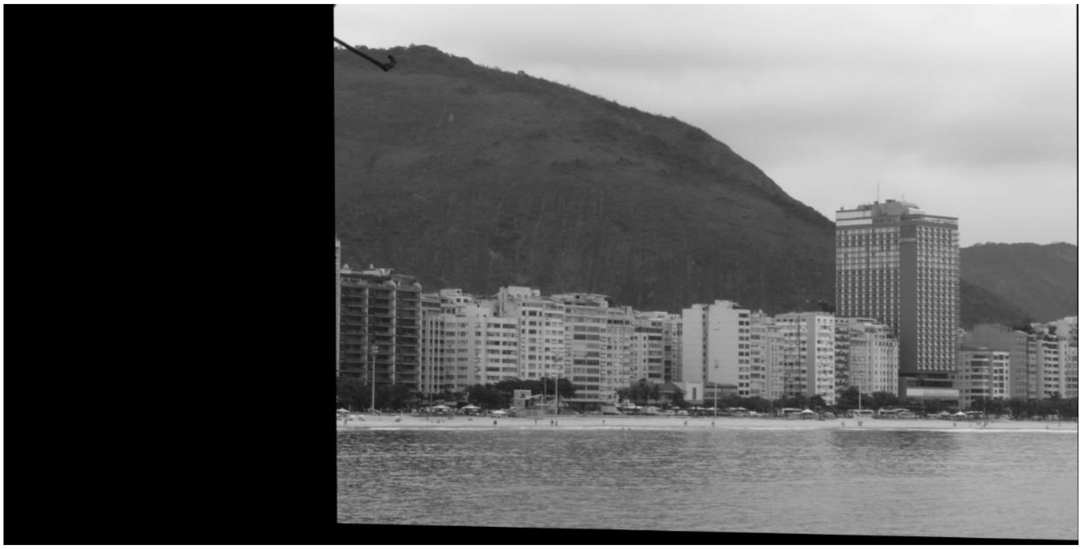
Andrew Garcia
8/11/2022

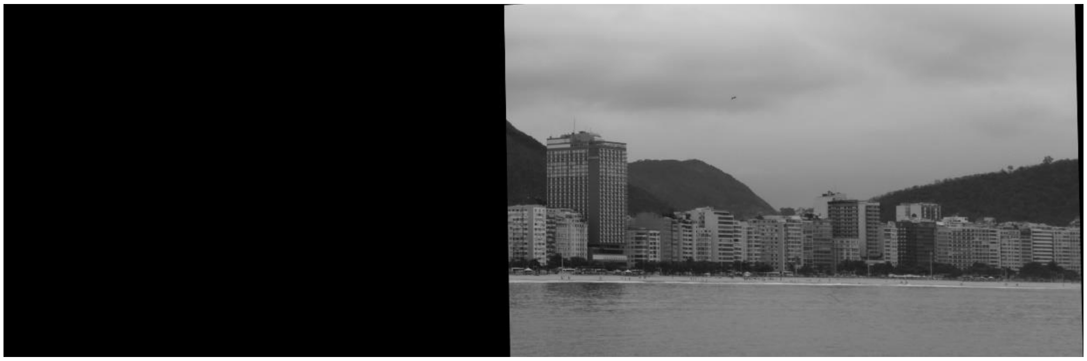
ECE 5554 SU22 – Dr. Jones – HW5

Image Output:

