Report for Image Stiching Using Ransac Algorithm

I am using opencv-python version 4.5.4.60 and python 3.9. Used np.random.seed(27).

The below code will generate the keypoints and descriptors for left and right images

```
gray = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray, None)
kp1 = np.float32([i.pt for i in kp1])

gray = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)
kp2, des2 = sift.detectAndCompute(gray, None)
kp2 = np.float32([i.pt for i in kp2])
```

For each key point in left image, I am calculating the top-2 nearest key points in right image. I did this using L2-norm.
For each key point in right image, repeated the same procedure.
I am storing key point index, its matching key point indexes of the other image, key point descriptor and its matching key point descriptors of the other image.

```
y = np.asarray(des2)
res = []
for index, ds in enumerate(des1):
    x = np.array(ds)
    dist = np.linalg.norm(x-y, axis=1)
    indices = sorted(range(len(dist)), key=lambda sub: dist[sub])[:2]
    top_2 = [dist[i] for i in indices]
    res.append((index, x, top_2, indices))

y = np.asarray(des1)
res2 = []
for index, ds in enumerate(des2):
    x = np.array(ds)
    dist = np.linalg.norm(x - y, axis=1)
    indices = sorted(range(len(dist)), key=lambda sub: dist[sub])[:2]
    top_2 = [dist[i] for i in indices]
    res2.append((index, x, top_2, indices))
```

I did the ratio test next using the res list calculated in the previous step. Using 0.75 as threshold.

```
goodmatches = []
for index, src, dest, indices in res:
    dist1 = np.linalg.norm(src-dest[0])
    dist2 = np.linalg.norm(src-dest[1])
    if dist1 < 0.75 * dist2:
        goodmatches.append([index, src, dest, indices])
```

I did the cross check next using ratio test result and res2 lists. The result of it is the source point and its top matching point if they match each other point as the best match.

```
goodmatches2 = []
for index, src, des, c_indices in goodmatches:
    found = False
    if index == res2[c_indices[0]][3][0]:
        goodmatches2.append((kp1[index], kp2[c_indices[0]]))
```

I have ran the RANSAC algorithm for 5000 iterations. Took 4 random source and its matching points. I have constructed the transformation matrix using these 4 points which gives me 8 rows of equations to solve further. Used svd function to solve the equation. Took the last row of vt and reshaped it to (3, 3) matrix. Normalized the H by H[2,2] value. Took all the remaining points unselected randomly and calculated the L2 norm between given left image and the generated left image points. Took the L2 norm value of 500 as threshold to select the inliers.

In each iteration updated the indices of the inliners if the increased inliners length is greather than the current best inliners length.

```
max_inliers = 0
max_inliers_list = []
probs = [1 / len(goodmatches2) for i in range(len(goodmatches2))]
indices = np.asarray([i for i in range(len(goodmatches2))])
for i in range(5000):
    n_samples = 4
    inds = np.random.choice(len(goodmatches2), n_samples, p=probs)
    rand_points = [goodmatches2[ind] for ind in inds]
    transformation_matrix = np.zeros((len(rand_points) * 2, 9))
    count = 0
    for index, (src, dest) in enumerate(rand_points):
        for j in range(2):
            if j == 0:
                transformation_matrix[count][0:3] = [dest[0], dest[1], 1]
                transformation_matrix[count][3:6] = [0, 0, 0]
            else:
                transformation_matrix[count][0:3] = [0, 0, 0]
                transformation_matrix[count][3:6] = [dest[0], dest[1], 1]
            transformation_matrix[count][6:] = [-1 * dest[0] * src[j],
                                                -1 * dest[1] * src[j],
                                                -1 * src[j]]
            count += 1
    u, sigma, vt = np.linalg.svd(transformation_matrix)
    H = vt[-1].reshape(3,3)
    if H[2,2] != 0:
        H = H/H[2, 2]
    remaining_points = np.asarray([goodmatches2[pt][0] for pt in np.delete(indices, inds)])
    left_pts = np.hstack((remaining_points, np.ones((remaining_points.shape[0], 1), dtype=remaining_points.dtype)))
    left_pts = left_pts/left_pts[:, -1].reshape(-1,1)
    remaining_points = np.asarray([goodmatches2[pt][1] for pt in np.delete(indices, inds)])
    right_pts = np.hstack((remaining_points, np.ones((remaining_points.shape[0], 1), dtype=remaining_points.dtype)))
    right_pts = right_pts / right_pts[:, -1].reshape(-1,1)
    cal_left_pts = np.dot(right_pts, H)
    diff = np.linalg.norm(left_pts - cal_left_pts, axis=1)
    inliers_idx = list(np.where(diff <= 500))
    count = (diff<=500).sum()
    if count>max_inliers:
        max_inliers = count
        max_inliers_list = inliers_idx.copy()
```

In the next step, I have selected all the matching key point pairs using the max_inliers_list generated in the previous step that is containing the corresponding indices of goodmatches2 list. I have got 87 matching pairs.
Generated the transformation matrix containing all the 174 equations and used svd to solve the equation. Normalized H again and reshaped to (3, 3) array. This is the final value of H which is used in the next steps.

```
all_points = [goodmatches2[i] for i in max_inliers_list[0]]
transformation_matrix = np.zeros((len(all_points) * 2, 9))
count = 0
```

```
for index, (src, dest) in enumerate(all_points):
    for j in range(2):
        if j == 0:
            transformation_matrix[count][0:3] = [dest[0], dest[1], 1]
            transformation_matrix[count][3:6] = [0, 0, 0]
        else:
            transformation_matrix[count][0:3] = [0, 0, 0]
            transformation_matrix[count][3:6] = [dest[0], dest[1], 1]
        transformation_matrix[count][6:] = [-1 * dest[0] * src[j],
                                            -1 * dest[1] * src[j],
                                            -1 * src[j]]
        count += 1
u, sigma, vt = np.linalg.svd(transformation_matrix)
H = vt[-1].reshape(3, 3)
if H[2, 2] != 0:
    H = H / H[2, 2]
```

Calculated the points of interest of left and right images and concatenated into one final list. Used perspectiveTransform method to transform the right image corners.

```
left_points = np.float32([[0, 0], [0, left_img.shape[0]], [left_img.shape[1], left_img.shape[0]], [left_img.shape[1], 0]]).reshape(-1, 1, 2)
temp = np.float32([[0, 0], [0, right_img.shape[0]], [right_img.shape[1], right_img.shape[0]], [right_img.shape[1], 0]]).reshape(-1, 1, 2)
right_points = cv2.perspectiveTransform(temp, H)
list_of_points = np.vstack((left_points, right_points))
```

Calculated the min_x, min_y, max_x, max_y used for the final warped image.

```
[min_x, min_y] = np.int32(list_of_points.min(axis=0).flatten())
[max_x, max_y] = np.int32(list_of_points.max(axis=0).flatten())

temp = np.array([[1, 0, -min_x], [0, 1, -min_y], [0, 0, 1]])
new_H = np.dot(temp, H)
```

Used warpPerspective function to form the transform the right image first and then storing the left image into the respective left position using the min and max values calculated in the previous step.

```
result_img = cv2.warpPerspective(right_img, new_H, (max_x - min_x, max_y - min_y))
result_img[-min_y:left_img.shape[0] + (-min_y), -min_x:left_img.shape[1] + (-min_x)] = left_img
return result_img
```

Output: