EE569: Introduction to Digital Image Processing

ASSIGNMENT 3

Geometric Modification
Morphological Processing

Suchismita Sahu | 3rd March 2019 | USCID: 7688176370

PROBLEM 1: GEOMETRIC MODIFICATION

GEOMETRIC TRANSFORMATIONS

ABSTRACT AND MOTIVATION

This task is kind of a puzzle matching task that is a form of geometric image modification where multiple geometrical transformation operations like scaling, rotation and scaling are used to fit a set of images into another. These techniques are used extensively in computer graphics and other Image Processing software.

Here, we are trying to implement a hole-filling algorithm for part 1, followed by Spatial Warping and Lens Distortion Correction.

APPROACH AND PROCEDURE

Theoretical Approach:

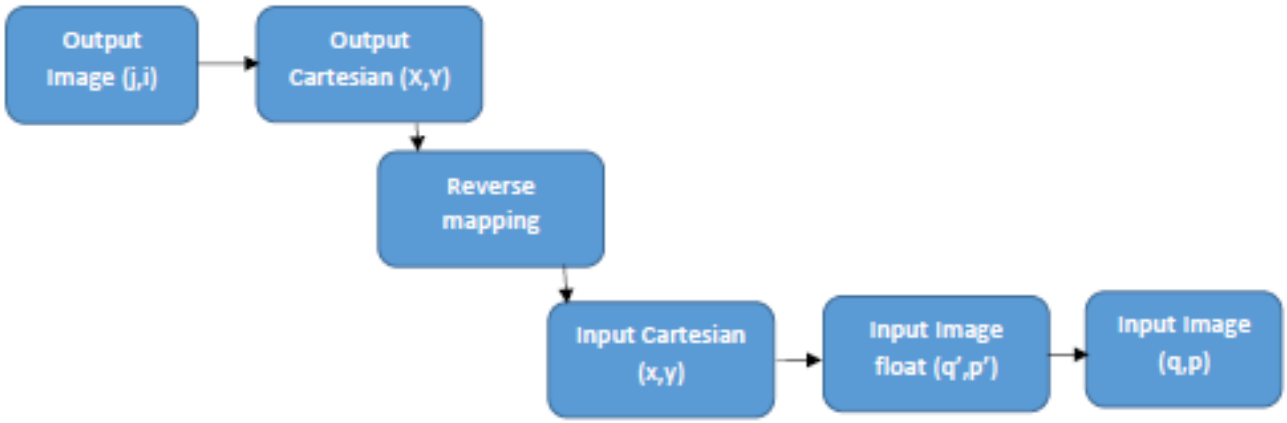


Figure 1 Core Steps involved in Geometric Transformation

1. **Conversion to Cartesian Coordinate System:**

We implement the morphological modifications in the cartesian coordinate system. The formula used to convert Image coordinates(i,j) to Cartesian Coordinates (X,Y):

$$X = i(centre) - i(image)$$

$$Y = j(image) - j(center)$$

Where, the I and j center values are values of the image center in the image spatial coordinate system.

For translating back to spatial coordinate system, we use:

$$I = i(center) + X$$

$$J = i(center) - Y$$

2. **Interpolation:**

   For some coordinate values the pixel values calculated are often not integers and are floating-point pixel locations. So, to estimate the pixel values for output image, we estimate the pixel value at that location by interpolation of neighboring pixels.

   Types:

   1. Nearest Neighbor
   2. Bilinear Interpolation

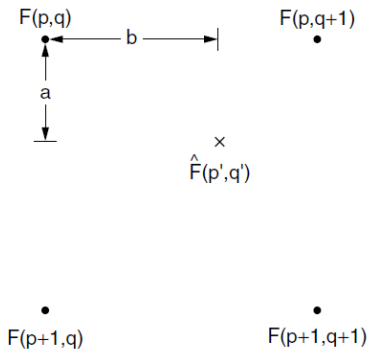


Figure 2 Bilinear Interpolation

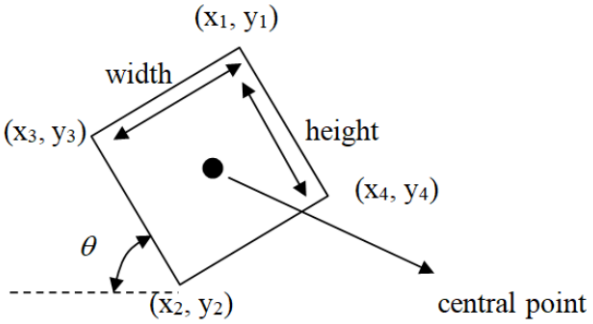
Equation for Bilinear Interpolation:

$$\hat{F}(p', q') = (1 - b)[(1 - a)F(p, q) + aF(p + 1, q)] + b[(1 - a)F(p, q + 1) + aF(p + 1, q + 1)]$$

3. Translation:

For translating an image, we must fine the corners of the image by scanning the image vertically and horizontally from front and back to detect the pixel value that is not 255. (Non- white)

Center is then found by taking the average of the corner distances. Also, from the corners, we find the height and width for scaling and the angle for rotation.



$$width = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}$$

$$height = \sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2}$$

$$\theta = \arctan\left(\frac{y_2 - y_3}{x_2 - x_3}\right)$$

$$center\ \ point = \left(\frac{x_3 + x_4}{2}, \frac{y_3 + y_4}{2}\right) or \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$

Figure 3 Calculating the Center and Angle of Rotation

4. Rotation:

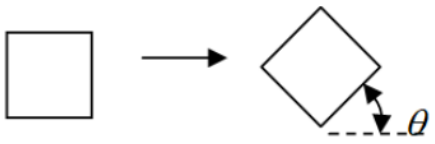


Figure 4 Rotation of an Image

For rotation of an image, we need to find the counterclockwise angle of rotation of the image using the arctan formula as given before. Equations followed are:

$$x_k = u_q cos\theta - v_p sin\theta$$

$$y_j = u_q sin\theta + v_p sin\theta$$

5. Scaling:

Scaling operation is performed using the following equations: sx and sy are scaling constants that are obtained by ratio of height and width of the gaps in actualimage and calculated height and width.

$$x_j = s_x u_p$$

$$y_k = s_y v_q$$

Going by predefined size of 160*160 can give artifacts.

Generalized vector space representation of scaling translation and rotation is given by the following matrices:

Translation:

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} u_q \\ v_p \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Scaling:

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix}$$

Rotation:

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix}$$

Algorithm followed for the geometric modifications:

1. First, read the lighthouse image lighthouse.raw1 image as a grayscale image byte by byte.

2. To find corners of the image, iterate through the image to locate the first non-white pixel up and down, right and left.
3. Convert the coordinate to cartesian system using the formulae x = i – 128 and y = 255 -j -128
4. Find the angle and center using the above corners. The center distance is the translation parameter and the angle is rotation parameter.
5. Translation and Rotation are performed to make image axes parallel to the output image.
6. Find the input coordinate (u,v) from (x,y) using reverse mapping and the values obtained (p,q) are then transformed into (i,j) using bilinear interpolation of neighboring 4 pixels.
7. Size is checked to be 160*160 or not and scaled to 162*162.
8. Now, output image (holed) is scanned to find the first block of 160*160 white pixels and the scaled lighthouse.raw1 image is filed into the white pixel hole to complete the image.
9. Same steps are repeated for other two images and the final filled image is obtained.