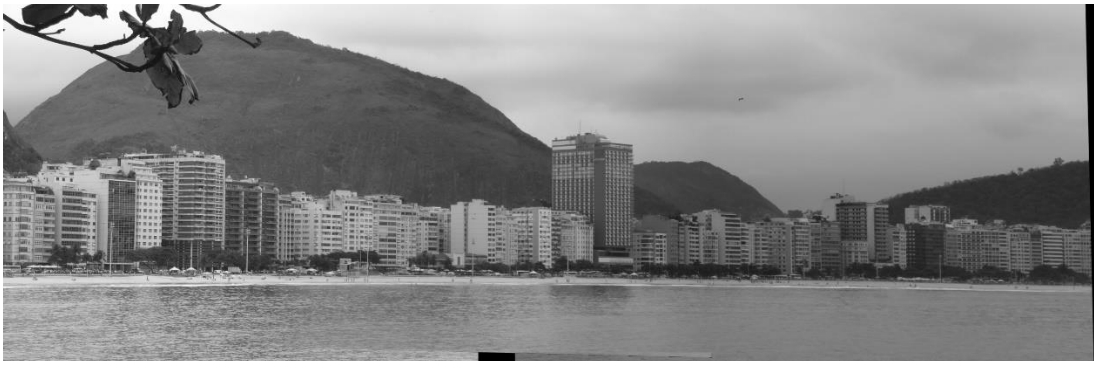
warped_rio-01.png



warped_rio-02.png

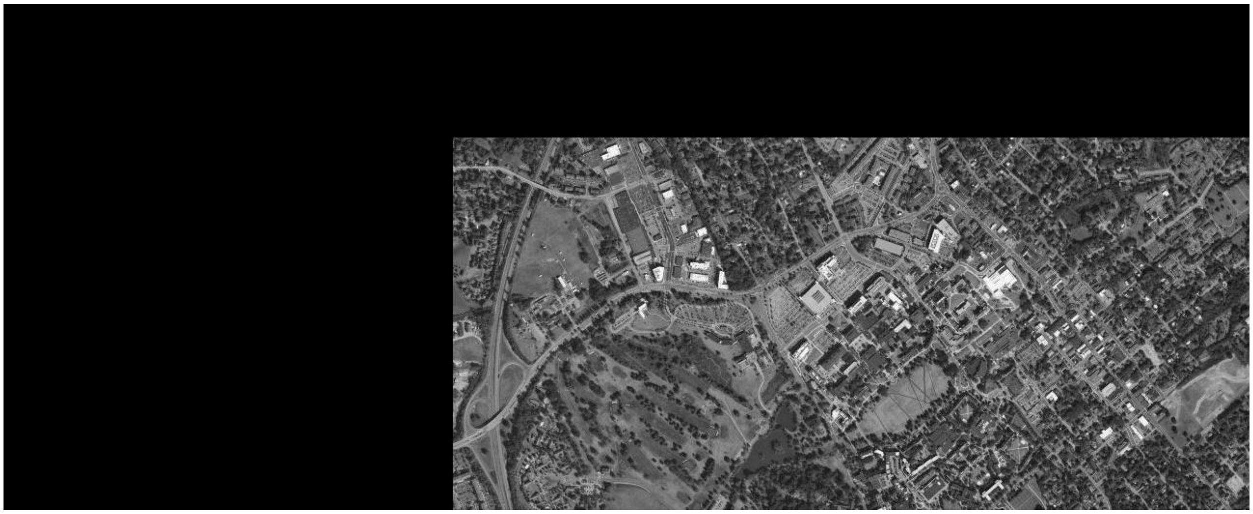


final_stitch_of_rio-00_rio-01_rio-01.png

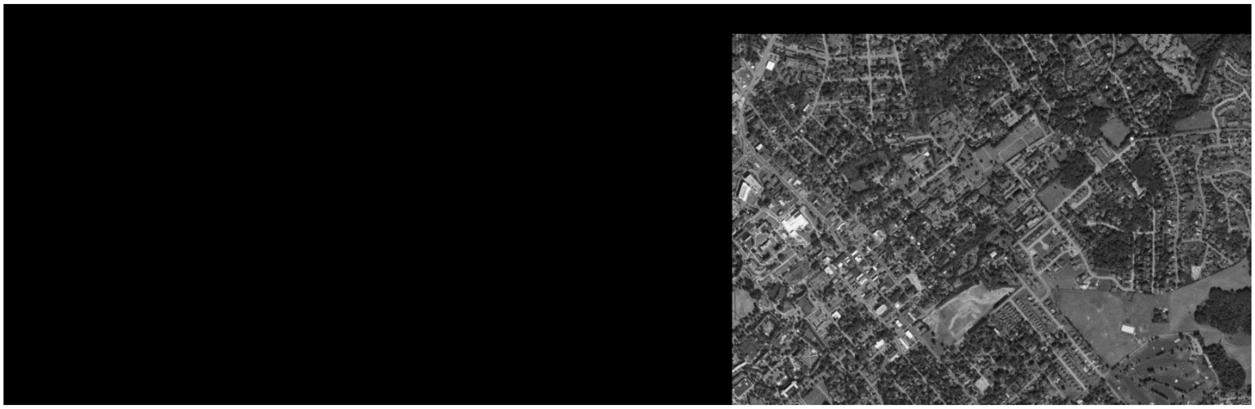


Andrew Garcia
8/11/2022

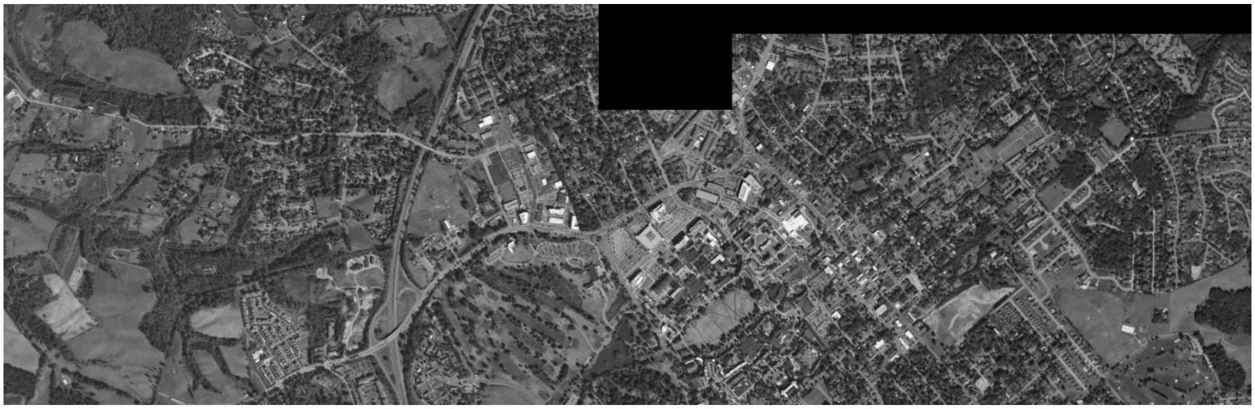
warped_blacksburg-01.png



warped_ blacksburg-02.png

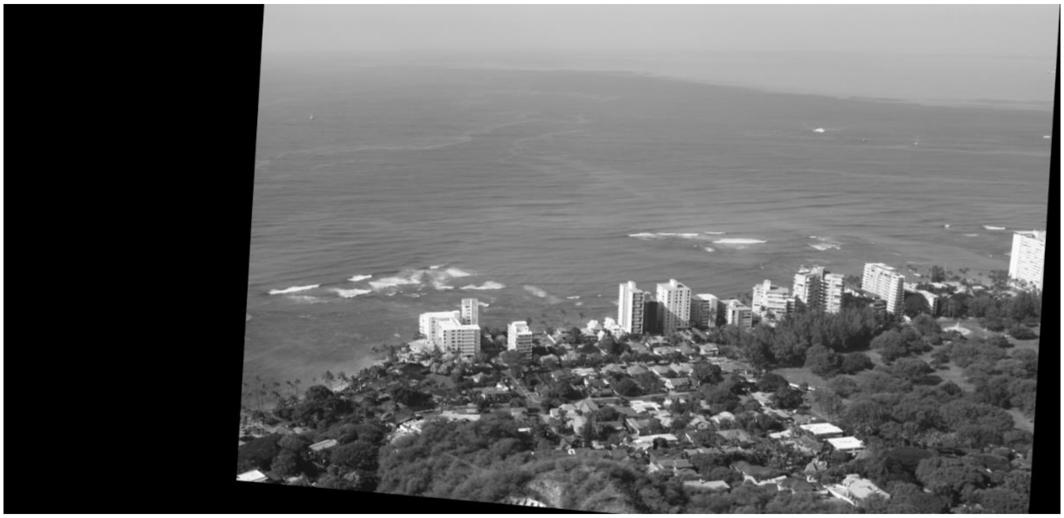


final_stitch_of_ blacksburg-00_ blacksburg-01_ blacksburg-02.png

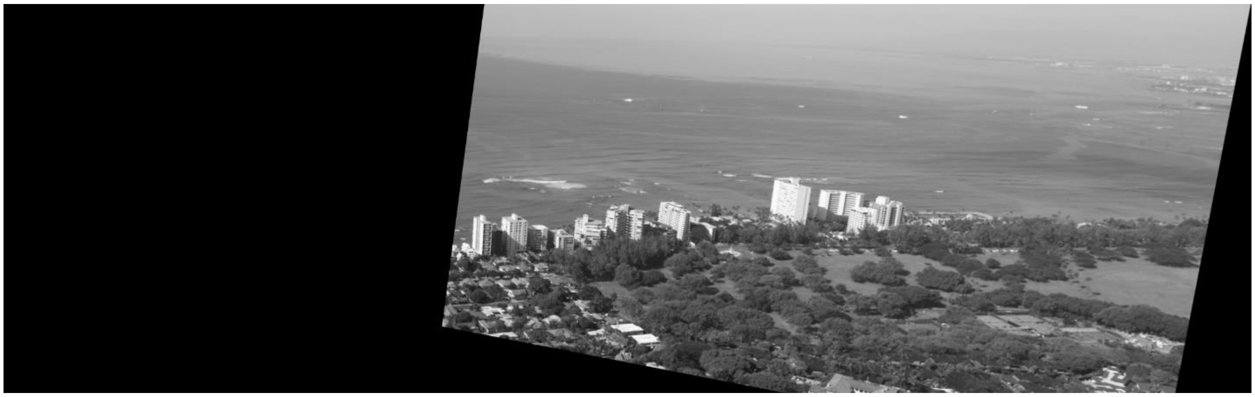


Andrew Garcia
8/11/2022

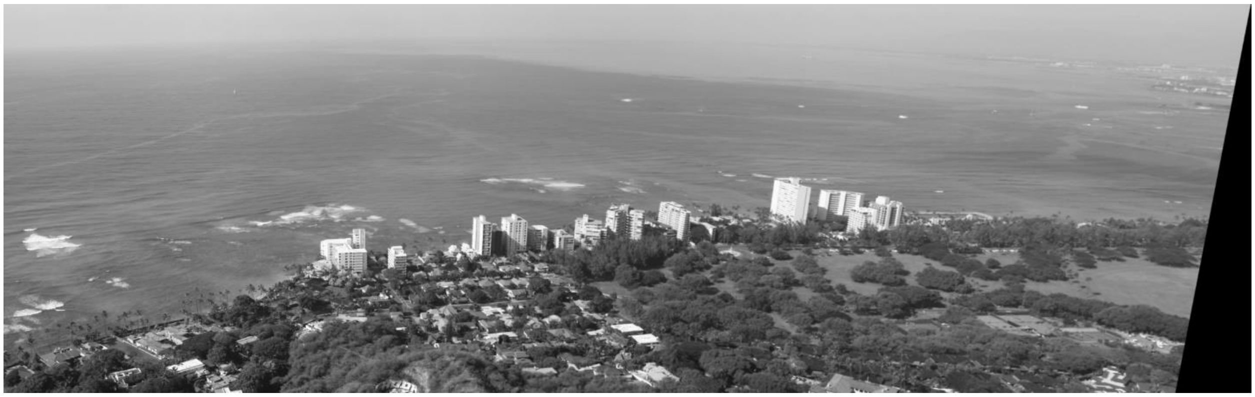
warped_ diamondhead-01.png



warped_ diamondhead-02.png



final_stitch_of_ diamondhead -00_ diamondhead -01_ diamondhead -02.png



Andrew Garcia
8/11/2022

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]]
```

Andrew Garcia
8/11/2022

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()
```