Computer Vision: Representation and Recognition
Assignment 3

171860607, Jinbin Bai, jinbin5bai@gmail.com

June 3, 2021

1 Image Mosaics

1.1 Getting correspondences

The code was implemented in the function

```
    def getting_correspondences(A,B,numPoints=8)
```

This function takes two image addresses as input and two lists of coordinates of some points (default is 8 in total and you can adjust it to any even number) as output. And this function mainly uses ginput to obtain some points you choose.

1.2 Computing the homography parameters

The code was implemented in the function

```
    def computing_homography_parameters(ptA,ptB)
```

This function takes two lists of coordinates from two images as input and homography matrix as output. In this function, we first nomalize the point sets, then try to solve an linear equation(1.1).

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_2 & y_1x_2 & x_2 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_2 & y_1y_2 & y_2 \end{bmatrix} H = 0 \tag{1.1}$$

Then we reshape H to a 3x3 matrix and add some normalization on it.

In order to verify our function, we wirte a function

```
    def verify_homography_matrix(imgA,imgB,H)
```

In this function, when you click on the point in the picture on the left, the corresponding point in the picture on the right will be marked. Figure 1 shows some points our function marked.

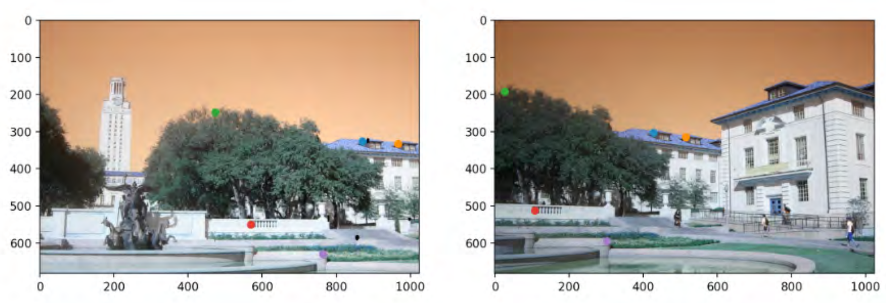

Figure 1: verify homography matrix

1.3 Warping between image planes

The code was implemented in the function

```
    def computing_homography_parameters(ptA, ptB)
```

In this function, we take the recovered homography matrix and an image as input, and we try to obtain the mapping relationship of each point coordinate, then we get a mapped picture size to finish transformation. Figure 2 shows transformed picture our function done.

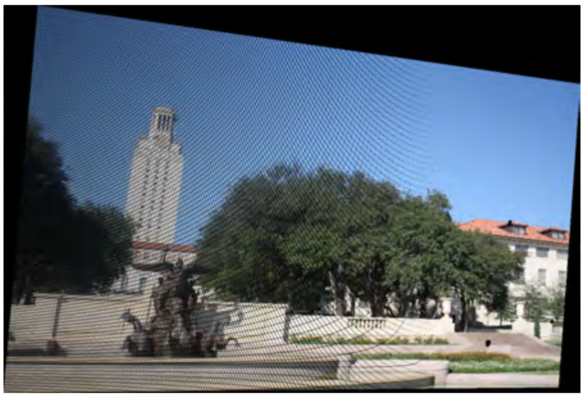

Figure 2: transformed picture

To avoid holes in the output, we then use an inverse warp to fill in the black area on the right side of the picture and the black holes in the middle. As a result, Figure 3 shows transformed picture after inverse warpping.

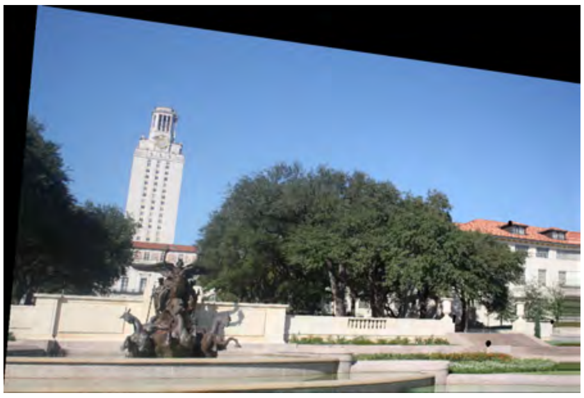

Figure 3: transformed picture after inverse warpping

1.4 Create the output mosaic

The code was implemented also in the part 3 of the function

```
    def computing_homography_parameters(ptA, ptB)
```

In this part, we first calculate the output size and then mapping both two images to the output images. As a result, Figure 4 shows our final output.