EXPERIMENTAL RESULTS

Figure 5 Lighthouse1.raw Input Image

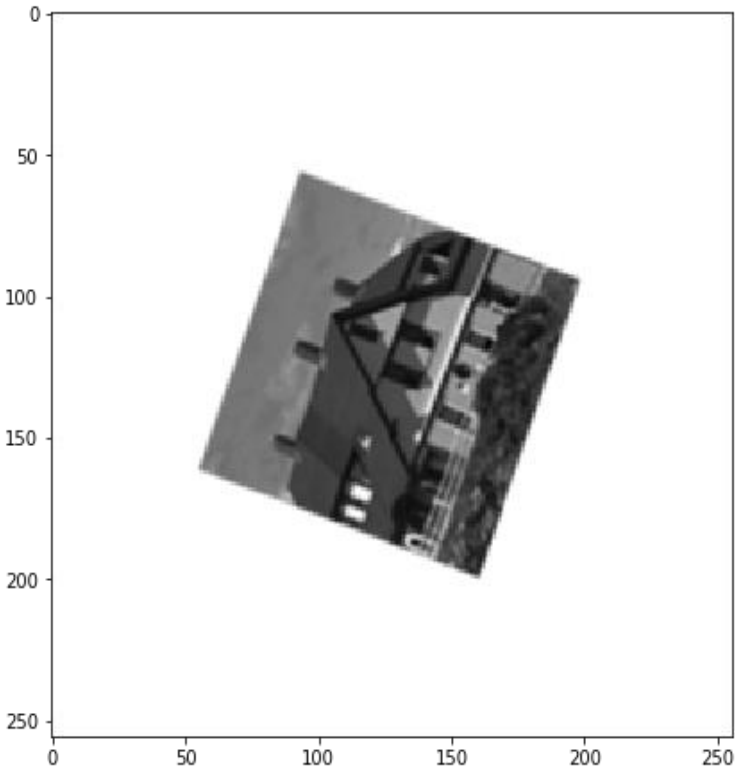


Figure 6 lighthouse1.raw after Translation

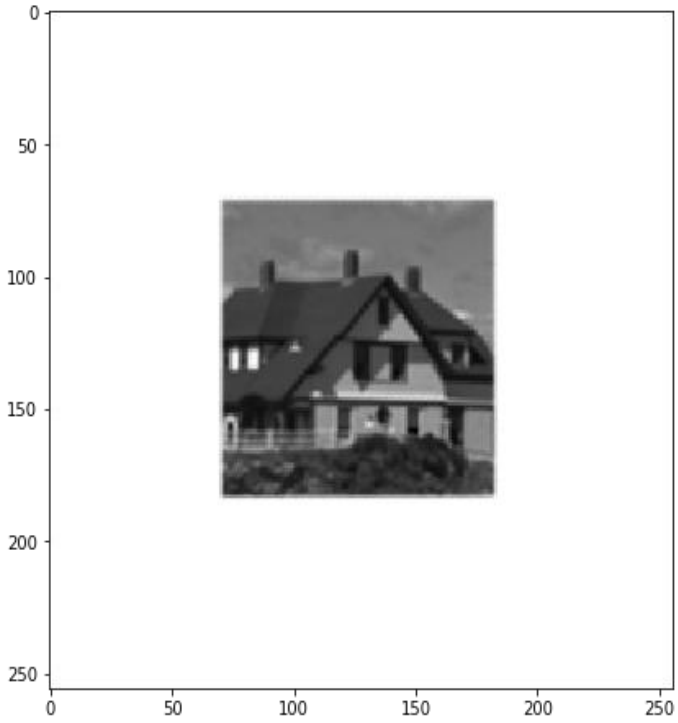


Figure 7 lighthouse1.raw after rotation

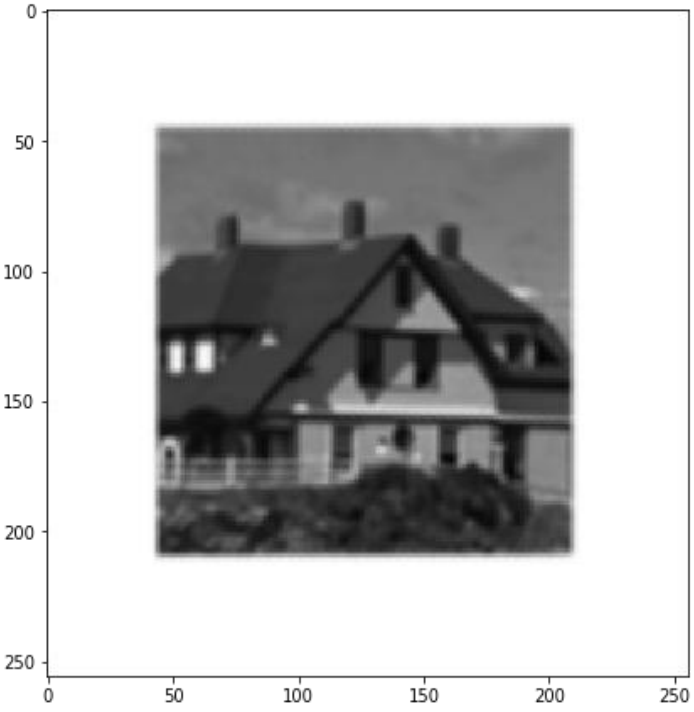


Figure 8 lighthouse1.raw after scaling

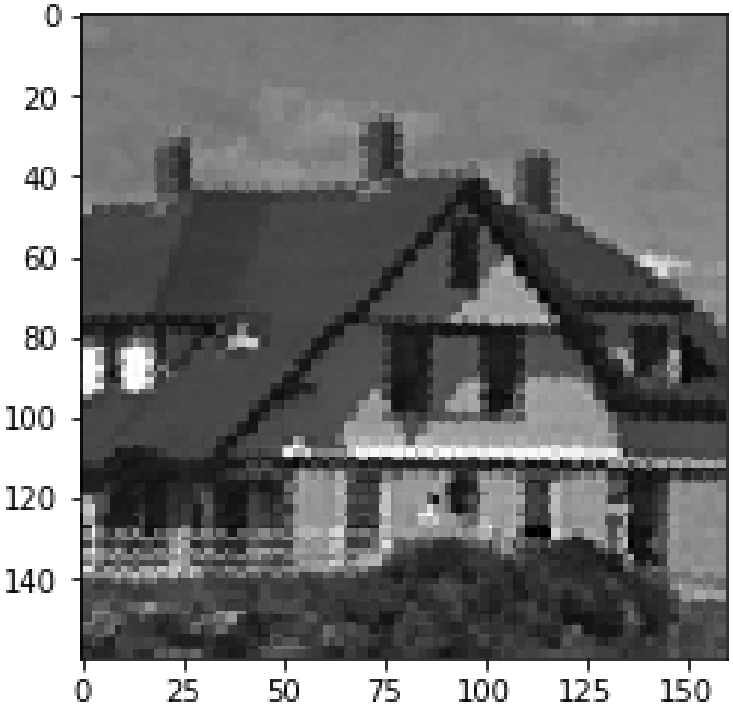


Figure 9 Lighthouse 1 sliced Image

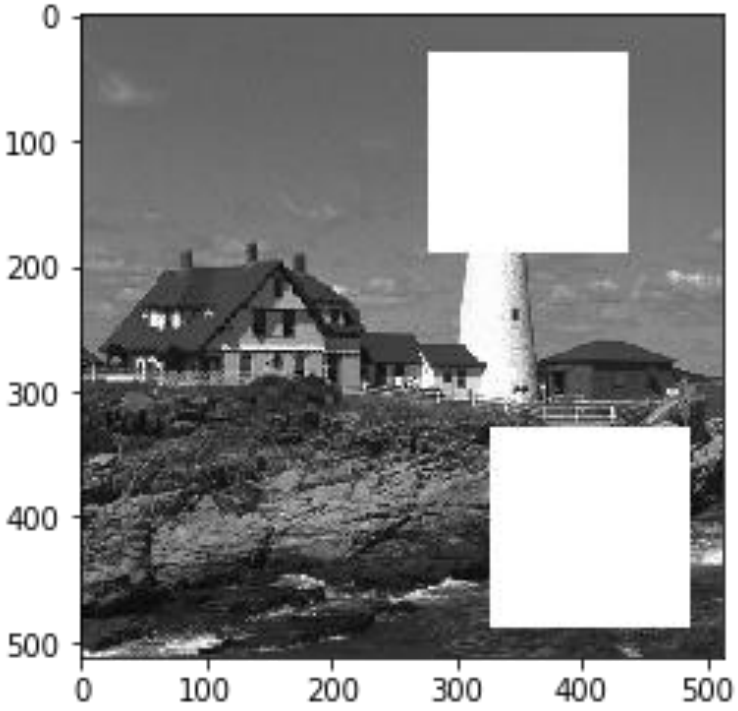


Figure 10 Lighthouse 1 House Image Fitted into the Lighthouse Image

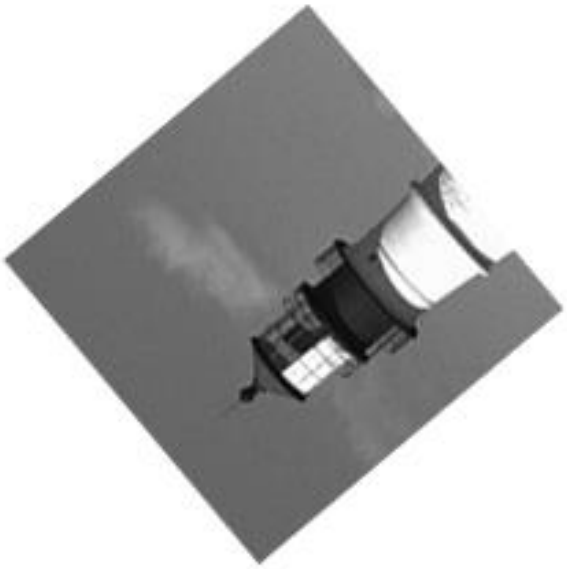


Figure 11 Lighthouse2.raw Input Image

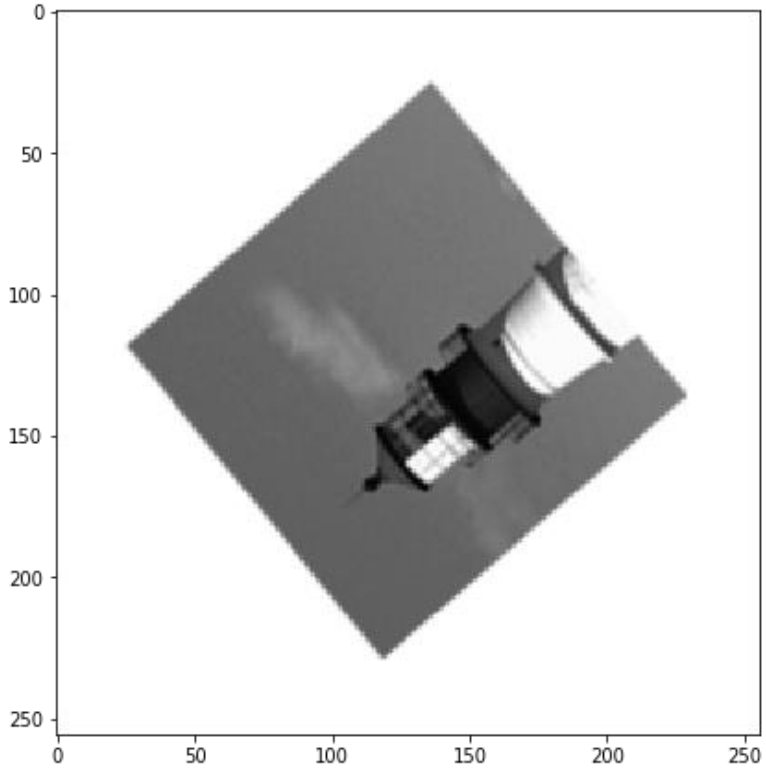


Figure 12 lighthouse2.raw after translation

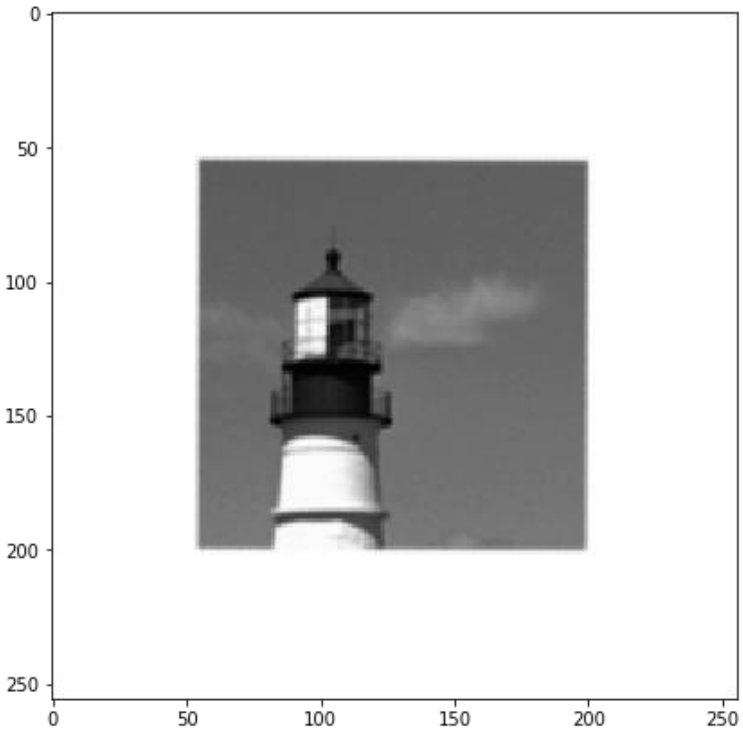


Figure 13 lighthouse2.raw after rotation

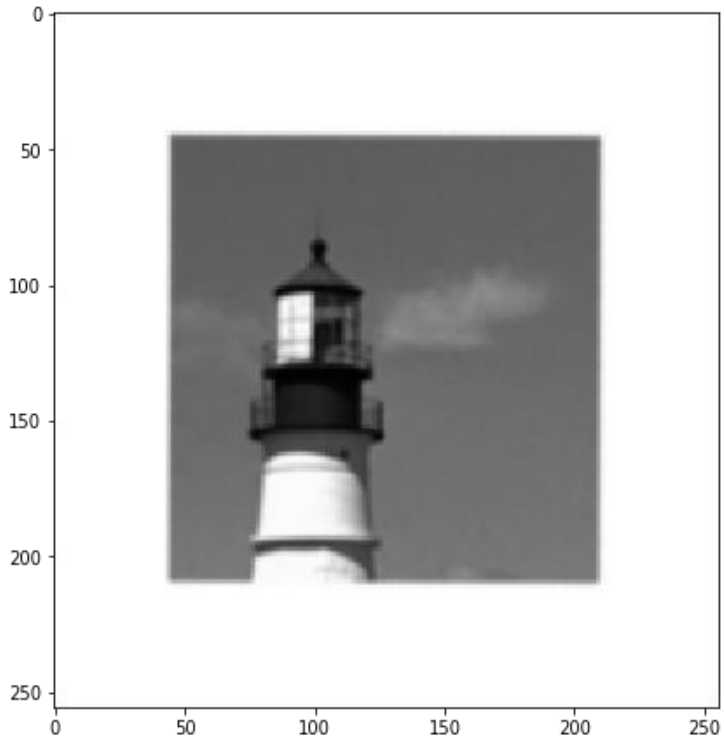


Figure 14 lighthouse2.raw after scaling

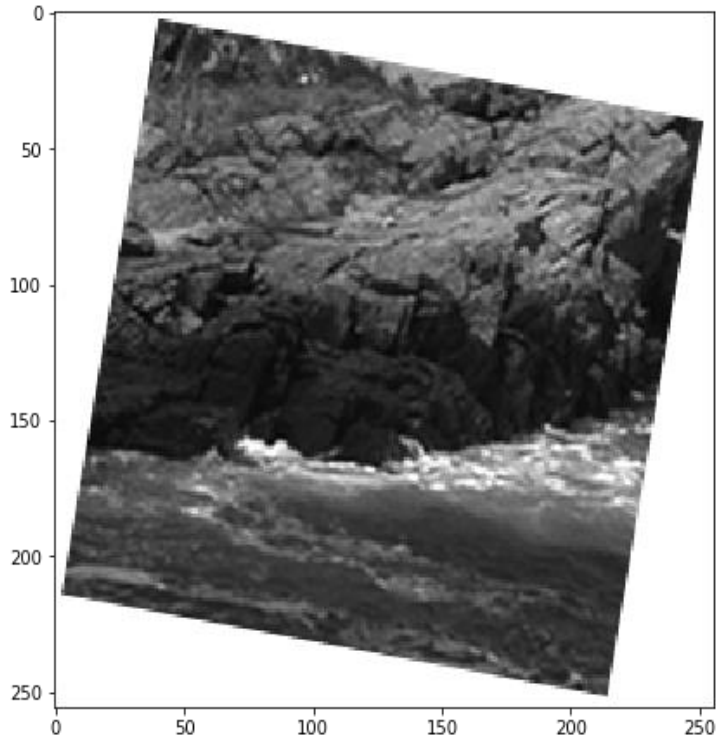


Figure 15 Lighthouse3.raw Input Image

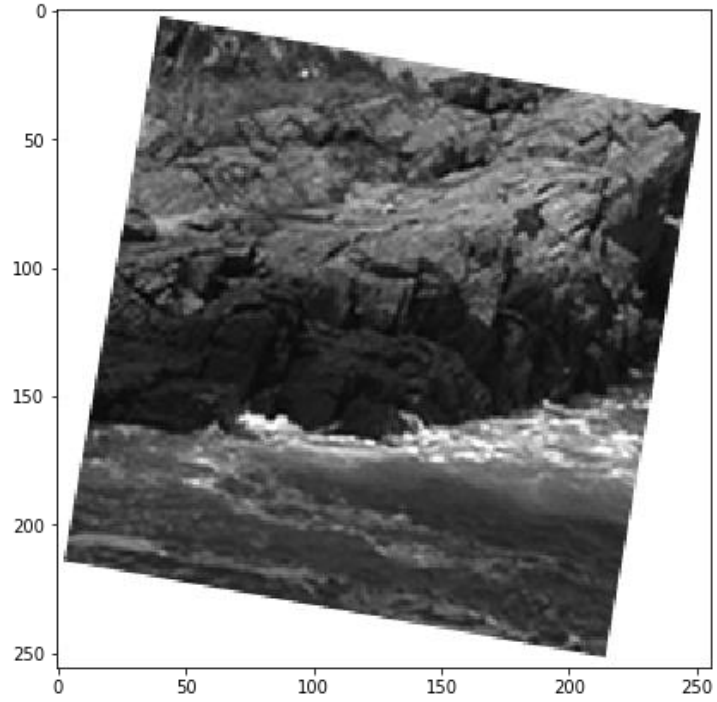


Figure 16 lighthouse3.raw after translation

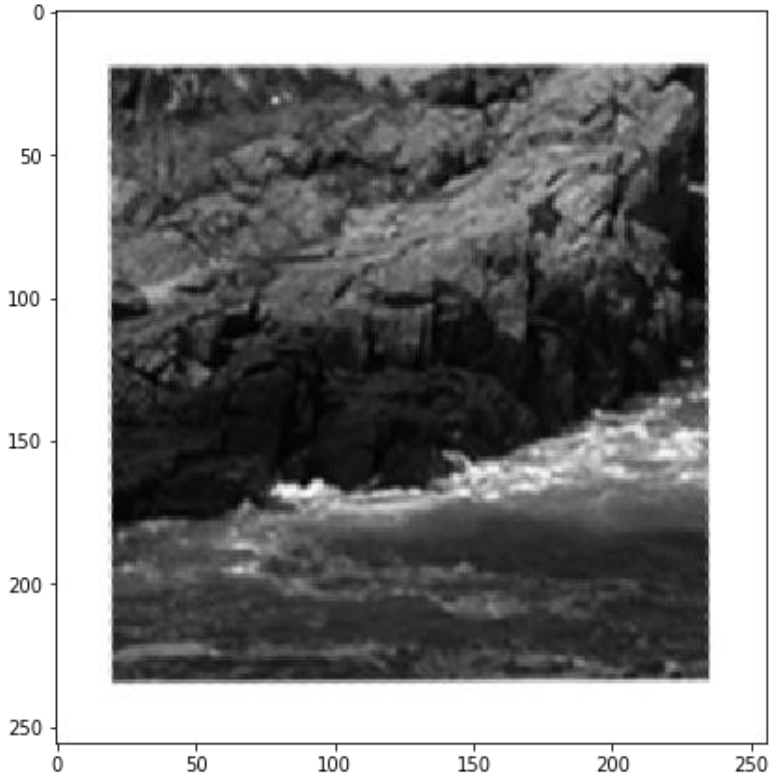


Figure 17 lighthouse3.raw after rotation

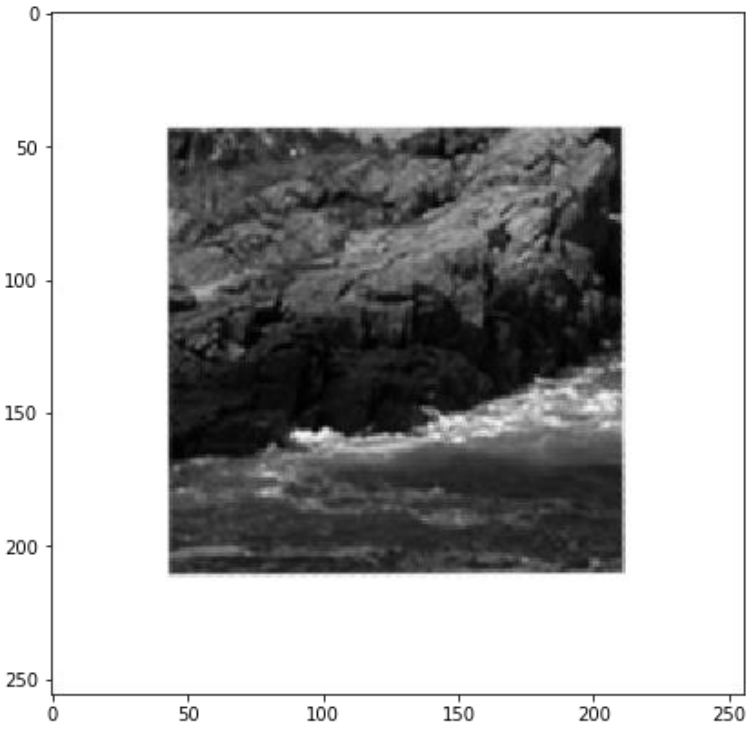
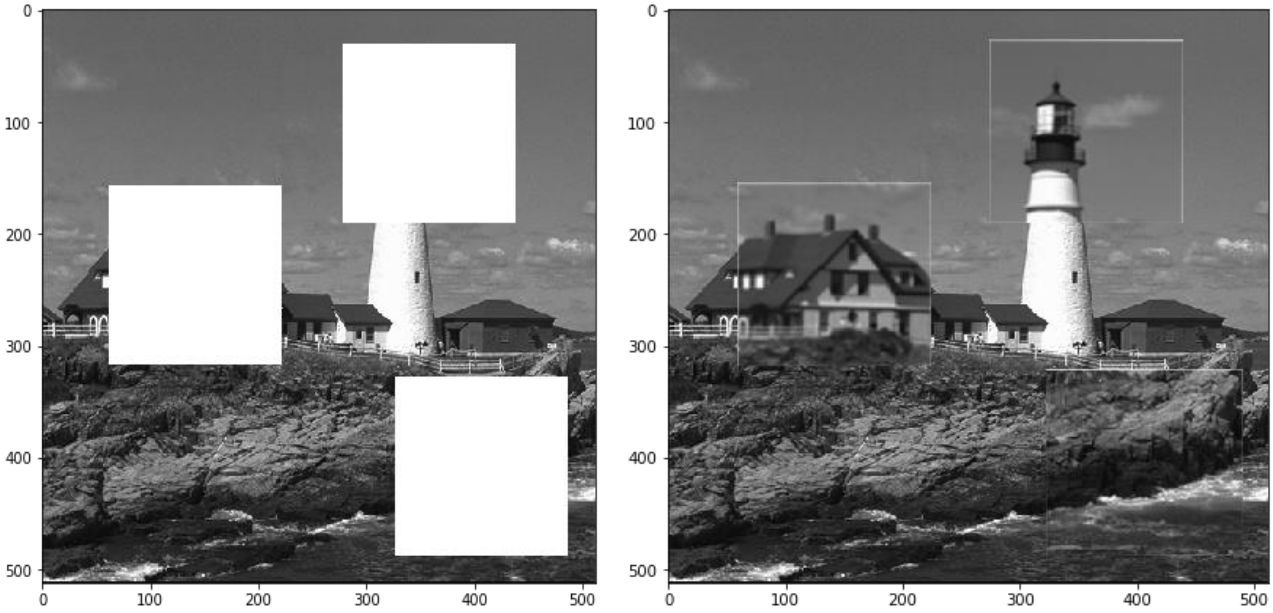


Figure 18 lighthouse3.raw after scaling

Final Fitting:



DISCUSSION

Observations for the procedure:

1. Had a tough time figuring the translation and rotation with trial and error.
2. The size of the holes in the image was 160*160 but on scaling the image exactly to 160*160 discontinuities and blurring of edges was observed while mapping it into the hole.
3. So, the scaling factor was increased to about 165*165 to allow the image to blend well when mapped into the hole.
4. Image Coordinate Corners:

	Lighthouse_1.raw	Lighthouse_2.raw	Lighthouse_3.raw
Top	(115, 80)	(125, 8)	(40, 3)
Bottom	(183, 223)	(108, 212)	(214, 251)
Left	(78, 182)	(15, 100)	(3, 209)

Right	(221, 119)	(218, 119)	(251, 46)

5. Cartesian coordinate Corners:

	Lighthouse_1.raw	Lighthouse_2.raw	Lighthouse_3.raw
Top	(-12, 47)	(-2, 119)	(-87, 124)
Bottom	(56, -96)	(-19, -85)	(87, -124)
Left	(-49, -55)	(-112, 27)	(-124, -82)
Right	(94, 8)	(91, 8)	(124, 81)

6. Translation Metrics :

	Lighthouse_1.raw	Lighthouse_2.raw	Lighthouse_3.raw
(t_x, t_y)	(22.5, -23.5)	(-10.5, 17.5)	(0.0, -0.5)
$\theta \; in \; radians$	7.5	2.27	6.1
(s_x, s_y)	(1.48, 1.46)	(1.14, 1.13)	(0.78, 0.78)

7. Hole Coordinates in input image:

	Hole - 1	Hole - 2	Hole -3
Top Left Corner	(31, 278)	(157, 62)	(328, 326)

8. The hole-filling helped explore the concepts of geometrical image transformations in cartesian coordinates and spatial image coordinates in depth.
9. Blurring: Scaling and bilinear interpolation cause loss of resolution in the image.
10. Other interpolation techniques can be used to reduce blurring
11. Translation is performed first so as not to exceed the image boundary on rotation and scaling. So need to bring image to centre first then scale or rotate.

SPATIAL WARPING

ABSTRACT AND MOTIVATION

Image warping involves common image processing operations such as rotation, translation and scaling of the image in the spatial domain to create a typical visual effect. This is used in computer graphics, image morphing techniques, and panorama stitching techniques extensively.

It is also referred to as rubber-sheet stretching. The second order polynomial address mapping is given as:

$$u = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2$$

$$v = b_0 + b_1 x + b_2 y + b_3 x^2 + b_4 xy + b_5 y^2$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix}$$

APPROACH AND PROCEDURE

To obtain the control points, image is divided into 4 triangles and each portion is mapped with the polynomial equation. Mapping is done for each triangle.

