Étude de la lecture des codes-barres

KAMARA Mohamed

Choix du modèle d'étude

- **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

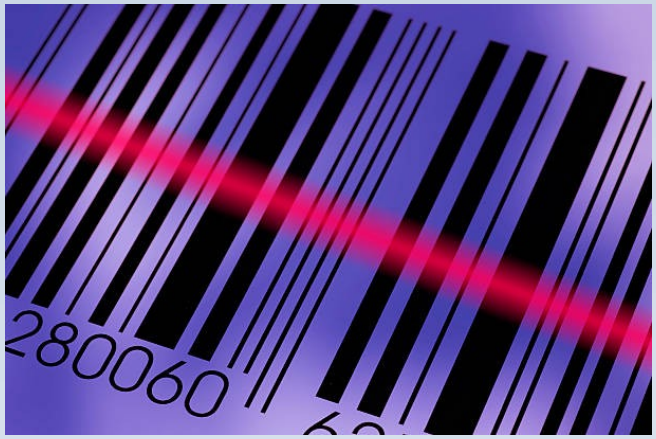
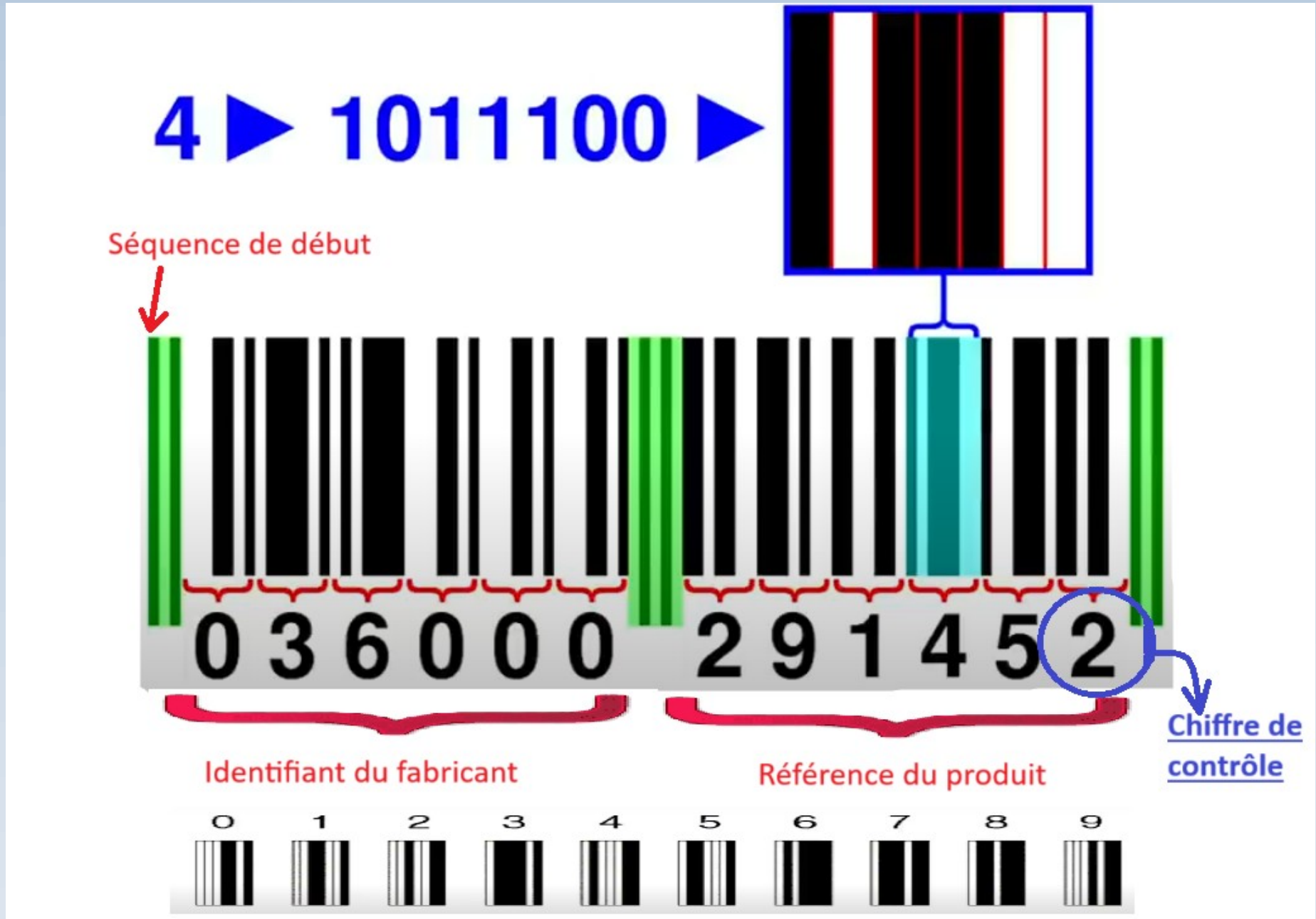


