



---

## Faculty of Engineering

**Module**

EE5731

**Name**

Zavier Ong Jin Jie

**Matriculation No.**

A0138993L

## Part 1: 2D Convolution

### Instructions to run code

Program files: part\_1.m, conv2d.m

Provided grayscale images: cat.jpg, bicyclewall.jpg

Output images folder: output\_img

Files are in the part\_1 folder. Run part\_1.m for image 2D convolution. Please ensure that the input image is in grayscale, .jpg format and are placed in the same directory as the part\_1 folder.

### Image 2D convolution

According to the lecture notes, image 2D convolution is done by padding the input images with 0s to allow for more space for the kernel to cover the image. 2D convolution follows the equation below:

$$(f * g)(x, y) = \sum_{v=0}^V \sum_{u=0}^U f(u, v)g(x - u, y - v)$$

Where  $f(u, v)$  is the 2D kernel and  $g$  is the padded image.

### Sobel kernel

The Sobel kernel is used mainly for edge detection. As shown in Figure 1,  $G_x$  and  $G_y$  are horizontal and vertical Sobel kernels with the scale of 3.

-1	0	1
-2	0	2
-1	0	1

$G_x$

-1	-2	-1
0	0	0
1	2	1

$G_y$

Figure 1. Horizontal ( $G_x$ ) and Vertical ( $G_y$ ) Sobel Kernels

These 2 kernels can be combined to find the absolute magnitude and orientation of the gradient at each point. The gradient magnitude is the Euclidean distance between  $G_x$  and  $G_y$ .

## Outputs and Conclusions

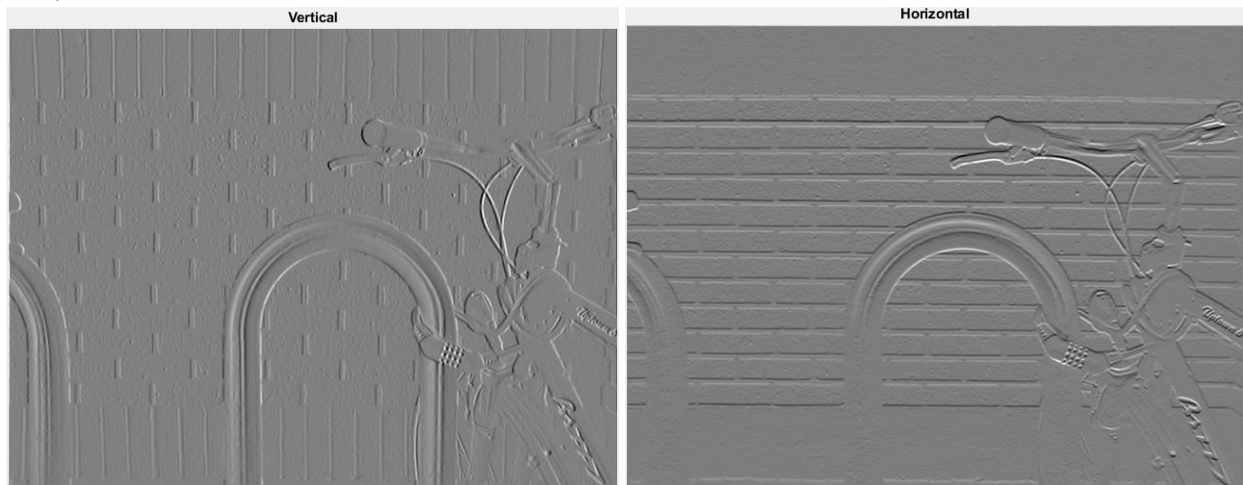


Figure 2. After convolution with  $G_x$  (left) and  $G_y$  (right) kernels

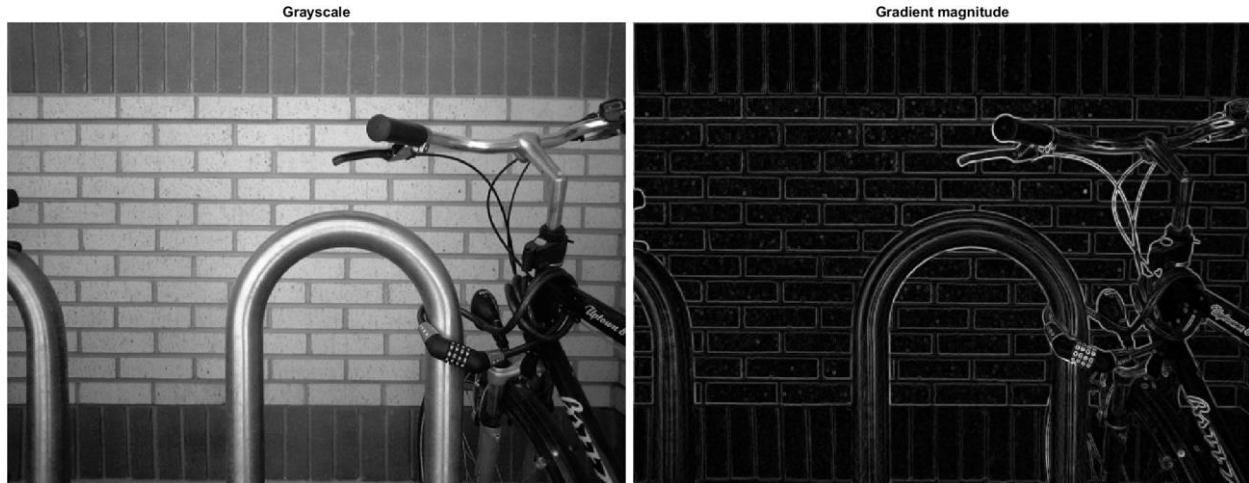


Figure 3. Image before convolution (left), Gradient magnitude (right)

### How the size of the kernel influences the outputs?

As the size of the kernel size increases, it would mean that more pixels are involved in the convolution process. Therefore, the edges in the outputs would become blurry and less defined. As shown in Figure 4 below.