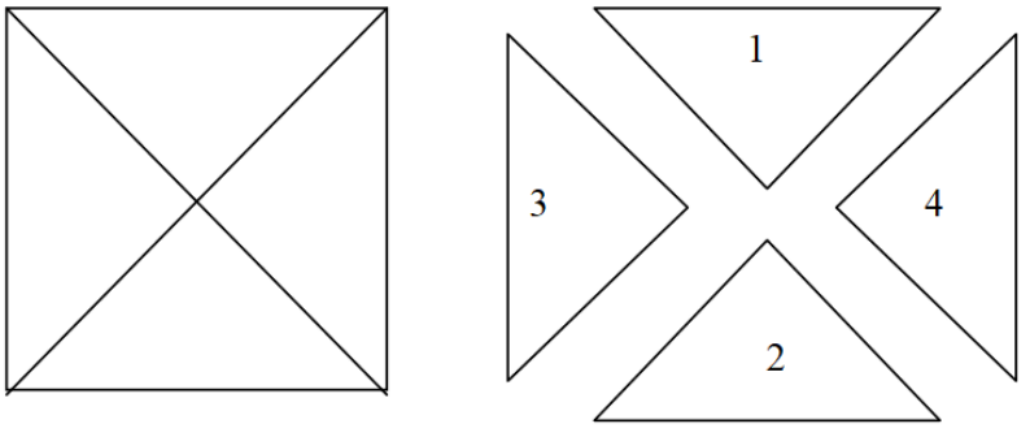


Figure 19 Division of image into Triangles

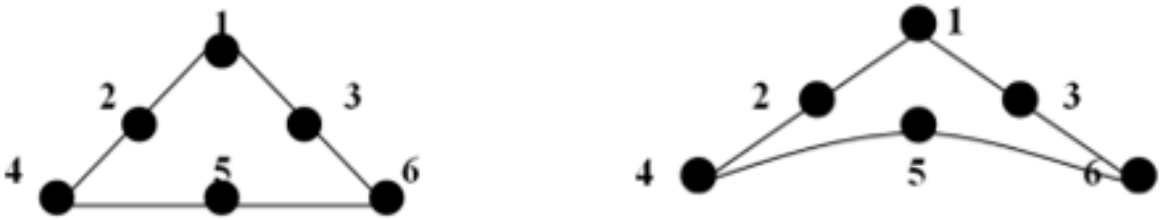


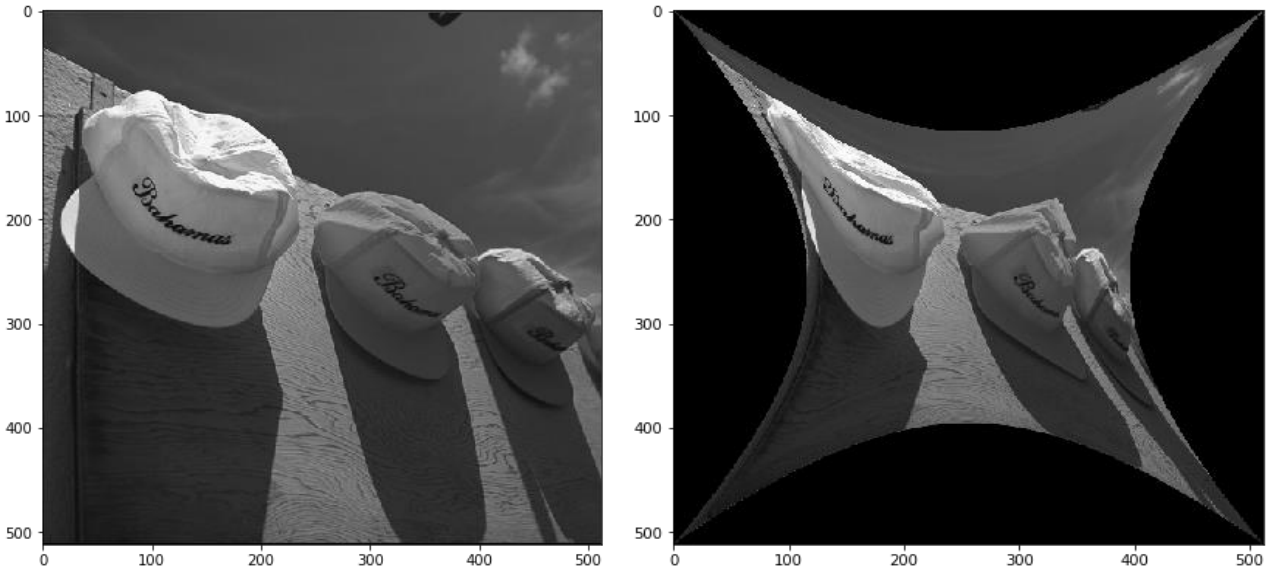
Figure 20 Control points in input and output

Algorithm Used for Spatial Warping:

1. Read input image as 3D array and obtain the height and width.
2. Divide the input image into four triangles as mentioned in the approach.
3. Find the height of the arc, six points for the polynomial equation for each of the triangles.
4. The coefficient values for each region are found using the equation.
5. Image is then converted into cartesian system and the warped image is obtained by mapping using the coefficients obtained from the polynomial equation as before.

6. The image is then converted back into image coordinate system and used bilinear interpolation to get the final output image.

EXPERIMENTAL RESULTS



DISCUSSION

1. The output image looks lightly distorted at the corners because of the fact that the forward mapping is not one to one. We also lose information when we transform longer line to a shorter line hence cannot recover all pixels lost in transformation.
2. We took 6 control points for the warping. Better warping could be obtained by taking more control points.

LENS DISTORTION CORRECTION

ABSTRACT AND MOTIVATION

This artifact occurs when an image is captured on the non-flat camera's focal plane. This distortion can be defined by the following set of equations:

$$x_d = \mathrm{x}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_d = \mathrm{y}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Where xd and yd are the distorted pixel locations. X and y are obtained as per:

$$x = \frac{u - u_c}{f_x}$$

$$y = \frac{v - v_c}{f_y},$$

Xd and yd can be calculated as per the Linear Regression Equations:

$$X_d = \boldsymbol{\alpha} \begin{bmatrix} X \\ Y \end{bmatrix} + \boldsymbol{\varepsilon}_1, Y_d = \boldsymbol{\beta} \begin{bmatrix} X \\ Y \end{bmatrix} + \boldsymbol{\varepsilon}_2$$

$$\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_N], \boldsymbol{\beta} = [\beta_1, \dots, \beta_N]$$

APPROACH AND PROCEDURE

Algorithm used :

1. Created an empty image for the output image.
2. Using the given constants k1, k2, k3, setup lens distortion.
3. Iterate through the input image, converting it into cartesian coordinate system and normalized by lens radius.
4. Corrected pixel values are then calculated and copied onto the output image.

EXPERIMENTAL RESULTS

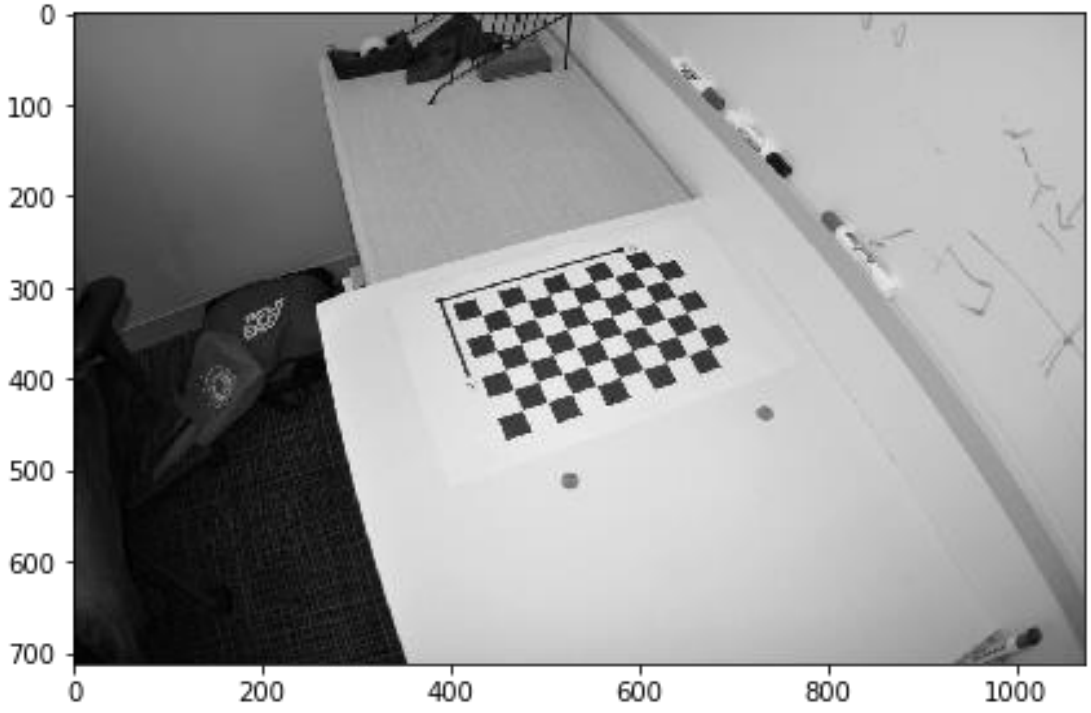


Figure 21 Input Image

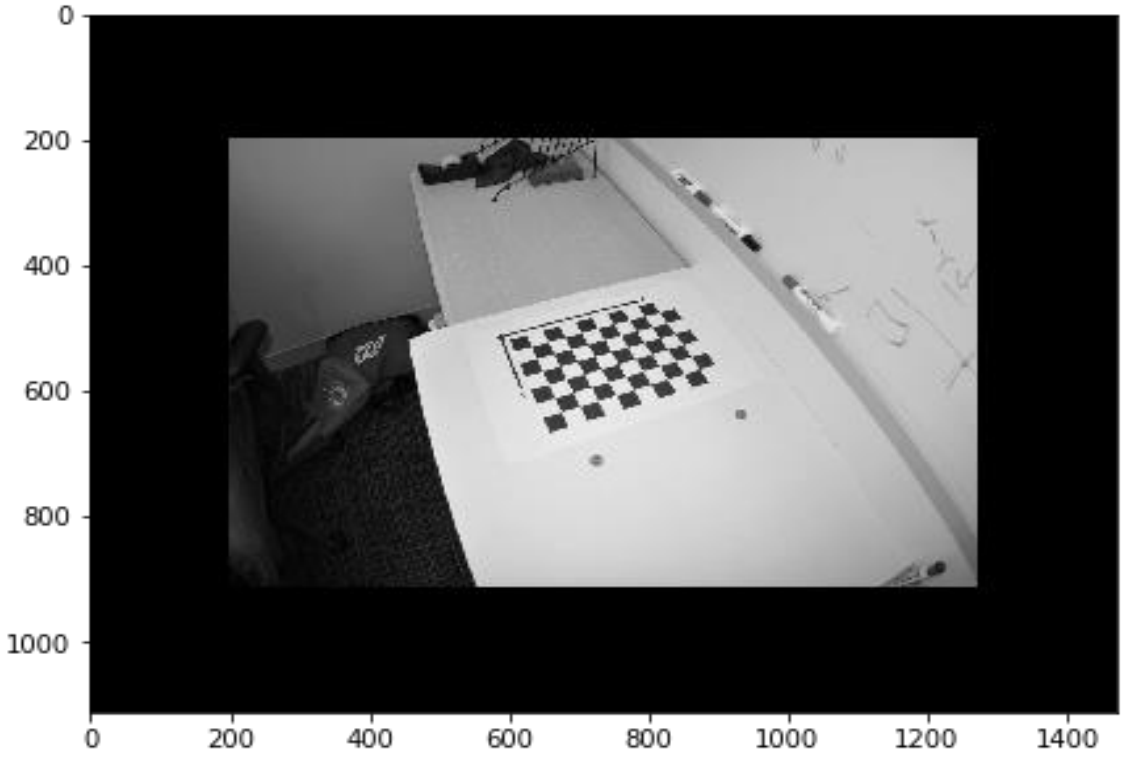


Figure 22 Input image with zero padding

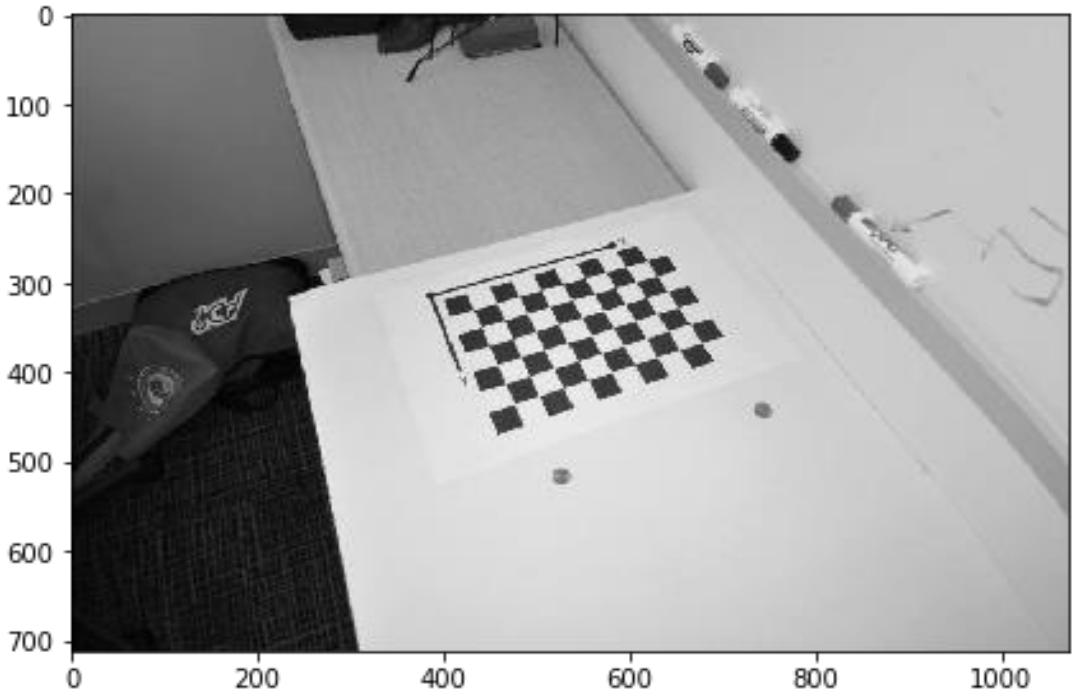


Figure 23 Output image without Padding

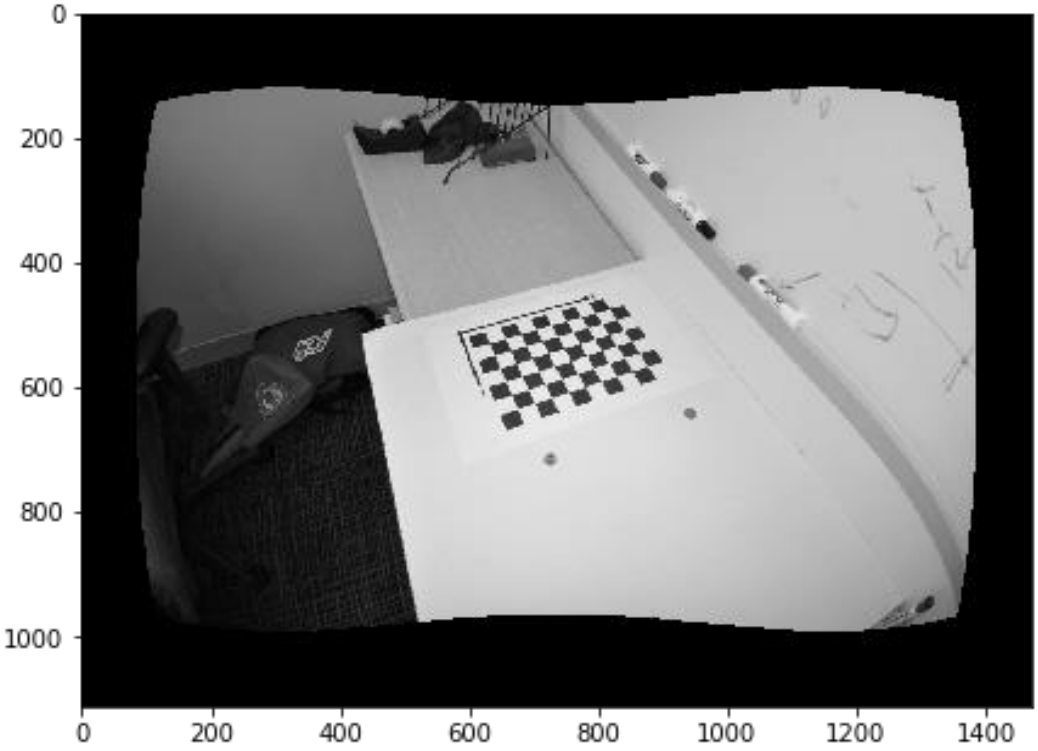


Figure 24 Output Image with Padding and Interpolation by Nearest Neighbor Method

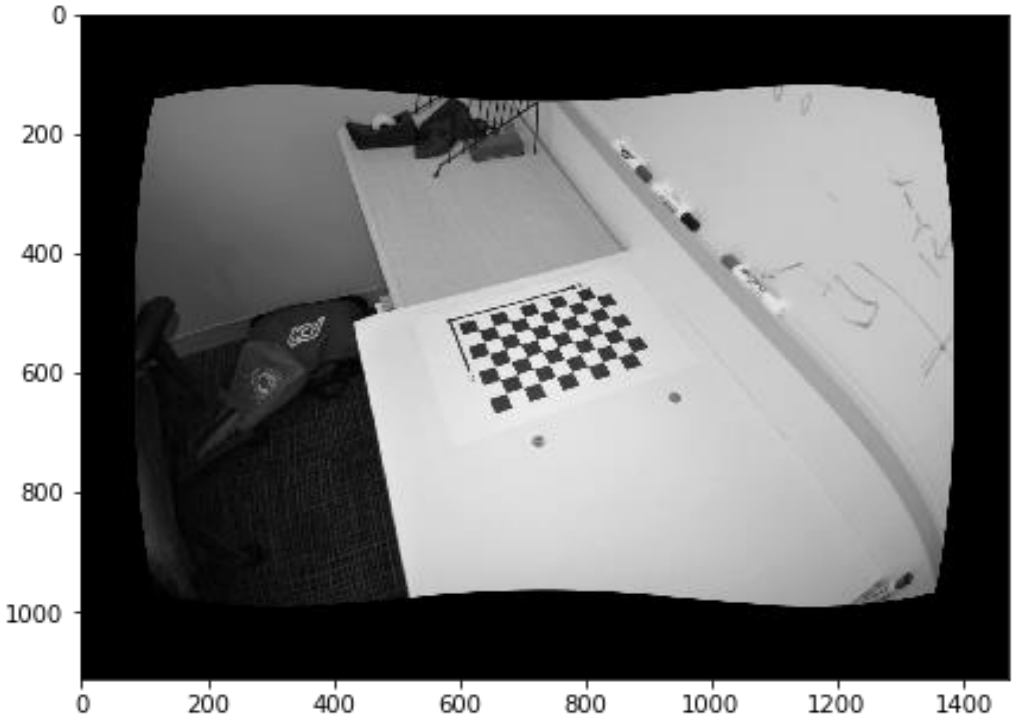


Figure 25 Output Image with Padding and Bilinear Interpolation Method

DISCUSSION

There is aliasing effects seen in the boundary regions since distortion correction is flattening the image into a sheet, it won't have a linear boundary.

