**ECE 5554 COMPUTER VISION HOMEWORK 3**

**ROHIT MURALI**

**SOLUTION 1**

**CODE:**

import numpy as np

import cv2

import math

pathName = "C:\Data\Software\ECE5554 FA19 HW3 images\VAoutline.png"

MAXCONTOUR = 5000

doLogging = False

def showImage(img, name):

cv2.imshow(name, img)

def saveImage(img, name):

cv2.imwrite(pathName + name + ".png", img)

def GaussArea(pts):

n = len(pts)

area = 0.0

for i in range(n):

j = (i + 1) % n

area += pts[i][0] \* pts[j][1] - pts[j][0] \* pts[i][1]

area = abs(area) / 2.0

return area

def length(contour, i):

size = contour.shape[0]

if i == size - 1:

l = math.sqrt((contour[i,0] - contour[0,0])\*\*2 + (contour[i,1] - contour[0,1])\*\*2)

else:

l = math.sqrt((contour[i,0] - contour[i-1,0])\*\*2 + (contour[i,1] - contour[i - 1,1])\*\*2)

return l

def theta(contour, i):

size = contour.shape[0]

if i == size - 1:

theta = abs(math.atan((contour[i,0] - contour[i-1,0])/(contour[i,1] - contour[i-1,1])) - math.atan((contour[0,0] - contour[i,0])/(contour[0,1] - contour[i,1])))

else:

theta = abs(math.atan((contour[i,0] - contour[i-1,0])/(contour[i,1] - contour[i-1,1])) - math.atan((contour[i+1,0] - contour[i,0])/(contour[i+1,1] - contour[i,1])))

return theta

def onePassDCE(contour):

size = contour.shape[0]

rele = []

for i in range(size):

if i < size - 1:

rel = (theta(contour, i)\*length(contour,i)\*length(contour,i+1))/(length(contour,i) + length(contour,i+1))

if i == size - 1:

rel = (theta(contour, i)\*length(contour,i)\*length(contour,0))/(length(contour,i) + length(contour,0))

rele.append(rel)

rele = np.asarray(rele)

rele = np.reshape(rele,(size,1))

a = min(rele)

index = np.argmin(rele)

trimmed\_contour = np.delete(contour,idx,0)

return trimmed\_contour

def getPoint(direction, a):

if direction == "up":

d = [(a[0] - 1,a[1] - 1),(a[0] - 1,a[1]),(a[0] - 1,a[1] + 1)]

elif direction == "down":

d = [(a[0] + 1,a[1] + 1),(a[0] + 1,a[1]),(a[0] + 1,a[1] - 1)]

elif direction == "left":

d = [(a[0] + 1,a[1] - 1),(a[0] ,a[1] - 1),(a[0] - 1,a[1] - 1)]

elif direction == "right":

d = [(a[0] - 1,a[1] + 1),(a[0] ,a[1] + 1),(a[0] + 1,a[1] + 1)]

return d

def getNextDirection(current\_direction, i):

if i == 0:

if current\_direction == "up":

nextDirection = "left"

elif current\_direction == "down":

nextDirection = "right"

elif current\_direction == "left":

nextDirection = "down"

elif current\_direction == "right":

nextDirection = "up"

else:

nextDirection = current\_direction

return nextDirection

def getNewDirection(current\_direction):

if current\_direction == "up":

nextDirection = "right"

elif current\_direction == "down":

nextDirection = "left"

elif current\_direction == "left":

nextDirection = "up"

elif current\_direction == "right":

nextDirection = "down"

return nextDirection

def Pavlidis(img, start):

contour\_point = []

count = 0

c = 0

direction = "up"

point = start

while True:

front\_pixel = getPoint(direction,point)

for i in range(len(front\_pixel)):

if binary[front\_pixel[i]] > 0:

contour\_point.append(front\_pixel[i])

count = count + 1

point = front\_pixel[i]

direction = getNextDirection(direction, i)

break

if count == 0:

direction = getNewDirection(direction)

if point == start:

break

count = 0

print(len(contour\_point))

return contour\_point

def showContour(ctr, img, name):

contourImage = img

length = ctr.shape[0]

row, col = img.shape

contour = np.zeros((row,col))

for count in range(length):

contourImage[ctr[count, 0], ctr[count, 1]] = 255

contour[ctr[count, 0], ctr[count, 1]] = 255

cv2.line(contour,(ctr[count, 1], ctr[count, 0]), \

(ctr[(count+1)%length, 1], ctr[(count+1)%length, 0]),(128,128,128),1)

cv2.line(contourImage,(ctr[count, 1], ctr[count, 0]), \

(ctr[(count+1)%length, 1], ctr[(count+1)%length, 0]),(128,128,128),1)

showImage(contourImage, name)

showImage(contour, "Detected")

saveImage(contourImage, name)

inputImage = cv2.imread(pathName, cv2.IMREAD\_GRAYSCALE)

thresh = 70

ret, binary = cv2.threshold(inputImage, thresh, 255, cv2.THRESH\_BINARY)

(height, width) = binary.shape

ystt = np.uint8(height/2)

for xstt in range(width):

if (binary[ystt, xstt] > 0):

break

start = (ystt, xstt)

contour = Pavlidis(binary, start)

contour = np.asarray(contour)

showContour(contour, inputImage, "CONTOUR")

print(contour, GaussArea(contour))

for step in range(6):

numLoops = math.floor(contour.shape[0]/2)

for idx in range(numLoops):

contour = onePassDCE(contour)

showContour(contour, np.zeros\_like(inputImage), "STEP"+str(step))

print(numLoops, contour.shape, GaussArea(contour))

cv2.waitKey(0)

TRACED CONTOUR



DISCRETE CONTOUR EVOLUTION OUTPUT

A picture containing dark

Description automatically generatedA display in a dark room

Description automatically generatedA picture containing indoor

Description automatically generatedA picture containing indoor, sitting

Description automatically generatedA picture containing indoor, sitting, looking

Description automatically generatedA picture containing indoor

Description automatically generated

AREA OF CONTOUR FOR 6 DIFFERENT STEPS

STEP 1 : AREA = 68307.0

STEP 2 : AREA = 68312.5

STEP 3 : AREA = 68337.5

STEP 4 : AREA = 68457.0

STEP 5 : AREA = 68211.5

STEP 6 : AREA = 69156.5

**SOLUTION 2**

**CODE:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from random import randrange

import math

import scipy.signal

def cross\_image(im1, im2):

im1\_gray = np.sum(im1.astype('float'), axis=2)

im2\_gray = np.sum(im2.astype('float'), axis=2)

im1\_gray -= np.mean(im1\_gray)

im2\_gray -= np.mean(im2\_gray)

return scipy.signal.fftconvolve(im1\_gray, im2\_gray[::-1,::-1], mode='same')

def stitch(img1, img2):

img1\_gray = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

img2\_gray = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

descriptor = cv2.ORB\_create()

kp1, des1 = descriptor.detectAndCompute(img1\_gray,None)

kp2, des2 = descriptor.detectAndCompute(img2\_gray,None)

bf = cv2.BFMatcher()

matches = bf.knnMatch(des1, des2, k=2)

good = []

for m in matches:

if m[0].distance < 0.5\*m[1].distance:

good.append(m)

matches = np.asarray(good)

if len(matches[:,0]) >= 4:

src = np.float32([ kp1[m.queryIdx].pt for m in matches[:,0] ]).reshape(-1,1,2)

dst = np.float32([ kp2[m.trainIdx].pt for m in matches[:,0] ]).reshape(-1,1,2)

H, masked = cv2.findHomography(src, dst, cv2.RANSAC, 5.0)

dst = cv2.warpPerspective(img1,H,(img2.shape[1] + img1.shape[1], img2.shape[0]))

dst[0:img2.shape[0], 0:img2.shape[1]] = img2

img3 = cv2.cvtColor(dst, cv2.COLOR\_BGR2GRAY)

row, col = img3.shape

a = 0

i = col - 1

while True:

if img3[int(row\*0.5),i] != 0:

a = i

break

i = i - 1

dst = dst[0:row, 0:a + 1]

return dst

img\_ = cv2.imread("C:\Data\Software\ECE5554 FA19 HW3 images\Goodwin1.png")

img = cv2.imread("C:\Data\Software\ECE5554 FA19 HW3 images\Goodwin0.png")

img\_p = cv2.imread("C:\Data\Software\ECE5554 FA19 HW3 images\Goodwin2.png")

c = stitch(img\_, img)

cv2.imwrite("output.png",c)

d = stitch(img\_p, img\_)

cv2.imwrite("output1.png",d)

final1 = cv2.imread("output1.png")

final2 = cv2.imread("output.png")

e = stitch(final1, final2)

cv2.imwrite("output2.png",e)

print("Displacement between first and second image")

a = cross\_image(img, img\_)

b = np.unravel\_index(np.argmax(a), a.shape)

print(b[1])

print("Displacement between merged image and third image")

a = cross\_image(c, img\_p)

b = np.unravel\_index(np.argmax(a), a.shape)

print(b[1])

**RESULTS:**

**FOR GOODWIN IMAGE SET**

FIRST & SECOND IMAGE MERGED

A city street filled with lots of traffic

Description automatically generated

ALL 3 IMAGES MERGED A view of a city

Description automatically generated

DISPLACEMENT BETWEEN IMAGES

A close up of a screen

Description automatically generated

The column displacement between the pairs of images is 1012 between first and second image and 524 between merged and third image.

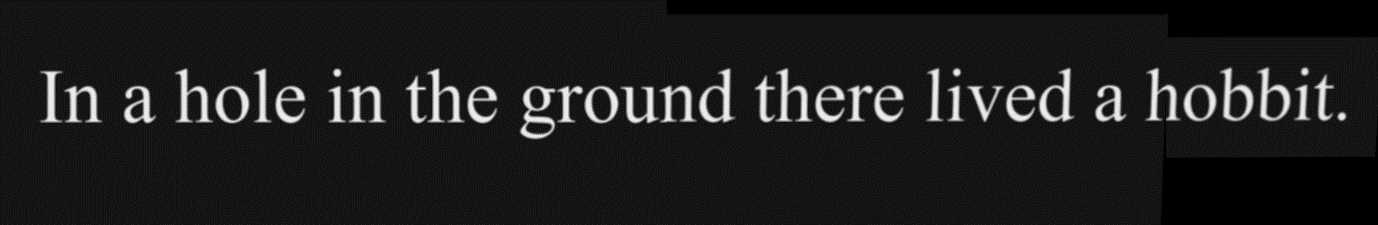
**FOR HOBBIT IMAGE SET**

FIRST & SECOND IMAGE MERGED

**A close up of a logo

Description automatically generated**

ALL 3 IMAGES MERGED



DISPLACEMENT BETWEEN IMAGES

A close up of a screen

Description automatically generated

The column displacement between the pairs of images is 496 between first and second image and 1236 between merged and third image.

**FOR BIGFOUR IMAGE SET**

FIRST & SECOND IMAGE MERGED

**A view of a snow covered mountain

Description automatically generated**

ALL 3 IMAGES MERGED

A blurry photo of a snow covered mountain

Description automatically generated

DISPLACEMENT BETWEEN IMAGES

A close up of a screen

Description automatically generated

The column displacement between the pairs of images is 1085 between first and second image and 1469 between merged and third image.

**Reference** : <https://towardsdatascience.com/image-stitching-using-opencv-817779c86a83>