EE 569 Homework 3

Problem 1: Geometric Modification

(a) Geometric Transformation

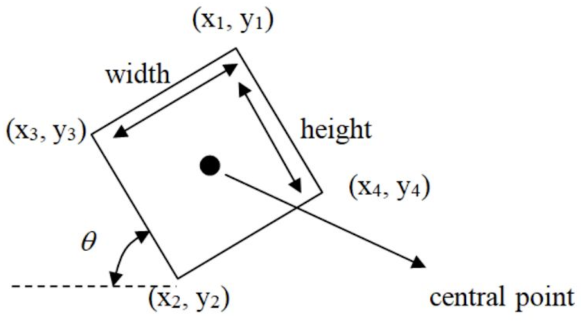
Motivation:

Understanding how to perform Geometric Transformations like Translation, rotation, etc on Image data and use them in an application. (Solving a puzzle in this case)

Approach:

Lets manipulate the pieces of puzzle first:

1. Read the image of a piece
2. The image is given in image coordinates. To rotate or translate, convert these to cartesian coordinates.
3. Find the corner points of given piece
4. Find width, height, angle of rotation and centre points of piece as follows:



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

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

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

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

5. Calculate (u,v) from the operation below. Ie. by using inverse transform

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Here, (x,y) are coordinates of the piece(cartesian) and (u,v) are coordinates we get after rotation and translation to centre. Convert the coordinates back to image coordinates and continue.

6. Now, find corner points of this new image formed by above(u,v)s.
7. Scale the image to size of hole(160x160) by bilinear interpolation.

   To find value of pixel at a fractional (x’,y’), we use weighted sum of surrounding pixels:

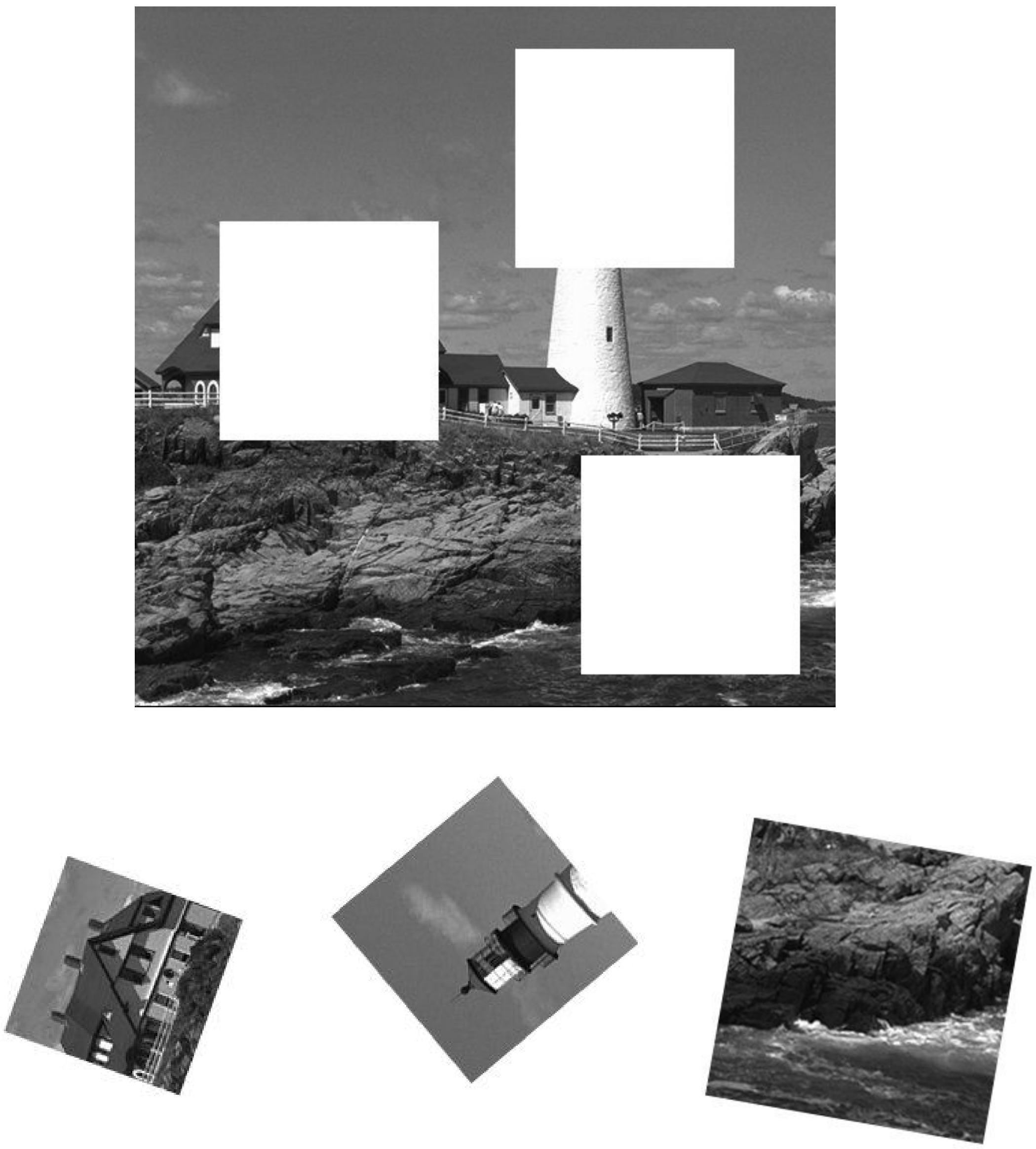
   Let, ratio_rows= width of original image/ width of scaled image
   ratio_columns= height of original image/ height of scaled image
   x’ = r’xratio_rows, where r’ are row coordinates of original image
   y’ = c’xratio_rows, where c’ are row coordinates of original image
   x= floor(x’)
   y= floor(y’)
   error_r = x’- x, error_c= y’-y