There are artifacts present as the output is wavy in the image which can be reduced using ML methods.

PROBLEM 2: MORPHOLOGICAL PROCESSING

ABSTRACT AND MOTIVATION

Usually, Images contain numerous imperfections. When we threshold the Image, binary regions produced are distorted by texture and noise. We aim to remove these imperfections using Morphological Image Processing techniques that account for the form and structure of the image.

Morphological Image Processing is the collection of non-linear operations relating to the shape or morphology of subsets in image.

Hit/Miss Transformation:

Most morphological techniques probe the image with a small template or mask called a structuring element which is positioned at all possible locations in the image and then it is compared to the corresponding neighborhood pixels and processing methods test whether the element 'fits' within the neighborhood and/or the element 'hits' or intersects.

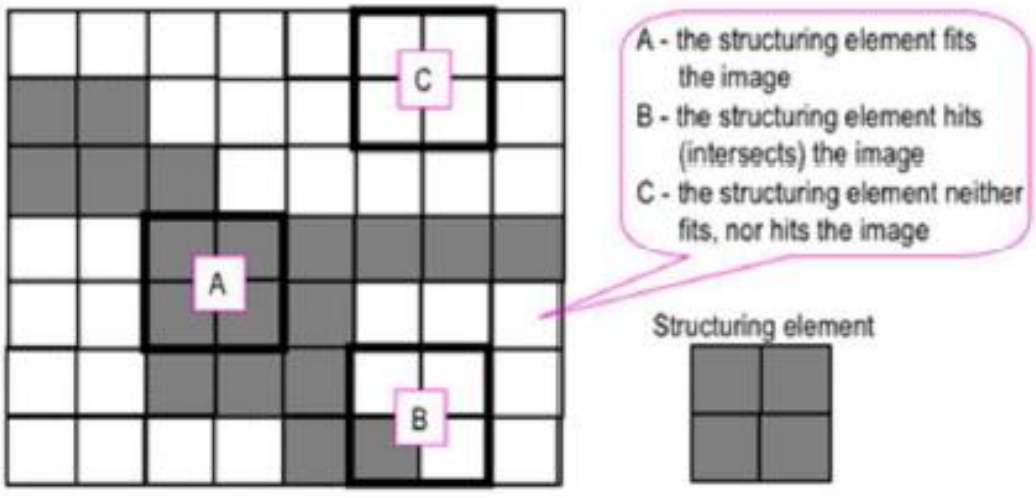


Figure 26 Probing Image using Structuring Mask

(Source: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)

When we do morphological processing, we create a binary image over which we convolve the structural mask.

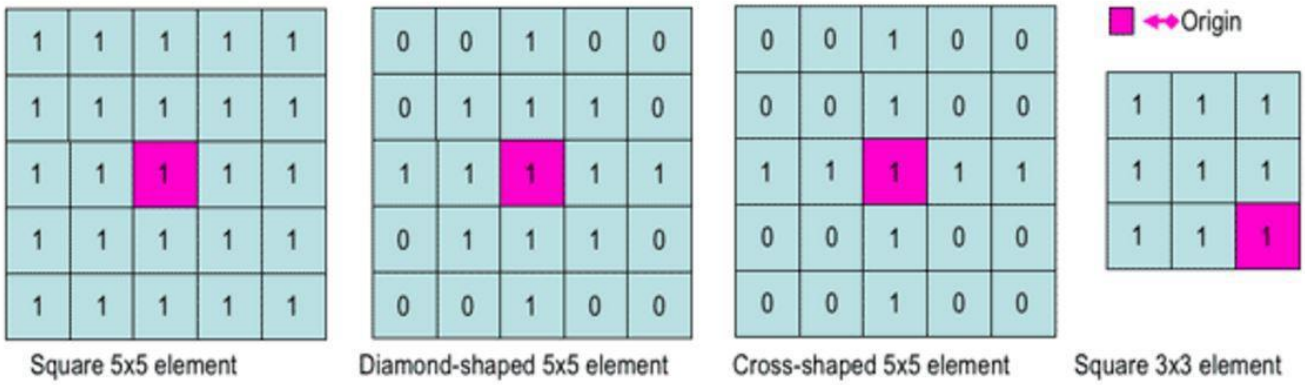


Figure 27 Types of Structuring Masks

(Source: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)

Defining the origin by center of the matrix, the mask is said to fit if for each of the pixels set to 1, the corresponding pixel in the image is also 1.

$$S_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad S_2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Figure 28 Hit and Fit of a Binarized Image with Structuring Elements S1 and S2

(Source: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)

Shrinking, Thinning and Skeletonizing are conditional erosion controlled by the following pattern Tables:

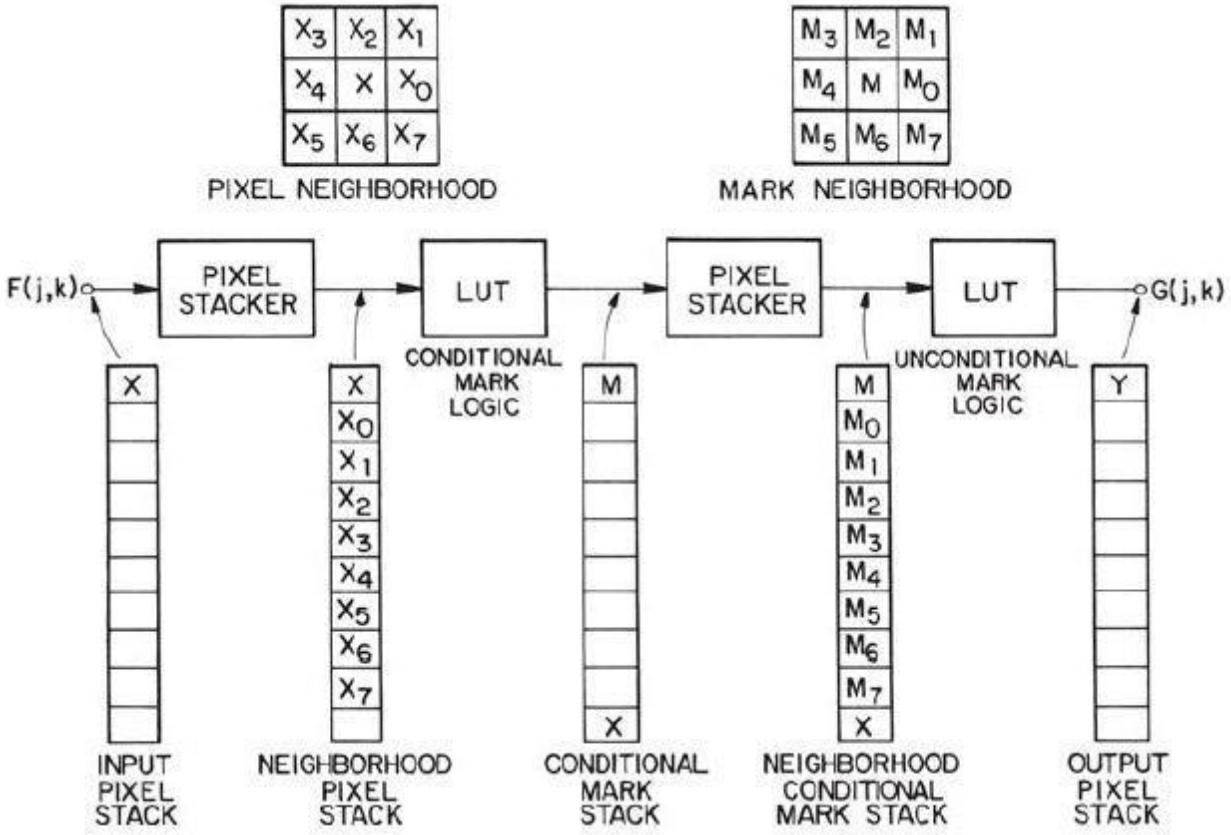


Figure 29 Look Up Table for Binary Conditional Mask Operations

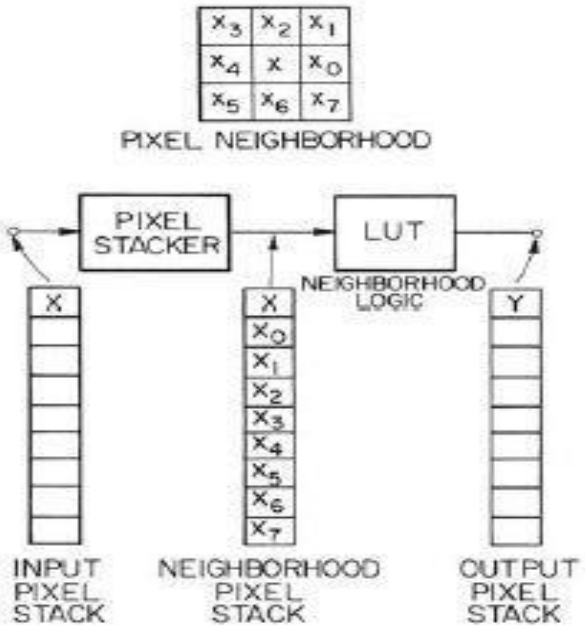


Figure 30 Look Up Table for Binary Un-Conditional Mask Operations

Image Close and Open Operations:

Close is dilation operation followed by erosion. Closing eliminates holes in object and short gaps and it can increase the spatial extent of an object.

Open is erosion followed by dilation that smooths contours and breaks narrow strokes.

Image Dilation and erosion:

Image dilation can be defined by the following equation:

$$G(j,k) = MAX(F(j,k), F(j,k+1), F(j-1,k+1), \dots, F(j+1,k+1))$$

Image erosion can be defined by the following equation:

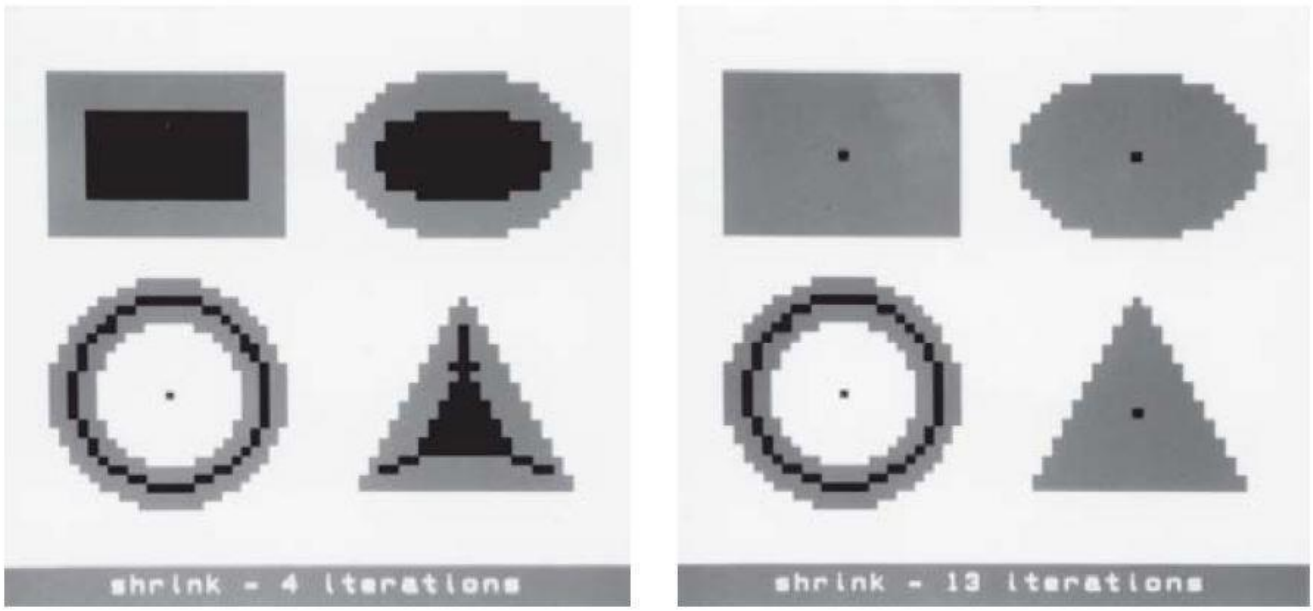
$$G(j,k) = MIN(F(j,k), F(j,k+1), F(j-1,k+1), \dots, F(j+1,k+1))$$

SHRINKING

Shrinking is the method of reducing the foreground (white) pixels from the image and letting the image be dominated by black pixels till the white or foreground pixels in a localized area get reduced to a single white dot.

We use a lot of Conditional and Unconditional Masks to achieve Shrinking. Thus, we intend to erase black pixels from an image such that an object without holes erodes to a single pixel at or near it's center of mass. Similarly, an object with holes erodes to a connected ring-like structure lying midway between each hole and it's nearest boundary on the outside.

Example:



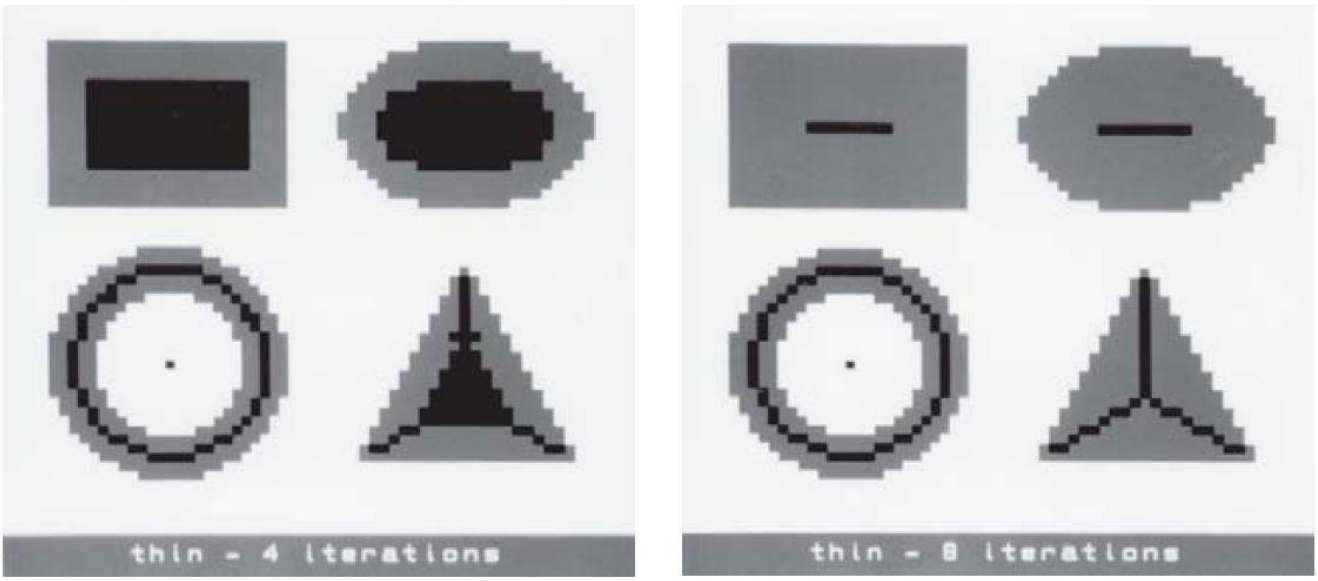
(*a*) Four iterations (*b*) Thirteen iterations

Figure 31 Shrinking of Binary Image

THINNING

Thinning is the method of reducing the foreground (white) pixels from the image and letting the image be dominated by black pixels till the white or foreground pixels in a localized area get reduced to a single line.

We use a lot of Conditional and Unconditional Masks to achieveThinning. Thus, we intend to erase black pixels from an image such that an object without holes erodes to a minimally connected line that is located equidistant from the nearest outer boundaries. Similarly, an object with holes erodes to a minimally connected ring-like structure lying midway between each hole and it's nearest boundary on the outside.



(*a*) Four iterations (*b*) Eight iterations

Figure 32 Thinning of a Binary Image

SKELETONIZING

Skeletonizing is the method of reducing the foreground (white) pixels from the image and letting the image be dominated by black pixels till the white or foreground pixels in a localized area get reduced to a single rooted line.

We use a lot of Conditional and Unconditional Masks to achieve Skeletonizing. It is a type of thinning. Here, we intend to erase black pixels from an image such that an object without holes erodes to a minimally connected rooted stroke line that is located equidistant from the nearest outer boundaries. Similarly, an object with holes erodes to a minimally connected ring-like structure lying midway between each hole and it's nearest boundary on the outside.

