CVE Assignment 4 Report

# Question 1 (PS4-1)

## Algorithm and Observations

The objective of the program was to stitch together three images after transforming them to align 4 common features on each image for each stitch. This involved selecting the common features manually on each image and using them to perform perspective transformation of the side images with reference to the centre image being stationary, and combining them. The results of picking the common features were stored in a json file in the same directory to reuse the same points while testing any changes in the source code.

The following algorithm was used to perform the task:

1. Ask the user for the dataset of the images to be used. Import the left, center and right images of that dataset.
2. If an existing datapoint set exists, choose to use them, or make new selections.
3. Choose 4 points on the right image (highlighted by red); once satisfied, confirm the selection, and repeat the process for the central image.
4. Repeat the process for the left image and the central image pair (features highlighted in blue).
5. Generate a black surface and place the centre image in the middle of the surface. Use perspective transformation to change the image and place it in the appropriate position of the black surface.
6. For pixels with input from multiple overlapping images, apply a weight at each input to normalize the values to 1.
7. Display and save the final image.

Observation from the images:

The images given along with the question were implemented in the code to create their stitiched equivalents. The following observations were made while performing the operations.

1. It was relatively easy to apply this image mosacing technique on images with a smaller number of distinctive feature like the “wall” dataset.
2. Geometric features like lines and grids were useful in testing the efficiency of the feature choice for that dataset (“door” dataset).
3. Features selected over a wider area and in a rough approximation of the targeting perspective transform yielded better results.
4. Images taken over a wider angle were more difficult to stitch without creating distinctive borders because of the variation in the brightness of the images (“house” dataset).
5. It was difficult to prevent overlap blurring in larger images with several distinctive features like the “pittsburgh” dataset. Instead of taking standard weigths for each pixel, weights created in favour of each image could have reduced the blurring in the images, giving a smoother transformation.
6. It was difficult to accurately select feature points for large images owing to human error while making the selection of the features with a mouse.

## Results

A picture containing ledge, concrete, cement

Description automatically generated

wall-stiched.png



door-stiched.png

A picture containing tree, sky, grass, outdoor

Description automatically generated

house-stiched.png



pittsburgh-stiched.png

## Source Code

# import the necessary packages

import cv2

import numpy as np

import json

def savePick():

global pick

data = {}

data["pick"] = pick

with open("JSON/"+f+'-result.json', 'w') as outfile:

json.dump(data, outfile)

def loadPick():

global pick

with open("JSON/"+f+'-result.json') as file:

data = json.load(file)

pick = data["pick"]

def combine():

global result, imageC, imageL, imageR, pick

(h,w) = imageC.shape[:2]

cng = cv2.cvtColor(result, cv2.COLOR\_BGR2GRAY)

th, mask\_c = cv2.threshold(cng, 1, 255, cv2.THRESH\_BINARY)

mask\_c = mask\_c / 255

# right

src\_pnts = np.empty([4,2], np.float32)

dst\_pnts = np.empty([4,2], np.float32)

for i in range(4):

src\_pnts[i][0] = float(pick[0][i][0])

src\_pnts[i][1] = float(pick[0][i][1])

dst\_pnts[i][0] = float(pick[1][i][0]+w)

dst\_pnts[i][1] = float(pick[1][i][1]+h)

M = cv2.getPerspectiveTransform(src\_pnts, dst\_pnts)

rn = cv2.warpPerspective(imageR, M, (w\*3,h\*3))

rng = cv2.cvtColor(rn, cv2.COLOR\_BGR2GRAY)

th, mask\_r = cv2.threshold(rng, 1, 255, cv2.THRESH\_BINARY)

mask\_r = mask\_r / 255

#left

src\_pnts = np.empty([4,2], np.float32)

dst\_pnts = np.empty([4,2], np.float32)

for i in range(4):

src\_pnts[i][0] = float(pick[2][i][0])

src\_pnts[i][1] = float(pick[2][i][1])

dst\_pnts[i][0] = float(pick[3][i][0]+w)

dst\_pnts[i][1] = float(pick[3][i][1]+h)

M = cv2.getPerspectiveTransform(src\_pnts, dst\_pnts)

ln = cv2.warpPerspective(imageL, M, (w\*3,h\*3))

lng = cv2.cvtColor(ln, cv2.COLOR\_BGR2GRAY)

th, mask\_l = cv2.threshold(lng, 1, 255, cv2.THRESH\_BINARY)

mask\_l = mask\_l / 255

mask = np.array(mask\_c + mask\_l + mask\_r, float)

ag = np.full(mask.shape, 0.0, dtype=float)

# weight: 1.0 / (num of picture)

ag = 1.0 / np.maximum(1,mask) # avoid 0 division

# generate result image from 3 images + alpha weight

result[:,:,0] = result[:,:,0]\*ag[:,:] + ln[:,:,0]\*ag[:,:] + rn[:,:,0]\*ag[:,:]

result[:,:,1] = result[:,:,1]\*ag[:,:] + ln[:,:,1]\*ag[:,:] + rn[:,:,1]\*ag[:,:]

result[:,:,2] = result[:,:,2]\*ag[:,:] + ln[:,:,2]\*ag[:,:] + rn[:,:,2]\*ag[:,:]

if dataset == 4:

cv2.imwrite("ps4-images/Test/"+f+"-stiched.png", result)

else:

cv2.imwrite("ps4-images/"+f+"-stiched.png", result)

cv2.imshow("result", result)

'''

pick 4 points from right image (red point)

'''

def right\_click(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONUP:

mousePick(x, y, 0)

'''

pick 4 points from center (correspond to right, red point)

'''

def center\_click\_r(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONUP:

mousePick(x, y, 1)

'''

pick 4 points from left (blue point)

'''

def left\_click(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONUP:

# add your code to select 4 points

mousePick(x, y, 2)

'''

pick 4 points from center (correspond to left, blue point)

'''

def center\_click\_l(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONUP:

# add your code to select 4 points

mousePick(x, y, 3)

'''

idea: handle mouse pick

idx

0: right

1: center (correspond to right)

2: left

3: center (correspond to left)

you can also create your own function for left + center selection

'''

def mousePick(x, y, idx):

global rn, cn, ln, imageR, imageC, imageL, pick

if idx == 0:

src = imageR

dst = rn

wn = "right"

elif idx == 1:

src = imageC

dst = cn

wn = "center"

# you need to add idx 2, 3 cases

elif idx == 2:

src = imageL

dst = ln

wn = "left"

elif idx == 3:

src = imageC

dst = cn

wn = "center"

pick[idx].append((x,y))

dst = src.copy()

# red BGR color in OpenCV, you need to set to blue on left side

if idx == 0 or idx == 1:

col = (0, 0, 255)

else:

col = (255,0,0)

# place circle on the picked point and text its serial (0-3)

for i in range(len(pick[idx])):

if idx == 3: #to retain the red points on the center image

for j in range(len(pick[1])):

dst = cv2.circle(dst, pick[1][j], 5, (0,0,255), 2)

dst = cv2.putText(dst, str(j), (pick[1][j][0]+10, pick[1][j][1]-10),

cv2.FONT\_HERSHEY\_SIMPLEX,1, (0,0,255), 1)

dst = cv2.circle(dst, pick[idx][i], 5, col, 2)

dst = cv2.putText(dst, str(i), (pick[idx][i][0]+10, pick[idx][i][1]-10),

cv2.FONT\_HERSHEY\_SIMPLEX,1, col, 1)

# please make sure when idx == 3, you need to show red color circle in dst

# this example erases red circle

cv2.imshow(wn, dst)

# to make sure image is updated

cv2.waitKey(1)

if len(pick[idx]) >= 4:

print('Is it OK? (y/n)')

i = input()

if i == 'y' or i == 'Y':

if idx >= 3:

savePick()

combine()

elif idx == 0:

print('center 4 points')

cv2.setMouseCallback("center", center\_click\_r)

elif idx == 1:

savePick()

print('left 4 points')

cv2.setMouseCallback("left", left\_click)

elif idx == 2:

savePick()

print('center 4 points')

cv2.setMouseCallback("center", center\_click\_l)

else:

pick[idx] = []

dst = src.copy()

cv2.imshow(wn, dst)

# Enter Dataset number to use: 0 for wall dataset, 1 for door data, 2 for house data, 3 for pittsburgh data

dataset = int(input("ENTER DATASET TO BE USED: "))

global f

if dataset == 0:

f = "wall"

imageL = cv2.imread("ps4-images/"+f+"-left.png")

imageC = cv2.imread("ps4-images/"+f+"-center.png")

imageR = cv2.imread("ps4-images/"+f+"-right.png")

elif dataset == 1:

f = "door"

imageL = cv2.imread("ps4-images/"+f+"-left.jpg")

imageC = cv2.imread("ps4-images/"+f+"-center.jpg")

imageR = cv2.imread("ps4-images/"+f+"-right.jpg")

elif dataset == 2:

f = "house"

imageL = cv2.imread("ps4-images/"+f+"-left.jpg")

imageC = cv2.imread("ps4-images/"+f+"-center.jpg")

imageR = cv2.imread("ps4-images/"+f+"-right.jpg")

elif dataset == 3:

f = "pittsburgh"

imageL = cv2.imread("ps4-images/"+f+"-left.jpg")

imageC = cv2.imread("ps4-images/"+f+"-center.jpg")

imageR = cv2.imread("ps4-images/"+f+"-right.jpg")

else:

f = "sorrel"

imageL = cv2.imread("ps4-images/Test/"+f+"-left.jpg")

imageC = cv2.imread("ps4-images/Test/"+f+"-center.jpg")

imageR = cv2.imread("ps4-images/Test/"+f+"-right.jpg")

result = cv2.copyMakeBorder(imageC,imageC.shape[0],imageC.shape[0],imageC.shape[1],imageC.shape[1],

borderType=cv2.BORDER\_CONSTANT,value=[0,0,0])

print(imageL.shape,imageC.shape,imageR.shape, result.shape)

cv2.namedWindow("left",cv2.WINDOW\_NORMAL)

cv2.namedWindow("center",cv2.WINDOW\_NORMAL)

cv2.namedWindow("right",cv2.WINDOW\_NORMAL)

cv2.namedWindow("result",cv2.WINDOW\_NORMAL)

ln = imageL.copy()

cn = imageC.copy()

rn = imageR.copy()

cv2.imshow("left", ln)

cv2.imshow("center", cn)

cv2.imshow("right", rn)

cv2.imshow("result", result)

pick = []

pick.append([])

pick.append([])

pick.append([])

pick.append([])

print('use saved points? (y/n)')

i = input()

if i == 'y' or i == 'Y':

loadPick()

combine()

else:

print("right 4 points")

cv2.setMouseCallback("right", right\_click)

cv2.waitKey()

# close all open windows

cv2.destroyAllWindows()

# System Specifications

Operating System: macOS Monterey Version 12.5.1

Hardware: MacBook Air 2017 (Intel Core i5)

Python: Conda environment utilizing Python 3.9.1

IDE: Visual Studio Code