CV HW3

0516006 賴奕善 0516218 軒轅照雯 0510134王泓仁

1. Introduction

Image stitching algorithms

Image stitching (image stitching) refers to the technique of stitching two or more overlapping images into a panoramic image or high-resolution image.differences.

Image stitching issues

The aspect ratio of a panorama image needs to be taken into account to create a visually pleasing composite. Since the illumination in two views cannot be guaranteed to be identical, stitching two images could create a visible seam. Other reasons for seams could be the background changing between two images for the same continuous foreground.

For panoramic stitching, the ideal set of images will have a reasonable amount of overlap (at least 15–30%) to overcome lens distortion and have enough detectable features.

The image stitching process can be divided into three main components: Keypoint detection, Registration, and blending:

Keypoint detection

Feature detection is necessary to automatically find correspondences between images.

SIFT and SURF are recent keypoint or interest point detector algorithms but a point to note is that these are patented and their commercial usage restricted. Once a feature has been detected, a descriptor method like SIFT descriptor can be applied to later match them.

Registration

To estimate a robust model from the data, a common method used is known as RANSAC. For the problem of homography estimation, RANSAC works by trying to fit several models using some of the point pairs and then checking if the models were able to relate most of the points. The best model – the homography, which produces the highest number of correct matches – is then chosen as the answer to the problem.

Alignment

Alignment may be necessary to transform an image to match the viewpoint of the image it is being composited with. Projective transformation is the farthest an image can transform (in the set of two-dimensional planar transformations).

Compositing

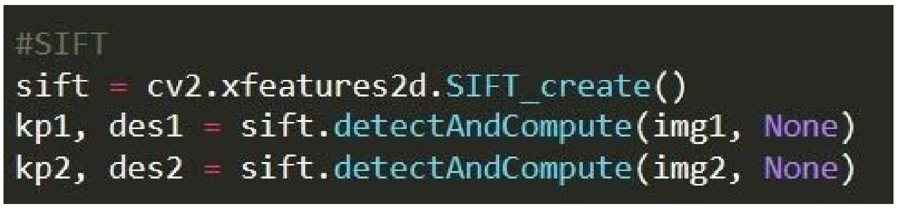
Compositing is the process where the rectified images are aligned in such a way that they appear as a single shot of a scene. Compositing can be automatically done since the algorithm now knows which correspondences overlap.

Blending

Image blending involves executing the adjustments figured out in the calibration stage, combined with the remapping of the images to an output projection. Images are blended together and seam line adjustment is done to minimize the visibility of seams between images.

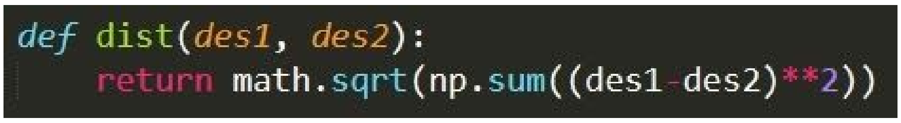
2. Implementation Procedure

(1) Interest points detection and feature description by SIFT

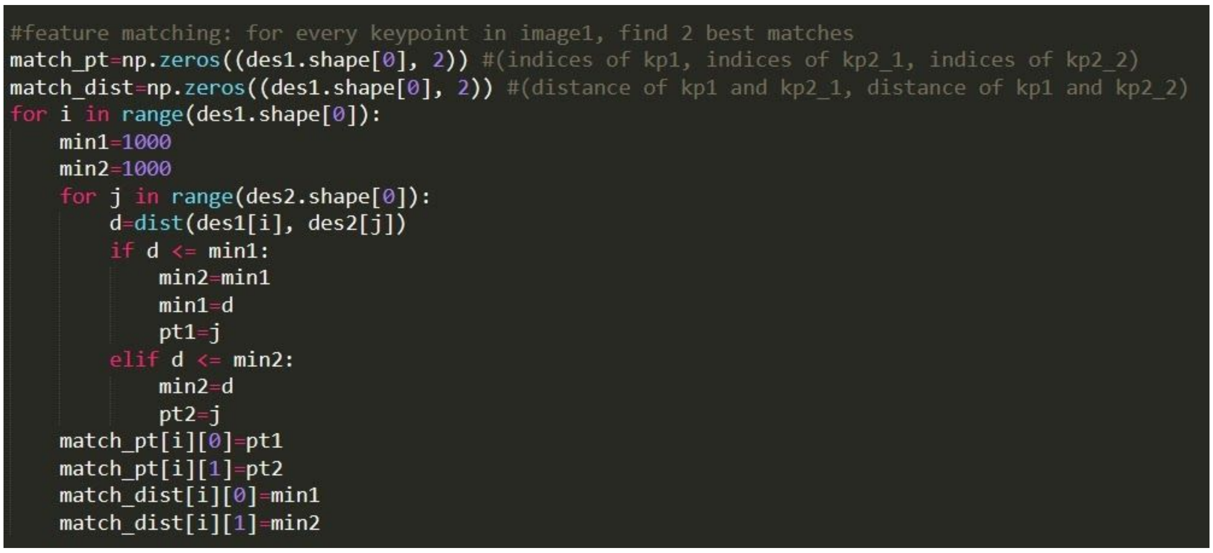


```
#SIFT
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
```

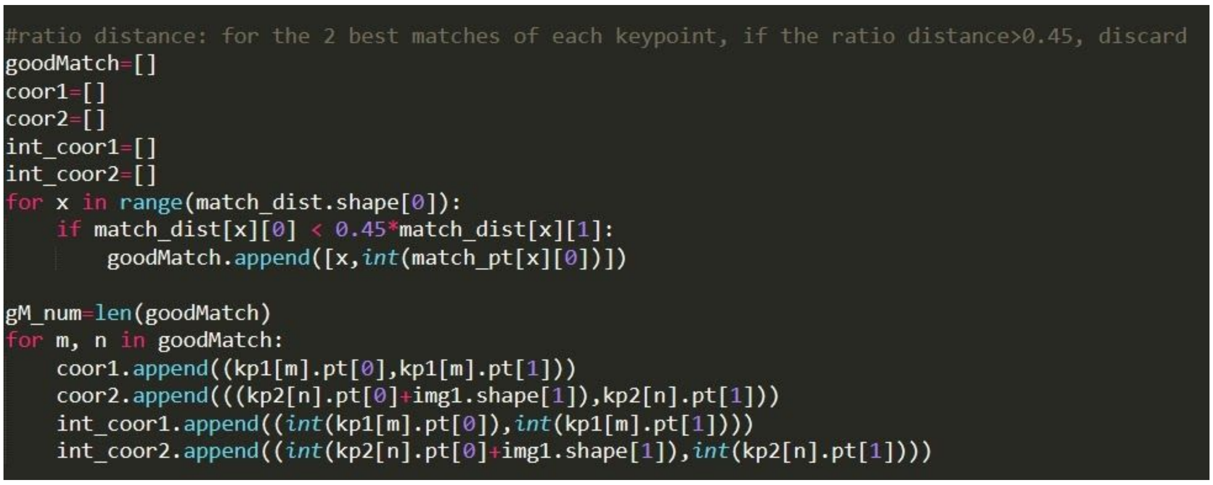
(2) Feature matching by SIFT features
For every keypoint in image1, find the 2 closest keypoints in image2. Set ratio distance threshold to get the best matches.



```
def dist(des1, des2):
    return math.sqrt(np.sum((des1-des2)**2))
```



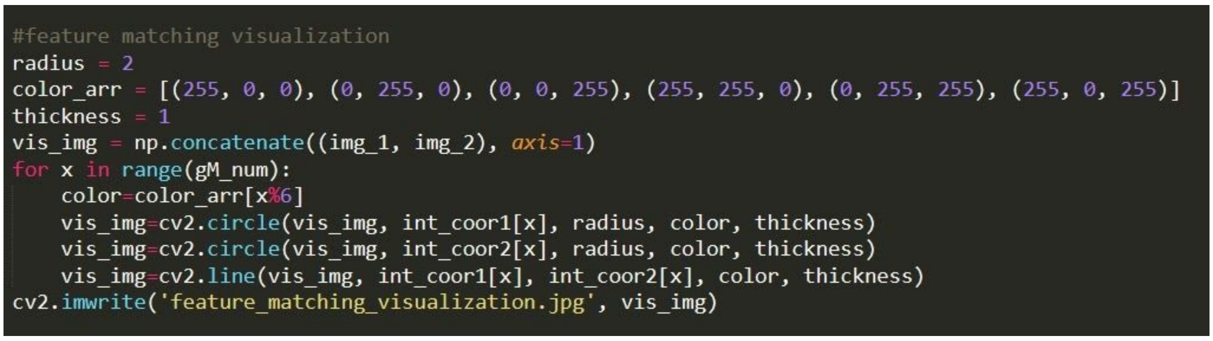
```
#feature matching: for every keypoint in image1, find 2 best matches
match_pt=np.zeros((des1.shape[0], 2)) #(indices of kp1, indices of kp2_1, indices of kp2_2)
match_dist=np.zeros((des1.shape[0], 2)) #(distance of kp1 and kp2_1, distance of kp1 and kp2_2)
for i in range(des1.shape[0]):
    min1=1000
    min2=1000
    for j in range(des2.shape[0]):
        d=dist(des1[i], des2[j])
        if d <= min1:
            min2=min1
            min1=d
            pt1=j
        elif d <= min2:
            min2=d
            pt2=j
    match_pt[i][0]=pt1
    match_pt[i][1]=pt2
    match_dist[i][0]=min1
    match_dist[i][1]=min2
```



```
#ratio distance: for the 2 best matches of each keypoint, if the ratio distance>0.45, discard
goodMatch=[]
coor1=[]
coor2=[]
int_coor1=[]
int_coor2=[]
for x in range(match_dist.shape[0]):
    if match_dist[x][0] < 0.45*match_dist[x][1]:
        goodMatch.append([x,int(match_pt[x][0])])

gM_num=len(goodMatch)
for m, n in goodMatch:
    coor1.append((kp1[m].pt[0],kp1[m].pt[1]))
    coor2.append(((kp2[n].pt[0]+img1.shape[1]),kp2[n].pt[1]))
    int_coor1.append((int(kp1[m].pt[0]),int(kp1[m].pt[1])))
    int_coor2.append((int(kp2[n].pt[0]+img1.shape[1]),int(kp2[n].pt[1])))
```