*Figure 4. Output for a 29x29 Sobel kernel*

## Gaussian kernel

The gaussian kernel is used to reduce image noise as well as reduce detail. The gaussian kernel can be defined as:

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where  $\sigma$  is the standard deviation of the gaussian distribution.

## Output and conclusion

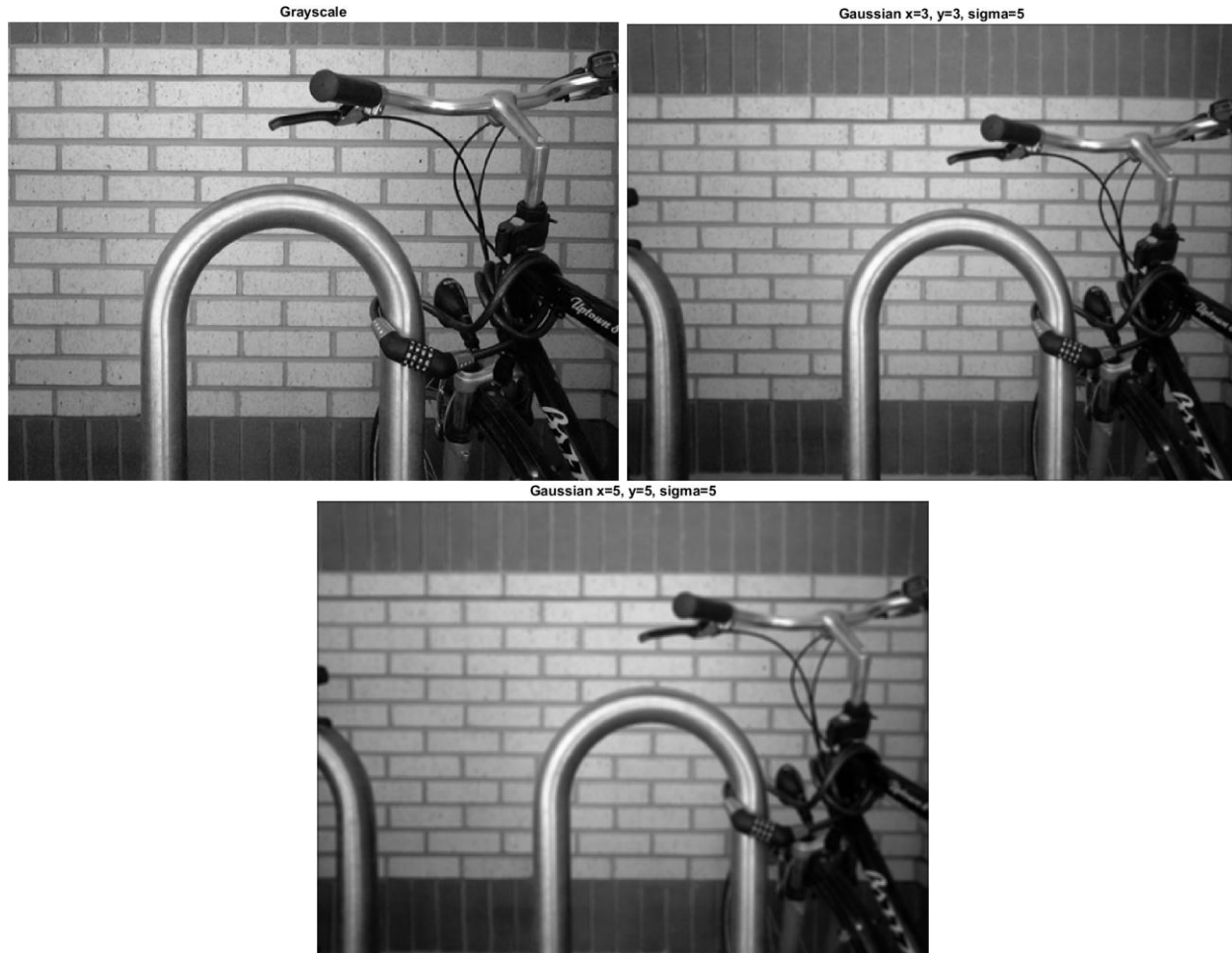


Figure 5. Original image (top left), After convolution with 3x3 gaussian kernel (top right), After convolution with 5x5 gaussian kernel (middle bottom)

## How the size of the kernel influences the outputs?

In a gaussian kernel, pixels that are close to the center of the kernel have more weight compared to those that are further away from the center. With larger kernel sizes, more pixel values are included when calculating the average, implying that a larger kernel will blur the input image more than a smaller kernel as shown in figure 5 where a 5x5 gaussian kernel is more blurred as compared to the 3x3 gaussian kernel.

## 5 Haar-like masks

There are 3 types of Haar-like masks as shown in Figure 6 below.

As shown in (a) in Figure 6, these Haar-like mask are used to detect edges. Edges are areas where the pixels transit from a lighter pixel to a darker pixel. An example of such edges in Figure 5 would be the bicycle handle where the silver metal frame transits to the rubber handle.

For line features (b) in Figure 6, they are used to detect lines as the name suggests. Pixel patterns can vary from white→black→white or black→white→black. An example of such lines in Figure 5 would be the pattern of the brick wall in the background.

For four-rectangle features (c) in Figure 6, it is useful for finding diagonal lines and highlights in an image. Such an example in Figure 5 would be the curves of the silver rails where the bicycle is chained to.

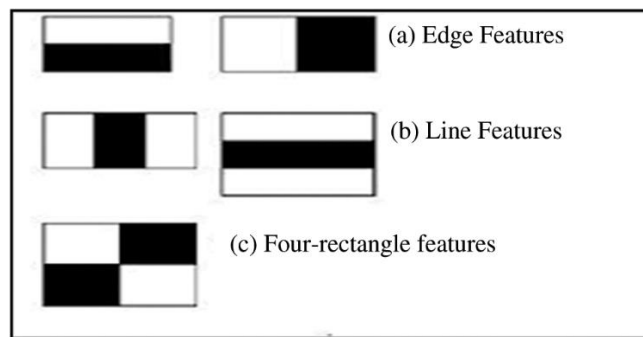


Figure 6. Types of Haar-like masks

## Output and conclusion

Figure 7 depicts the five types of Haar-like masks at different scales. As it may be too small to view in this document, kindly refer to the output images in the output\_img folder for each individual photo.

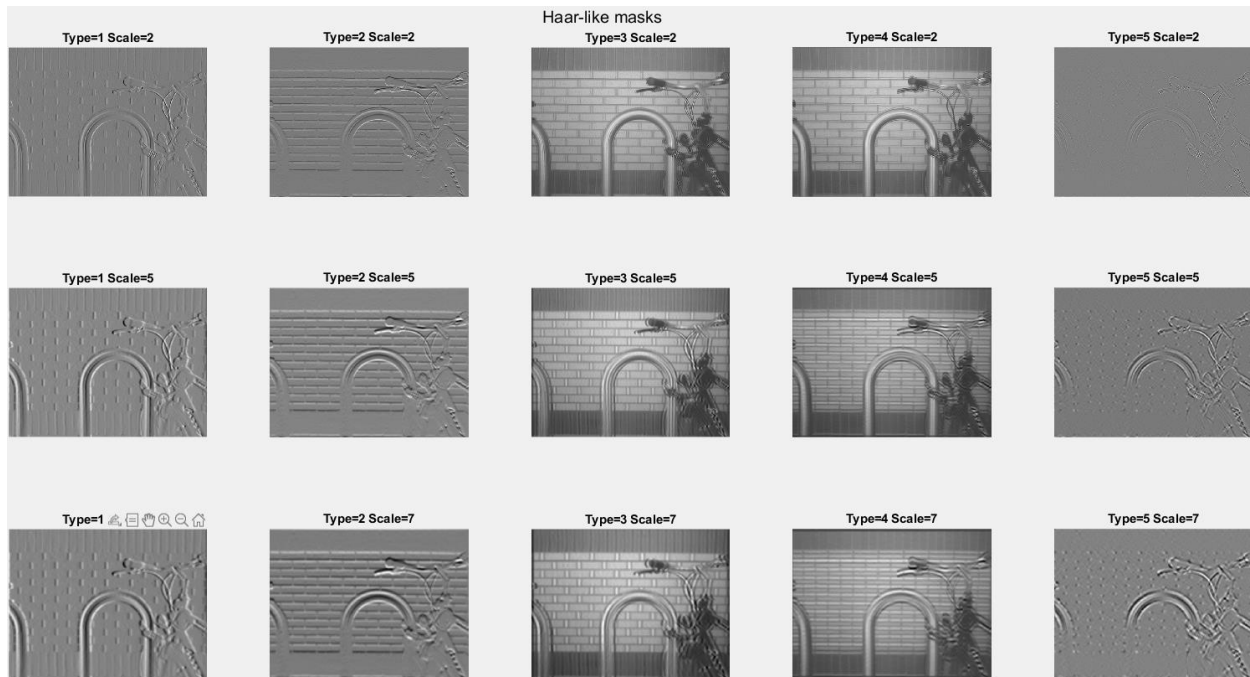


Figure 7. Haar-like masks at scale 2 (top row), 5 (middle row), 7 (bottom row)

### How the size of the kernel influences the outputs?

As shown in Figure 7, type 1, 2 and 5 haar-like masks become more defined and pronounced as the scale increases while type 3,4 become more blur and happen to create a double image effect. With increasing sizes of the haar-like kernel, it will try to locate features that are at larger scale. For instance:

- Type 1 and 2 → thicker and longer edges
- Type 3 and 4 → thicker and longer lines that transit from white→black→white or vice versa.
- Type 5 → larger diagonal lines or corners

For example, for type 5, scale 2, the corners of the bricks in the brick walls cannot be identified as the scales are too small. However, at scale 7, it is observed that the masks can extract the corners of the bricks. Therefore, depending on the features and the resolution of the image, to achieve ideal feature extraction, it may be better to choose different scales for different types of Haar-like masks.



## Part 2: SIFT Features and Descriptors

### Instructions to run code

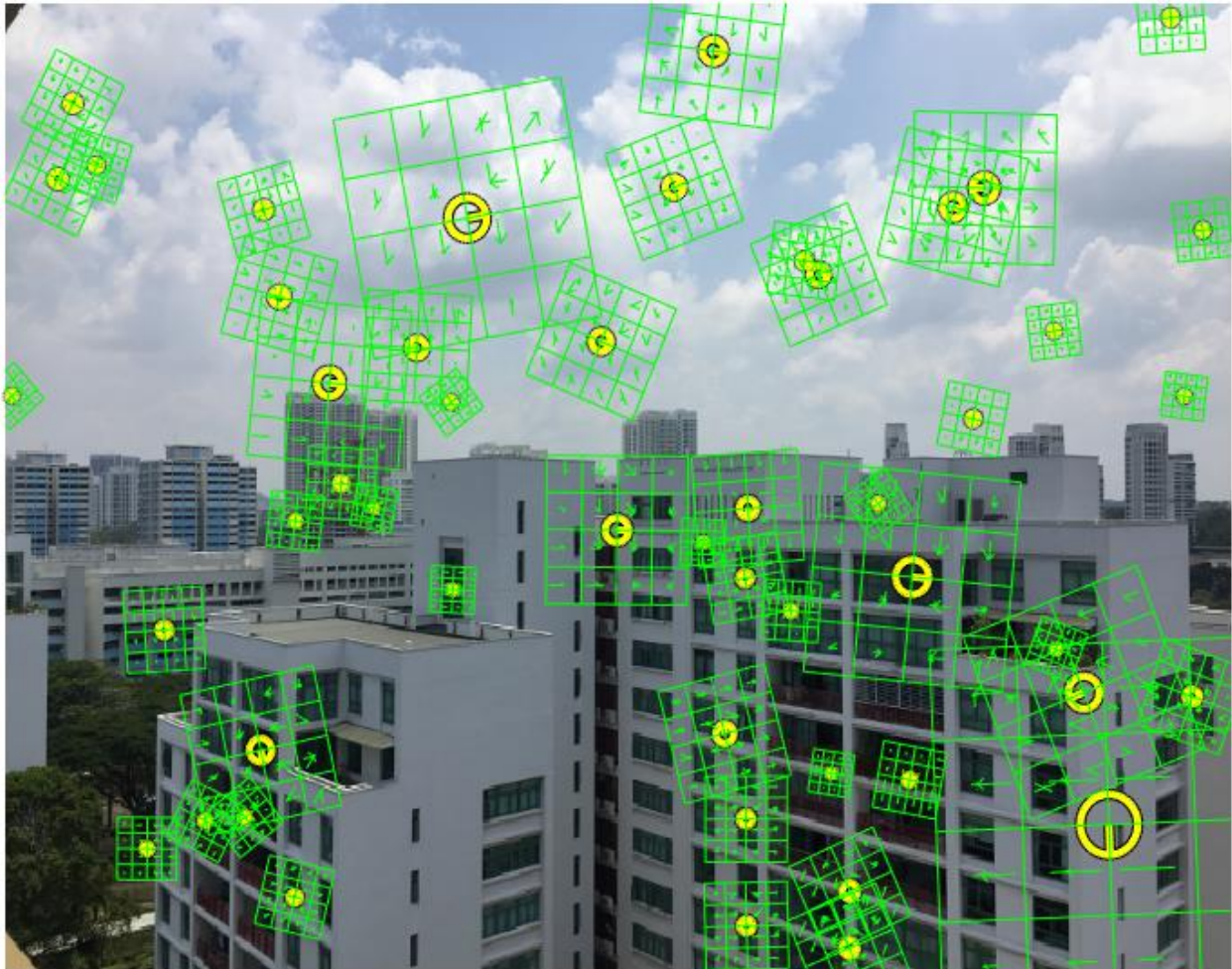
Program files: part\_2.m

Public SIFT code used: vlfeat-0.9.21

Output images folder: output\_img

Files are in the part\_2 folder. Please ensure that the vlfeat-0.9.21 folder is placed in the same directory as the part\_2 folder.

**Show key points on the images.**



*Figure 8. 50 randomly select SIFT key points*

In Part 2, the public SIFT code used is vlfeat-0.9.21 which can be found in this website (<https://www.vlfeat.org/download.html>). After running SIFT to extract the key points, 50 points are randomly selected and displayed as shown in Figure 8.