1.5 After writing and debugging your system

Apply your system to the provided pair of images, and display the output mosaic. The picture was show in Figure 4.

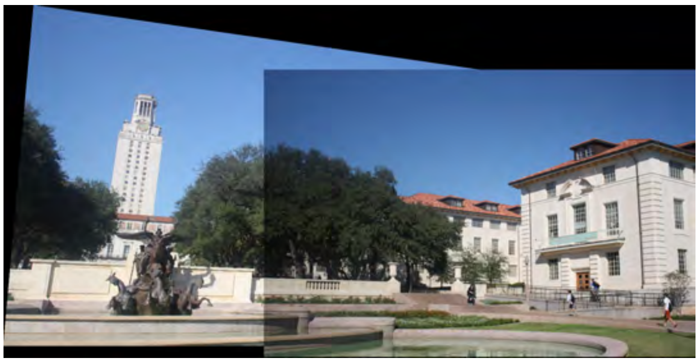

Figure 4: mosaic picture

Show one additional example of a mosaic you create using images that you have taken. The picture was show in Figure 5 and Figure 6.

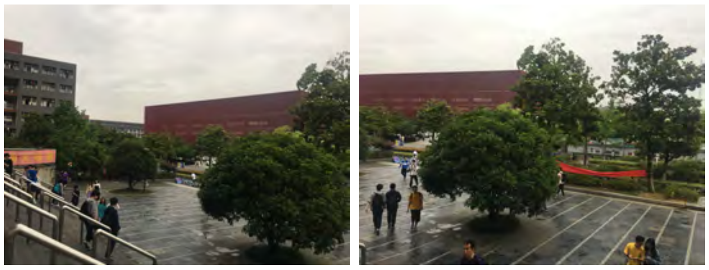

Figure 5: Source Images

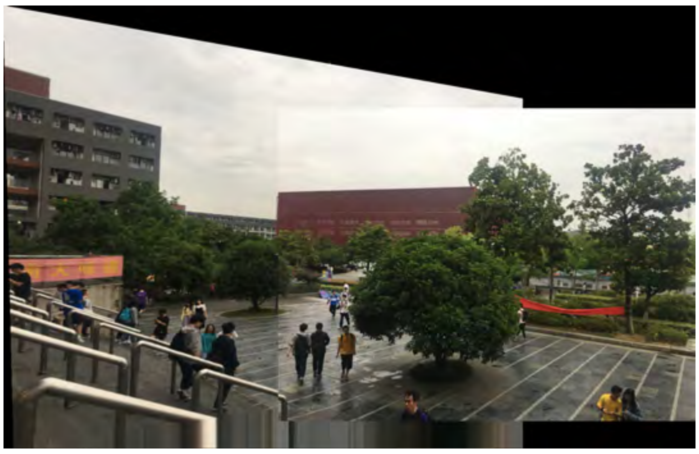

Figure 6: Target Image

Warp one image into a "frame" region in the second image. The picture was show in Figure 7 and Figure 8. One thing to note is in order to meet the requirements of this task, we slightly modified the code of part 3, we first print the picture on the right, and then print the picture on the left to prevent the picture on the left from being overwritten. In addition, part 2 fills the black parts and holes of the image generated after the transformation of the left image, which modifies the shape of the left image, so we commented out part 2 temporarily in this topic.

