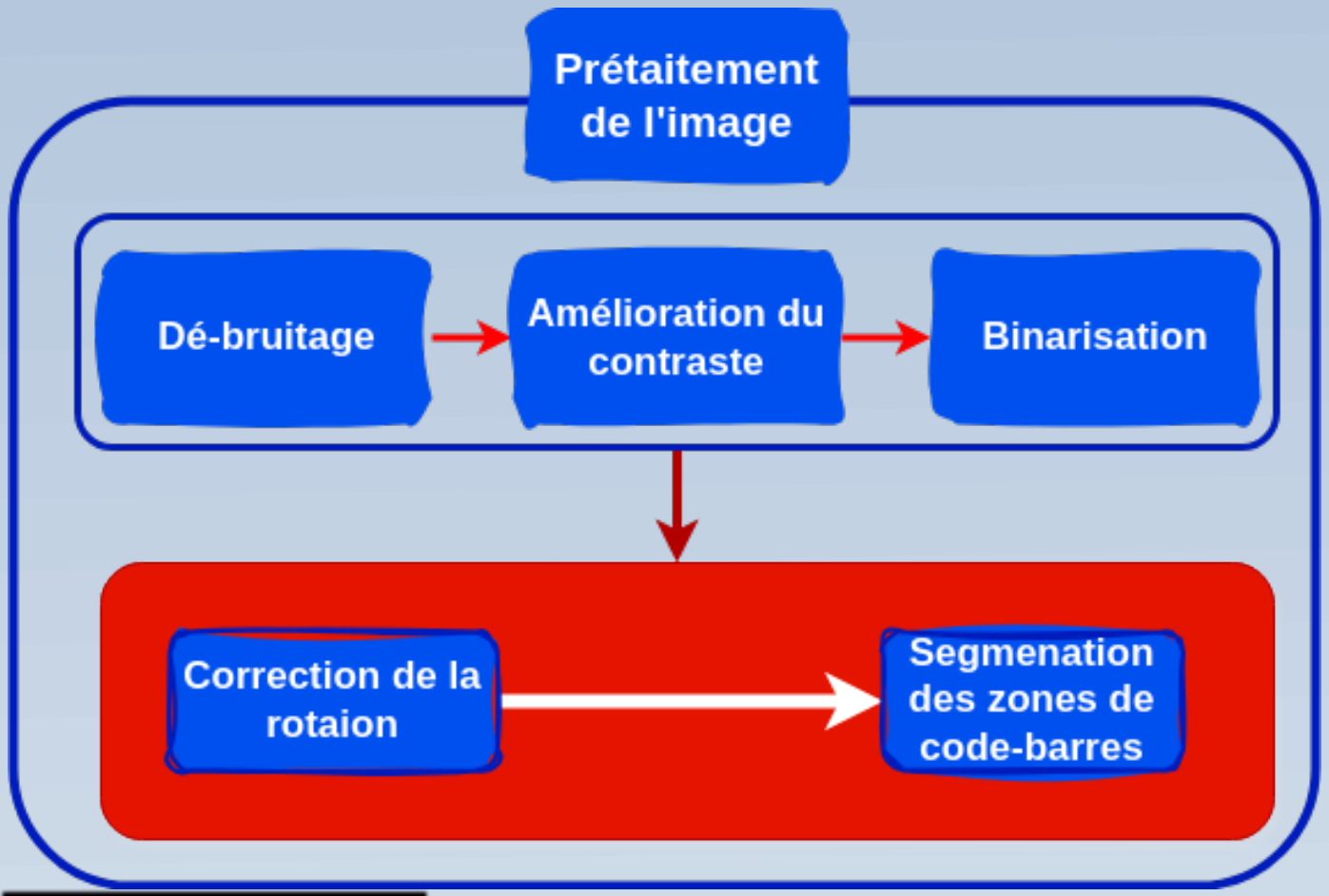
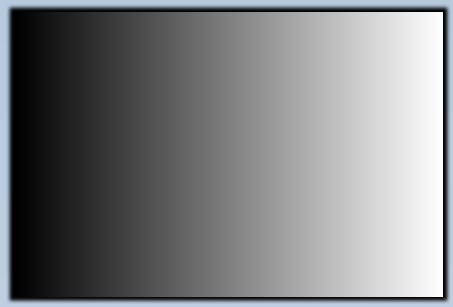