## Part 3: Homography

### Instructions to run code

Program files: part\_3.m, computeH.m

Provided images: h1.jpg, h2.jpg

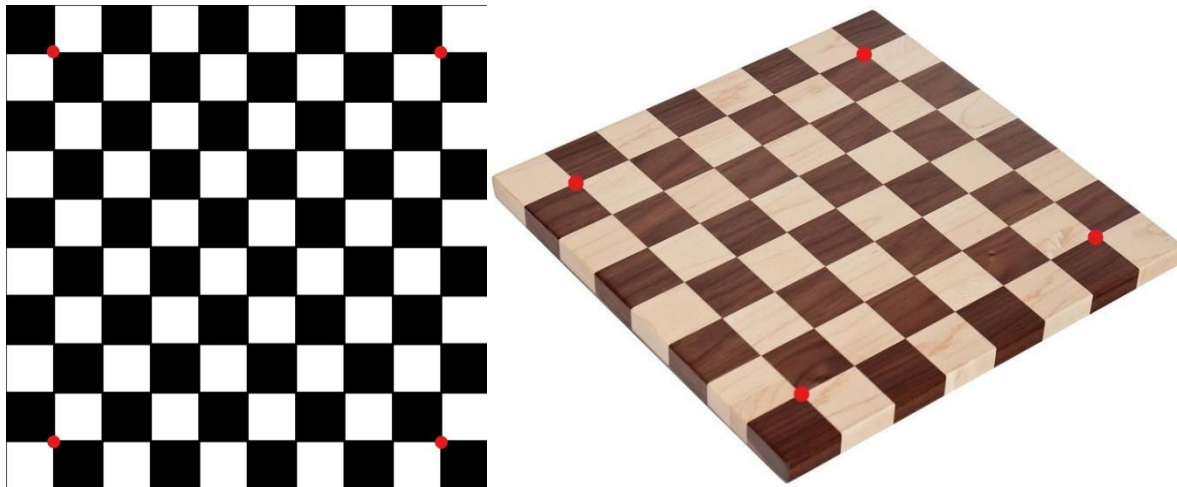
Output images folder: output\_img

Files are in the part\_3 folder. Please ensure that images are placed in the same directory as in part\_3 folder.

### Steps to follow:

1. Click on 4 homography points in h1.jpg
2. Click on 4 homography points in h2.jpg
3. Transformed images will be displayed and outputted to output\_img folder.

The 4 points clicked on h1 and h2 is should in Figure 9.



*Figure 9. 4 selected homography points on h1 (left), 4 selected homography points on h2 (right)*

## Outputs

### Show the homography matrix and result (H1 to H2)

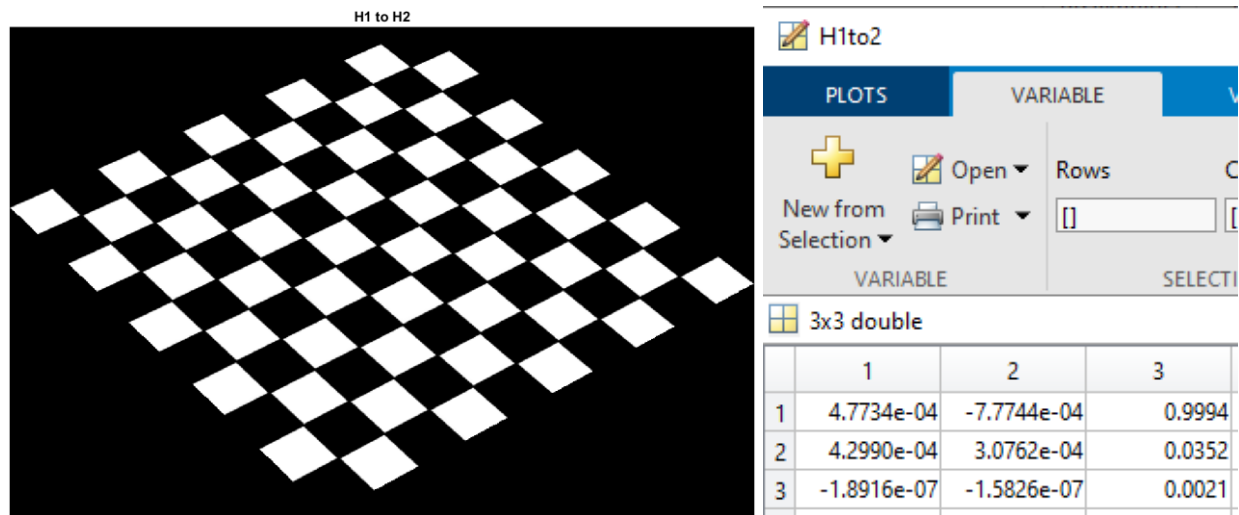


Figure 10. H1 transformed to H2 (left), homography matrix (right)

As shown in Figure 10, after the selection of 4 points in both h1 and h2, the homography matrix is calculated in the function `computeH.m`, which is then used to transform `h1.jpg` to `h2.jpg`. The homography matrix used to calculate this transformation is in the variable `H1to2`.

