**Homework 3**

**Hough Transform**

**Sahithi Muppiri**

**Hough Line Transform:**

This is an algorithm to detect lines in an image.

**Input Images:**

A blue car with a white background

Description automatically generated with medium confidence A black and white checkered pattern

Description automatically generated with low confidence A large metal tower

Description automatically generated with low confidence

**Implementation using OpenCV functions:**

**Two kinds of Hough line transform:**

1. Standard Hough transform

 It gives you as result a vector of couples (θ,rθ)

1. Probabilistic Hough transform

It gives as output the extremes of the detected lines (x0,y0,x1,y1)

**Code:**

Using both cv2.HoughLines() and cv2.HoughLinesP

import math

import cv2 as cv

from matplotlib import image

import numpy as np

# Loads an image

img = cv.imread('image1.jpg', cv.IMREAD\_GRAYSCALE)

canny\_image = cv.Canny(img, 50, 200, None, 3)

# Copy edges to the images that will display the results in BGR

standard\_ht = cv.cvtColor(canny\_image, cv.COLOR\_GRAY2BGR)

Probabilistic\_ht = np.copy(standard\_ht)

lines = cv.HoughLines(canny\_image, 1, np.pi / 180, 150, None, 0, 0)

if lines is not None:

    for i in range(0, len(lines)):

        rho = lines[i][0][0]

        theta = lines[i][0][1]

        a = math.cos(theta)

        b = math.sin(theta)

        x0 = a \* rho

        y0 = b \* rho

        pt1 = (int(x0 + 1000\*(-b)), int(y0 + 1000\*(a)))

        pt2 = (int(x0 - 1000\*(-b)), int(y0 - 1000\*(a)))

        cv.line(standard\_ht, pt1, pt2, (0,0,255), 3, cv.LINE\_AA)

linesP = cv.HoughLinesP(canny\_image, 1, np.pi / 180, 50, None, 50, 10)

if linesP is not None:

    for i in range(0, len(linesP)):

        l = linesP[i][0]

        cv.line(Probabilistic\_ht, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv.LINE\_AA)

cv.imshow("Source", img)

cv.imshow("Standard Hough Line Transform", standard\_ht)

cv.imshow("Probabilistic Line Transform", Probabilistic\_ht)

cv.waitKey()

**Output:**

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**Code for Hough Line detection without using OpenCV:**

**Code**

from matplotlib import pyplot as plt

import numpy as np

import math

import cv2

def hough\_line(edge):

    # Theta ranging from 0 - 180 degree

    theta = np.arange(0, 180, 1)

    cos = np.cos(np.deg2rad(theta))

    sin = np.sin(np.deg2rad(theta))

    # Generate a accumulator matrix to store the values

    rho\_range = round(math.sqrt(edge.shape[0]\*\*2 + edge.shape[1]\*\*2))

    accumulator = np.zeros((2 \* rho\_range, len(theta)), dtype=np.uint8)

    # Get threshold to get edges pixel location along x, y

    edge\_pixels = np.where(edge == 255)

    coordinates = list(zip(edge\_pixels[0], edge\_pixels[1]))

    # Determine the rho value for the edge location along x, y with all the theta range

    for p in range(len(coordinates)):

        for t in range(len(theta)):

            rho = int(round(coordinates[p][1] \* cos[t] + coordinates[p][0] \* sin[t]))

            accumulator[rho, t] += 2 # Suppose add 1 only, Just want to get clear result

    return accumulator

# Read image, Conver to grey scale, Find the edges using canny edge detection

image = cv2.imread('image4.png')

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

edges = cv2.Canny(grayscale,50, 200, None, 3)

# Calculate and perform the hough line detection

accumulator = hough\_line(edges)

# Set the threshold and draw the lines for the same image

edge\_pixels = np.where(accumulator > 110)

coordinates = list(zip(edge\_pixels[0], edge\_pixels[1]))

# Line equation is used to draw the lines which are detected on the original image

for i in range(0, len(coordinates)):

    a = np.cos(np.deg2rad(coordinates[i][1]))

    b = np.sin(np.deg2rad(coordinates[i][1]))

    x0 = a\*coordinates[i][0]

    y0 = b\*coordinates[i][0]

    x1 = int(x0 + 1000\*(-b))

    y1 = int(y0 + 1000\*(a))

    x2 = int(x0 - 1000\*(-b))

    y2 = int(y0 - 1000\*(a))

    cv2.line(image,(x1,y1),(x2,y2),(0,255,0),1)

