Image Stitching Part 2

Kareem Tiwari

For this part of the image stitching assignment we are now getting we are focusing on matching of good points and the stitching of images 1 and 2 in a new panoramic image that we create as we go through our code. In my code I am using the OpenCV computed H.

```
img1 = cv2.imread('mountainLeft.jpg') # queryImage
img2 = cv2.imread('mountainRight.jpg') # trainImage

im2_Rows = img2.shape[0]
im2_cols = img2.shape[1]

img1_Rows = img1.shape[0]
img1_Cols = img1.shape[1]

# Initiate SIFT detector
sift = cv2.SIFT_create()
##sift = cv2.xfeatures2d.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

# cv2.drawMatchesKnn expects list of lists as matches.
img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2) #NOTE: 'None' parameter has to be added (not in documentation)

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


pts1 = np.zeros((len(good),2), np.float32)
pts2 = np.zeros((len(good),2), np.float32)
for m in range(len(good)):
        pts1[m] = kp1[good[m][0].queryIdx].pt
        pts2[m] = kp2[good[m][0].trainIdx].pt

opencvH, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC, 5.0)
print("H matrix estimated by OpenCV (for comparison):\n", opencvH)
```

Above, is where I created my images as well as creating the variables to store the Rows and Columns. As well as the key points in and the array for the good matches. I also created the pano image setting for the pano image that has been stitched together.

In the screenshot below is where I create all of the mins, maxes, as well as the array to store all of the “mapped corners” in the pano image. I then use a for loop to find 4 corners. Then we look through the “mapped corners” and set all of the mins and maxes to the mins and maxes of the x and y coordinates that had been gathered and stored from the loops prior. I then printed out the x and y coordinate arrays to make sure that they were being populated. (I later commented this out).

Then I create the “pano” image and show that image. We then go through the rows and columns and for every value that falls within the range we are shifting that value in from image2 into the “pano” image. I then print out the “pano rows” and the “pano Cols” just so I could make sure that there would not be an index out of bounds error. (I later would comment these lines out as well).

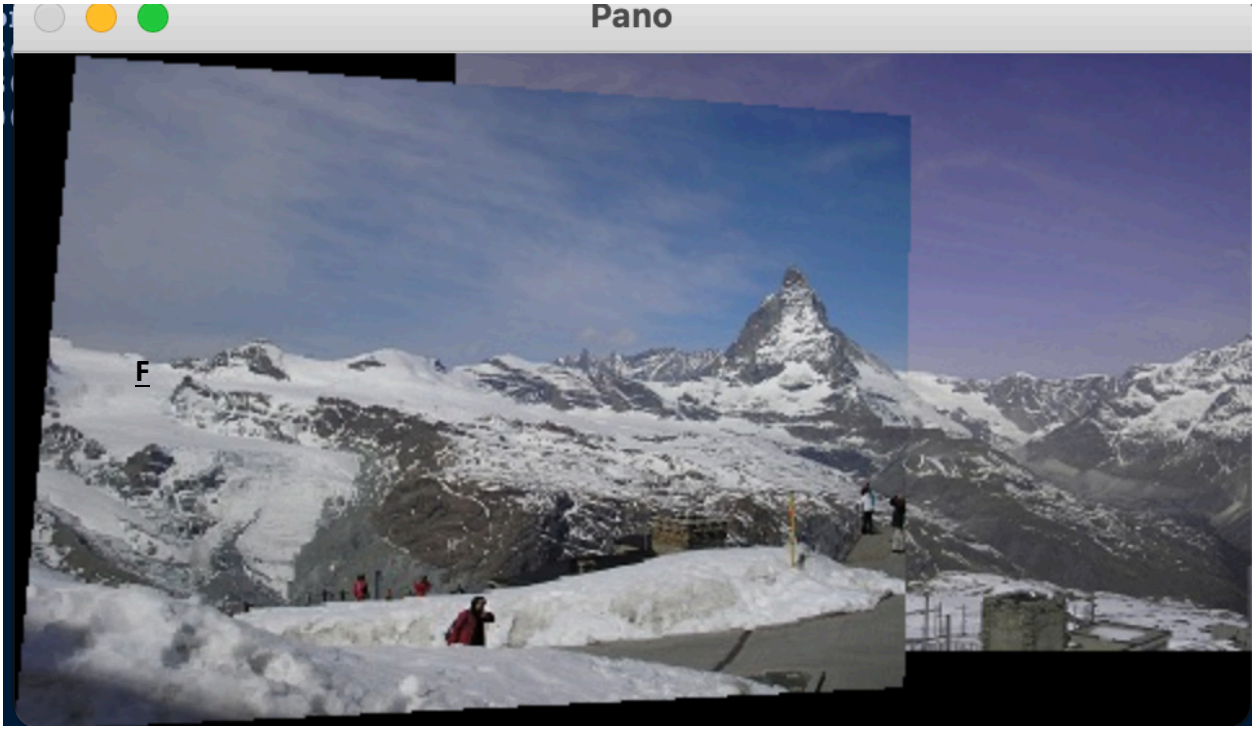
```
mapped_corners = []
corners= ([0,0,1],[img1.shape[1],0,1],[0,img1.shape[0],1],[img1.shape[1],img1.shape[0],1])
min_x = 0
max_x = 0
min_y = 0
max_y = 0
all_y = []
all_x = []

for r in range(4):
    new_H =  np.dot(opencvH,corners[r])
    mapped_corners.append(new_H/new_H[2])


for i in mapped_corners:
    all_x.append(i[0])
    all_y.append(i[1])

for i in corners:
    all_x.append(i[0])
    all_y.append(i[1])


min_x = min(all_x)
max_x = max(all_x)
min_y = min(all_y)
max_y = max(all_y)

p_cols =  (int(max_x) - int(min_x))
p_rows = int(max_y) - int(min_y)


##print(all_x)
##print(all_y)


pano = np.zeros((p_rows,p_cols,3), np.float32)

plt.imshow(pano),plt.show()


for i in range(im2_Rows):
    for j in range(im2_cols):

        pano[i+int(abs(min_y))][j+int(abs(min_x))][0] = img2[i][j][0]
        pano[i+int(abs(min_y))][j+int(abs(min_x))][1] = img2[i][j][1]
        pano[i+int(abs(min_y))][j+int(abs(min_x))][2] = img2[i][j][1]

print(p_rows)
print(p_cols)
Hinv = np.linalg.inv(opencvH)
```