### Show the homography matrix and result (H2 to H1)

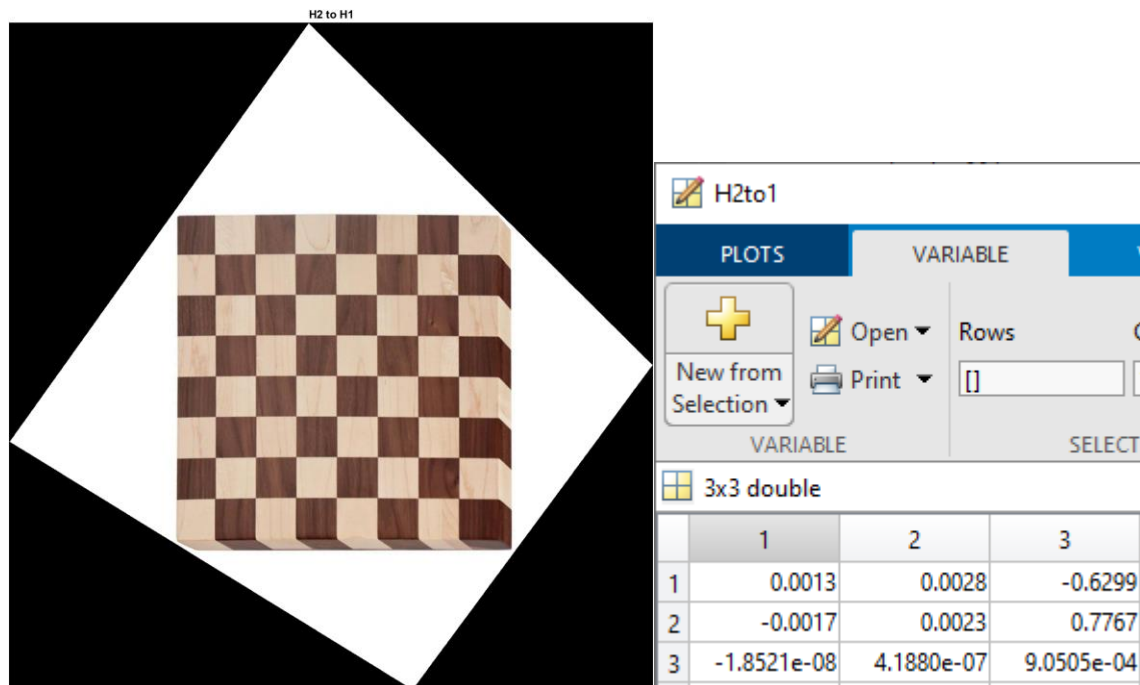


Figure 11. H2 transformed to H1 (left), homography matrix (right)

Similar process is used for this transformation. The results and homography matrix is shown in Figure 11. The homography matrix used in this transformation is stored in variable `H2to1`.

## Part 4: Manual Homography + Stitching

### Instructions to run code

Program files: part\_4.m, computeH.m, doStitch.m, image\_transform.m

Provided images: im01.jpg, im02.jpg

Output images folder: output\_img

Files are in the part\_4 folder. Please ensure that images are placed in the same directory as in part\_4 folder.

### Steps to follow:

1. Click 4 points on im01.jpg
2. Click 4 points on im02.jpg
3. Stitch images will be displayed and outputted to the output\_img folder

The 4 points clicked on im01 and im02 is should in Figure 12.



*Figure 12. 4 Selected homography points in im01 (left), 4 selected homography points selected in im02 (right)*

Show the result of the stitched images using computed homography matrix

Manual Homography + Stitching (img2 to img1)



Manual Homography + Stitching (img1 to img2)



Figure 13. Result of stitched image im02 to im01 (top), im01 to im02 (bottom)

**Explain why there is an effect of double edges in the overlapping regions between im01.jpg and im02.jpg**

When observing the result of im02 to im01 in Figure 13, it is possible to see the effect of double edges in the overlapping regions. Figure 14 shows a closer view of the double edges.



*Figure 14. Double edges effect in the overlapping regions*

There is such an effect as the 4 points selected in the first image will not exactly match the same points as the selected 4 points in the second image due to human error. Therefore, this leads to a small offset occurring, causing a double edge in the overlapping regions.



## Part 5: Homography + RANSAC

### Instructions to run code

Program files: part\_5.m, computeH.m, doMatch.m, doRANSAC, doStitch.m, image\_transform.m

Provided images: im01.jpg, im02.jpg

Public SIFT code used: vlfeat-0.9.21

Output images folder: output\_img

Files are in the part\_5 folder. Please ensure that images are placed in the same directory as part\_5 folder. Please ensure that the vlfeat-0.9.21 folder is placed in the same directory as the part\_5 folder.

### Show all the matches

The threshold parameter when doing the matching is set to 1.5. Descriptors ( $D1, D2$ ) are only matched if the squared Euclidean distance between the descriptors multiplied by the threshold is not greater than the distance of  $D1$  to all other descriptors.

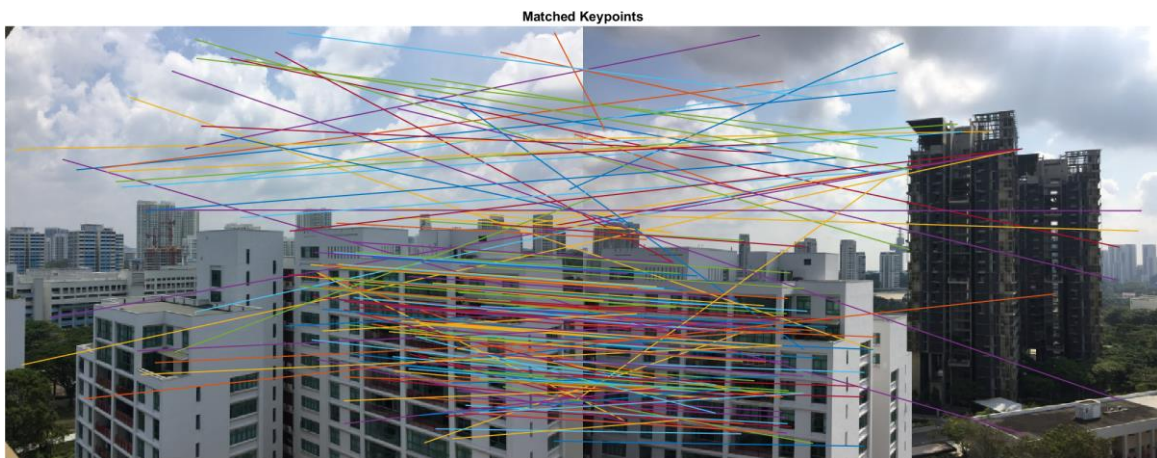


Figure 15. All matches found

### Show all inlier matches using RANSAC

Parameters used for RANSAC: iteration = 1000; epsilon = 1; n = 5;