# display the result

plt.title('Hough Line'), plt.imshow(image)

plt.show()

**Output:**

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**Voting:**

import cv2

import numpy as np

import imageio

import math

import matplotlib.pyplot as plt

import cv2 as cv

def fast\_hough\_line(img, angle\_step=1, lines\_are\_white=True, value\_threshold=5):

    # Rho and Theta ranges

    thetas = np.deg2rad(np.arange(-90.0, 90.0, angle\_step)) #can be changed

    #width, height = col.size  #if we use pillow

    width, height = img.shape

    diag\_len = int(np.ceil(np.sqrt(width \* width + height \* height)))   # max\_dist

    rhos = np.linspace(-diag\_len, diag\_len, diag\_len \* 2)

    # Cache some resuable values

    cos\_theta = np.cos(thetas)

    sin\_theta = np.sin(thetas)

    num\_thetas = len(thetas)

    # Hough accumulator array of theta vs rho

    accumulator = np.zeros((2 \* diag\_len, num\_thetas))

    are\_edges = img > value\_threshold if lines\_are\_white else img < value\_threshold

    #are\_edges = cv2.Canny(img,50,150,apertureSize = 3)

    y\_idxs, x\_idxs = np.nonzero(are\_edges)  # (row, col) indexes to edges

    # Vote in the hough accumulator

    xcosthetas = np.dot(x\_idxs.reshape((-1,1)), cos\_theta.reshape((1,-1)))

    ysinthetas = np.dot(y\_idxs.reshape((-1,1)), sin\_theta.reshape((1,-1)))

    rhosmat = np.round(xcosthetas + ysinthetas) + diag\_len

    rhosmat = rhosmat.astype(np.int16)

    for i in range(num\_thetas):

        rhos,counts = np.unique(rhosmat[:,i], return\_counts=True)

        accumulator[rhos,i] = counts

    return accumulator, thetas, rhos

def show\_hough\_line(img, accumulator, thetas, rhos):

    plt.imshow(accumulator, cmap='gray', extent=[np.rad2deg(thetas[-1]), np.rad2deg(thetas[0]), rhos[-1], rhos[0]])

    plt.title('Hough transform')

    plt.xlabel('Angles degree')

    plt.ylabel('Distance in pixels')

    plt.axis('image')

    plt.show()

image = cv.imread('image1.jpg')

greying = cv.cvtColor(image, cv2.COLOR\_BGR2GRAY)

bluring = cv.GaussianBlur(greying, (5, 5), 0)

edgeing = cv2.Canny(bluring, 50, 150)

cv2.imshow("image", image)

cv2.waitKey(0)

acc, theta, rho = fast\_hough\_line(edgeing)

show\_hough\_line(image, acc, theta, rho)

Voting for image 1:

A blue car with a white background

Description automatically generated with medium confidenceChart

Description automatically generated

Voting for image 2:

A black and white checkered pattern

Description automatically generated with low confidenceChart

Description automatically generated

Voting for image 3:

A large metal tower

Description automatically generated with low confidenceGraphical user interface

Description automatically generated

Comparison between Hough transform using OpenCV and without using any functions:

Using the cv2.HoughLines which is a standard approach to detect the lines in an image, uses a one single command line to give the input which is of 9 parameters. This function finds the lines in a binary image. With the use of the OpenCV algorithm we use Polar system. With the use of (r, θ).

With the custom-built function, it is a 5-step process. Which is:

1. Read the image, convert to grey scale.
2. Apply the canny edge algorithm to detect the edges of the image.
3. Initialize the accumulator to 0 and generate a matrix by theta and rho.
4. Convert the x, y coordinate system to polar coordinate system.
5. Use the threshold to select the line.

The time complexity for running either of the code, we can see a good computation time with the use of the OpenCV method. However, the current implementation can further be improved with the selection of the points which can be only responsible for line detection instead of randomly selecting the points.

**Non-maximum suppression**

Non-maximum suppression is the process of local maxima of the magnitude of the gradient, in the direction of gradient in the image.

This is used to predict the next best possible line.