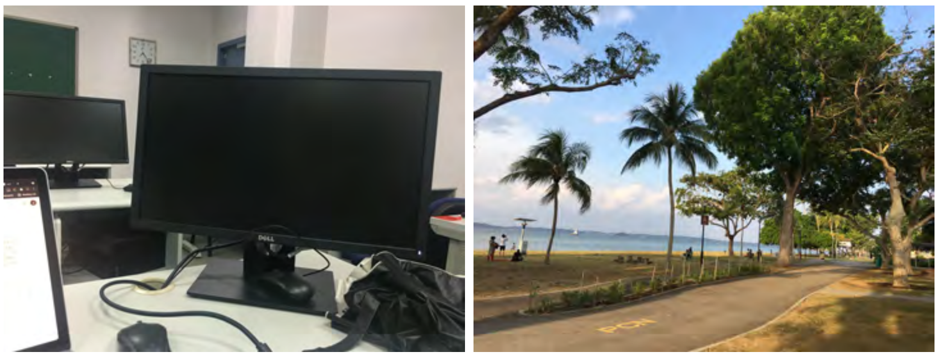

Figure 7: Source Images

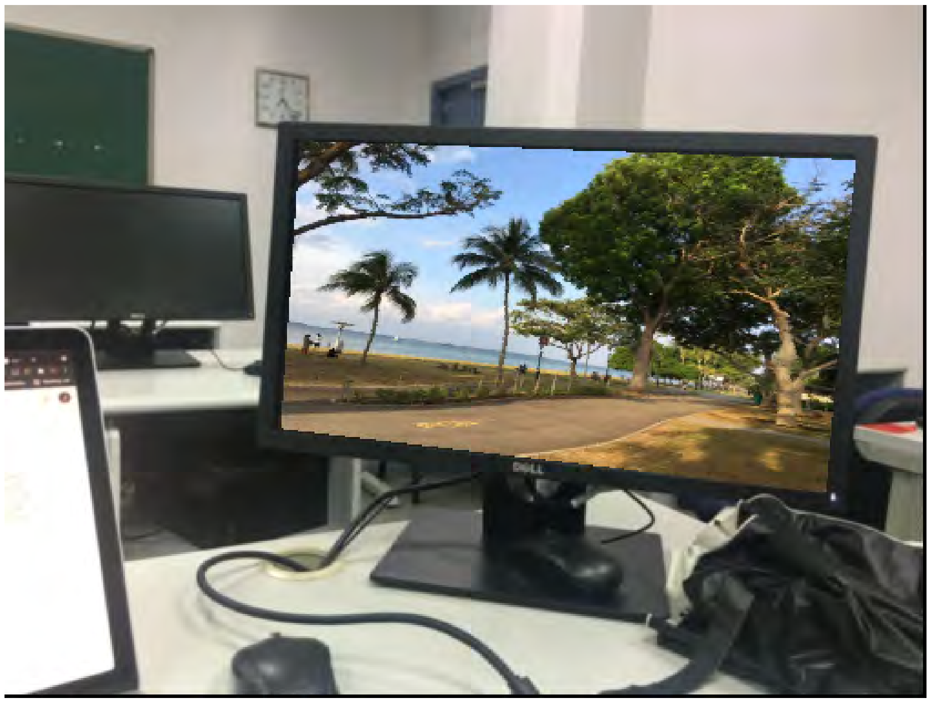

Figure 8: Target Image

2 Automatic Image Mosaics

2.1 Use VLFeat to automatically obtain interest points and descriptors

The code was implemented in the function

```python
def getting_correspondences_sift(img1_url, img2_url)
```

In this part, we first use cv2.SIFT_create to obtain key points and descriptors, then we use knnMatch algorithm to obtain some pairs of points which is the most matched. Some results are drawn in Figure 1.1