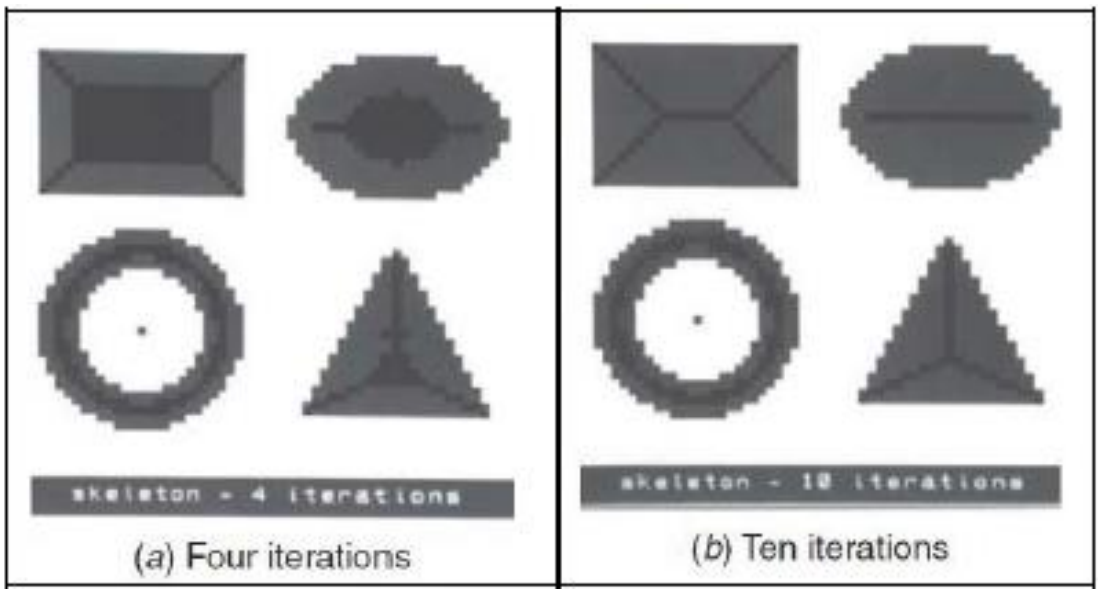


Figure 33 Skeletonization of Binary Image

APPROACH AND PROCEDURE

SHRINKING

The idea is to apply the Conditional and Unconditional masks based on predefined masks given that can be logically expressed as:

$$G(j,k) = X \cap [\bar{M} \cup P(M, M_0, \dots, M_7)]$$

Where,

P = Erasure inhibiting logical variable

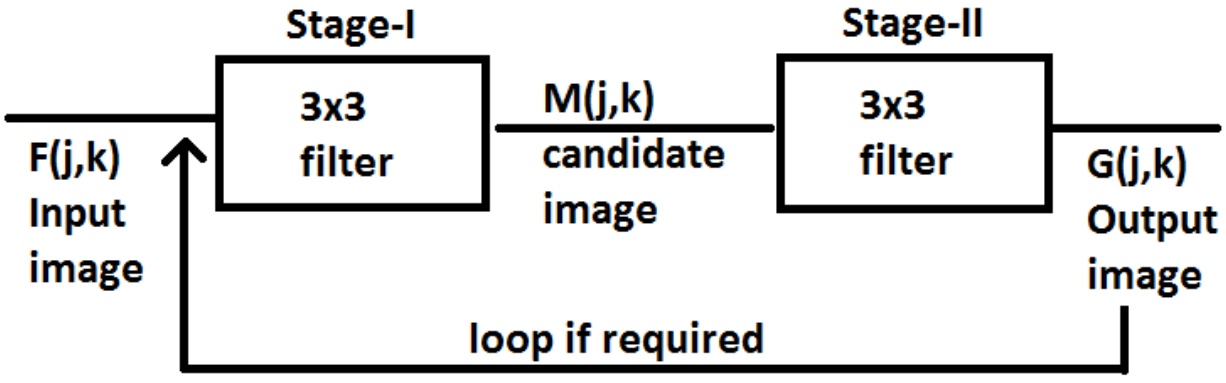


Figure 34 Flow Chart of Procedure for Shrinking of a Binary Image

Algorithm used for Shrinking:

1. Make the mask table for the conditional and unconditional masks for shrinking.
2. Reduce one layer of pixels to black if there is a hit according to the LUT else, set it to black.
3. Run the masks through the entire image.
4. If there's a hit, copy the pixels else set it to black.
5. The final array gives the 1st level shrinked image.
6. Run this for iterations till the object shrinks to a single pixel.

THINNING

The central idea is to apply the conditional and unconditional masks as given before.

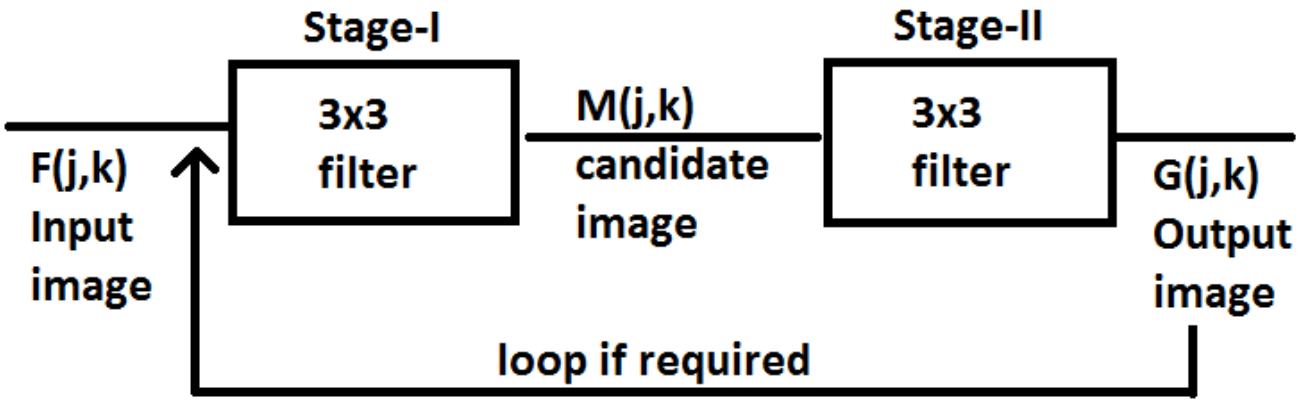


Figure 35 FLowchart for Thinning of Binary Image

1. Make the mask table for the conditional and unconditional masks for Thinning.
2. Reduce one layer of pixels to black if there is a hit according to the LUT else, set it to black.
3. Run the masks through the entire image.
4. If there's a hit, copy the pixels else set it to black.
5. The final array gives the 1st level Thinned image.
6. Run this for iterations till the object shrinks to a single pixel.

SKELETONIZING

The central idea is to apply the conditional and unconditional masks as given before and following it up with bridging expressions.

$$G(j,k) = X \cup [P_1 \cup P_2 \cup \cdots \cup P_6]$$

where

$$P_1 = \bar{X}_2 \cap \bar{X}_6 \cap [X_3 \cup X_4 \cup X_5] \cap [X_0 \cup X_1 \cup X_7] \cap \bar{P}_Q$$

$$P_2 = \bar{X}_0 \cap \bar{X}_4 \cap [X_1 \cup X_2 \cup X_3] \cap [X_5 \cup X_6 \cup X_7] \cap \bar{P}_Q$$

$$P_3 = \bar{X}_0 \cap \bar{X}_6 \cap X_7 \cap [X_2 \cup X_3 \cup X_4]$$

$$P_4 = \bar{X}_0 \cap \bar{X}_2 \cap X_1 \cap [X_4 \cup X_5 \cup X_6]$$

$$P_5 = \bar{X}_2 \cap \bar{X}_4 \cap X_3 \cap [X_0 \cup X_6 \cup X_7]$$

$$P_6 = \bar{X}_4 \cap \bar{X}_6 \cap X_5 \cap [X_0 \cup X_1 \cup X_2]$$

and

$$P_Q = L_1 \cup L_2 \cup L_3 \cup L_4$$

$$L_1 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap \bar{X}_7$$

$$L_2 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap \bar{X}_7$$

$$L_3 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap X_7$$

$$L_4 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap X_7$$

Algorithm used:

1. Make the mask table for the conditional and unconditional masks for skeletonization.
2. Reduce one layer of pixels to black if there is a hit according to the LUT else, do nothing.
3. Run the masks through the entire image.
4. If there's a hit, copy the pixels else set it to black.
5. Bridge the output array with the input image using the bridging operations.
6. The final array gives the 1st level skeletonized image
7. Run this for iterations till the object shrinks to a root line
8. Stop when line starts breaking.