Image 1 : Exemple

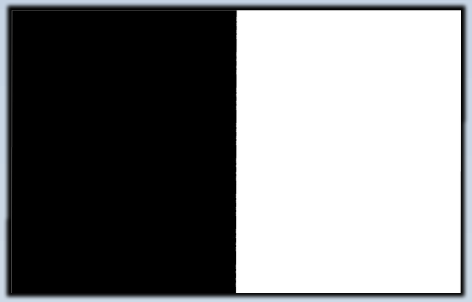
pré-traitement

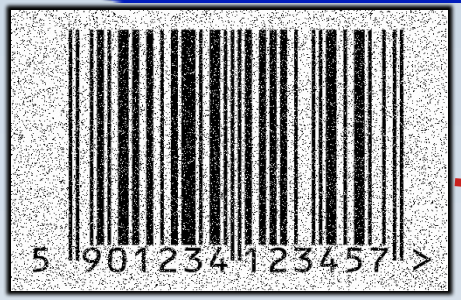
Image 2 : Résultat





Binarisation





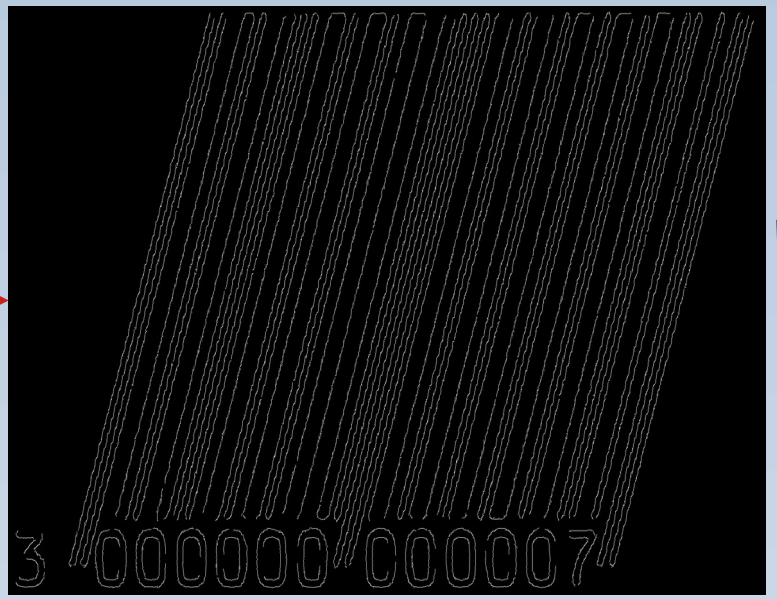
Dé-bruitage

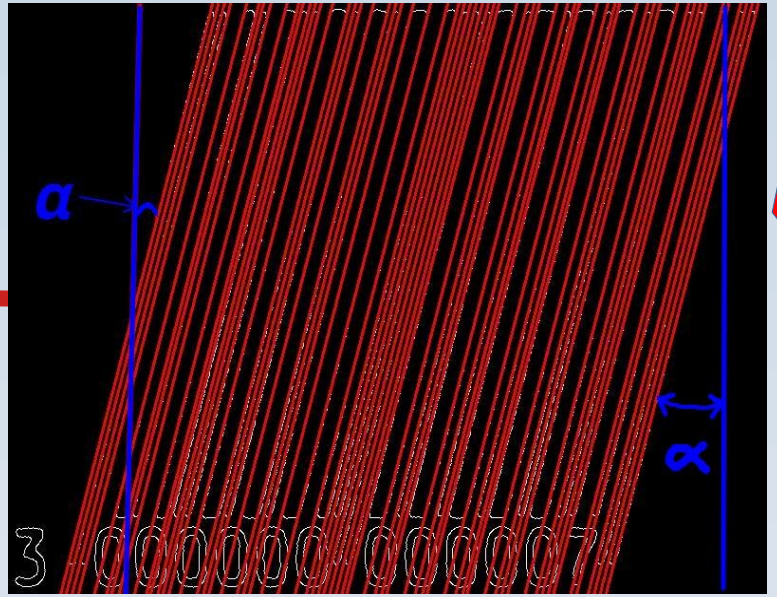
Amélioration du contraste

Image souhaitée

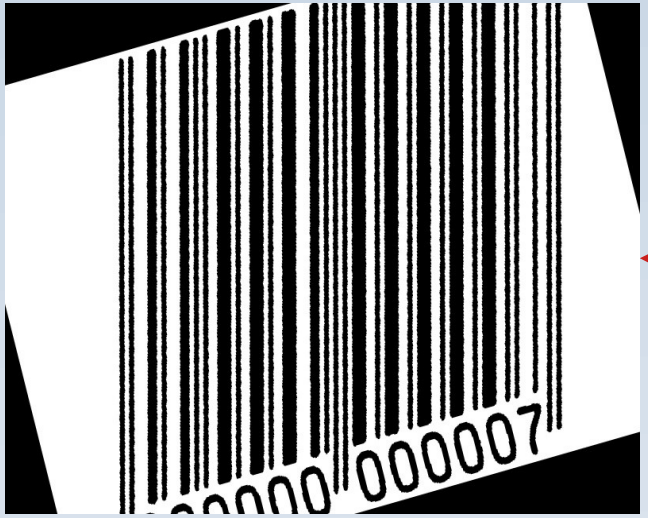
Correction de la rotation

Détection des Contours





Redressement

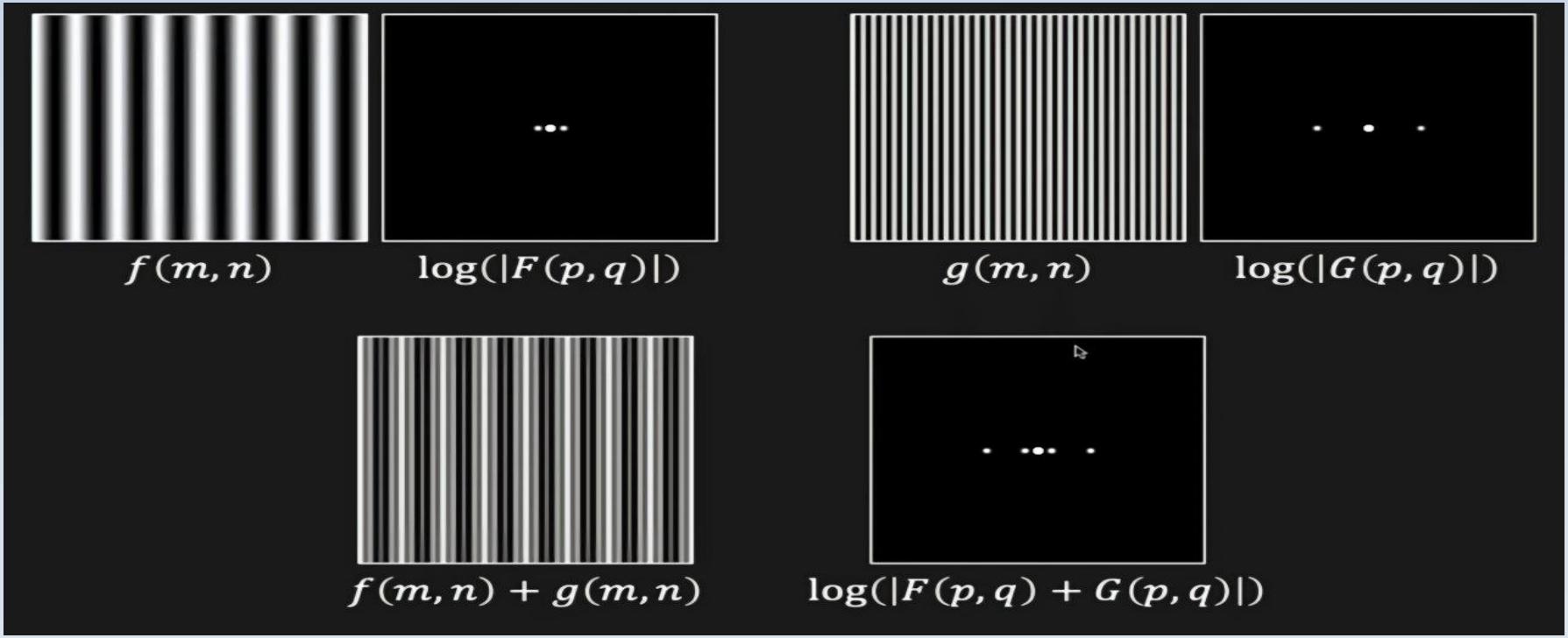


Résultat

Détection des lignes et calcul de l'angle moyen d'inclinaison

Segmentation : De la musique… à l'image

- **Alternance régulière des barres → Fréquence dominante détectable**
- **Transformée de Fourier 2D → Carte des fréquences de l'image (spectre)**
- **Chaque point de la carte → Fréquence + intensité associée**
- **Motifs du code-barres → Pics caractéristiques dans le spectre**



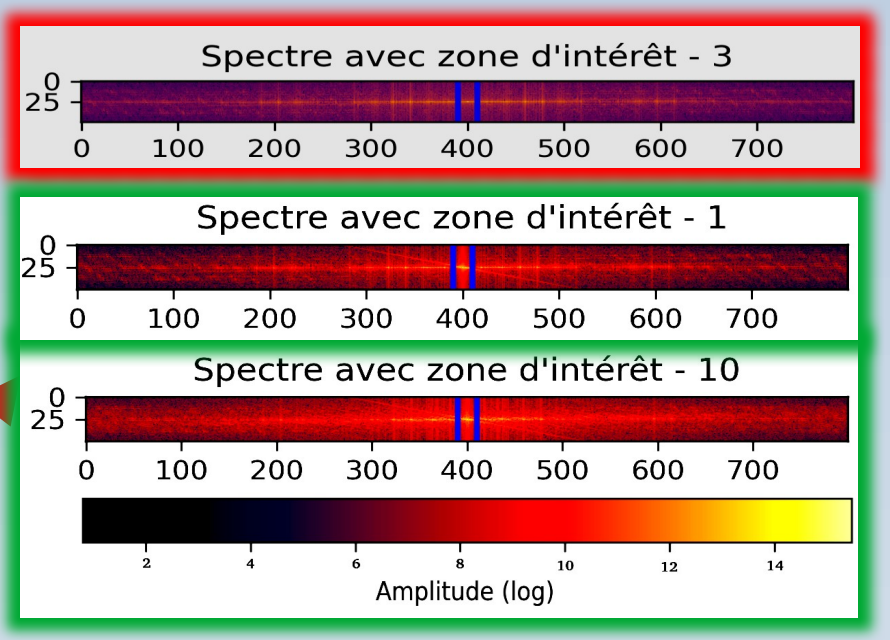
Exemple d'une Transformation de Fourrier Discrète 2D

Segmentation

Application de la Transformée De Fourrier discrète 2D

Fenêtrage

Fenêtre



Exemple de résultat

Détection par amplitude : si l'amplitude dans la zone centrale dépasse un seuil basé sur la moyenne globale, on retient la fenêtre.

Conclusion

- Teste avec des images
- D’autres méthodes possibles et plus évoluées

L’importance de l’ordre d’application des filtres

Plan de l'annexe

Annexe : Dé-bruitage

Code proposé pour le dé-bruitage afin de facilité l'analyse de l'image

```python
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
```

Annexe : Amélioration du contraste

```python
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
```

Annexe : Binarisation

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

        return black_and_white_img
```

Annexe : Correction de la rotation

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

Annexe : Correction de la rotation(suite)

```python
# 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
```

Annexe : Segmentation

```python
def segmentation(straightened_img):
        print(" ----------- SEGMEMENTATION -----------")

        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
```

Annexe : Détecte code-barres texture

```python
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))

        # 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
        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
        edianBlur(bilateral_denoised, ksize=3)

        return final_denoised
```