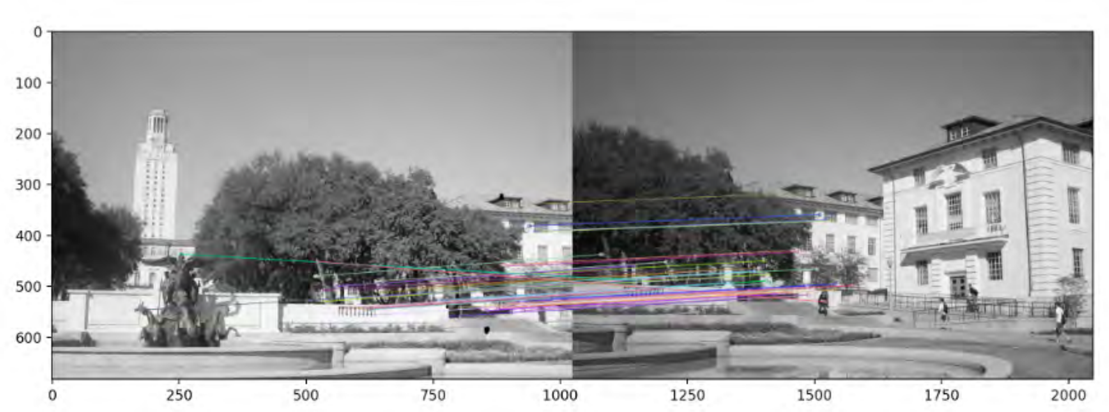

Figure 9: SIFT

2.2 Implement RANSAC for robustly estimating the homography matrix from noisy correspondences

In this part, we first use SIFT to obtain enough key points pairs, then use a RANSAC method to process our point pairs and remove some outliers. Figure 10 and Figure 11 show difference between whether using RANSAC or not.

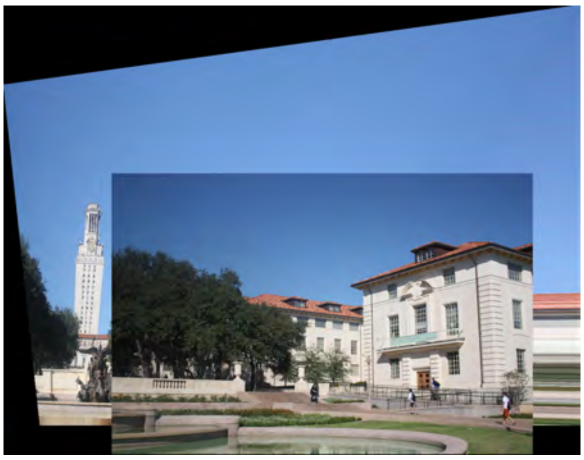

Figure 10: Without using RANSAC

We can easily find that Figure 10 is worse. That is because if we donnot remove some outliers points which affect the parameters of H Significantly, these points will greatly change the value of H, which makes the resulting image very poor.

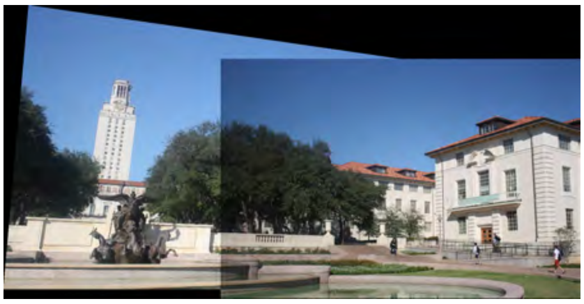

Figure 11: Using RANSAC

More details about H are listed in following. H1 means without using RANSAC and H2 means using RANSAC.

```
H1: [[ 1.22562159e-01 -1.00657850e+00  4.66088966e+02]
 [ 7.15784504e-02 -5.71116019e-01  2.89298874e+02]
 [ 1.60548148e-04 -1.96684707e-03  1.00000000e+00]]
H2: [[ 1.26819326e+00 -9.46937197e-02 -5.47546091e+02]
 [ 1.63669337e-01  1.17493177e+00 -1.53238342e+02]
 [ 2.70966020e-04 -1.76559314e-05  1.00000000e+00]]
Difference between the two matricies:
[[-1.14563110e+00 -9.11884785e-01  1.01363506e+03]
 [-9.20908865e-02 -1.74604779e+00  4.42537216e+02]
 [-1.10417872e-04 -1.94919114e-03  0.00000000e+00]]
Total error =  162.0
```