   Then, Scaled_Image(x’,y’)= Original_image(x,y)x(1- error_r)x(1- error_c) +
   Original_image(x,y+1)x(1- error_r)x(error_c) +
   Original_image(x+1,y)x(error_r)x(1- error_c) +
   Original_image(x+1,y+1)x(error_r)x(error_c)

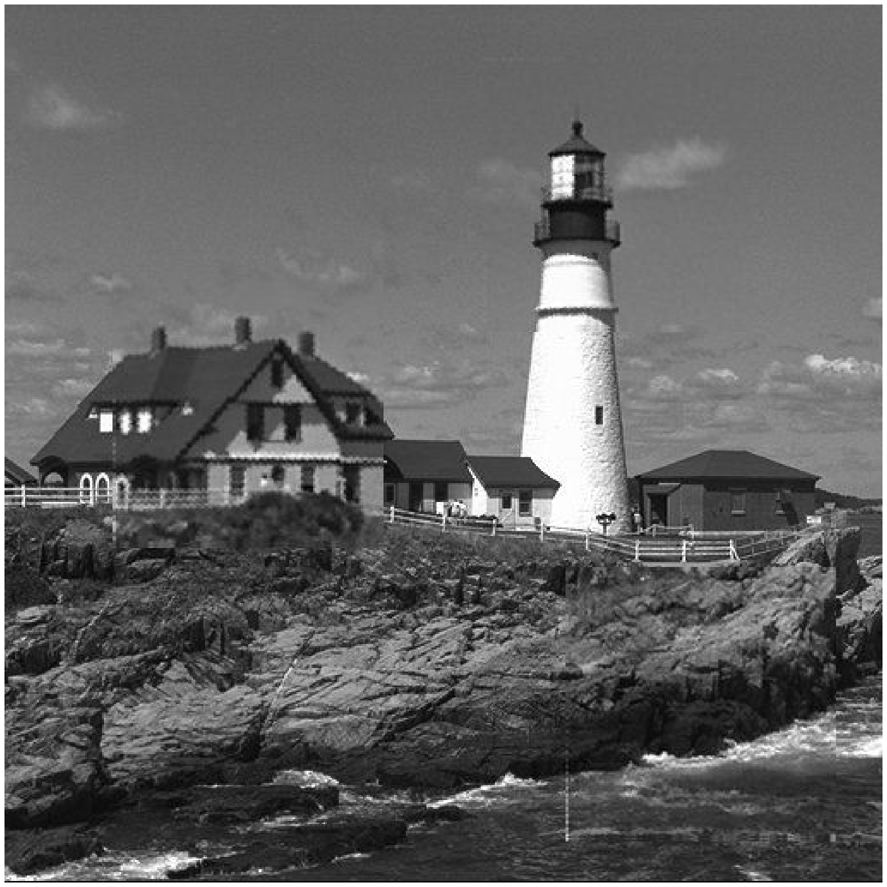
Now, for the puzzle,

1. Locate the corners of holes in puzzles
2. Fill the scaled piece in the hole.
3. Fill the extra white borders by replicating the next row/column into the white line.

