

# Assignment 1: Panoramic Image Stitching

Name: LAI YICHENG

Student ID: A0206573E

## Part 1: 2D convolution

### 1. Sobel kernel



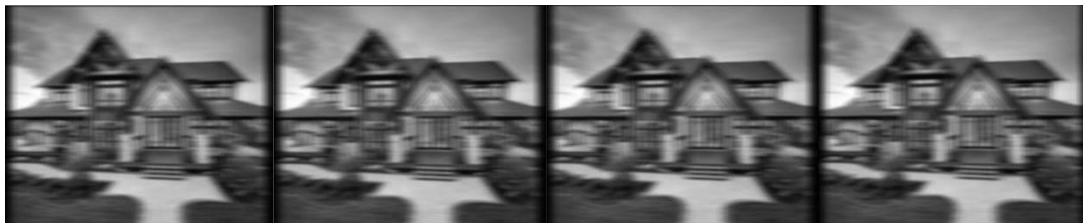
Figure 1-1: convolution image using sobel kernel(size=1)

### 2. Gaussian kernel



Figure 1-2: convolution image using Gaussian kernel(left\_size=1,right\_size=4)

### 3. Haar-like masks



1\*2(size=4)

2\*1(size=4)

1\*3(size=4)

3\*1(size=4)



2\*2(size=4)

2\*2(size=10)

Figure 1-3: convolution image using 5 Haar-like masks respectively

### Conclusion:

The outputs show that Sobel kernel can be used in edge detection since the contour of the sample house is quite clear (figure 1). Besides, the Gaussian kernel is used to blur an image shown in figure 2. when scaling up the Gaussian kernel, image become more blurry after convolution which means blurriness could be operated by setting the mask size.

As displayed in figure 3, when scaling up the mask size, haar-like kernel could compute larger patches to get the features of different sizes.

## Part 2: SIFT Features and Descriptors



Figure 2-1: Descriptors of keypoints extracted by SIFT

## Part 3: Homography

### 1. GUI to click 4 points on each of the images

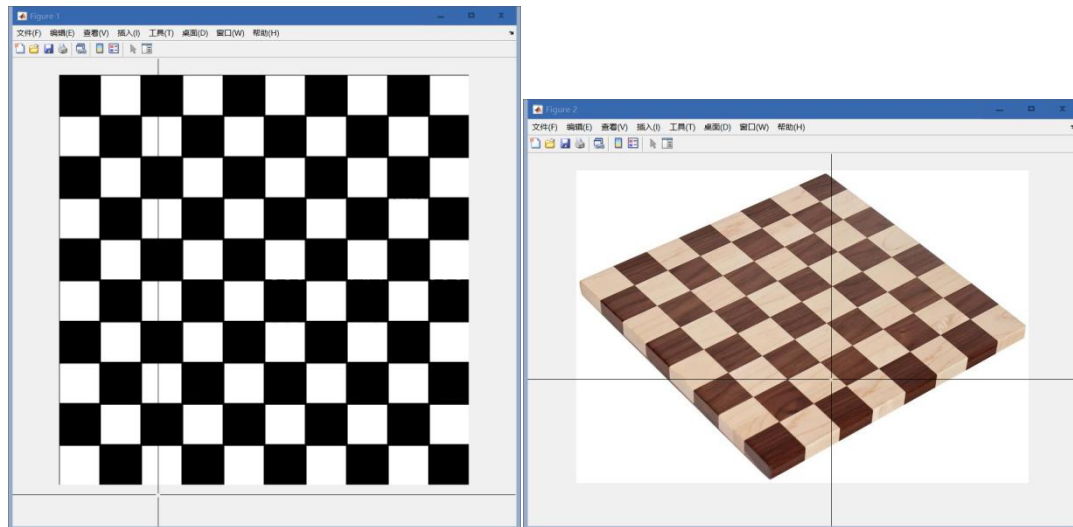


Figure 3-1: GUI of choosing key points for transformation

### 2. Homography matrix

H =

-0.4251	0.1698	-0.0001
-0.2580	-0.2347	0.0001
-6.1247	-210.7402	1.0000

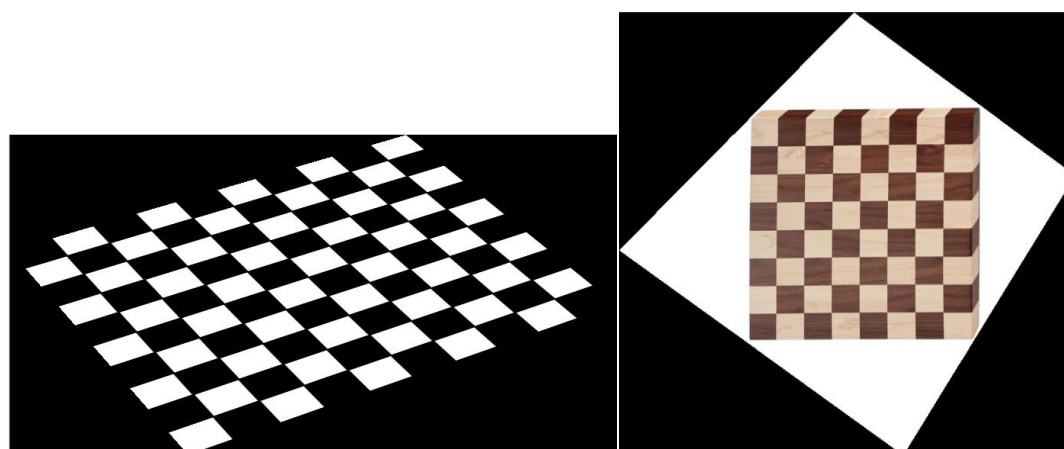
Figure 3-2: Homography matrix h1-to-h2

inv\_H =

-1.6415	-1.1711	-0.0000
1.9929	-3.2962	0.0005
409.9198	-701.8042	1.1120

Figure 3-3: Homography matrix h2-to-h1

### 3. Transform results



(1) h1 to h2

(2) h2 to h1

Figure 3-4: Transformed images

## Part 4: Manual Homography + Sticthing

### 1.GUI to click 4 points on each of the images

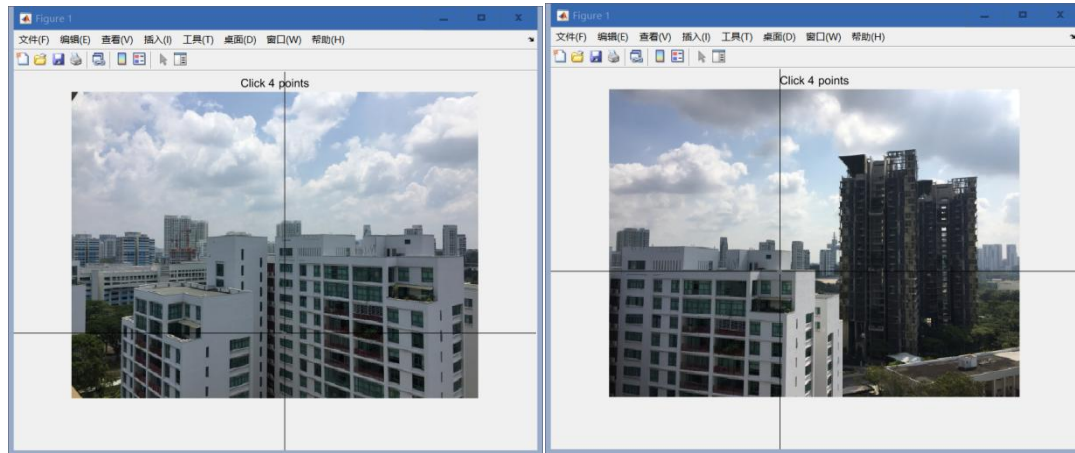


Figure 4-1: GUI of choosing key points for transformation

### 2.Stitched image



Figure 4-2: Results for manual image stitching

### 3.Effect of double edges

Because only 4 points are used for image transformation and the points were chosen manually, the key points can not be matched well enough for stitching which results in the double edges in the overlapping region. In addition, the homography is not the best homography matrix.