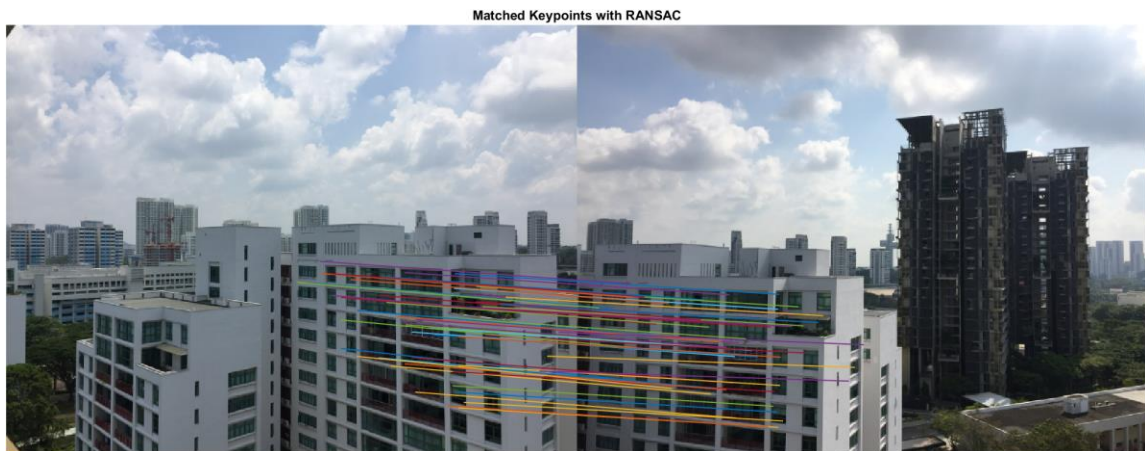


Figure 16. Showing all inlier matches found using RANSAC



## Stitch images together

Homography + RANSAC (Img1 to Img2)



Homography + RANSAC (Img2 to Img1)



Figure 17. Im1 stitched to im2 (top), Im2 stitched to im1 (bottom)

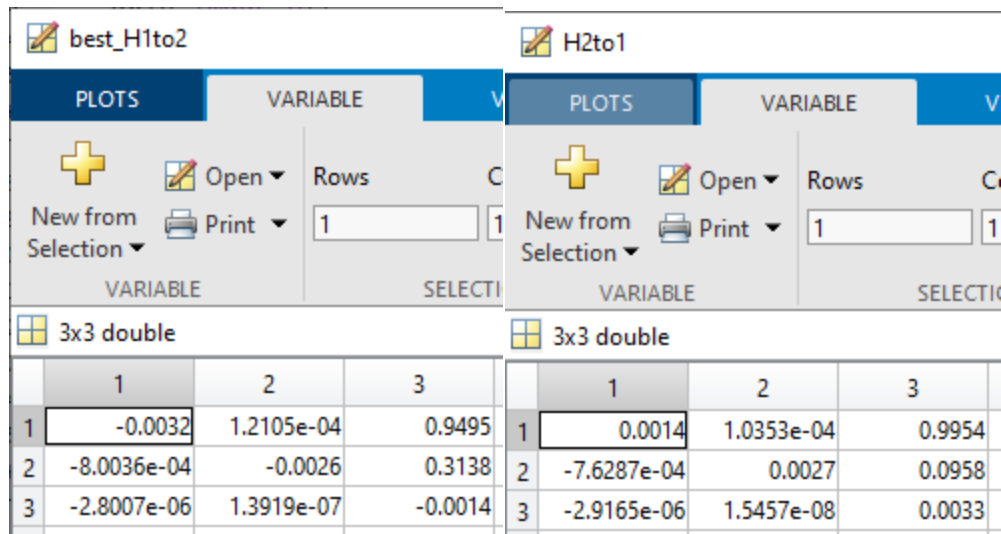


Figure 18. Homography matrix calculated from inliers for H1 to H2 (left), H2 to H1 (right)

Figure 18 shows the best homography matrices calculated from all in the inliers. The homography matrix used for transforming im01 to im02 is stored in the variable *best\_H1to2* while the matrix used for transforming im02 to im01 is stored in the variable *H2to1*.

## Part 6: Basic Panoramic Image

### Instructions to run code

Program files: part\_6.m, computeH.m, doMatch.m, doRANSAC, doStitch.m, image\_transform.m, generateStitchOrder.m

Provided Datasets:

- Dataset 1: im01.jpg, im02.jpg, im03.jpg, im04.jpg, im05.jpg (5 images)
- Dataset 2: 1.jpg, 2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg, 7.jpg, 8.jpg (8 images)

Public SIFT code used: vlfeat-0.9.21

Output images folder: output\_img

Files are in the part\_6 folder. Please ensure that images are placed in the same directory as part\_6 folder. Please ensure that the vlfeat-0.9.21 folder is placed in the same directory as the part\_6 folder.

### Outputs

Like previous parts all intermediate as well as final outputs will be located in the output\_img folder. Please change the parameters to fit whichever dataset for testing.

The current parameter for matching threshold is set to 1.5. RANSAC parameters will be set to: (1) iteration = 3000, (2) epsilon = 1, (3) n = 5.

When stitching panoramic images, it would result in a more balanced, less distorted image by starting from the center of the set of images. This is to prevent excessive distortion when stitching images from the side. As a result, a total of 8 images can be stitched together to form the panoramic image as shown in Figure 19.