Experimental Results:



Before



After

Discussion:

We use inverse mapping instead of forward mapping to avoid black lines(holes) in the output. In forward mapping, if a 3x3 image is mapped to a 7x7 image, pixels from original image are directly assigned to their scaled pixel eg. (0,0) → (0,0), (0,1) →(0,3), (0,2) → (0,6). Now, (0,1),(0,2),(0,4),(0,5) are not assigned any values. Hence, they remain as holes.
In reverse mapping, we can solve this problem. The problem after finding inverse mapping is we might fractional indices, to find the value of original image at a fractional pixel location, we take weighted sum of surrounding 4 pixels based on Euclidean distance. Procedure explained in approach.

We see white lines at border of pieces once they are filled in the puzzle, I replicated the value next to that column to fill in the holes.

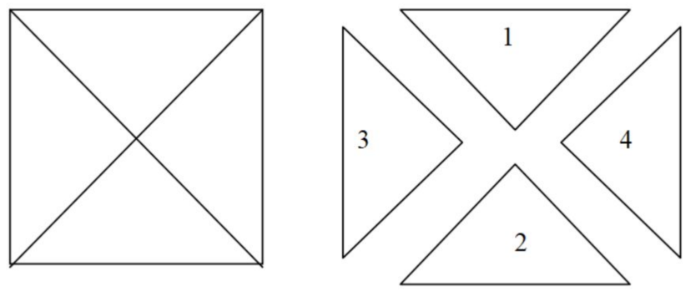
(b) Spatial Warping

Motivation:

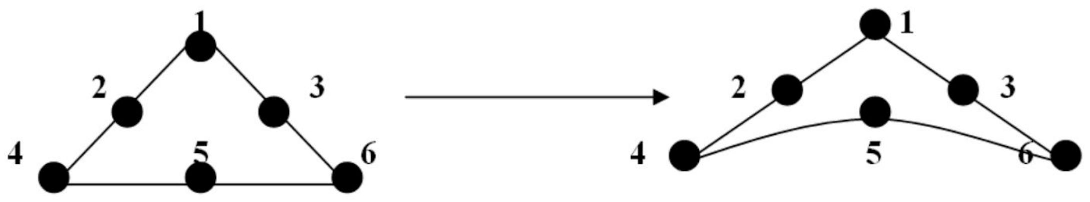
Warping is a process in which you take the image and assign it to another image with a different spatial distortion. It can be used as a technique to correct lens distortion.

Approach:

1. Divide image into 4 triangular images



2. Get six coordinates of original and warped triangles :



3. Find coefficients for warping using:

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} u_0 & u_1 & u_2 & u_3 & u_4 & u_5 \\ v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ y_0 & y_1 & y_2 & y_3 & y_4 & y_5 \\ x_0^2 & x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \\ x_0y_0 & x_1y_1 & x_2y_2 & x_3y_3 & x_4y_4 & x_5y_5 \\ y_0^2 & y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 \end{bmatrix}^{-1}$$

   Here, (u,v)s are the 6 coefficients of warped image and (x,y)s are 6 coefficients of original image

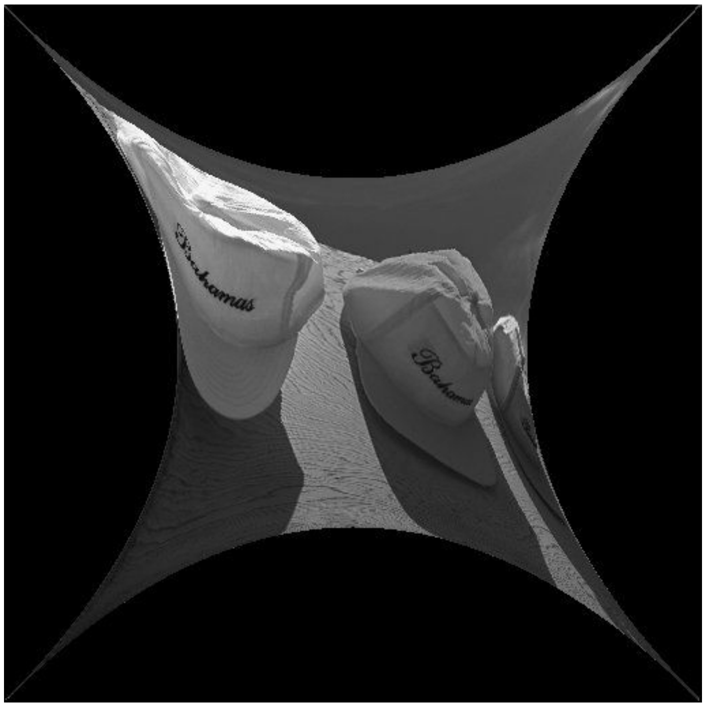
4. Once you have coefficients, you can find indices of warped image corresponding to every index in each triangle