## Part 5: Homography + RANSAC

1. Show all the matches



Figure 5-1: All the matches between im01 and im02

2. Show all the inlier matches (support best homography matrix)



Figure 5-2: Best inlier matches between im01 and im02

3. images stitching using the best homography matrix



(1) im2 to im1

(2) im1 to im2

Figure 5-3: stitched image using RANSAC

## Part 6: Basic Panoramic Image

### 1.Stitch 3 pictures



Figure 6-1: stitched image (im01,im02,im03)

### 2.Stitch 4 pictures

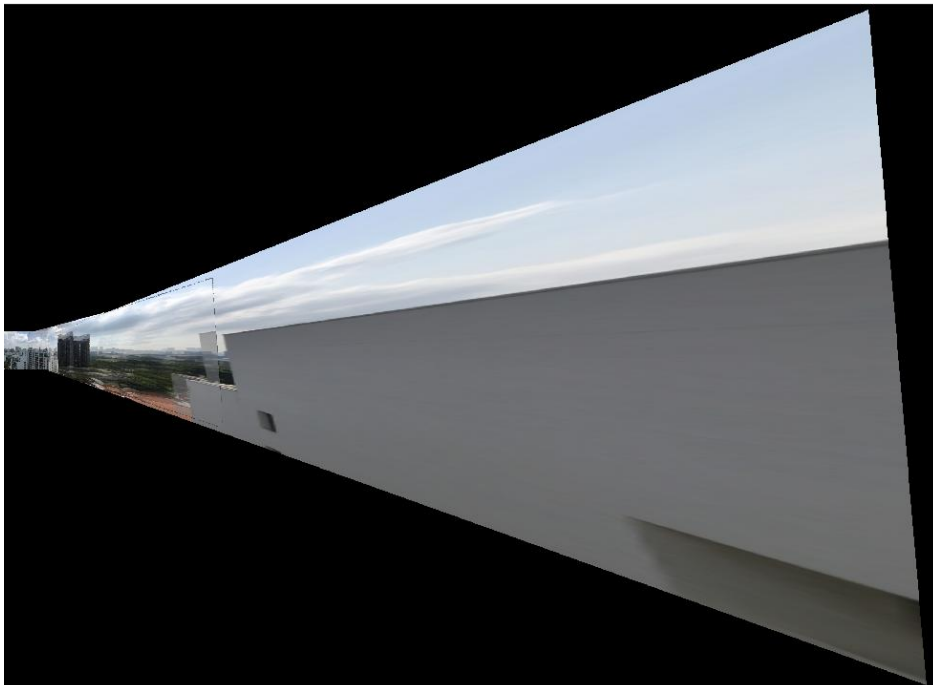


Figure 6-2: stitched image (im01,im02,im03,im04)

3. Stitch 5 pictures(put im03 at the center)



Figure 6-3: stitched image (im01,im02,im03,im04,im05)

## Part 7: Advanced Panoramic Image

### Handling unordered images

#### 1. Algorithm

1) Calculate the distance between each images

Distance = numbers of inliers for the best matches

Then based on the distance,create a matrix (ismatch) to judge if they are matched

If inliers > 0.3\*all-matches the two images could be matched

Set ismatch(i,j) = 1

next x ismatch x sorted_images x						
5x5 double						
	1	2	3	4	5	
1	0	1	0	1	1	
2	1	0	0	0	1	
3	0	0	0	0	1	
4	1	0	0	0	0	
5	1	1	1	0	0	

Figure 7-1: ismatch matrix

2) Forward sort the images

Put one image in the sequence list e.g. `sequence = [1]`

Get the matched images for the last image of the sequence (potential next images)

If the image in next have not been in the sequence, then add it in the sequence.

