## ****TP3 : Détection de lignes Droites avec OpenCV et HoughVG****

## ****1. Convertir en niveaux de gris et flouter une image****

### Objectif :

* Passer d’une image couleur à une image en noir et blanc
* Appliquer un flou pour adoucir l’image

import cv2

# Charger les images

img\_url = r'I:\Mon Drive\ENSEIGNEMENT SUPERIEUR ODG\Informatique\Python\ESI Traitement Image\paysage.jpg'

image = cv2.imread(img\_url)  # Charger l'image

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # Convertir en niveaux de gris

blurred = cv2.GaussianBlur(gray, (5, 5), 0)  # Appliquer un flou gaussien

cv2.imshow("Grayscale", gray)

cv2.imshow("Blurred", blurred)

cv2.waitKey(0)

cv2.destroyAllWindows()

* cv2.cvtColor(...) : convertit l’image BGR (Bleu-Vert-Rouge) en niveaux de gris.
* cv2.GaussianBlur(...) : applique un flou en utilisant une matrice (ou noyau) 5x5.
  + (5, 5) est la taille du noyau.
  + 0 est l’écart-type du noyau (calculé automatiquement ici).

## 2. ****Détection de contours avec Canny****

### Objectif :

* Extraire les contours d’une image pour détecter des formes

import cv2

# Charger les images

img\_url = r'I:\Mon Drive\ENSEIGNEMENT SUPERIEUR ODG\Informatique\Python\ESI Traitement Image\paysage.jpg'

image = cv2.imread(img\_url)  # Charger l'image

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # Convertir en niveaux de gris

gauss = cv2.GaussianBlur(gray, (5, 5), 0)  # Appliquer un flou gaussien

edges = cv2.Canny(gauss, 50, 150)

cv2.imshow("Contours", edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

* cv2.Canny(image, seuil\_bas, seuil\_haut) :
  + Utilise la détection de contours de Canny.
  + 50 = seuil bas, 150 = seuil haut.
* Il faut flouter l’image avant, sinon trop de bruit est détecté.

## 3. ****Détection de droites avec OpenCV****

**Objectifs :**

* Charger une image
* Appliquer Canny pour détecter les contours
* Détecter les droites avec la transformée de Hough (cv2.HoughLines ou cv2.HoughLinesP)
* Afficher les droites détectées

**La transformée de Hough Probabiliste**

import cv2

import numpy as np

# Charger l'image

image = cv2.imread(r'./TP-ESI-Master2/immeuble.png')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Appliquer le flou pour réduire le bruit

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Détection de contours

edges = cv2.Canny(blurred, 50, 150)

# Détection de lignes avec la transformée de Hough probabiliste

lines = cv2.HoughLinesP(edges,

                        rho=1,             # Résolution de la distance en pixels

                        theta=np.pi/180,   # Résolution angulaire en radians

                        threshold=100,     # Seuil minimum d'accumulateur

                        minLineLength=50,  # Longueur minimale de ligne

                        maxLineGap=10)     # Distance max entre segments liés

# Dessiner les lignes détectées

if lines is not None:

    for line in lines:

        x1, y1, x2, y2 = line[0]

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

# Affichage

cv2.imshow("Image avec lignes", image)

cv2.imshow("Contours", edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

La transformée de Hough Standard

import cv2

import numpy as np

# Charger l'image

image = cv2.imread(r'./TP-ESI-Master2/immeuble.png')  # Remplace par le bon chemin

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Appliquer un flou pour réduire le bruit

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Détection des contours avec Canny

edges = cv2.Canny(blurred, 50, 150)

# Détection des lignes avec la Transformée de Hough standard

lines = cv2.HoughLines(edges,

                       rho=1,

                       theta=np.pi/180,

                       threshold=150)

# Tracer les lignes détectées

if lines is not None:

    for rho\_theta in lines:

        rho, theta = rho\_theta[0]

        a = np.cos(theta)

        b = np.sin(theta)

        x0 = a \* rho

        y0 = b \* rho

        # Convertir les coordonnées polaires en cartésiennes pour tracer

        x1 = int(x0 + 1000 \* (-b))

        y1 = int(y0 + 1000 \* (a))

        x2 = int(x0 - 1000 \* (-b))

        y2 = int(y0 - 1000 \* (a))

        # Tracer la ligne

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

# Affichage

cv2.imshow("Contours", edges)

cv2.imshow("Lignes détectées", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

* cv2.HoughLines donne les lignes au format (rho, theta), mais nécessite des conversions trigonométriques pour tracer les lignes.
* cv2.HoughLinesP (version probabiliste) donne directement les coordonnées des segments.

## 4. ****Détection de droites avec HoughVG (Hough transform on Virtual Grids)****

HoughVG est une boîte à outils de transformation de Hough basée sur une grille virtuelle pour la détection de lignes droites et la reconnaissance d'empreintes digitales. Elle regroupe plusieurs variantes innovantes de la transformation de Hough basée sur une grille virtuelle, notamment les transformations de Hough rectangulaires, triangulaires, hexagonales et octogonales pour la détection de lignes droites, ainsi que la transformation de Hough généralisée utilisant une grille rectangulaire virtuelle, spécialement adaptée à la reconnaissance d'empreintes digitales.

OS : Unix

La Transformée de Hough Octogonale

import cv2

import sys

import os

from HoughVG import HoughLine

# === Parameters ===

Rate = 0.30

gamma = 2

Threshold = 53

EdgesThresholdMax = 500

EdgesThresholdMin = 200

colors = (0, 255, 0)

# === Paths ===

# Full path to the image

image\_path = r'./TP-ESI-Master2/immeuble.png'

# Output folder for results

output\_folder = os.path.join(os.path.dirname(\_\_file\_\_), '..', 'results')

os.makedirs(output\_folder, exist\_ok=True)

# === Load the image ===

imge\_Build = cv2.imread(image\_path)

if imge\_Build is None:

    print("[ERROR] Image not found or unreadable:", image\_path)

    sys.exit(1)

else:

    print("Image loaded successfully:")

# === Preprocessing (Canny) ===

def ImgPreprocessing(imge, EdgesThresholdMin, EdgesThresholdMax):

    return cv2.Canny(imge, EdgesThresholdMin, EdgesThresholdMax, None, 3)

img = ImgPreprocessing(imge\_Build, EdgesThresholdMin, EdgesThresholdMax)

# === Application of OHT ===

accum, accum\_max, lines = HoughLine.Octogonal(img, gamma, Rate, Threshold)

# === Display  results ===

img2 = HoughLine.PlotHoughLine(imge\_Build, lines, colors)

cv2.imshow("Image affichée", img2)

cv2.waitKey(0)

cv2.destroyAllWindows()

# === Save results ===

filename\_base = f'THO{int(Rate\*100)}\_T{Threshold}\_G{gamma}\_C{EdgesThresholdMin}-{EdgesThresholdMax}'

cv2.imwrite(os.path.join(output\_folder, filename\_base + '\_accum.png'), accum)

cv2.imwrite(os.path.join(output\_folder, filename\_base + '\_accum\_max.png'), accum\_max)

cv2.imwrite(os.path.join(output\_folder, filename\_base + '\_lines.png'), img2)

print("Results saved in:", os.path.abspath(output\_folder))

**Paramètres :**

* **Rate** : Taux minimal de pixels actifs d’un cellule active
* **gamma** : paramètre definissant la taille des cellules octogonales
* **Threshold** : Seuil de détection (nombre minimum de votes)
* **EdgesThresholdMax** : Seuil max pour Canny
* **EdgesThresholdMin** : Seuil min pour Canny

**Valeurs retour :**

* **accum** : accumulateur complet
* **accum**\_**max** : carte des maxima locaux
* **lines** : liste des coordonnées polaires des lignes détectées