## CS4475 Assignment 8 Panorama

In this assignment, I use three photos that I took at Hollywood mountain hiking trail in LA this past spring break.

- Input

- Output
  After cropping

- blendImagePair

```
col_left = 0
col_right = 100000
row_left = 0
row_right = 100000

warp_copy_1 = np.copy(warped_image)
warp_copy_2 = np.copy(warped_image)    ##blending below

warp_copy_1[warp_copy_1 > 0] = 127
warp_copy_2[:,:,:] = 0

output_image = np.copy(warped_image)
output_image[point[1]:point[1] + image_2.shape[0],
        point[0]:point[0] + image_2.shape[1]] = image_2
```

```
    warp_copy_1[warp_copy_1 > 0] = 127
    warp_copy_2[:,:,:] = 0
    warp_copy_2[point[1]:point[1]+image_2.shape[0],
point[0]:point[0]+image_2.shape[1]] = 127

    overlap = np.copy(warp_copy_2)
    overlap = warp_copy_1 + warp_copy_2
    overlap[overlap > 127] = 255
    overlap[overlap < 128] = 0

    for x in range(overlap.shape[0]):
        for y in range(overlap.shape[1]):
            if(y>col_left):
                if(overlap[x,y,0]==255):
                    col_left = y
            if(y<col_right):
                if(overlap[x,y,0]==255):
                    col_right = y
            if(x>row_left):
                if(overlap[x,y,0]==255):
                    row_left = x
            if(x<row_right):
                if(overlap[x,y,0]==255):
                    row_right = x

    for x in range(output_image.shape[0]):
        for y in range(output_image.shape[1]):
            if (overlap[x, y, 0] > 0):
                x_size = (float(y)-col_right)/(0-col_right)
                output_image[x, y] = x_size*output_image[x, y] + (1-
x_size)*warped_image[x, y]
                if (x>=row_right and x<=row_right+100):
                    y_size_1 = (float(x)-row_right)/100
                    output_image[x, y] = y_size_1*output_image[x, y] + (1-
y_size_1)*warped_image[x, y]
                if (x<=0 and x>=0-150):
                    y_size_1 = 1.0 - ((0-float(x))/150)
                    output_image[x, y] = y_size_1*image_2[x-point[1], y-point[0]] + (1-
y_size_1)*output_image[x, y]

    return output_image
```

This is the blending function, which takes in an image that has been warped and an image that needs to be inserted into the warped image. Then, it takes in a point where the new image will be inserted. I implemented a function to

average the pixels where the images overlap. I added a size function from the start of the overlap point to create a “fade” effect and properly transition between two images. It calculates both horizontally and vertically (x and y) between the boundaries. However, this method has a long execution time (around 10 minutes) because it must iterate over every pixel in the output image and calculate the new pixel value. I would love to use this method for my project, so I will look into a more efficient way to do this.

Sample test:

- Input:

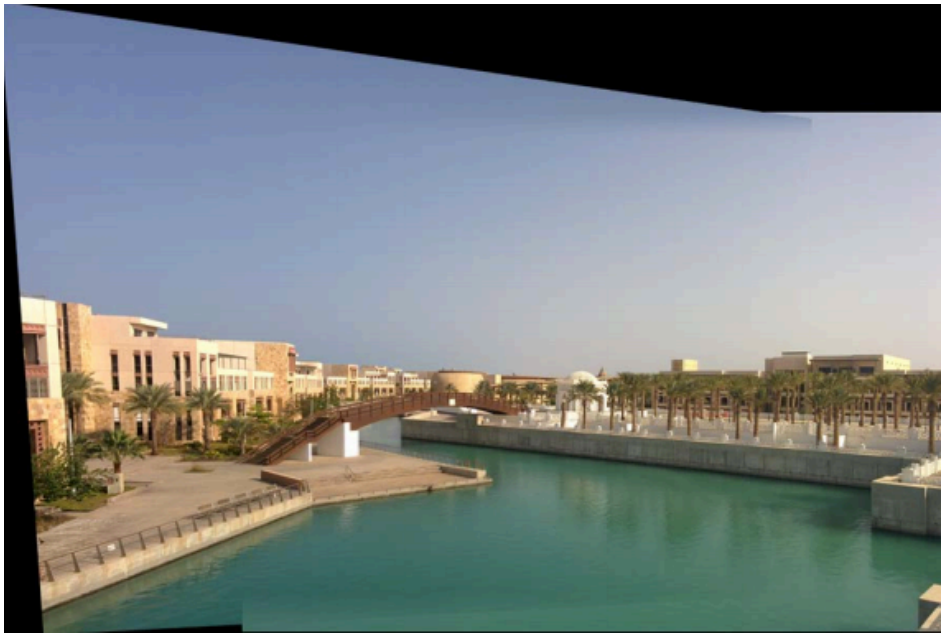
Output:
Before cropping

After cropping

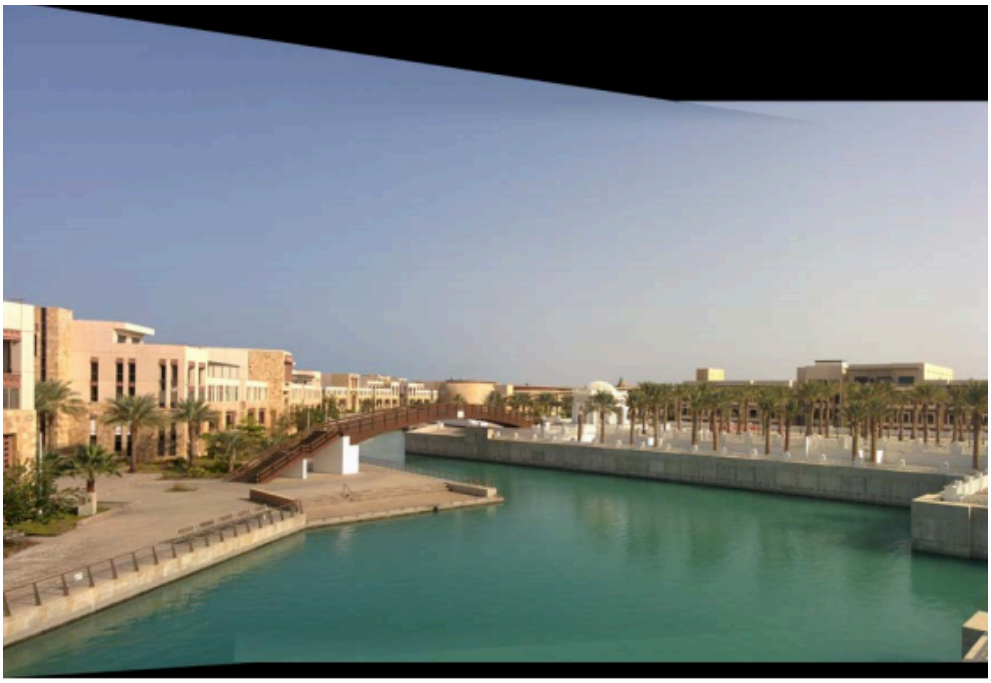
Answers:

- The number of matched features has definitely has an impact on the output result. The output would warp differently due to the different numbers because there are more features to affect the warping and translation. The panorama is generated with a number of matches equal to 20 in the test code. I tried to increase the number to 50 and 100. The image in my opinion looks pretty nice when increased the value, the color looks more defined and clear, but this would change based on the input images and camera rotation. Also, the execution times are longer with additional matches because there is more data to sift through.

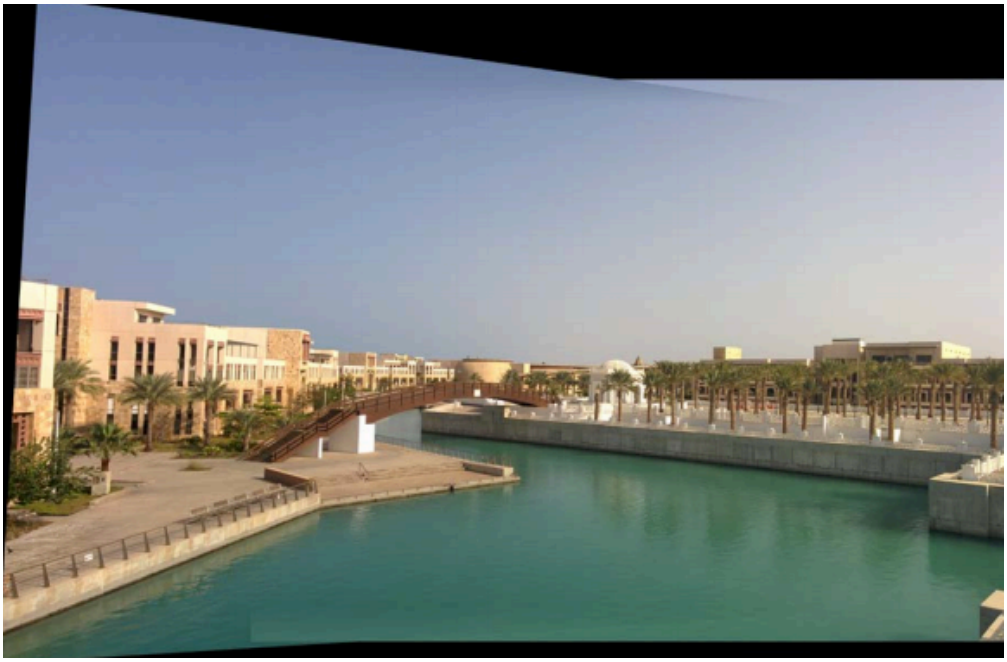
  20:



50:



100:



- I took rotational panorama for the assignment.

- In general, I am happy with the result for this assignment. The sample images came out really nicely for the panorama, because the input images are taken perfectly for the panorama. My own panorama, is okay, and I think it's because the quality of my pictures are not as good. I believe this is due to occlusions in features taken at different

angles. I could probably fix this by taking images of a relatively “flat” subject with lots of features to match with. In the future, I’d like to experiment with my pyramid blending from previous assignment to make the blended images more seamless.

- Multiplying the x_min and y_min by -1 in warpImagePair() Reason: These values are multiplied by negative 1 because of properties of the matrix dot product. The combined translation and homography matrix needs to be inversed for the output warped image pair to be oriented properly. These negative values help to remove the offsets of the x and y minimums. The function cv2.warpPerspective() is then called using the dot product of the homography and the translation matrix. The resulting warped image and image 2 are then stitched/blended together by calling blendImagePair().