`Sequence = [sequence, next(i)]`

3) Backward sort the images

`Sequence = [next(i), sequence]`

## 2. Results

1)

Input: [2 3 5 1 4]

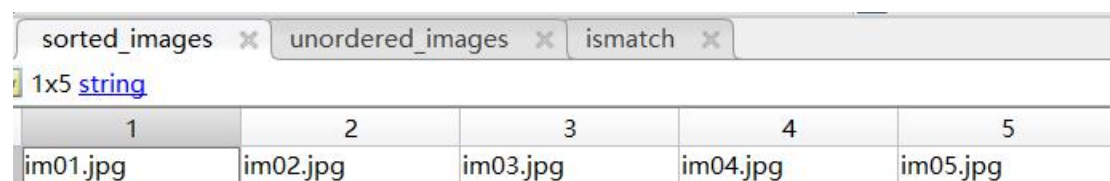


The screenshot shows a Jupyter Notebook interface with two tabs: 'sorted\_images' and 'unordered\_images'. The 'unordered\_images' tab is active, displaying a 1x5 array of strings. Below the array is a table with 5 columns and 2 rows. The first row contains indices 1 through 5, and the second row contains the corresponding image filenames.

1	2	3	4	5
im02.jpg	im03.jpg	im05.jpg	im01.jpg	im04.jpg

Figure 7-1: Unordered sequence 1

Output: [1 2 3 4 5]



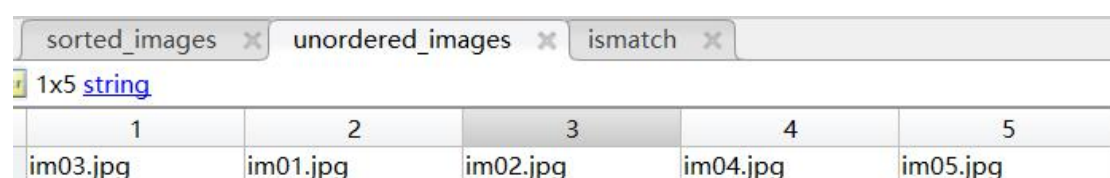
The screenshot shows a Jupyter Notebook interface with three tabs: 'sorted\_images', 'unordered\_images', and 'ismatch'. The 'sorted\_images' tab is active, displaying a 1x5 array of strings. Below the array is a table with 5 columns and 2 rows. The first row contains indices 1 through 5, and the second row contains the corresponding image filenames in sorted order.

1	2	3	4	5
im01.jpg	im02.jpg	im03.jpg	im04.jpg	im05.jpg

Figure 7-2: Sorted sequence 1

2)

Input: [3 1 2 4 5]

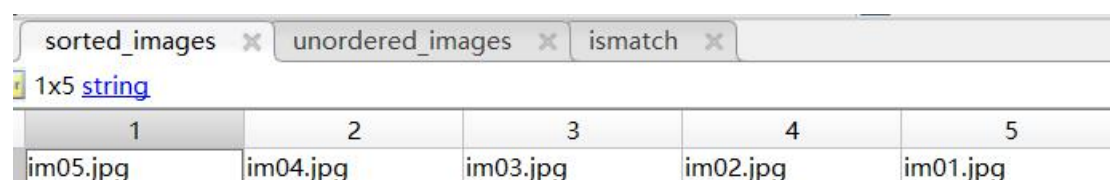


The screenshot shows a Jupyter Notebook interface with three tabs: 'sorted\_images', 'unordered\_images', and 'ismatch'. The 'unordered\_images' tab is active, displaying a 1x5 array of strings. Below the array is a table with 5 columns and 2 rows. The first row contains indices 1 through 5, and the second row contains the corresponding image filenames.

1	2	3	4	5
im03.jpg	im01.jpg	im02.jpg	im04.jpg	im05.jpg

Figure 7-3: Unordered sequence 2

Output: [5 4 3 2 1]



The screenshot shows a Jupyter Notebook interface with three tabs: 'sorted\_images', 'unordered\_images', and 'ismatch'. The 'sorted\_images' tab is active, displaying a 1x5 array of strings. Below the array is a table with 5 columns and 2 rows. The first row contains indices 1 through 5, and the second row contains the corresponding image filenames in sorted order.

1	2	3	4	5
im05.jpg	im04.jpg	im03.jpg	im02.jpg	im01.jpg

Figure 7-4: Sorted sequence 2