EXPERIMENTAL RESULTS

Shrinking

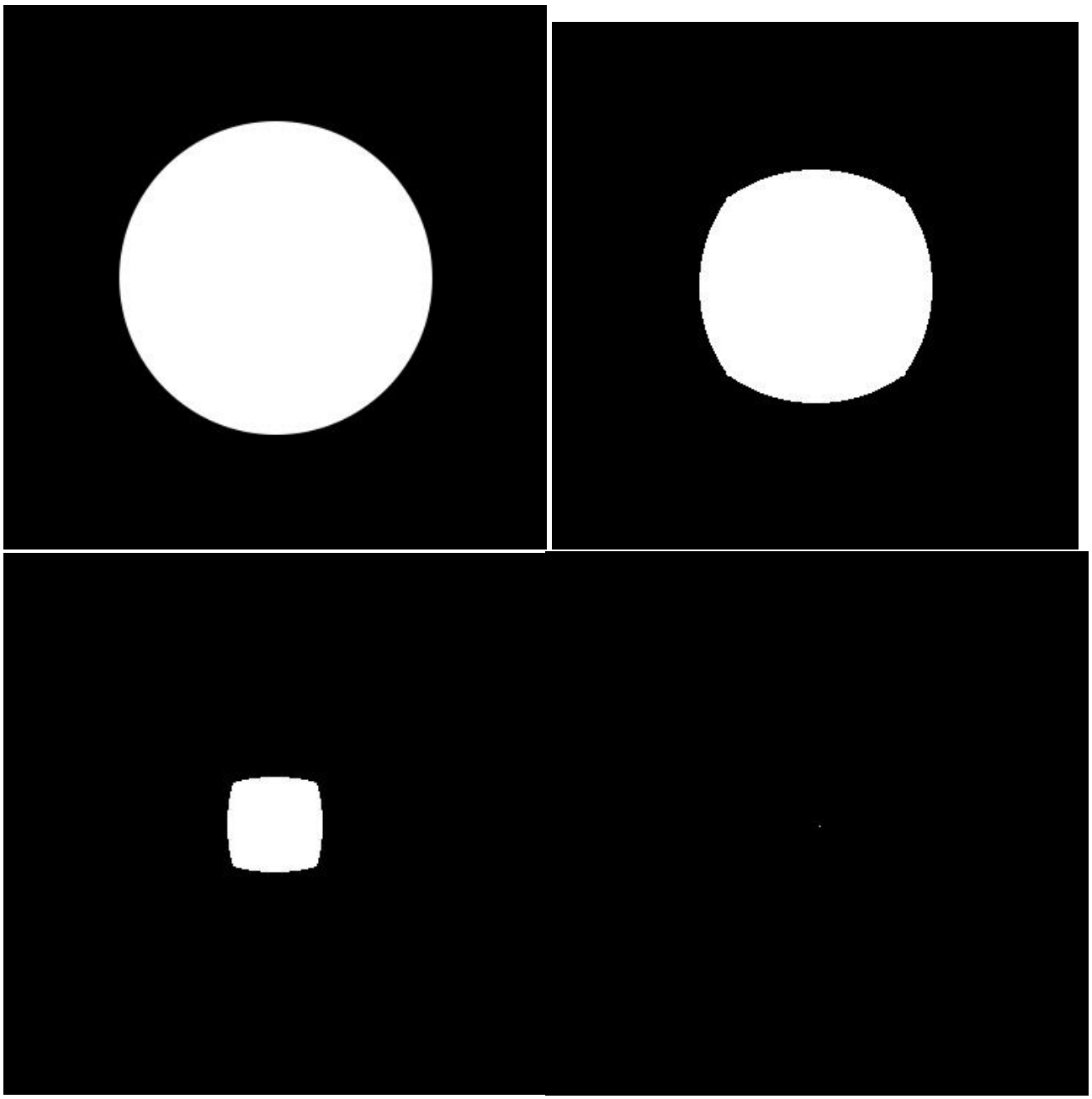


Fig: Shrinking output of Pattern 1 25, 50 iterations and final otput

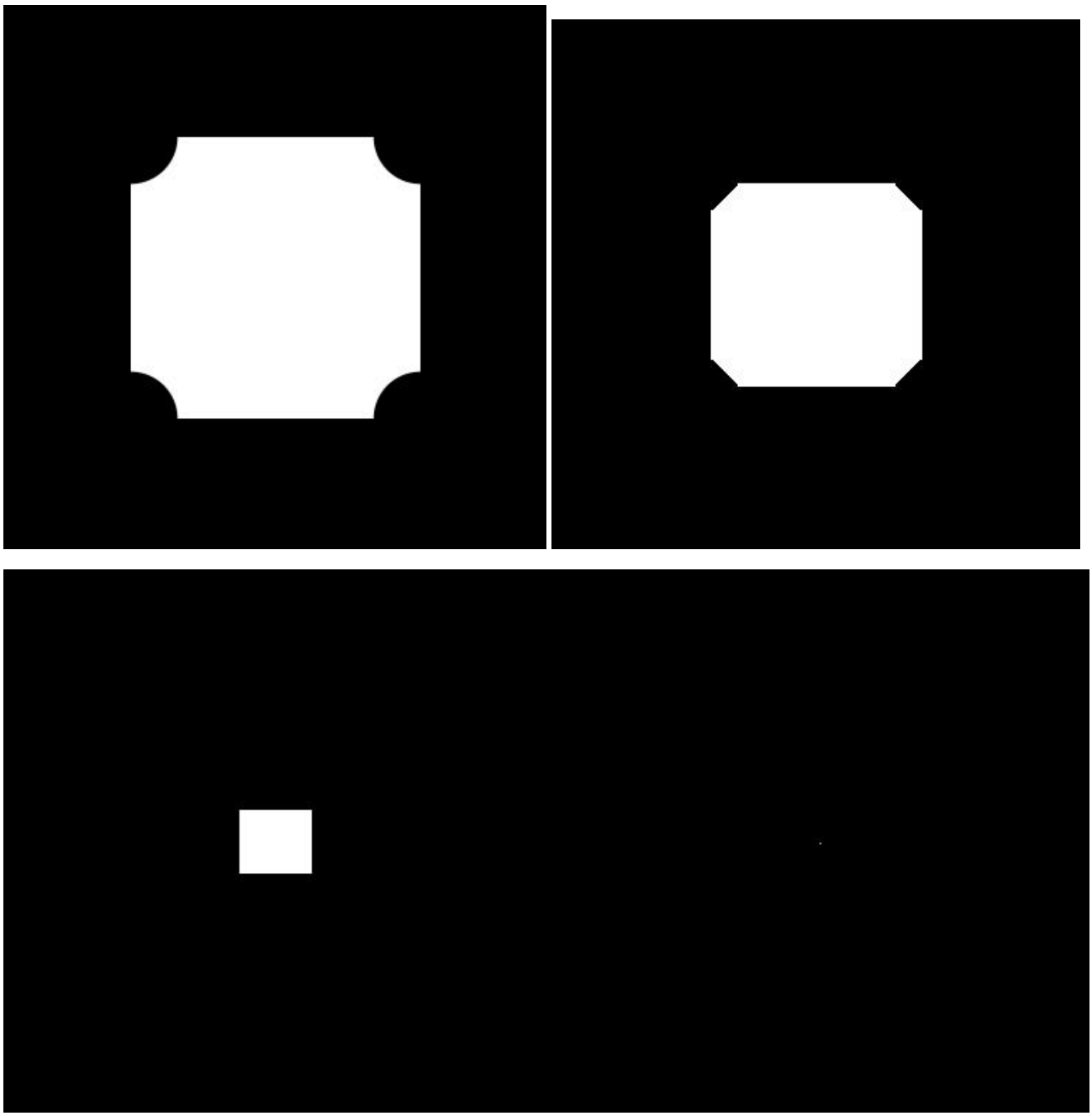


Fig: Shrinking output of Pattern 2

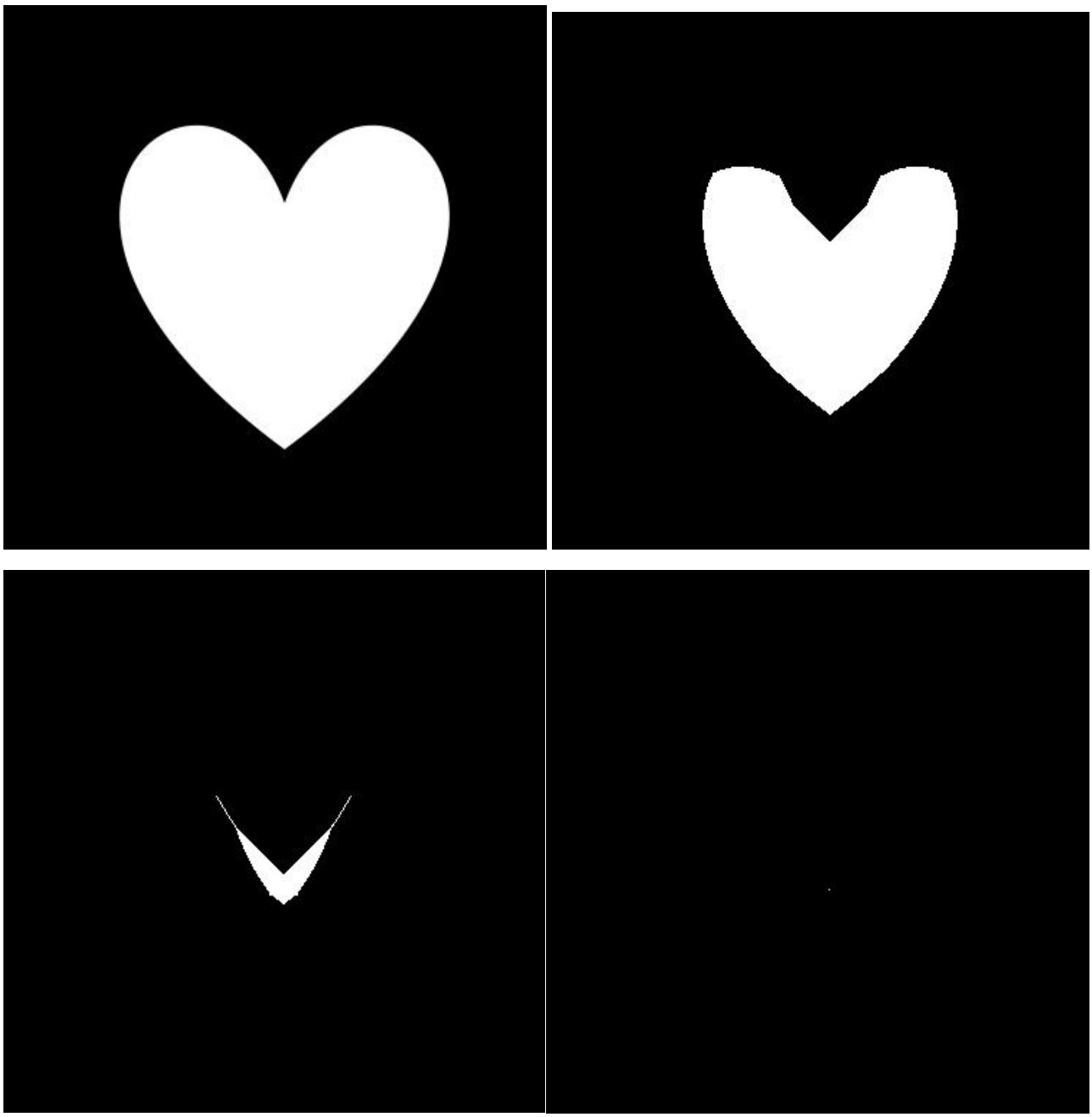


Fig: Shrinking output of Pattern 3

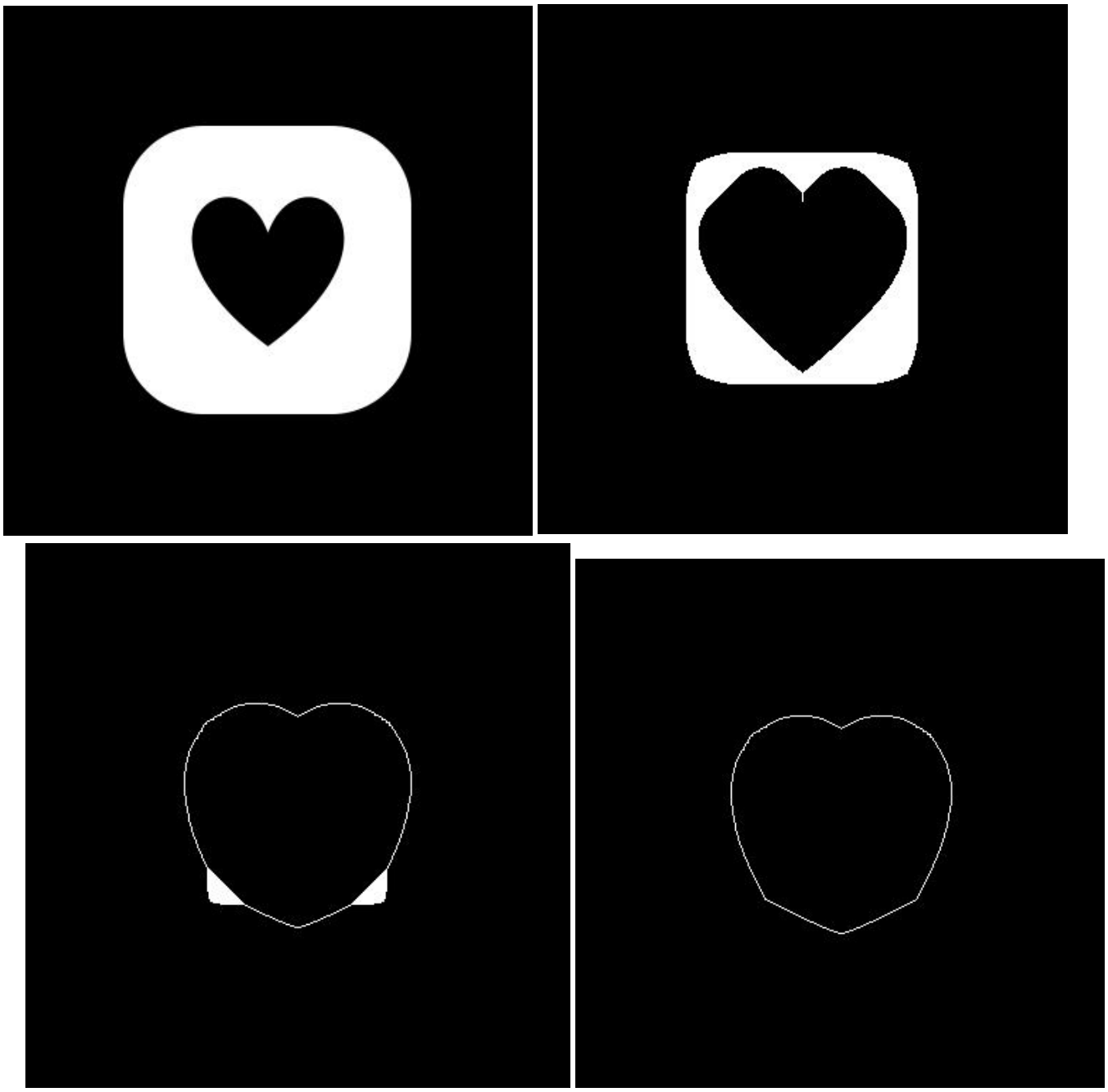


Fig: Shrinking output of Pattern 4

Thinning

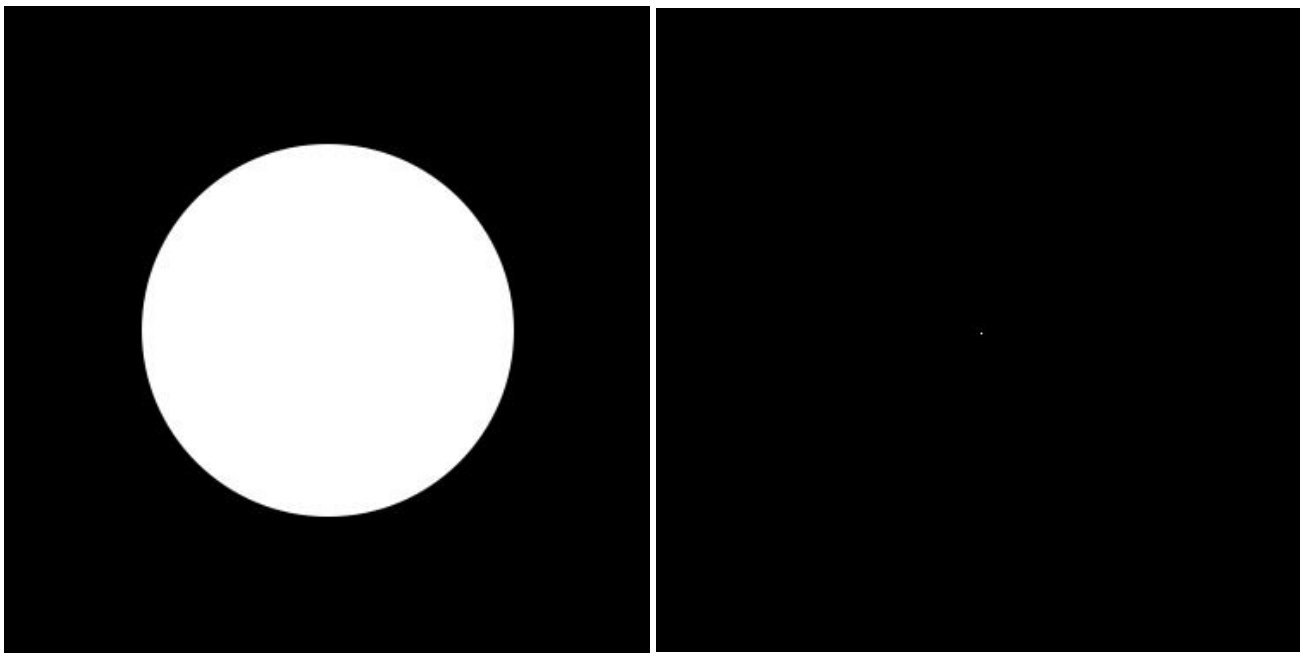


Fig: Thinning output of Pattern 1

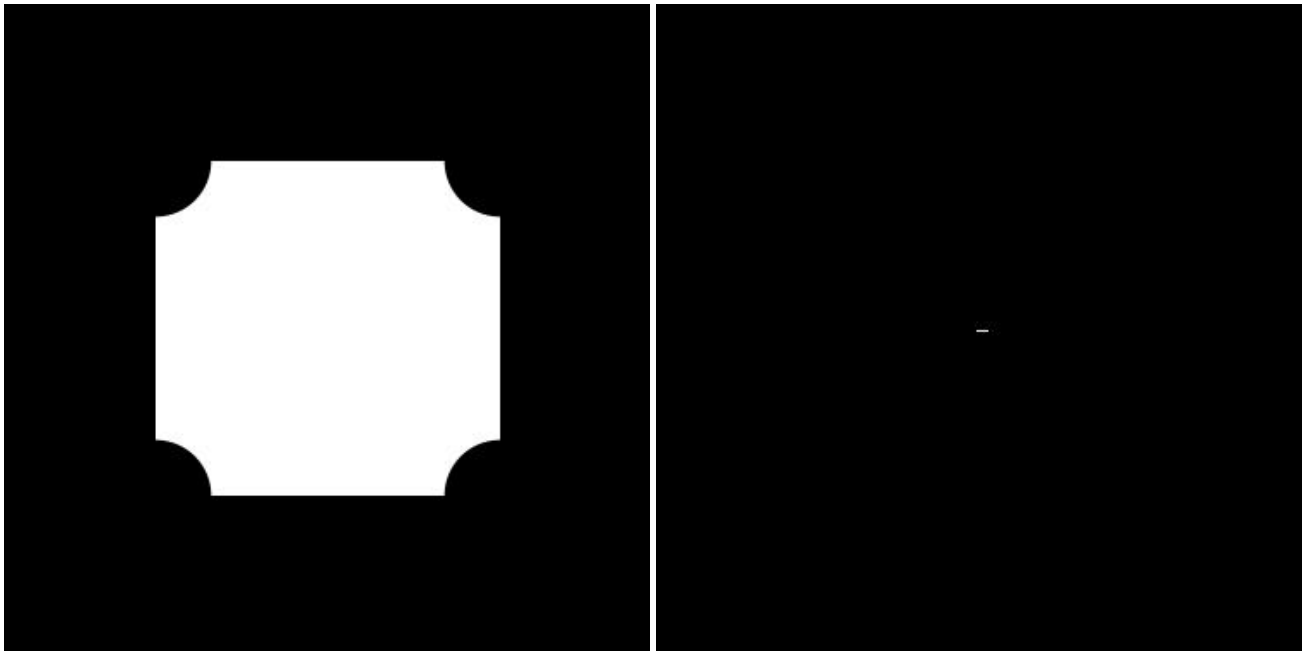


Fig: Thinning output of Pattern 2

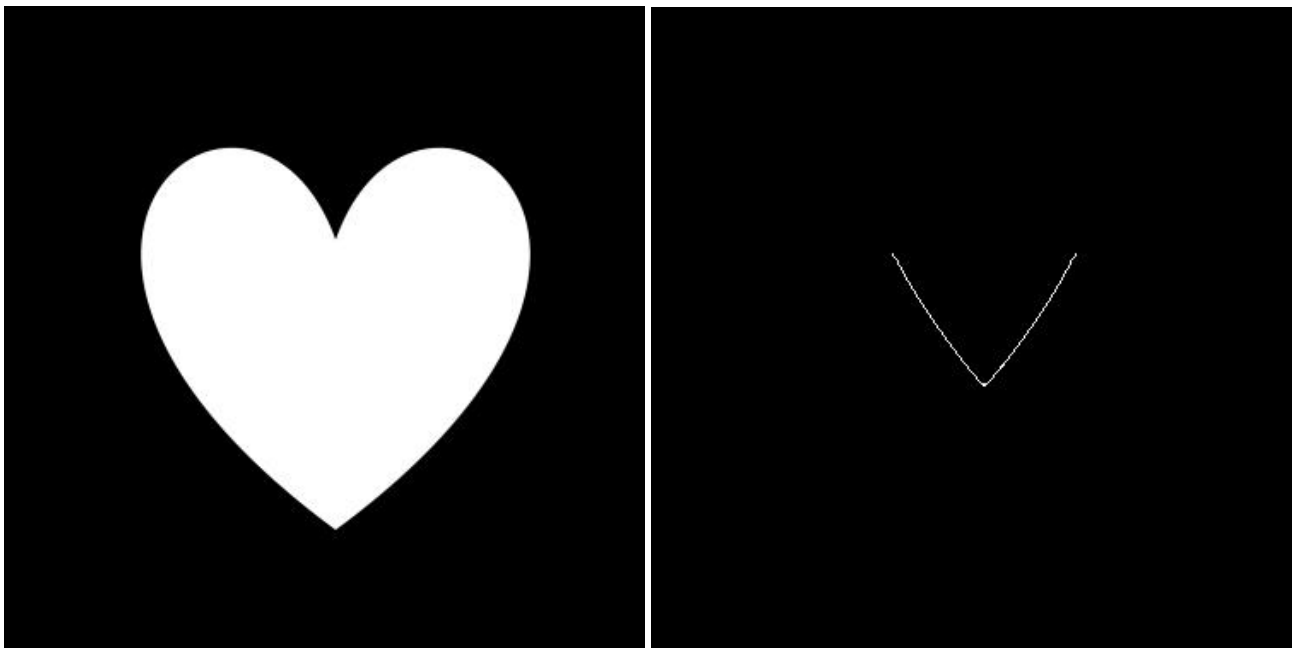


Fig: Thinning output of Pattern 3

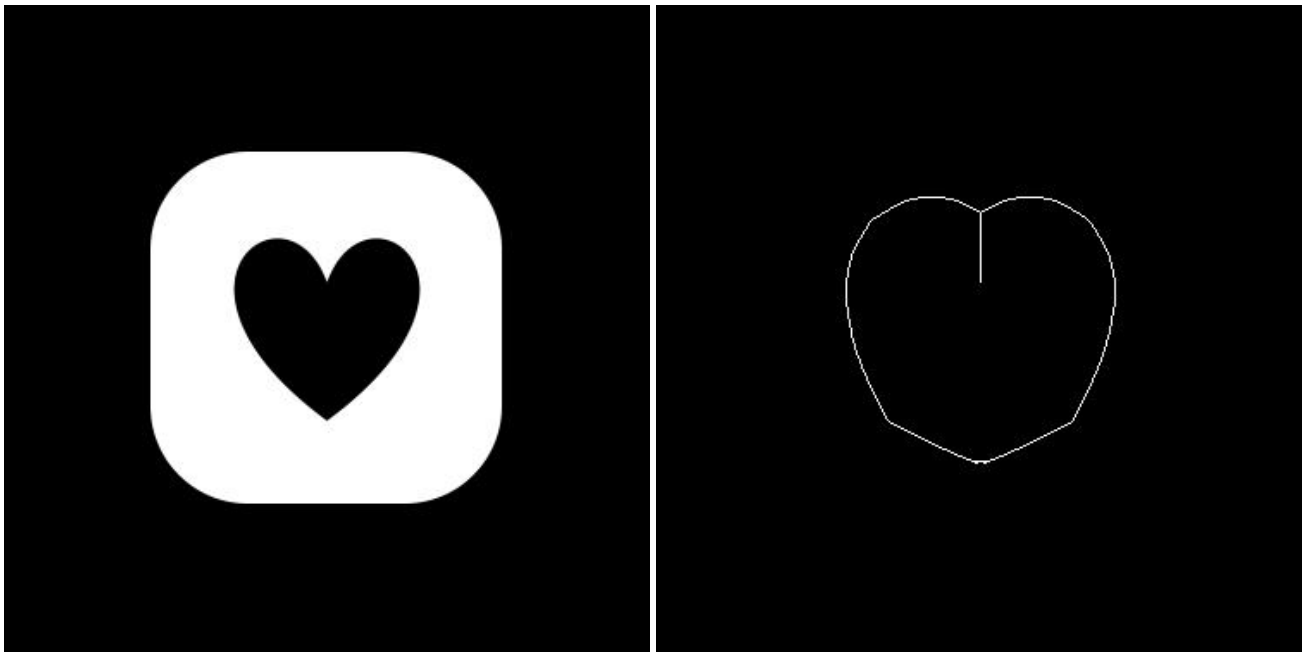


Fig: Thinning output of Pattern 4

Skeletonizing

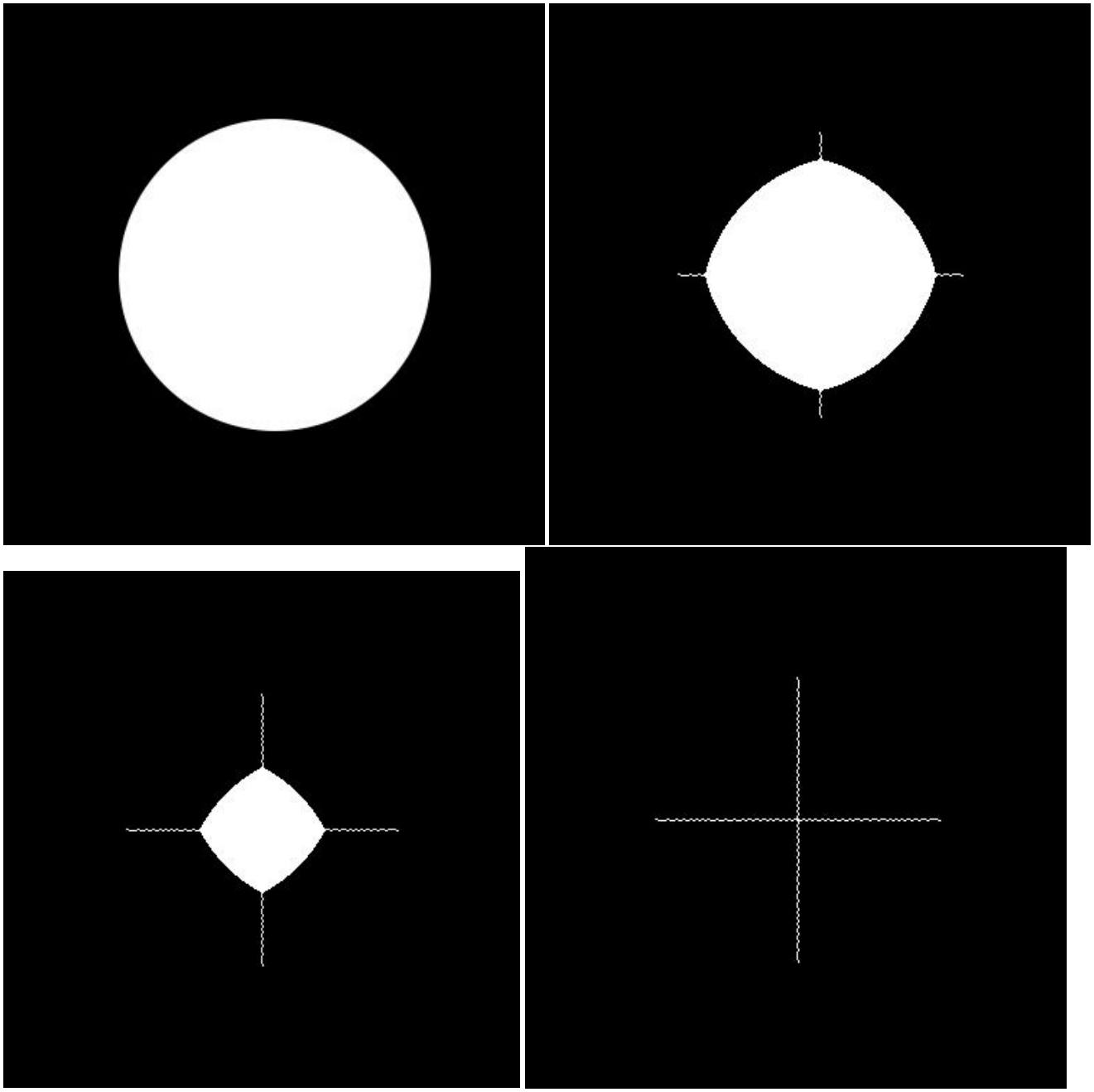


Fig: Skeletonizing output of Pattern 1

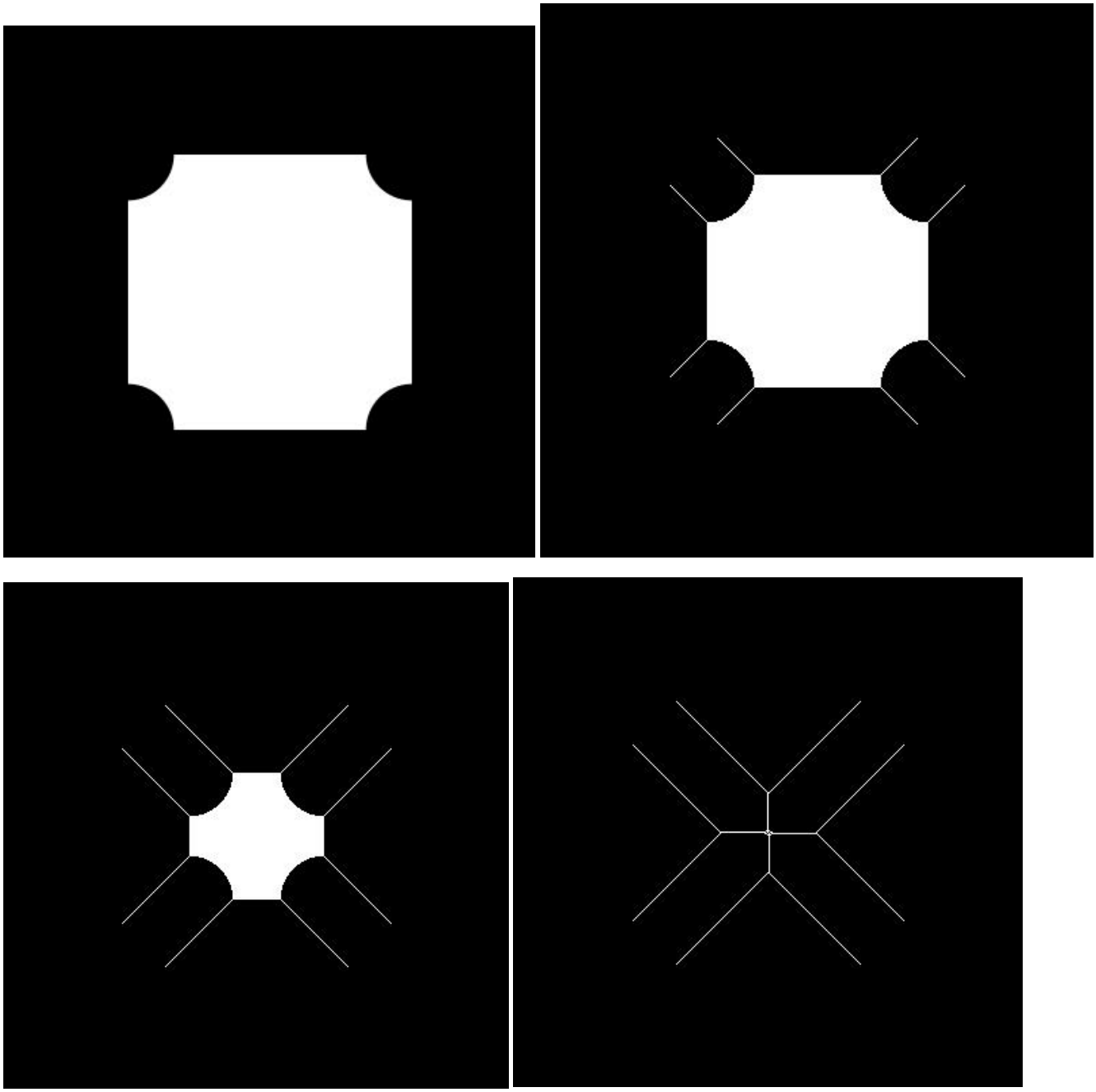


Fig: Skeletonizing output of Pattern 2

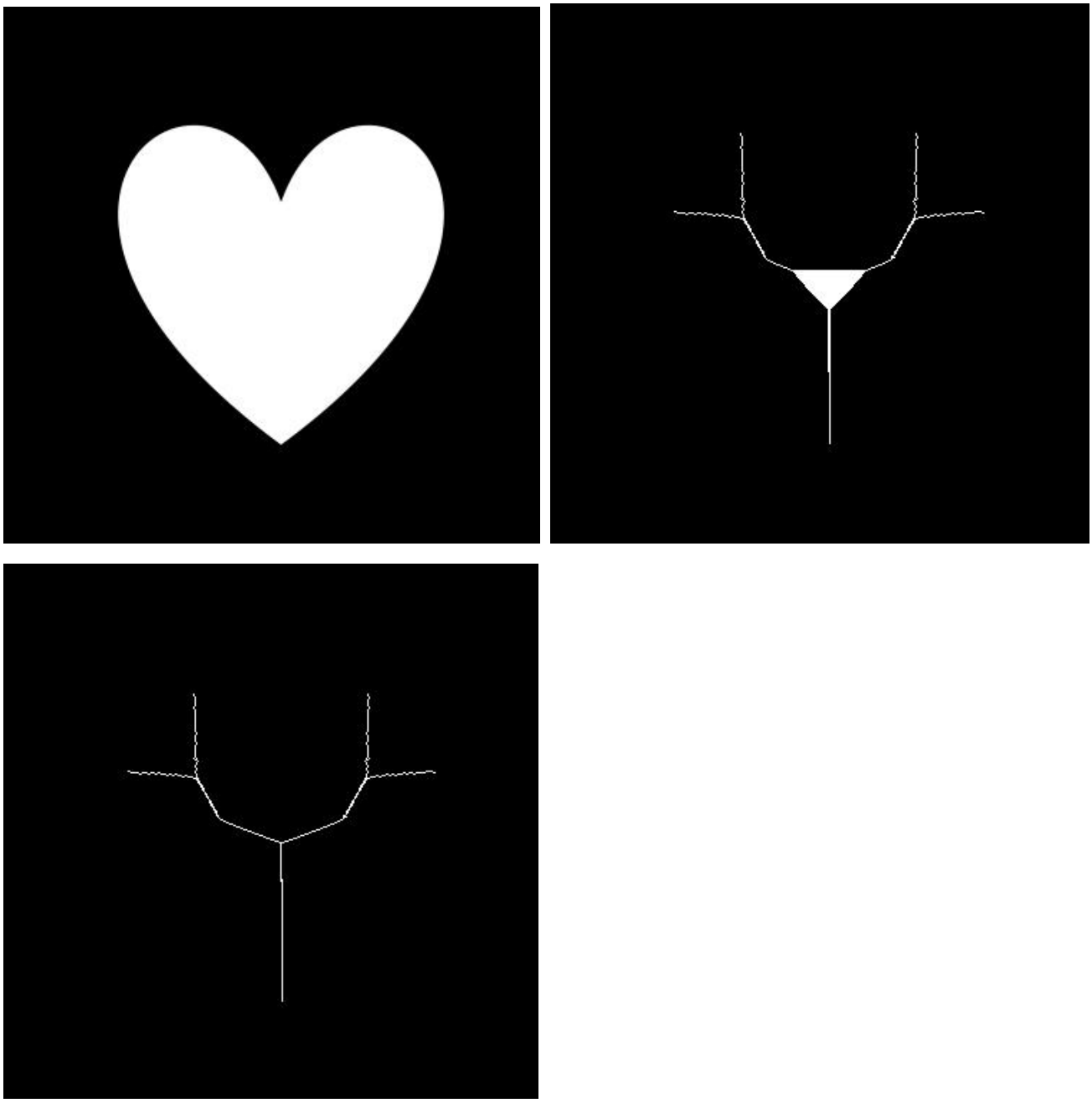


Fig: Skeletonizing output of Pattern 3

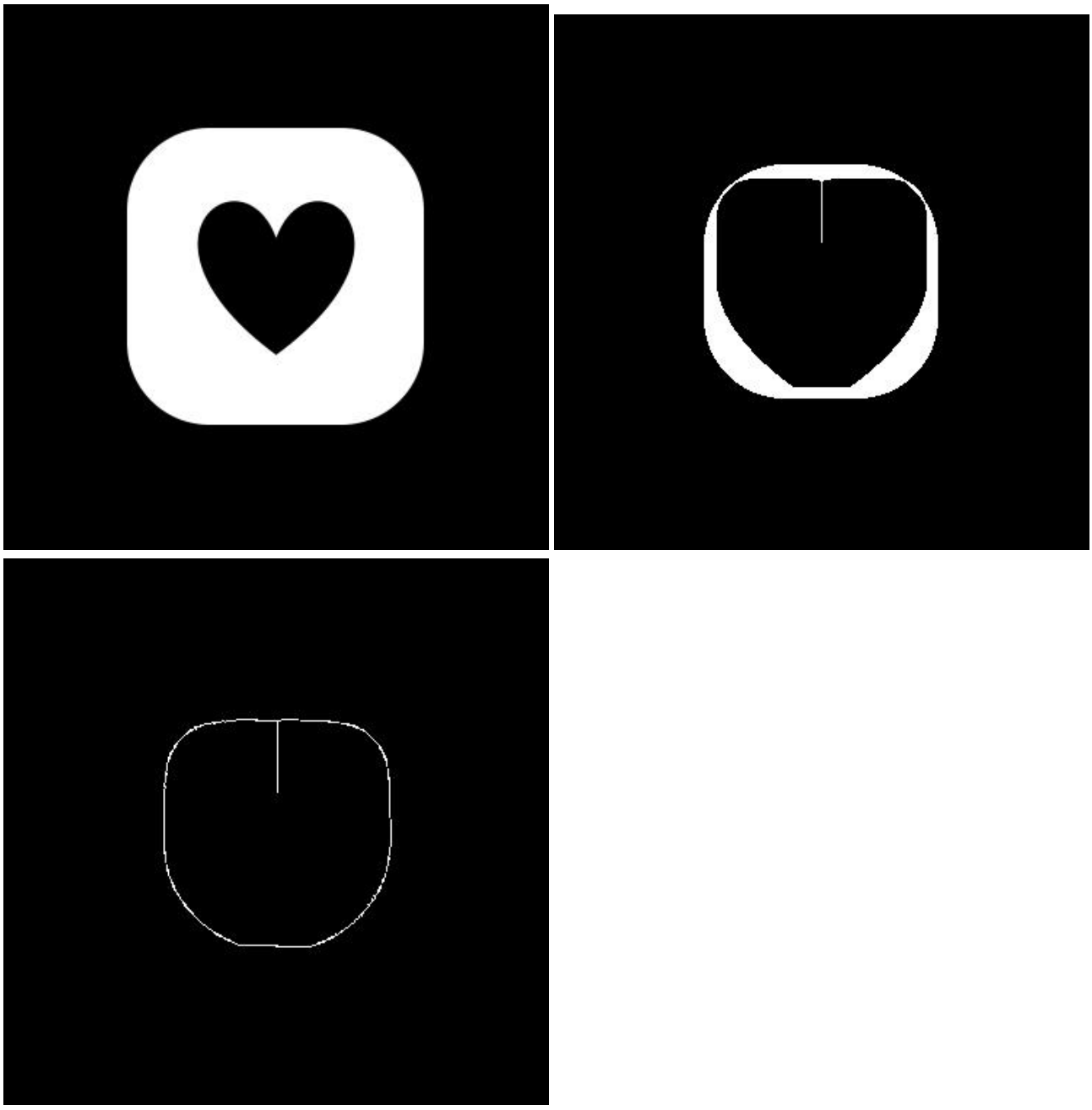


Fig: Skeletonizing output of Pattern 4

DISCUSSION

- We use a lot of Conditional and Unconditional Masks to achieve Shrinking. Thus, we intend to erase black pixels from an image such that an object

without holes erodes to a single pixel at or near it's center of mass. Similarly, an object with holes erodes to a connected ring-like structure lying midway between each hole and it's nearest boundary on the outside.