In the code below, I uncomment out the code that was given to us to find the inverse of the H that was calculated earlier by the OpenCV function. Then I am iterating through the rows and columns of the “pano” image and in that image, I am setting a new value for pt1 by taking the x and y values and subtracting them from the “minX” and “minY” values. I am also setting a new value for the pt2 value by multiplying pt1 by the inverse of the H matrix. I then compared the values in pt1 the index zero to the integer zero and the value in pt2 at the index zero to the image 1 columns. As well as comparing the index 1 of pt2 to zero and the index 1 of pt2 of the rows of image 1. I then set the image of Pano at index [I,j] to the value stored at the integer value of the index 1 at pt2 and the integer value of the index 0 at pt2 at image1.

```
Hinv = np.linalg.inv(opencvH)

for i in range(p_rows): # iterate over the height of the image, y-coordinates
    for j in range(p_cols): # iterate over the width of the image, x-coordinates

        pt1 = [j-int(abs(min_x)),i-int(abs(min_y)),1]
        pt2 =  np.dot(Hinv,pt1)
        pt2 = pt2/pt2[2]

        if pt2[0] > 0 and pt2[0] < img1_Cols:
            if pt2[1] > 0 and pt2[1] < img1_Rows:

                pano[i,j] = img1[int(pt2[1]),int(pt2[0])]



##
        #access va;ues at pixel:
##        pano[i][j][0] = img1[int(pt2)] # blue value stored at pixel(i,j)
##        k +=1
##        pano[i][j][1] = img1[int(pt2)] # green value stored at pixel(i,j)
##        k +=1
##        pano[i][j][2] = img1[int(pt2)]# red value stored at pixel(i,j)




cv2.imshow("Pano",pano/255)
```