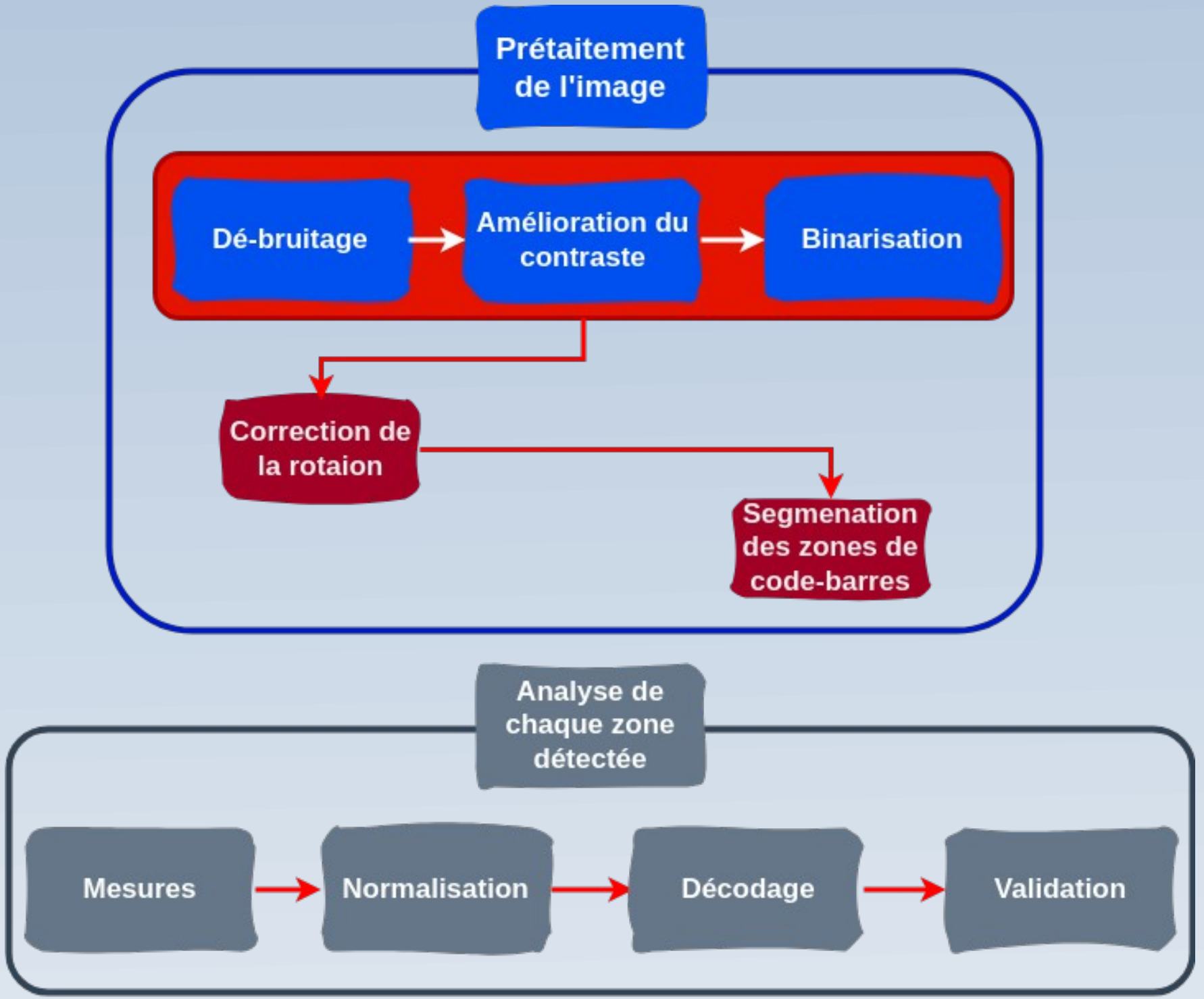
Fig1 : Exemple

pré-traitement

Fig2 : Modèle

La norme EAN-13




Prétaitement de l'image
Dé-bruitage
Amélioration du contraste
Binarisation
Correction de la rotaion
Segmenation des zones de code-barres
Analyse de chaque zone détectée
Mesures
Normalisation
Décodage
Validation

Dé-bruitage

128	140	115	140	130
140	255	140	109	155
115	133	155	109	115
130	155	140	115	109

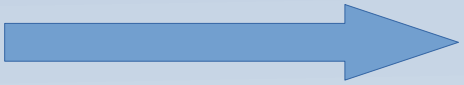
fenêtre

Median de: 128, 140, 115, 140, 255, 140, 115, 133, 155 → **140**

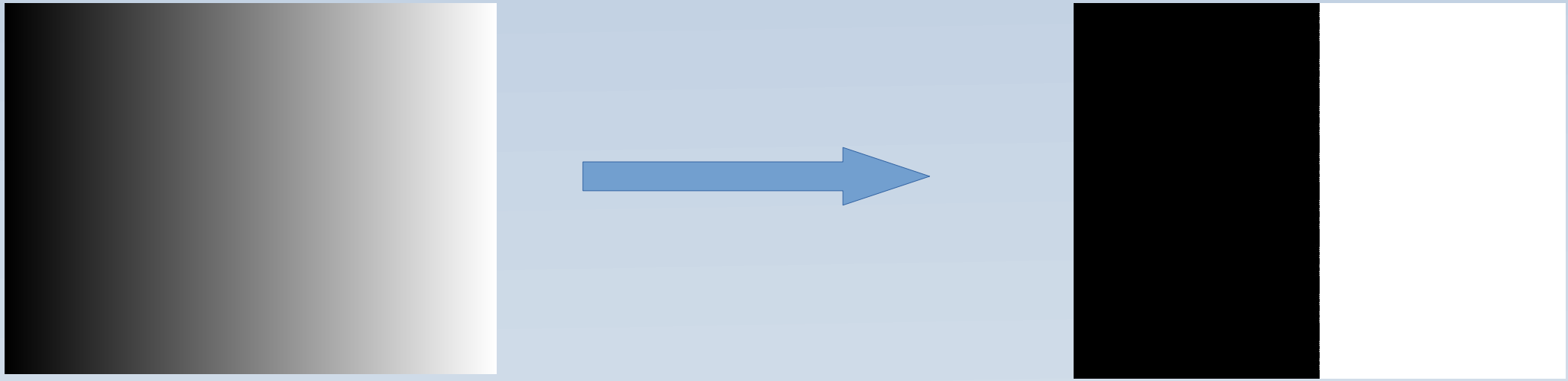
Filtre médian

128	140	115	140	130
140	140	140	109	155
115	133	155	109	115
130	155	140	115	109

Amélioration du contraste



Binarisation



Segmentation

Bruit, contraste, binarisation

Plan de l'annexe

Dé-bruitage

```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
```

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
```

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(gray_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        return black_and_white_img
```

Correction de la rotation

```python
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:
```

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

```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
```

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)) # 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
```