- We use a lot of Conditional and Unconditional Masks to achieveThinning. Thus, we intend to erase black pixels from an image such that an object without holes erodes to a minimally connected line that is located equidistant from the nearest outer boundaries. Similarly, an object with holes erodes to a minimally connected ring-like structure lying midway between each hole and it's nearest boundary on the outside.
- We use a lot of Conditional and Unconditional Masks to achieve Skeletonizing. It is a type of thinning. Here, we intend to erase black pixels from an image such that an object without holes erodes to a minimally connected rooted stroke line that is located equidistant from the nearest outer boundaries. Similarly, an object with holes erodes to a minimally connected ring-like structure lying midway between each hole and it's nearest boundary on the outside.
- For Pattern 1, it took 109 Iterations to get completely shrinked image and Thinning image. 77-Skeleton
- For Pattern 2, it took 101 Iterations to get completely shrinked image and 98 iterations to give thinned image.97 – skeleton
- For Pattern 3, it took 143 Iterations to get completely shrinked image 89 iterations to give thinned image.85 -Skeleton
- For Pattern 4, it took 49 Iterations to get completely shrinked image and 50 iterations to give thinned image. 48 – Skeleton
- Shrining was quick.
- After one pixel remaining, it doesn't shrink further however much the iterations are increased.

DEFECT DETECTION AND CORRECTION (DEER PROBLEM)

ABSTRACT AND MOTIVATION

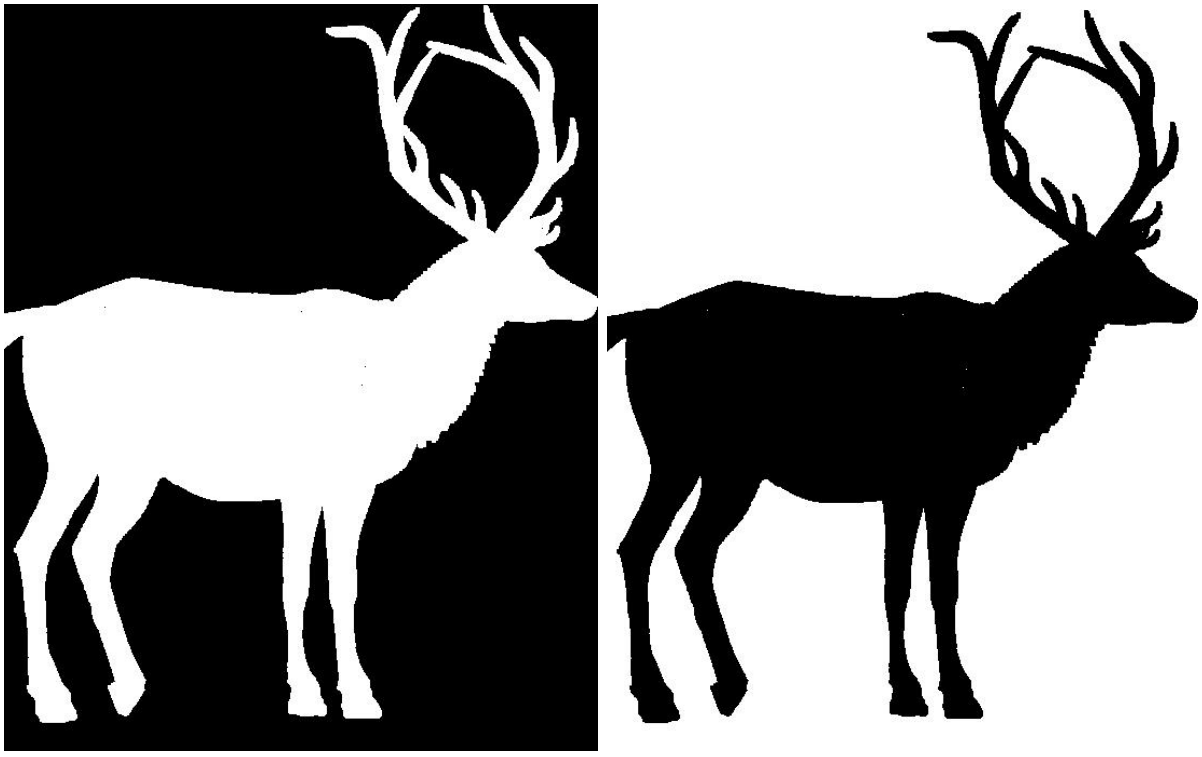
In this task, we are supposed to design an algorithm that uses morphological processing operations to count the number of defect in the image i.e. holes and then correct the holes. We can try multiple approaches.