**Code:**

from matplotlib import pyplot as plt

import numpy as np

import math

import cv2

import copy

def hough\_line(edge):

    # Theta ranging from 0 - 180 degree

    theta = np.arange(0, 180, 1)

    cos = np.cos(np.deg2rad(theta))

    sin = np.sin(np.deg2rad(theta))

    # Generate a accumulator matrix to store the values

    rho\_range = round(math.sqrt(edge.shape[0]\*\*2 + edge.shape[1]\*\*2))

    accumulator = np.zeros((2 \* rho\_range, len(theta)), dtype=np.uint8)

    # Get threshold to get edges pixel location along x, y

    edge\_pixels = np.where(edge == 255)

    coordinates = list(zip(edge\_pixels[0], edge\_pixels[1]))

    # Determine the rho value for the edge location along x, y with all the theta range

    for p in range(len(coordinates)):

        for t in range(len(theta)):

            rho = int(round(coordinates[p][1] \* cos[t] + coordinates[p][0] \* sin[t]))

            accumulator[rho, t] += 2 # Suppose add 1 only, Just want to get clear result

    return accumulator

def NMS(lines, overlapThresh = 0.4):

    #return an empty list, if no boxes given

    if len(lines) == 0:

        return []

    x1 = lines[:, 0]  # x coordinate of the top-left corner

    y1 = lines[:, 1]  # y coordinate of the top-left corner

    x2 = lines[:, 2]  # x coordinate of the bottom-right corner

    y2 = lines[:, 3]  # y coordinate of the bottom-right corner

    # compute the area of the bounding boxes and sort the bounding

    # boxes by the bottom-right y-coordinate of the bounding box

    areas = (x2 - x1 + 1) \* (y2 - y1 + 1) # We have a least a box of one pixel, therefore the +1

    indices = np.arange(len(x1))

    for i,box in enumerate(lines):

        temp\_indices = indices[indices!=i]

        xx1 = np.maximum(box[0], lines[temp\_indices,0])

        yy1 = np.maximum(box[1], lines[temp\_indices,1])

        xx2 = np.minimum(box[2], lines[temp\_indices,2])

        yy2 = np.minimum(box[3], lines[temp\_indices,3])

        w = np.maximum(0, xx2 - xx1 + 1)

        h = np.maximum(0, yy2 - yy1 + 1)

        # compute the ratio of overlap

        overlap = (w \* h) / areas[temp\_indices]

        if np.any(overlap) > treshold:

            indices = indices[indices != i]

    return lines[indices].astype(int)

def possible\_line(image, template):

    (tH, tW) = template.shape[:2]             # getting height and width of template

    imageGray = cv2.cvtColor(image, 0)        # convert the image to grayscale

    templateGray = cv2.cvtColor(template, 0)  # convert the template to grayscale

    result = cv2.matchTemplate(imageGray, templateGray, cv2.TM\_CCOEFF\_NORMED)  # template matching return the correlatio

    (y1, x1) = np.where(result >= treshold)  # object is detected, where the correlation is above the treshold

    lines = np.zeros((len(y1), 4))      # construct array of zeros

    x2 = x1 + tW                       # calculate x2 with the width of the template

    y2 = y1 + tH                       # calculate y2 with the height of the template

    # fill the bounding boxes array

    lines[:, 0] = x1

    lines[:, 1] = y1

    lines[:, 2] = x2

    lines[:, 3] = y2

    return lines.astype(int)

def draw\_lines(image,im):

    for line in im:

        image = cv2.line(copy.deepcopy(image),line[:2], line[2:], (255,0,0), 3)

    return image

treshold = 0.8837 # the correlation treshold, in order for an object to be recognised

image1 = plt.imread(r"im1.jpg")

image2 = plt.imread(r"image1.jpg")

lines\_redundant = possible\_line(image2, image1) # calculate bounding boxes

lines = NMS(lines\_redundant)                                            # remove redundant bounding boxes

overlapping\_line\_image = draw\_lines(image2,

                                            lines\_redundant)  # draw image with all redundant bounding boxes

segmented\_image = draw\_lines(image2,lines)           # draw the bounding boxes onto the image

plt.imshow(overlapping\_line\_image)

plt.show()

**Output:**

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

The line detects the possible proper line in the image using non maximum suppression.