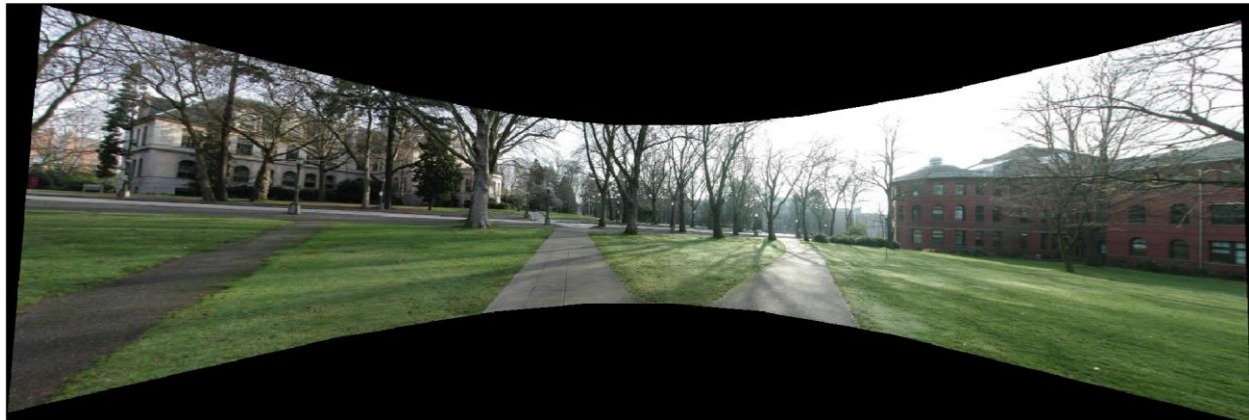


Figure 19. Panoramic Image of 8 stitched images

## Part 7: Advanced Panoramic Image

### Instructions to run code

Program files: part\_7.m, computeH.m, doMatch.m, doRANSAC, doStitch.m, image\_transform.m, generateStitchOrder.m

Provided Datasets:

- Dataset 2: 1.jpg, 2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg (6 images)

Public SIFT code used: vlfeat-0.9.21

Output images folder: output\_img

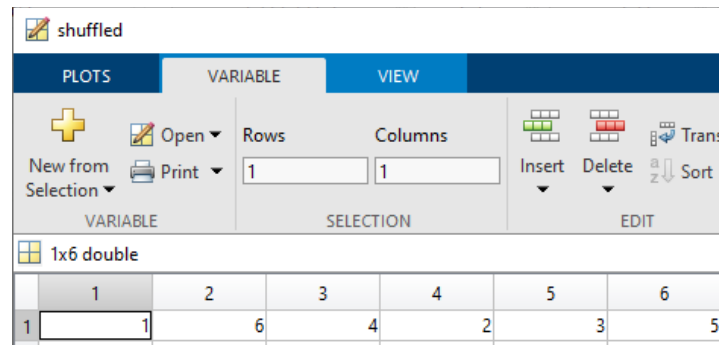
Files are in the part\_7 folder. Please ensure that images are placed in the same directory as part\_7 folder. Please ensure that the vlfeat-0.9.21 folder is placed in the same directory as the part\_7 folder.

For this part, unordered image stitching is implemented based on [0] and [1].

### Handling unordered images

#### Shuffling of images

To ensure that the loaded images are shuffled, the *randperm* function is used to scramble the images and load them into an image list. As shown in figure 20 below, this is one of the possible permutations when shuffling the index of the images to load.



The image shows a MATLAB window titled 'shuffled'. It has tabs for 'PLOTS', 'VARIABLE', and 'VIEW'. The 'VARIABLE' tab is active, showing a '1x6 double' array. The array contains the values [1, 6, 4, 2, 3, 5], which are the shuffled indices of the 6 images. The interface also shows options for 'Rows' and 'Columns' (both set to 1) and a 'SELECTION' area.

	1	2	3	4	5	6
1	1	6	4	2	3	5

Figure 20. Shuffled sequence of loading images

#### Generating stitching order

Based on [0], SIFT features would first be extracted for all the images. All descriptors generated will be added into a k-d tree, which would be used to find the nearest neighbors. In this case, the number of nearest neighbors select is four.

In the next part, we would need to find all matching images and according to [0],  $m$  number of images that have the greatest number of feature matches to the current image would be considered as a potential match. In this experiment,  $m=2$ .

Subsequently, a probabilistic model is used for image matching verification. According to [0] and [1], there are 2 different types of model to follow. Based on [0] and [1], the model follows:

$$n_i > \alpha + \beta n_f$$

Where  $n_i$  is the total number of inliers and  $n_f$  is the total number of features in the area of overlap. However, in [0] the values  $\alpha$  and  $\beta$  are 8.0 and 0.3 respectively. While in [1], the values  $\alpha$  and  $\beta$  are 5.9 and 0.22 respectively. In this experiment, parameters from [1] seems to produce a more accurate result.

Finally, the stitching order will be determined by the images that have the highest number of matches. While the image with the highest number of matches assumed to be to base image (center) image.

#### Output result

As shown in Figure 22, the output of the unordered stitching is similar to the results in part 6. However, since the final order of the stitching sequence differs from the stitching order generated from part 6, the total number stitched images are decreased to 6 instead of 8 as matlab would run out of memory.



*Figure 21. Output of 6 image unordered stitching*

## References

- [0] Brown, M., Lowe, D.G. Automatic Panoramic Image Stitching using Invariant Features. *Int J Comput Vision* **74**, 59–73 (2007). <https://doi.org/10.1007/s11263-006-0002-3>
- [1] Brown, Matthew & Lowe, David. (2003). Recognising Panoramas. Proceedings of the 9th International Conference on Computer Vision (ICCV2003). 2. 1218-1227.