Computer Vision: Representation and Recognition
Assignment 3

211240076, Liu Jiaxin, 211240076@smail.nju.edu.cn

June 20, 2024

1 Question 1

1.1 question1.1

```
def get_correspondences(img1, img2, num_points=4)
```

takes the addresses of two images and returns the coordinates of selected points. plt.ginput is used to get the coordinates of selected points.

1.2 question1.2

We assume $\mathbf{H}$ is the homography matrix, i.e.

$$\lambda \mathbf{p} = \mathbf{H}\mathbf{p}' \tag{1.1}$$

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.2}$$

Eliminate λ, set $h_{33} = 1$, we have

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{1.3}$$

We assume

$$\mathbf{A} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' \end{bmatrix},$$

$$\mathbf{x} = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix},$$

$$\mathbf{b} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

The function

```
def compute_homography_paras(pt1, pt2):
```

uses np.linalg.lstsq to solve it and get the homography matrix.

```
def verify_H(img1, img2, H)
```

To verify the homography matrix we compute, we can simply choose point in image1, apply **H** on it and see if the corresponding point is correct or not.

1.3 question1.3

```
def warp_between_planes(img1_path, H)
```

takes the recovered homography matrix and an image, and returns a new image that is the warp of the input image using H we computed before.
Here I use np.meshgrid to compute the coordinates for the grid. Then I map the grid using H and get the bounding box of the warped image.

```
x, y = np.meshgrid(np.arange(w), np.arange(h))
coords = np.stack([x.flatten(), y.flatten(), np.ones_like(x.flatten())], axis=1)
# Apply the homography transformation to the grid
   new_coords = (H @ coords.T).T # shape: (.., 3)
```

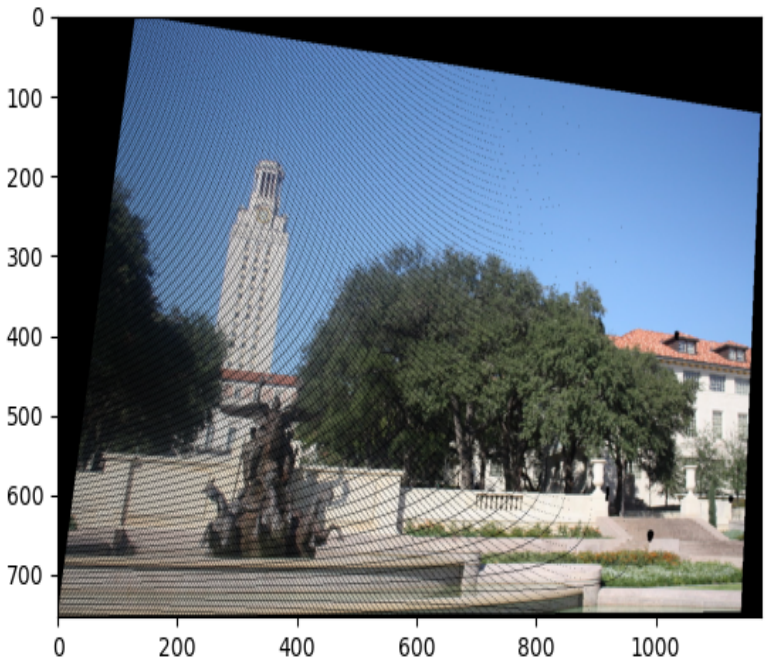

Figure 1: warped image without inverse warp

then I interpolate the pixel values in the warped image.
In order to avoid holes in the output, an inverse warp is used to fill in the black area. Result is shown as follows:

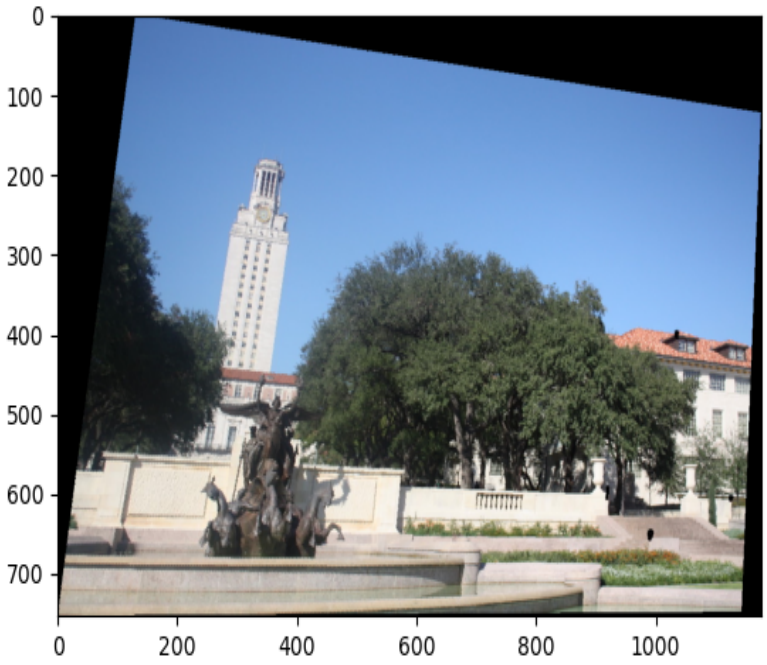

Figure 2: warped image with inverse warp

1.4 question1.4

The function

```
def merge_images(warped_image, boundingbox, img2_path)
```

creates a merged image showing the mosaic. Specifically, this function combined the warpped image and the original second image

(1) Apply your system to the provided pair of images, and display the output mosaic.

See "output mosaic1"

Figure 3: output mosaic1

(2) Show one additional example

See "output mosaic2"

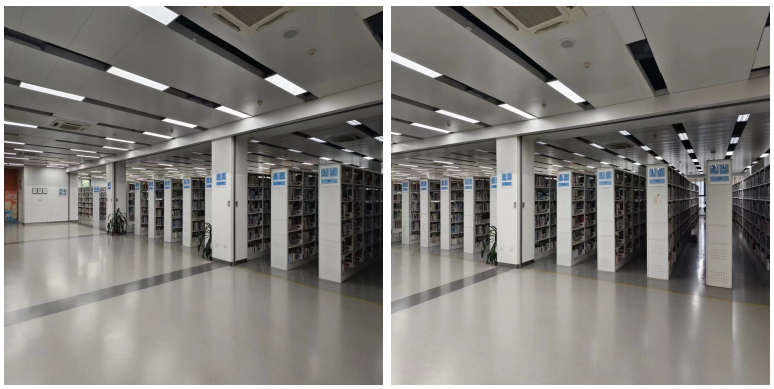

Figure 4: images used

Figure 5: output mosaic2

(3) Warp one image into a "frame" region

We can warp the image to the frame by clicking the points of the frame and the four corners of the desired image.
Notice we should overwrite the frame image. So we need to switch the order in the function merge_images

```
    for j in range(img2.shape[0]):
        for i in range(img2.shape[1]):
            merged_image[j-y_zero][i-x_zero] = img2[j][i]
for j in range(warped_image.shape[0]):
  for i in range(warped_image.shape[1]):
    if np.any(warped_image[j][i]): ## don't write black pixel
      merged_image[j+min_y-y_zero][i+min_x-x_zero] = warped_image[j][i]
```

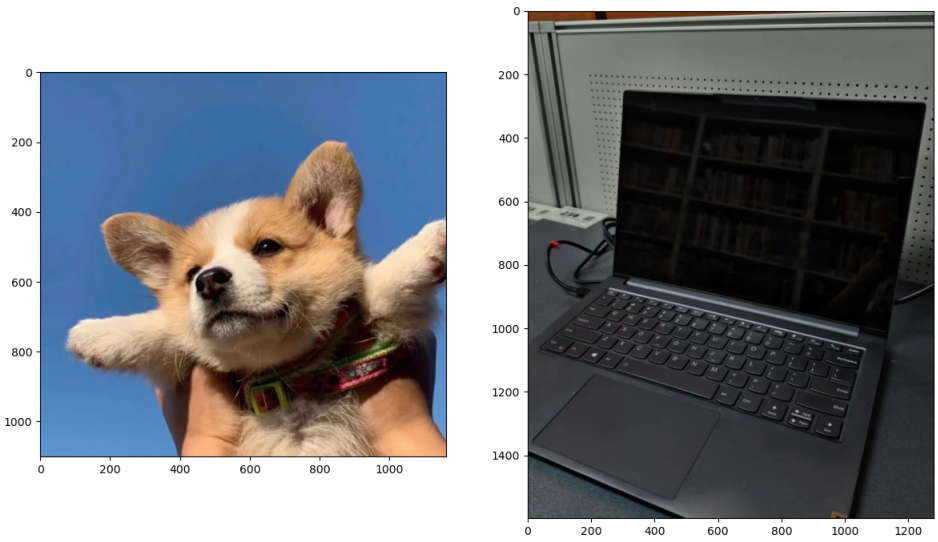

Figure 6: frame and image

Figure 7: Output image

2 Question 2

2.1 question2.1

The function

```
def get_correspondences_sift(image1, image2)
```

uses sift to automatically identify matching descriptors in the two input images.
Specifically, it first converts two images into grayscale and compute SIFT keypoints and descriptors for both images. Then it finds matching descriptors using brute-force matching. After that, ratio test is applied to filter good matches. SO we can get the desired correspondences.