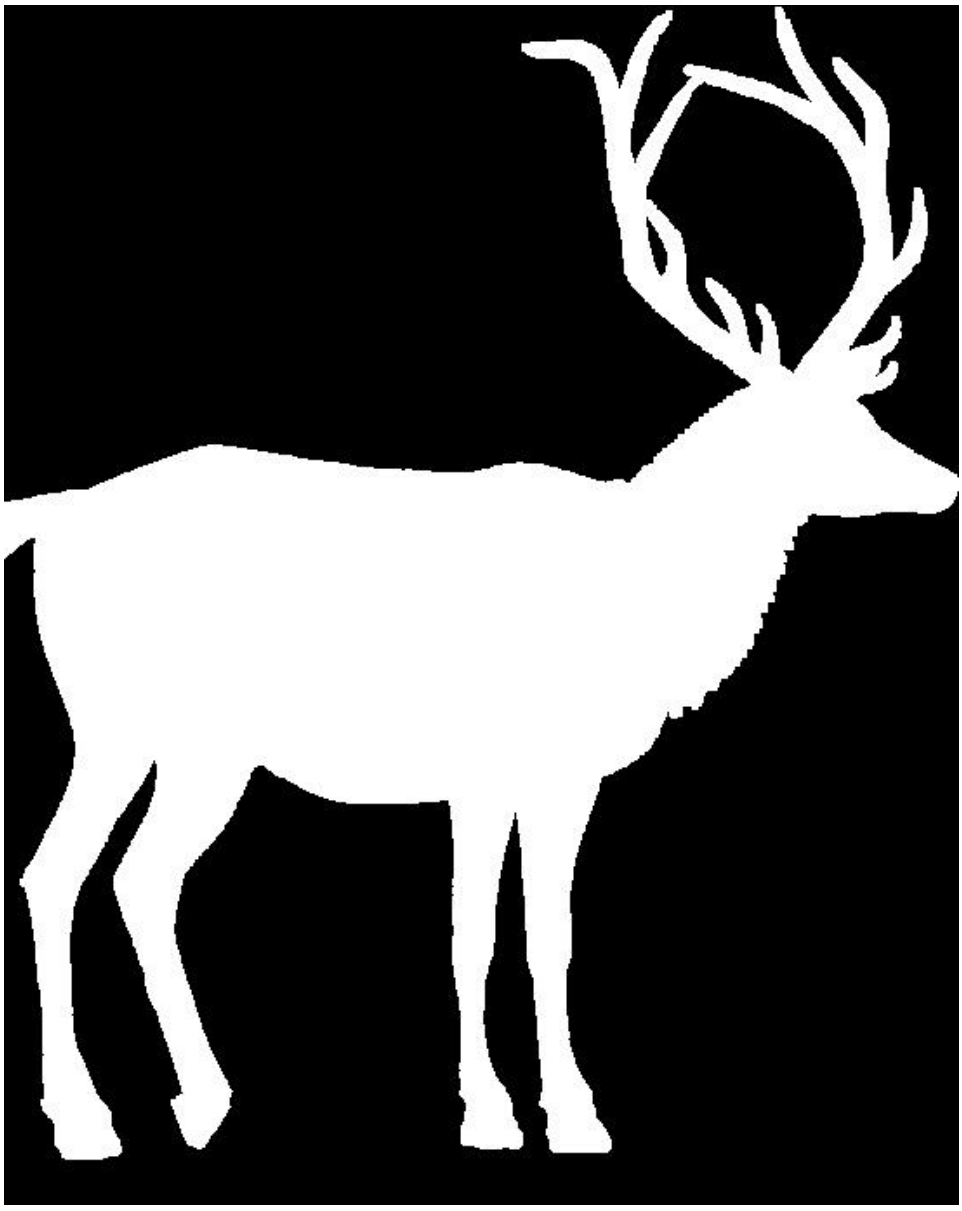
APPROACH AND PROCEDURE

Algorithm:

1. Implement mask for checking if a pixel has eight 1's surrounding a 0 pixel.
2. Scan this mask throughout image and keep count of hits.
3. Count is the number of defects.
4. Fill this defects to 1 in the next iteration.

EXPERIMENTAL RESULTS





DISCUSSION

Deer is not defectless. There are 5 black dots.

Coordinates : (208, 499), (281, 94), (285, 276), (336, 335), (353, 332)

RICE- GRAIN PROBLEM

APPROACH AND PROCEDURE

To count the rice grains, I did the following things in C++:

1. Convert into grayscale image and perform dilation followed by erosion to close any open edges.
2. Threshold and binarize the image. Threshold is fixed so that the darker rice grains are also visible.
3. Set output as 1 for the pixel values above 127 and below 26. This captured all the rice grain visibility.
4. Then image is shrinked and filter is convolved to detect defect section and increment count when found hit. This counts = 55.

Also, used MATLAB inbuilt functions to sort the grains according to type.

Steps:

1. Detect edge using canny and perform dilation
2. Fill the holes
3. Perform erosion
4. Fill the area using regionprops
5. Measure stats of each rice grain as area.
6. Then remove small objects using bwareaopen
7. Count the connected componets using bwconncomp, 4 connectivity.

EXPERIMENTAL RESULTS

Figure 36 Original Rice Grain Image

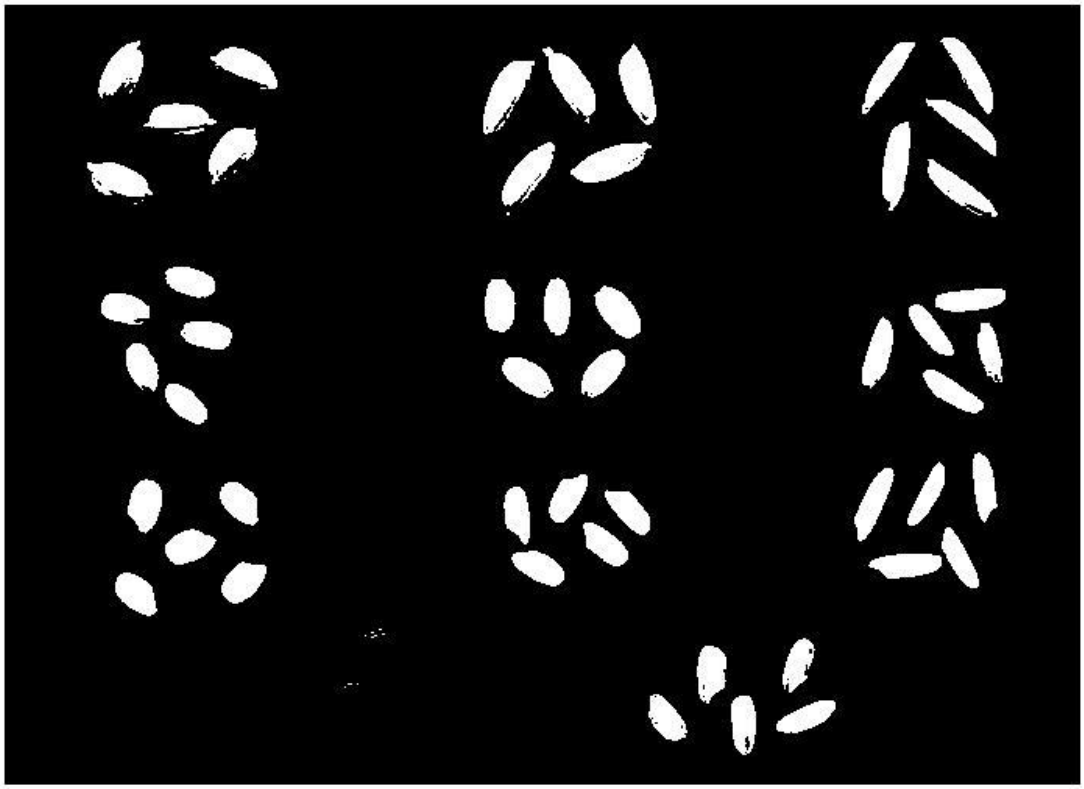


Figure 37 Binarized Rice Grain Image

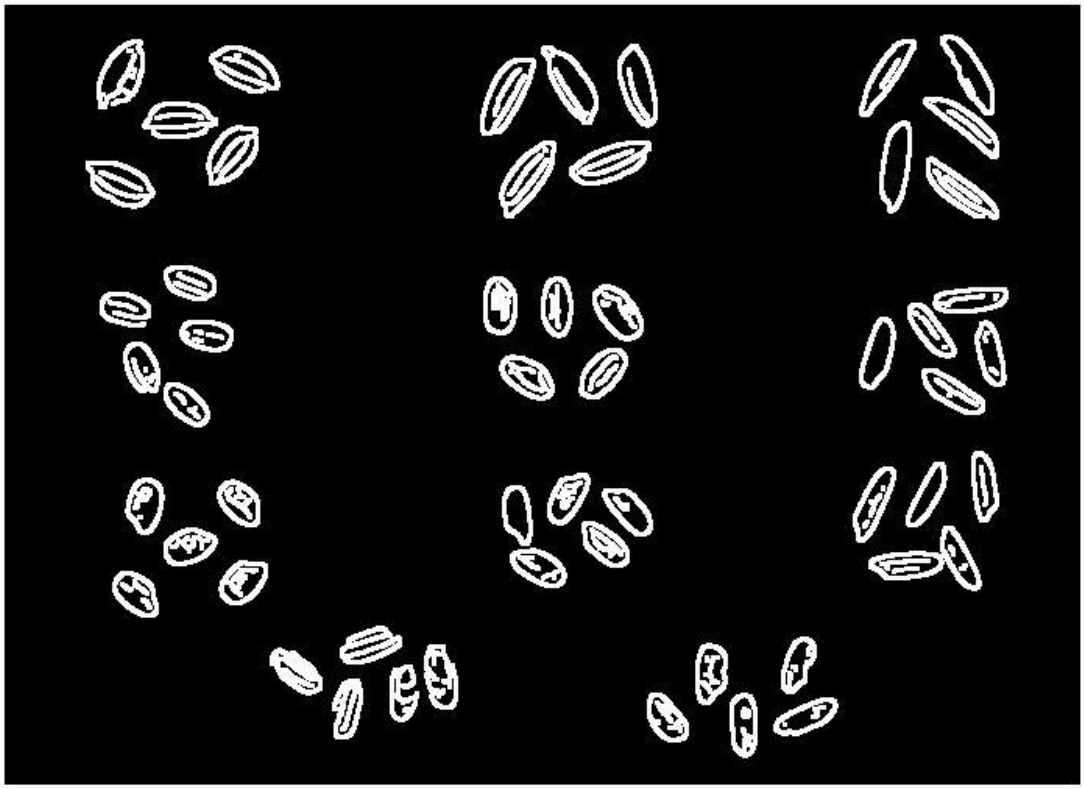


Figure 38 Structural Element Dilated Rice Grain Image

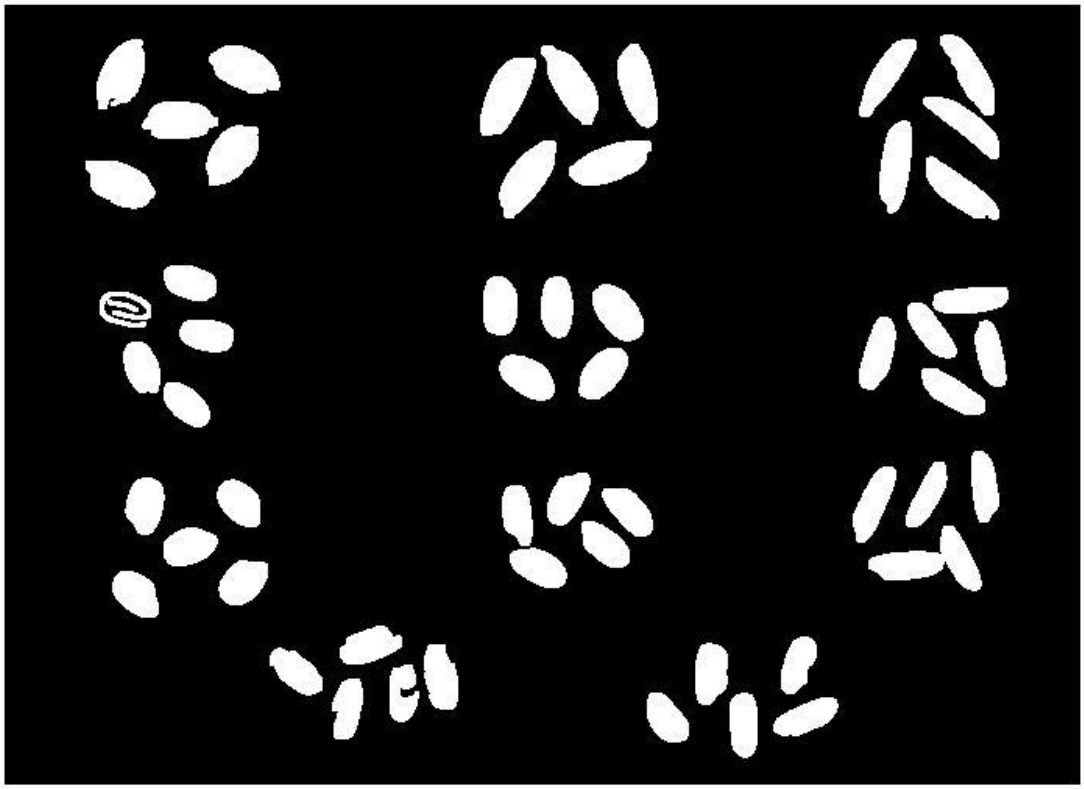


Figure 39 Filled Rice Grain Image

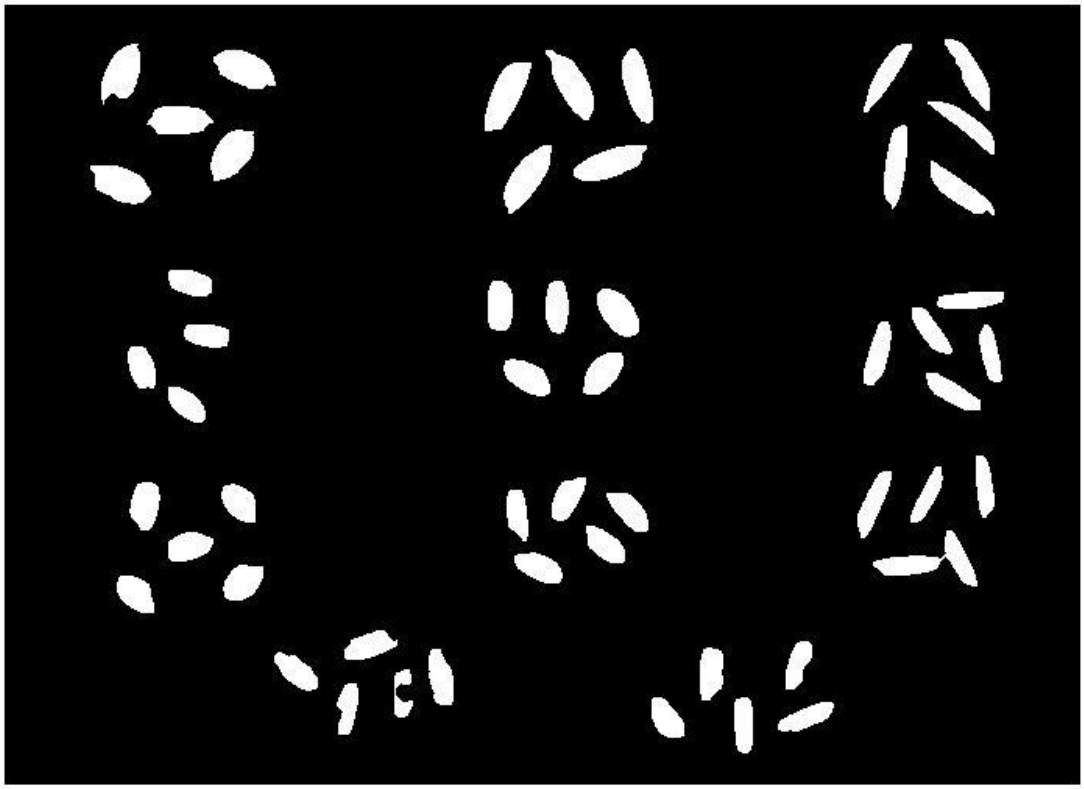


Figure 40 Eroded Final Rice Grain Image

Figure 41 Rice Grain Sizes Image

```
60 -    size(stats)
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> DIP_rice

ans =

    55     2
```

DISCUSSION

There are total 55 rice grains, as shown in the above output.

This was obtained by shrinking and counting the dots in the shrunk image.

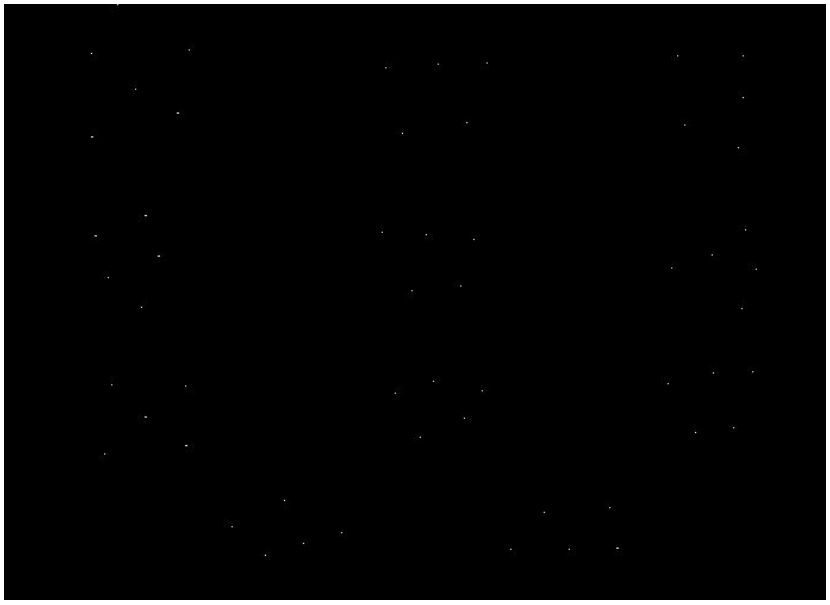


Figure 42 Rice dots after shrinking

REFERENCES:

1. http://eeweb.poly.edu/~yao/EL5123/lecture12_ImageWarping.pdf
2. https://arxiv.org/ftp/arxiv/papers/0906/0906.3770.pdf
3. https://www.mathworks.com/matlabcentral/answers/375096-calculate-the-size-of-the-smallest-rice-grain-in-the-image
4. https://www.mathworks.com/help/images/correcting-nonuniform-illumination.html
5. W. K. Pratt, "Geometrical Image Modification," , 2002. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/0471221325.ch13/summary. [Accessed 1 3 2019].
6. Class Notes and assignments
7. http://www.tannerhelland.com/4743/simple-algorithm-correcting-lens-distortion/