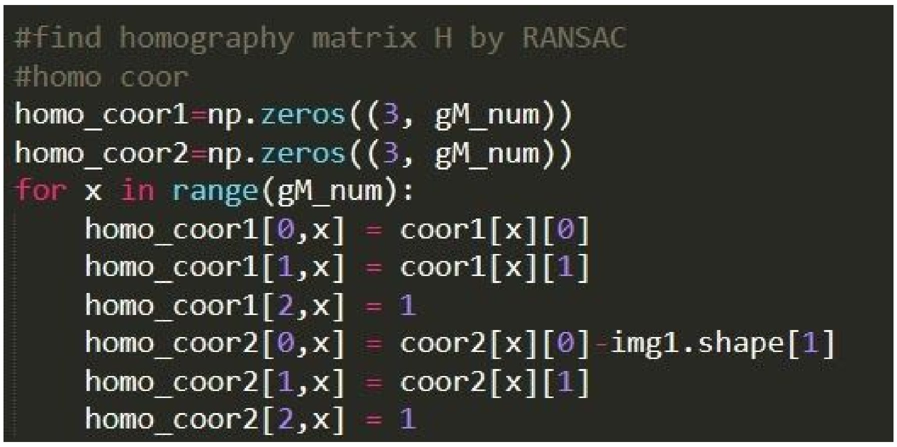
Visualization:



```
#feature matching visualization
radius = 2
color_arr = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0, 255, 255), (255, 0, 255)]
thickness = 1
vis_img = np.concatenate((img_1, img_2), axis=1)
for x in range(gM_num):
    color=color_arr[x%6]
    vis_img=cv2.circle(vis_img, int_coor1[x], radius, color, thickness)
    vis_img=cv2.circle(vis_img, int_coor2[x], radius, color, thickness)
    vis_img=cv2.line(vis_img, int_coor1[x], int_coor2[x], color, thickness)
cv2.imwrite('feature_matching_visualization.jpg', vis_img)
```

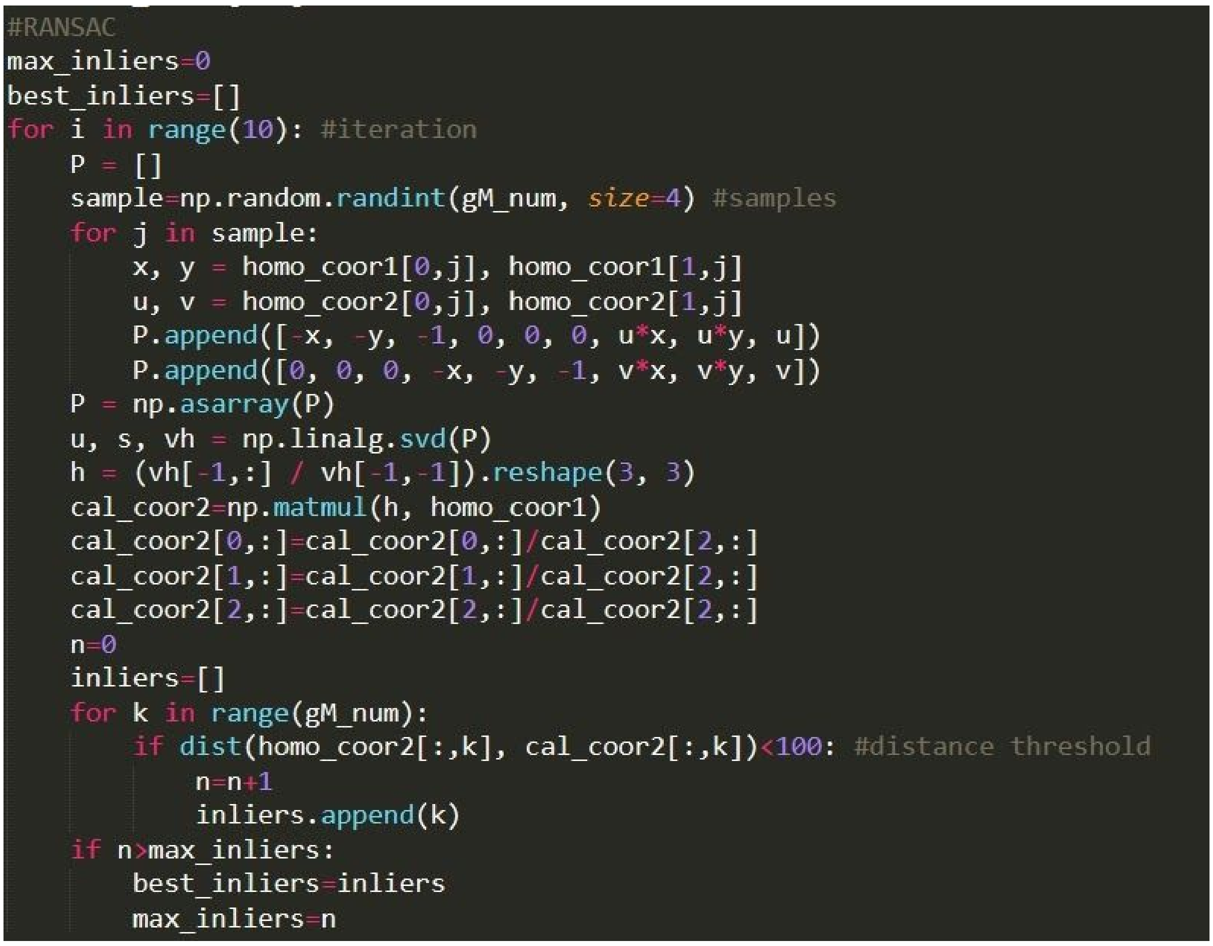
(3) Find homograpy matrix:

a. Get homogeneous coordinate

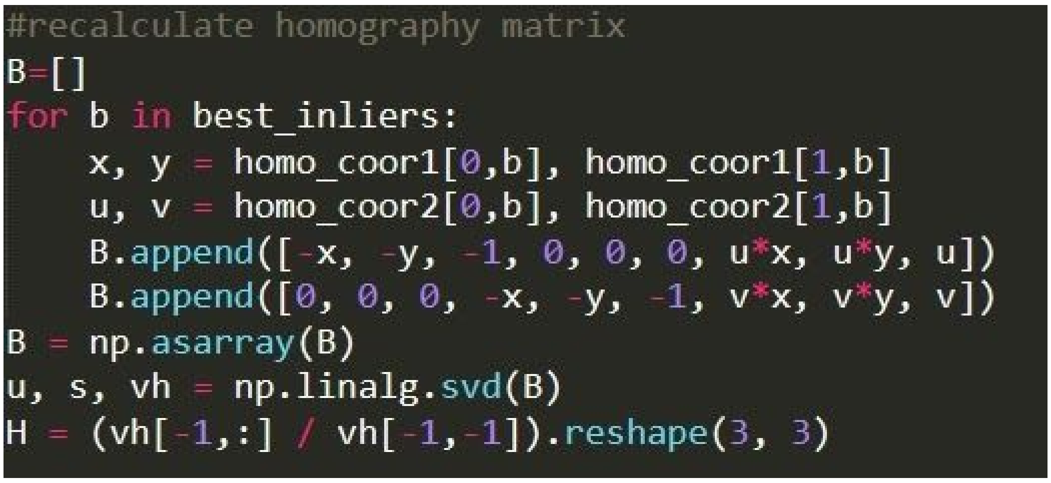


```python
#find homography matrix H by RANSAC
#homo coor
homo_coor1=np.zeros((3, gM_num))
homo_coor2=np.zeros((3, gM_num))
for x in range(gM_num):
    homo_coor1[0,x] = coor1[x][0]
    homo_coor1[1,x] = coor1[x][1]
    homo_coor1[2,x] = 1
    homo_coor2[0,x] = coor2[x][0]-img1.shape[1]
    homo_coor2[1,x] = coor2[x][1]
    homo_coor2[2,x] = 1
```

b. RANSAC implementation: Randomly choose 4 pairs of points to calculate homography matrix. Transform the features from Image1’s coordinate to Image2’s coordinate with the homography matrix and calculate the distance between the matching features on Image2, and set distance threshold to determine inliers and outliers. Finally, choose the largest number of inliers and recalculate the homography matrix with them.

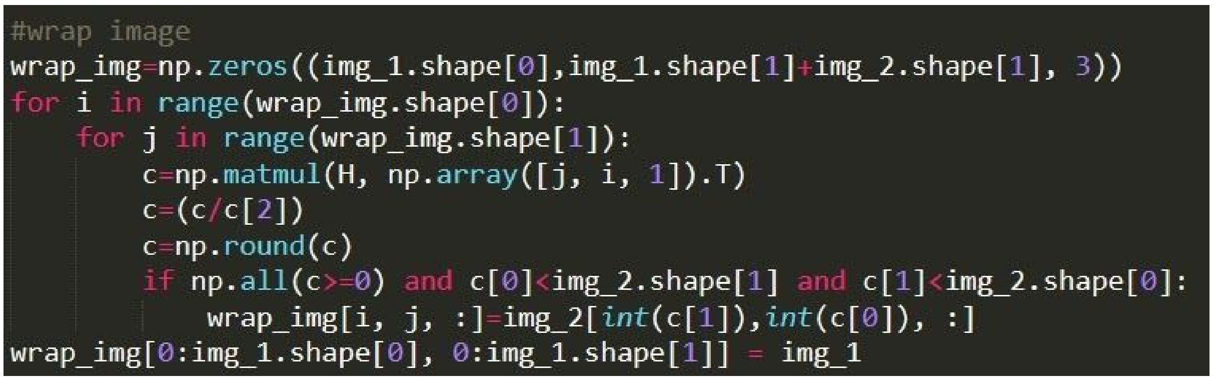


```python
#RANSAC
max_inliers=0
best_inliers=[]
for i in range(10): #iteration
    P = []
    sample=np.random.randint(gM_num, size=4) #samples
    for j in sample:
        x, y = homo_coor1[0,j], homo_coor1[1,j]
        u, v = homo_coor2[0,j], homo_coor2[1,j]
        P.append([-x, -y, -1, 0, 0, 0, u*x, u*y, u])
        P.append([0, 0, 0, -x, -y, -1, v*x, v*y, v])
    P = np.asarray(P)
    u, s, vh = np.linalg.svd(P)
    h = (vh[-1,:] / vh[-1,-1]).reshape(3, 3)
    cal_coor2=np.matmul(h, homo_coor1)
    cal_coor2[0,:]=cal_coor2[0,:]/cal_coor2[2,:]
    cal_coor2[1,:]=cal_coor2[1,:]/cal_coor2[2,:]
    cal_coor2[2,:]=cal_coor2[2,:]/cal_coor2[2,:]
    n=0
    inliers=[]
    for k in range(gM_num):
        if dist(homo_coor2[:,k], cal_coor2[:,k])<100: #distance threshold
            n=n+1
            inliers.append(k)
    if n>max_inliers:
        best_inliers=inliers
        max_inliers=n
```



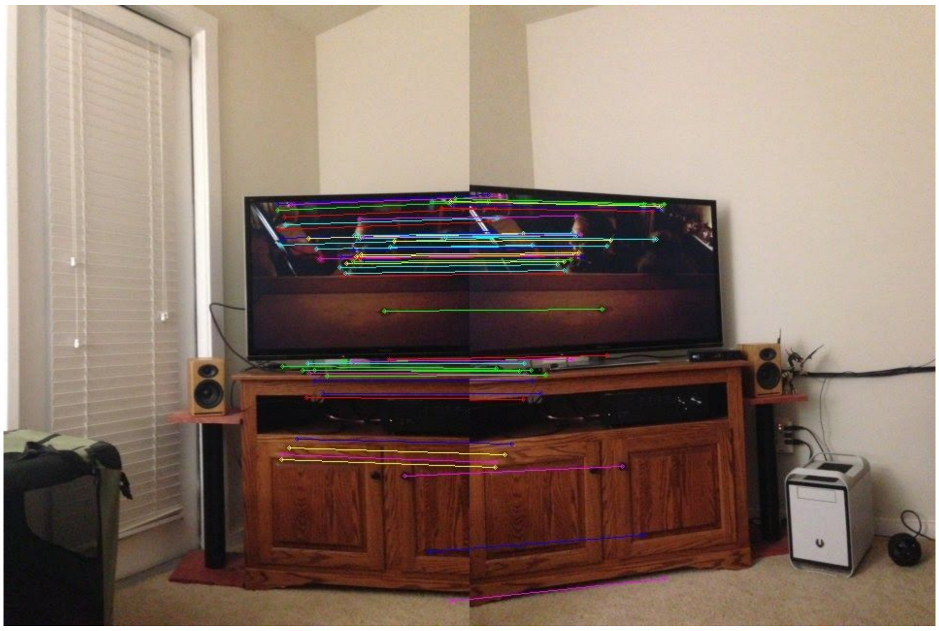
```python
#recalculate homography matrix
B=[]
for b in best_inliers:
    x, y = homo_coor1[0,b], homo_coor1[1,b]
    u, v = homo_coor2[0,b], homo_coor2[1,b]
    B.append([-x, -y, -1, 0, 0, 0, u*x, u*y, u])
    B.append([0, 0, 0, -x, -y, -1, v*x, v*y, v])
B = np.asarray(B)
u, s, vh = np.linalg.svd(B)
H = (vh[-1,:] / vh[-1,-1]).reshape(3, 3)
```

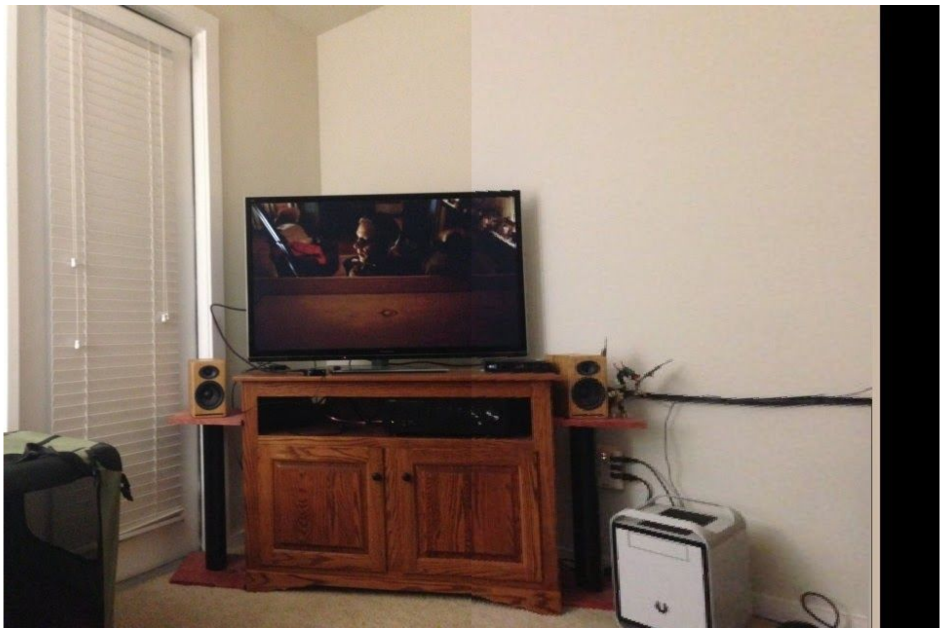
(4) Warp perspective of Image2 to fit Image1

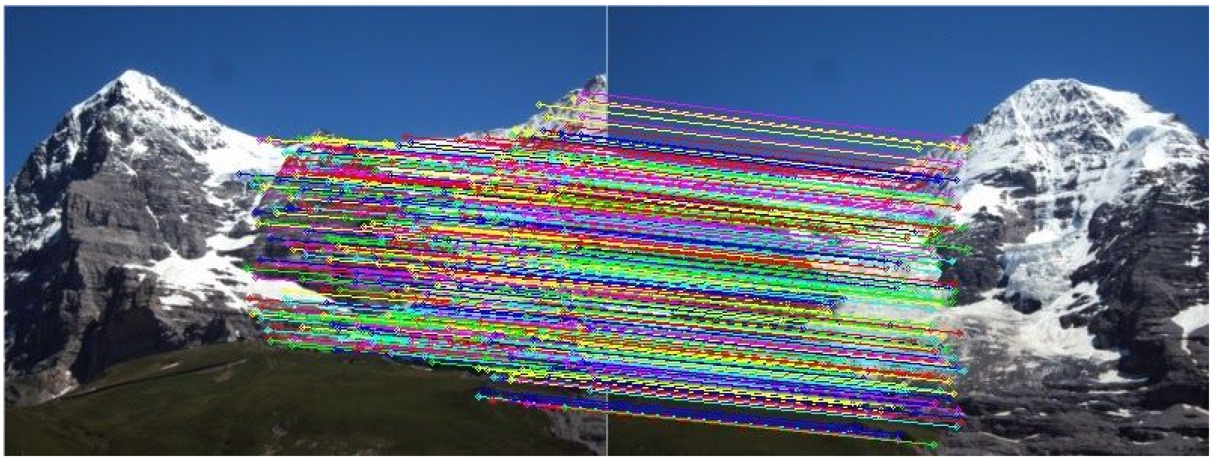
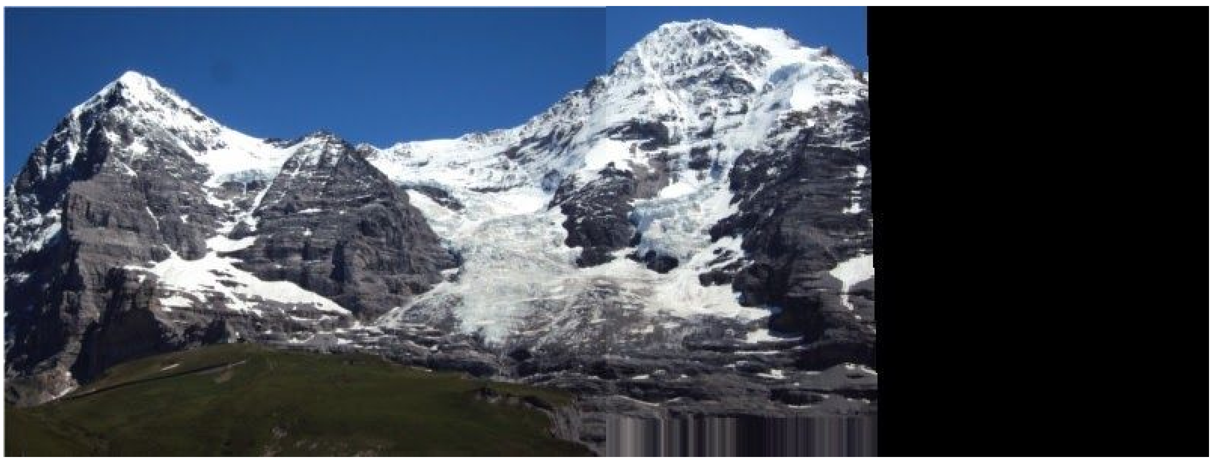


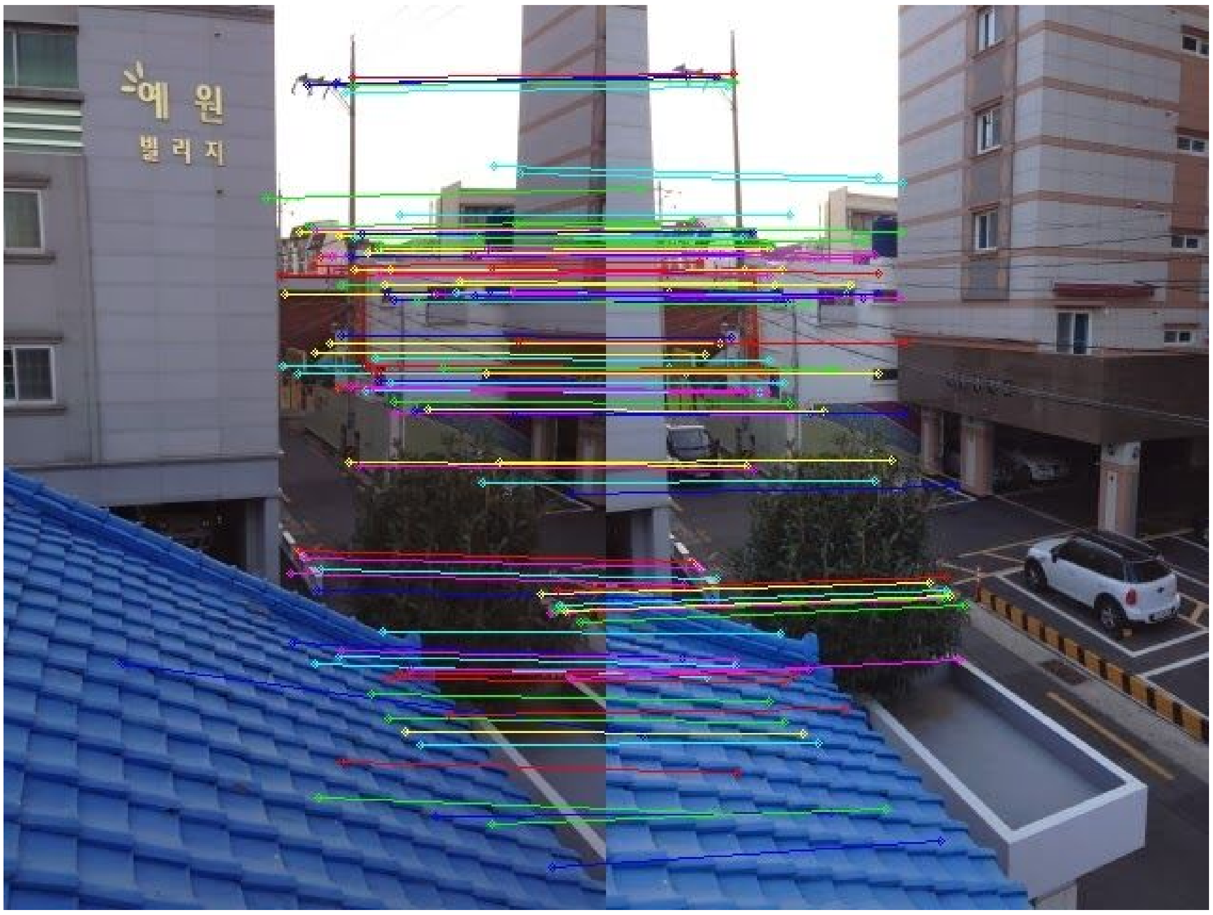
```python
#wrap image
wrap_img=np.zeros((img_1.shape[0],img_1.shape[1]+img_2.shape[1], 3))
for i in range(wrap_img.shape[0]):
    for j in range(wrap_img.shape[1]):
        c=np.matmul(H, np.array([j, i, 1]).T)
        c=(c/c[2])
        c=np.round(c)
        if np.all(c>=0) and c[0]<img_2.shape[1] and c[1]<img_2.shape[0]:
            wrap_img[i, j, :]=img_2[int(c[1]),int(c[0]), :]
wrap_img[0:img_1.shape[0], 0:img_1.shape[1]] = img_1
```

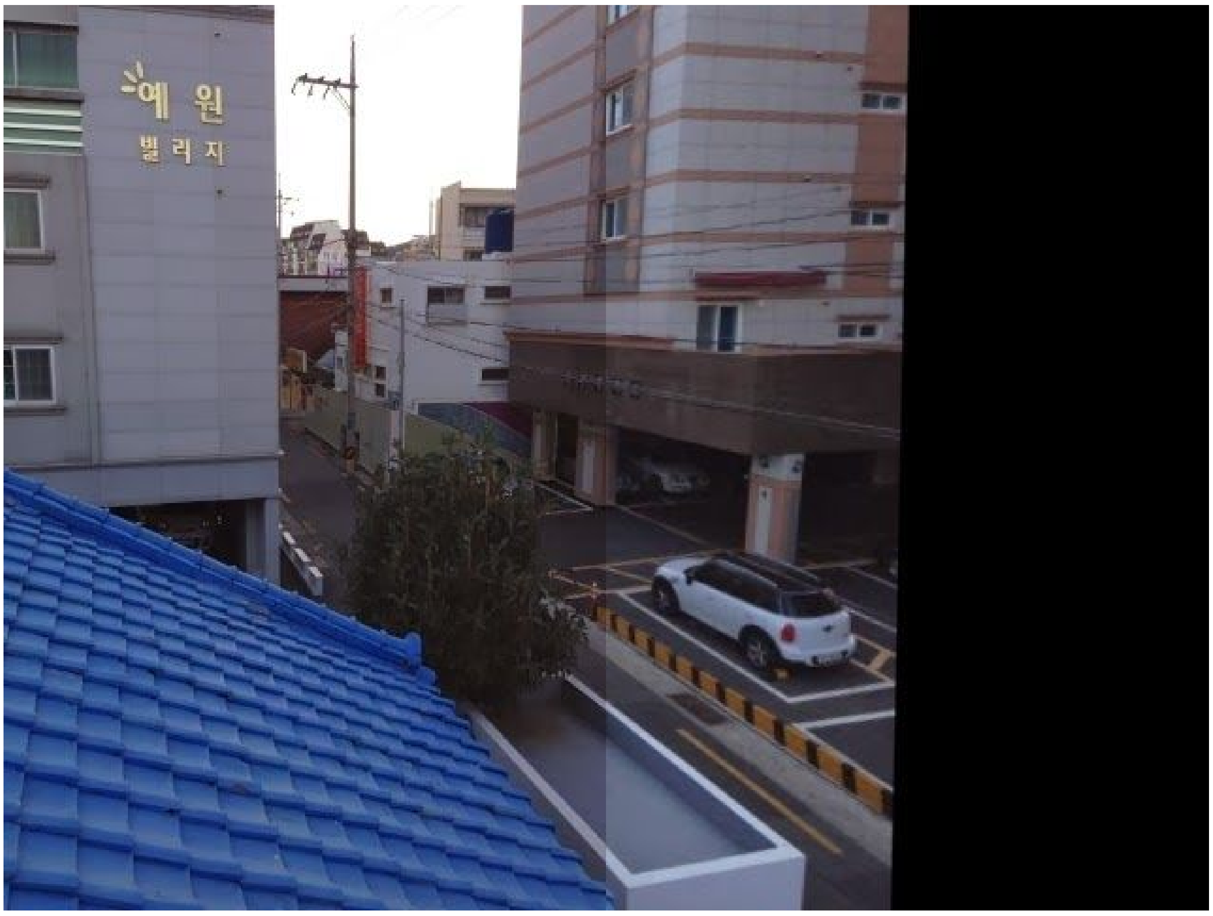
3. Experimental Result

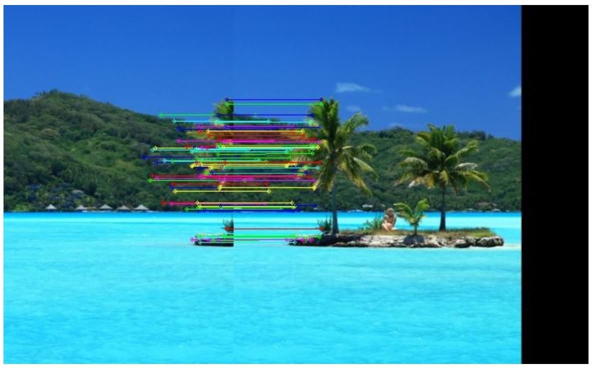


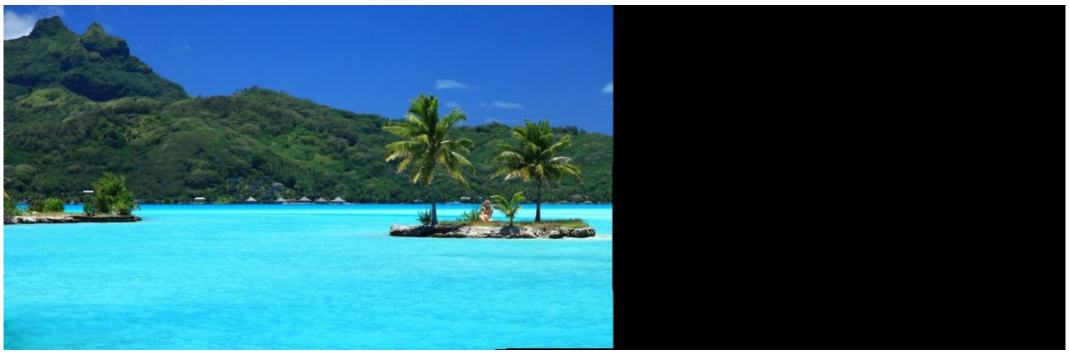



예원
빌리지


예원
빌리지





4. Discussion

In the warp function, the differences between coordinate and array index took us lots of time to find out the correct relation. For instance, (i,j) in coordinate need to be written as [j,i] in array index.

After all the calculation, unfortunately, the result of the panorama image cannot be shown by **cv2.imshow**, however, the result can be written into an image file by **cv2.imwrite** first, and then it can be loaded with **cv2.imread** for the further process, we still haven't figured out the reason.

5. Conclusion

Automatic panoramic image stitching is now widely available in the camera software of mobile phones. It is a method that can take several photos and then connect these photos to make a wide-angle view or a representation of physical space. The photos to be connected are usually taken in the same direction, but there are Panoramic from all the directions (Photo Sphere).

For a series of photos taken in the same direction, it's easy to make a complete panoramic image by aligning the overlapping parts of the series of photos. But in fact, it still needs a series of methods to get a better panoramic image.

Digital photography of the late twentieth century greatly simplified this assembly process, which is now known as image stitching. Such stitched images may even be fashioned into forms of like virtual reality movies.

6. Work Assignment

(1) 軒轅照雯： coding step 1~ step 4

(2) 賴奕善： coding step 4 & report

(3) 王泓仁： report & testing