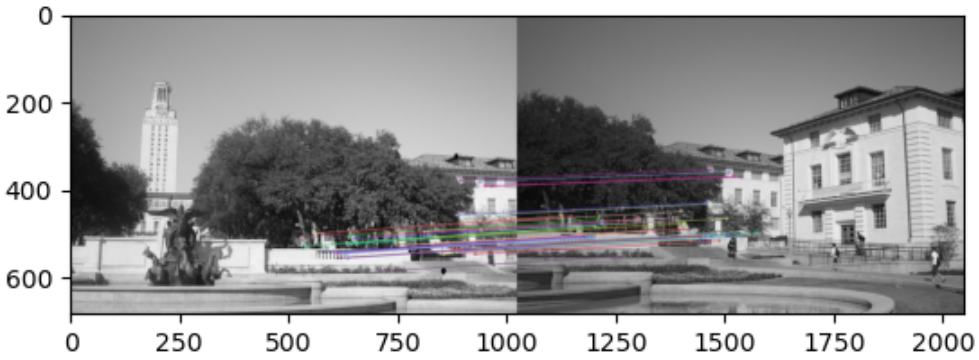

Figure 8: automatically obtain interest points and descriptors with SIFT

Reusing the codes (still use compute_homography_paras(pt1, pt2) to compute homography matrix H), we can get the results:

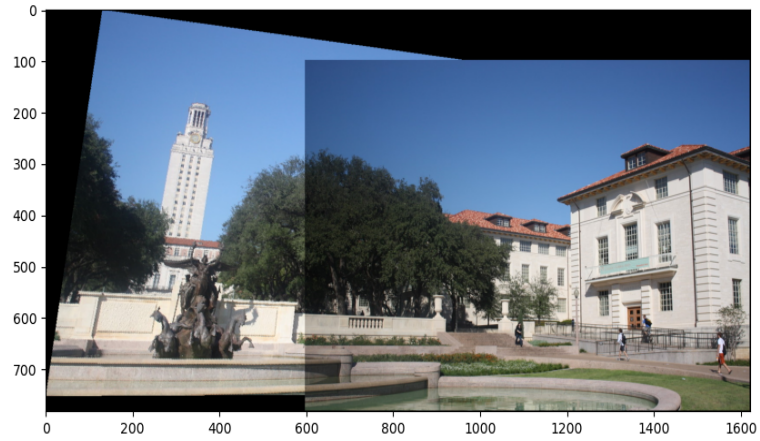

Figure 9: results using SIFT

2.2 question2.2

```
# ordinary method RANSAC (4th parameters should be in [1, 10])
    H, mask = cv2.findHomography(np.asarray(pt1), np.asarray(pt2), 0, 5.0)
    # RANSAC
    H, mask = cv2.findHomography(np.asarray(pt1), np.asarray(pt2), cv2.RANSAC, 5.0)
```

Below figures shows the difference between whether or not using RANSAC.

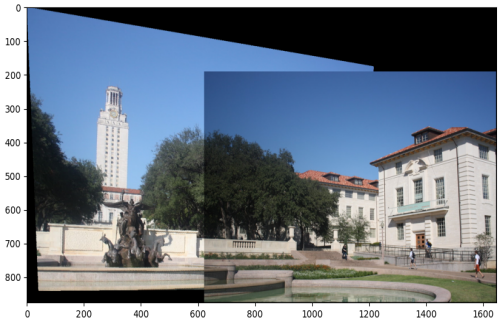

Figure 10: results without using RANSAC

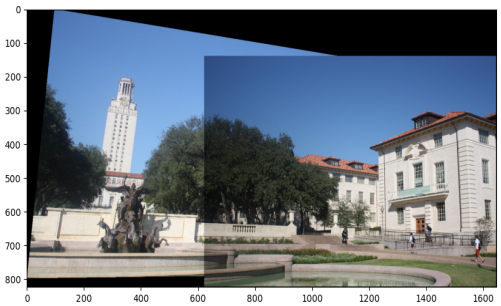

Figure 11: results using RANSAC

We can see in this case RANSAC slightly improves the alignment, since RANSAC removes some outliers. Here are more obvious results showing the advantages of RANSAC:

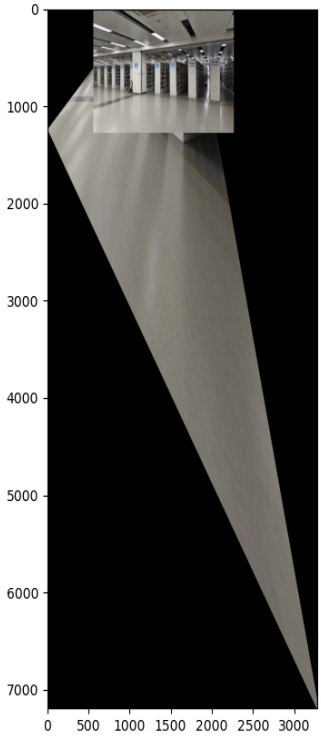

Figure 12: another result without using RANSAC

We see that the mosaic image is pretty worse. That is because some outliers make bad influence on the output.

Figure 13: another result using RANSAC