ECE 5554 SU22 – Dr. Jones – HW5

**Image Output:**

warped\_rio-01.png

warped\_rio-02.png



final\_stitch\_of\_rio-00\_rio-01\_rio-01.png

warped\_blacksburg-01.png



warped\_ blacksburg-02.png



final\_stitch\_of\_ blacksburg-00\_ blacksburg-01\_ blacksburg-02.png

A picture containing text, outdoor

Description automatically generated

warped\_ diamondhead-01.png

**A city next to the ocean

Description automatically generated with low confidence**

warped\_ diamondhead-02.png

**A picture containing outdoor, nature, overlooking, hillside

Description automatically generated**

final\_stitch\_of\_ diamondhead -00\_ diamondhead -01\_ diamondhead -02.png

**A picture containing outdoor, mountain, field, full

Description automatically generated**

**Console Output:**

Running rio

Homography Matrix:

[[ 9.19445162e-01 8.32761553e-03 4.69159507e+02]

[-3.22339692e-02 9.76056738e-01 -1.05353958e+01]

[-7.34228131e-05 3.68033965e-06 1.00000000e+00]]

Homography Matrix:

[[ 8.39176970e-01 1.21370715e-02 1.09018595e+03]

[-7.69487370e-02 1.00523437e+00 3.52701241e+00]

[-1.59869382e-04 -4.89901448e-06 1.00000000e+00]]

Running blacksburg

Affine Matrix:

[[ 9.99999325e-01 -2.92998451e-06 4.66000799e+02]

[ 3.48852541e-06 9.99992667e-01 1.38001644e+02]]

Affine Matrix:

[[ 1.00000101e+00 -4.01336404e-06 9.51001586e+02]

[ 6.94099115e-08 9.99996161e-01 3.90008919e+01]]

Running diamondhead

Homography Matrix:

[[ 8.92452166e-01 -5.51077761e-02 3.50264568e+02]

[ 1.45061916e-02 9.65492527e-01 -2.03212066e+01]

[-1.04610142e-04 2.86673961e-06 1.00000000e+00]]

Homography Matrix:

[[ 7.46444379e-01 -9.24087596e-02 8.58030223e+02]

[ 3.67048980e-02 1.00175258e+00 -1.00006446e+02]

[-2.59227335e-04 4.66338627e-05 1.00000000e+00]]

Code

Homework5\_AndrewGarcia.py:

import cv2

import numpy as np

import math

from tqdm import tqdm

import matplotlib.pyplot as plt

RED = (0, 0, 255)

GREEN = (0, 255, 0)

BLUE = (255, 0, 0)

def main():

rio\_list = ['rio-00.png', 'rio-01.png', 'rio-02.png']

blacksburg\_list = ['blacksburg-00.png', 'blacksburg-01.png', 'blacksburg-02.png']

diamondhead\_list = ['diamondhead-00.png', 'diamondhead-01.png', 'diamondhead-02.png']

filename\_list = [rio\_list, blacksburg\_list, diamondhead\_list]

for stitch\_list in filename\_list:

print(f"Running {stitch\_list[0][:-7]}")

if stitch\_list == blacksburg\_list:

method = 'affine'

else:

method = 'homography'

# Part a: Load images and convert to greyscale

img\_list = [cv2.imread(this\_filename,0) for this\_filename in stitch\_list]

# Stitch First IMG to Second IMG

stitch\_result1, warped1 = stitch\_img(img\_list[0], img\_list[1], method)

cv2.imshow(f'Warped {stitch\_list[1]} ', warped1), cv2.waitKey(0)

cv2.imwrite(f'warped\_{stitch\_list[1]} ', warped1)

cv2.imshow(f'Sticth of {stitch\_list[0][:-4]} with {stitch\_list[1]}', stitch\_result1), cv2.waitKey(0)

cv2.imwrite(f'stitch\_of\_{stitch\_list[0][:-4]}\_with\_{stitch\_list[1]}', stitch\_result1)

# Stitch Stitched IMG to Third IMG

stitch\_result2, warped2 = stitch\_img(stitch\_result1, img\_list[2], method)

cv2.imshow(f'Warped {stitch\_list[2]} ', warped2)

cv2.waitKey(0)

cv2.imwrite(f'warped\_{stitch\_list[2]}', warped2)

cv2.imshow(f'final\_sticth\_of\_{stitch\_list[0][:-4]}\_{stitch\_list[1][:-4]}\_{stitch\_list[2]}', stitch\_result2)

cv2.waitKey(0)

cv2.imwrite(f'final\_stitch\_of\_{stitch\_list[0][:-4]}\_{stitch\_list[1][:-4]}\_{stitch\_list[2]}', stitch\_result2)

def stitch\_img(img1, img2, method):

# Part b: Implement the image stitching process described in lecture 9

# Citation for sift and bfmatcher code: https://docs.opencv.org/3.4/dc/dc3/tutorial\_py\_matcher.html

sift = cv2.SIFT\_create()

keypoints1, descriptors1 = sift.detectAndCompute(img1, mask = None)

keypoints2, descriptors2 = sift.detectAndCompute(img2, mask = None)

# BFMatcher with default params

bf = cv2.BFMatcher()

matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Apply ratio test

good = []

good\_matches = []

for m,n in matches:

if m.distance < 0.25\*n.distance:

good.append([m])

good\_matches.append(m)

# cv.drawMatchesKnn expects list of lists as matches.

img3 = cv2.drawMatchesKnn(img1, keypoints1,

img2, keypoints2,

good, None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

#plt.imshow(img3),plt.show()

# Source for Stitching: https://stackoverflow.com/questions/61146241/how-to-stitch-two-images-using-homography-matrix-in-opencv

src\_pts = 0

dst\_pts = 0

if len(good\_matches) > 10:

dst\_pts = np.float32([keypoints1[m.queryIdx].pt for m in good\_matches]).reshape(-1, 2)

src\_pts = np.float32([keypoints2[m.trainIdx].pt for m in good\_matches]).reshape(-1, 2)

# Part b.i Write the resulting transformation matrix to the console

width = img2.shape[1] + img1.shape[1]

height = img1.shape[0]

if method == 'homography':

H, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC)

print(f"Homography Matrix: \n{H}")

warped = cv2.warpPerspective(img2, H, (width, height))

warped = remove\_extra(warped)

elif method == 'affine':

H, mask = cv2.estimateAffine2D(src\_pts, dst\_pts, cv2.RANSAC)

print(f"Affine Matrix: \n{H}")

warped = cv2.warpAffine(img2, H, (width, height))

warped = remove\_extra(warped)

# Stitch together

result = warped.copy()

# Add zeros at end of img1 to match size of warped img

missig\_cols = result.shape[1] - img1.shape[1]

missing\_rows\_arr = np.zeros((img1.shape[0], missig\_cols), dtype = np.uint8)

img1\_ = np.concatenate((img1, missing\_rows\_arr), axis = 1)

# Average the two images together

result = np.zeros(warped.shape, dtype= np.uint8)

for r in range(result.shape[0]):

for c in range(result.shape[1]):

val1 = img1\_[r,c]

val2 = warped[r,c]

if val1 == 0:

result[r,c] = val2

elif val2 == 0:

result[r,c] = val1

else:

# Put first img on top of second img

result[r,c] = np.average([val1, val2])

warped\_seam = find\_horizontal\_seams(warped)

img1\_seam = find\_horizontal\_seams(img1\_)

result\_fthr = add\_feathering(result, img1\_, warped, warped\_seam, img1\_seam)

return result\_fthr, warped

def add\_feathering(output\_img, img\_left, img\_right, left\_seam, right\_seam):

rst\_img = output\_img.copy()

for r in range(len(left\_seam)-1):

for c in range(img\_right.shape[1]-1):

try:

if c >= left\_seam[r] and c <= right\_seam[r]:

left\_seam\_col = left\_seam[r]

right\_seam\_col = right\_seam[r]

total\_dist = right\_seam\_col - left\_seam\_col

dist\_right = right\_seam\_col - c

dist\_left = c - left\_seam\_col

pix\_val\_left = img\_left[r,c]

pix\_val\_right = img\_right[r,c]

rst\_img[r,c] = np.uint8((pix\_val\_left\*dist\_right)/total\_dist + (pix\_val\_right\*dist\_left)/total\_dist)

except:

#print(f'Error on Feather {r}, {c}')

pass

return rst\_img

def find\_horizontal\_seams(img):

seam\_rc = {}

for r in range(img.shape[0]-1):

for c in range(img.shape[1]-1):

if c != 0:

value1 = img[r,c-1]

value2 = img[r,c]

value3 = img[r,c+1]

if value1 == 0 and value2 == 0 and value3 != 0:

seam\_rc[r] = c

break

elif value1 != 0 and value2 == 0 and value2 == 0:

seam\_rc[r] = c

break

return seam\_rc

def remove\_extra(img):

for col in reversed(range(img.shape[1])):

if img[:,col].sum() != 0:

col = col+1

break

img\_final = img[:,:col]

return img\_final

if \_\_name\_\_ == '\_\_main\_\_':

main()