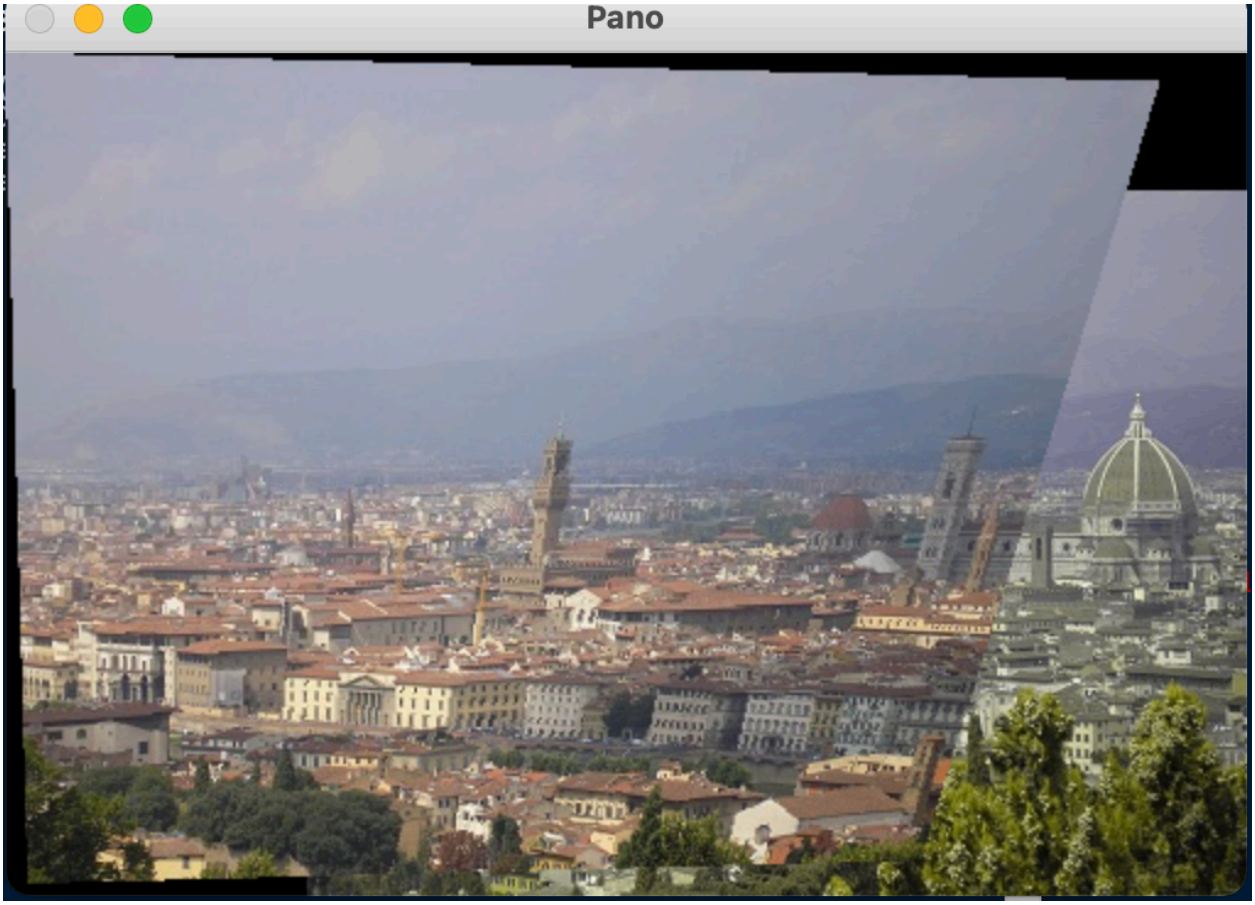
I then show the “Pano” image based on all of the values that have been calculated and the stitching of the two images should be apparent. The image pairs Panos that are being represented are going to be shown below.

Mountains:

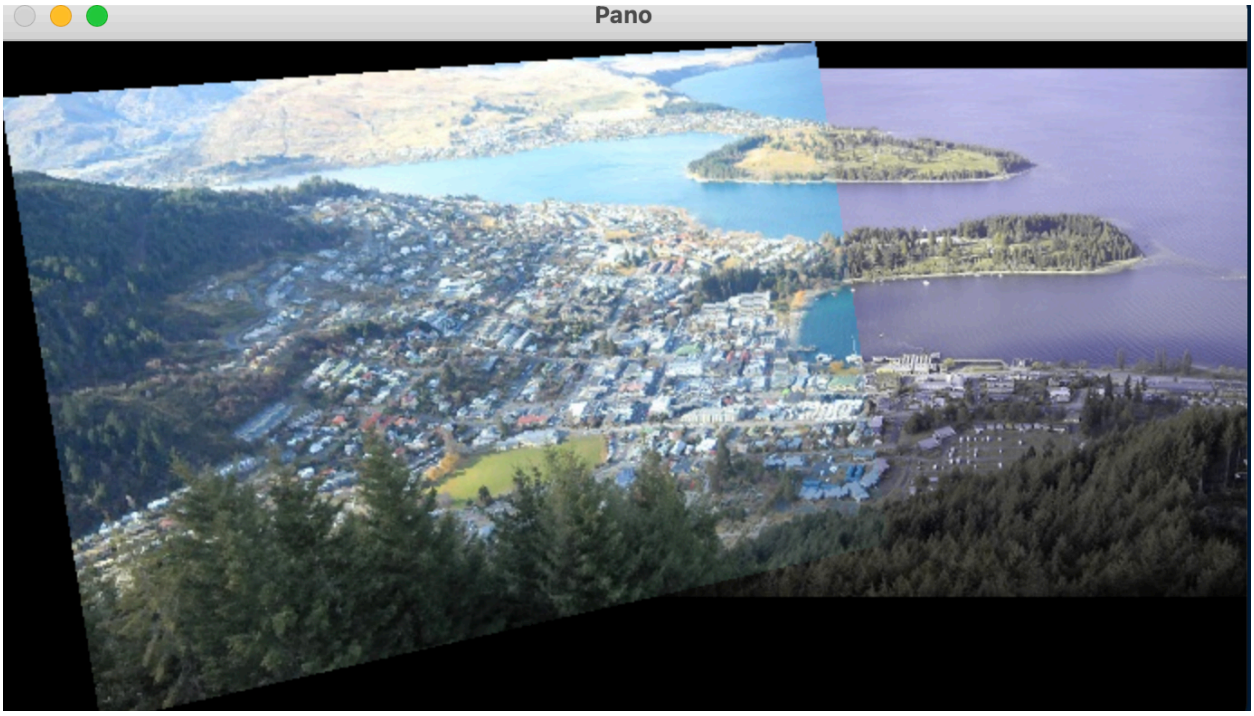


F

Florence:



Queenstown:



Waterfall: