Assignment-3

Hough Transform

By Harsh Sharma

1. Code with comments

import cv2

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.lines as mlines

# line detection method in vectorized format

def line\_detection\_vectorized(image, edge\_image, num\_rhos=180, num\_thetas=180, t\_count=220):

# Get the dimensions of the edge image

edge\_height, edge\_width = edge\_image.shape[:2]

edge\_height\_half, edge\_width\_half = edge\_height / 2, edge\_width / 2

# Calculate the maximum distance from the origin to the image corners

d = np.sqrt(np.square(edge\_height) + np.square(edge\_width))

# Calculate the increments of theta and rho

dtheta = 180 / num\_thetas

drho = (2 \* d) / num\_rhos

# Generate arrays of theta and rho values

thetas = np.arange(0, 180, step=dtheta)

rhos = np.arange(-d, d, step=drho)

# Precompute cosine and sine values of theta

cos\_thetas = np.cos(np.deg2rad(thetas))

sin\_thetas = np.sin(np.deg2rad(thetas))

# Create an accumulator array to store Hough transform values

accumulator = np.zeros((len(rhos), len(rhos)))

# Create a figure and subplots for visualization

figure = plt.figure(figsize=(12, 12))

subplot1 = figure.add\_subplot(1, 4, 1)

subplot2 = figure.add\_subplot(1, 4, 2)

subplot3 = figure.add\_subplot(1, 4, 3)

subplot4 = figure.add\_subplot(1, 4, 4)

# Display the original image and edge image

subplot1.imshow(image)

subplot2.imshow(edge\_image, cmap="gray")

# Set the background color of the Hough space subplot

subplot3.set\_facecolor((0, 0, 0))

# Display the original image in the last subplot

subplot4.imshow(image)

# Adjust edge points based on the center of the image

edge\_points = np.argwhere(edge\_image != 0)

edge\_points = edge\_points - np.array([[edge\_height\_half, edge\_width\_half]])

# Compute rho values for edge points

rho\_values = np.matmul(edge\_points, np.array([sin\_thetas, cos\_thetas]))

# Perform 2D histogram accumulation

accumulator, theta\_vals, rho\_vals = np.histogram2d(

np.tile(thetas, rho\_values.shape[0]),

rho\_values.ravel(),

bins=[thetas, rhos]

)

accumulator = np.transpose(accumulator)

# Find lines with sufficient votes in the accumulator

lines = np.argwhere(accumulator > t\_count)

rho\_idxs, theta\_idxs = lines[:, 0], lines[:, 1]

r, t = rhos[rho\_idxs], thetas[theta\_idxs]

# Plot the detected lines in Hough space

for ys in rho\_values:

subplot3.plot(thetas, ys, color="white", alpha=0.05)

subplot3.plot([t], [r], color="yellow", marker='o')

# Plot the detected lines on the original image

for line in lines:

y, x = line

rho = rhos[y]

theta = thetas[x]

a = np.cos(np.deg2rad(theta))

b = np.sin(np.deg2rad(theta))

x0 = (a \* rho) + edge\_width\_half

y0 = (b \* rho) + edge\_height\_half

x1 = int(x0 + 1000 \* (-b))

y1 = int(y0 + 1000 \* (a))

x2 = int(x0 - 1000 \* (-b))

y2 = int(y0 - 1000 \* (a))

subplot3.plot([theta], [rho], marker='o', color="yellow")

subplot4.add\_line(mlines.Line2D([x1, x2], [y1, y2]))

# Invert y-axis and x-axis for Hough space plot

subplot3.invert\_yaxis()

subplot3.invert\_xaxis()

# Set titles for the subplots

subplot1.title.set\_text("Original Image")

subplot2.title.set\_text("Edge Image")

subplot3.title.set\_text("Hough Space")

subplot4.title.set\_text("Detected Lines")

# Show the plot

plt.show()

# Return the accumulator, rhos, and thetas for further analysis if needed

return accumulator, rhos, thetas

if \_\_name\_\_ == "\_\_main\_\_":

# Process three images for line detection

for i in range(1):

# Read an image file

image = cv2.imread(f"circle.jpg")

# Convert the image to grayscale

edge\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian blur to reduce noise

edge\_image = cv2.GaussianBlur(edge\_image, (3, 3), 1)

# Apply Canny edge detection

edge\_image = cv2.Canny(edge\_image, 100, 200)

# Dilate the edges for better line detection

edge\_image = cv2.dilate(

edge\_image,

cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5)),

iterations=1

)

# Erode the edges to reduce thickness

edge\_image = cv2.erode(

edge\_image,

cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5)),

iterations=1

)

# Perform line detection using the vectorized method

line\_detection\_vectorized(image, edge\_image)

1. **Output - Four images for showing results**
2. Line

**A picture containing text, line, screenshot, diagram

Description automatically generated**

**A picture containing screenshot, text, diagram

Description automatically generated**

1. Pentagon

**Chart, shape, polygon

Description automatically generated**

**Chart

Description automatically generated**

1. Star

**Shape, polygon

Description automatically generated**

**Chart

Description automatically generated**

1. Circle

**A picture containing screenshot, circle, diagram, text

Description automatically generated**

**A picture containing text, screenshot, rectangle, diagram

Description automatically generated**

1. **Hough transform code using OpenCV and Hough Line method**

import cv2

import numpy as np

img = cv2.imread('dave.jpg')

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

edges = cv2.Canny(gray,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,200)

for rho,theta in lines[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(img,(x1,y1),(x2,y2),(0,0,255),2)

cv2.imwrite('houghlines.jpg',img)

**Analysis**: The coded version of Hough lines is able to define lines in image with more accuracy. The difference between traditional and OpenCV method is **time complexity** and Space complexity.