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

5. Carefully write the image pixels from original image to a new image with (u,v) generated. You have to access every pixel in the triangle and fill corresponding pixel in warped image.

Experimental Results:

Original Image


Warped Image

Discussion:
We want a given image in a warped shape. We get the new indexes as shown in procedure. We have used second order polynomial address mapping for warping. It is extension of reverse address mapping for linear case we used in the first part.
Images appear such that they have been projected onto a certain curved surface or reflecting from a certain curved surface.
It is widely used for creative tasks like a snapchat filter. It can also be used in correction of lens distortion, cause due to the curve of lens capturing picture.

(c) Lens Distortion Correction

Motivation:
The lenses we use to capture digital images are curved. The rays that enter the centre of lens are straight but towards the corner light rays curve causing this phenomenon of lens distortion.

Approach:
1. Load the image. You can pad the image (Optinal).
2. Get camera coordinates of image using:

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

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

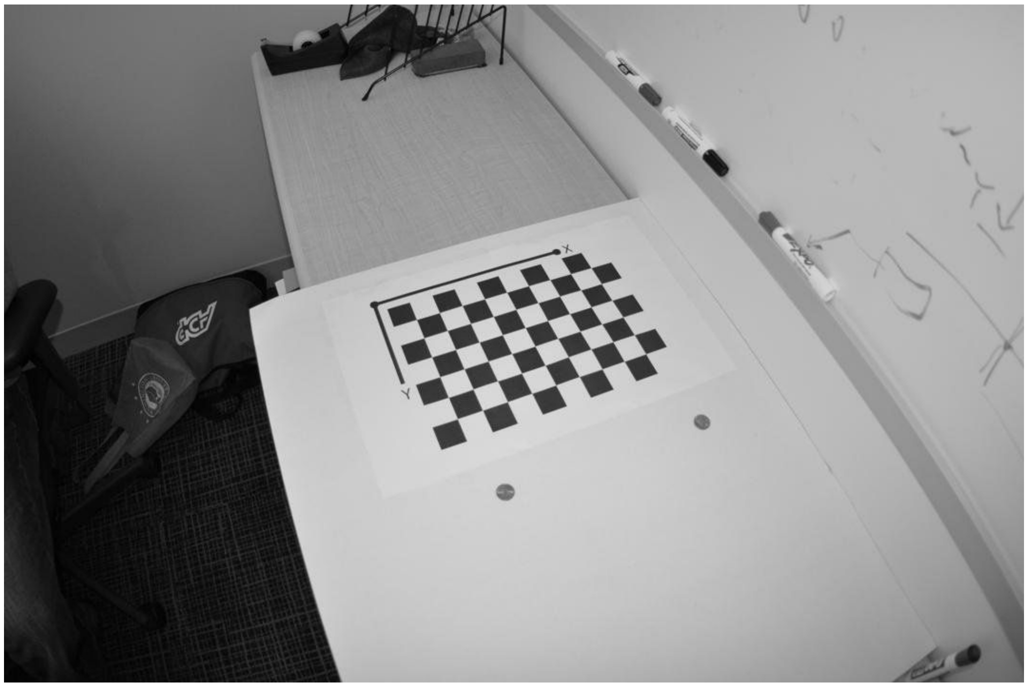
Here, (u,v) are image coordinates of original distorted image. (uc,vc) is the centre point of the image and (x,y) are camera coordinates. Also fx= fy= 600

3. Use the following forward model to find undistorted coefficients(xd,yd):

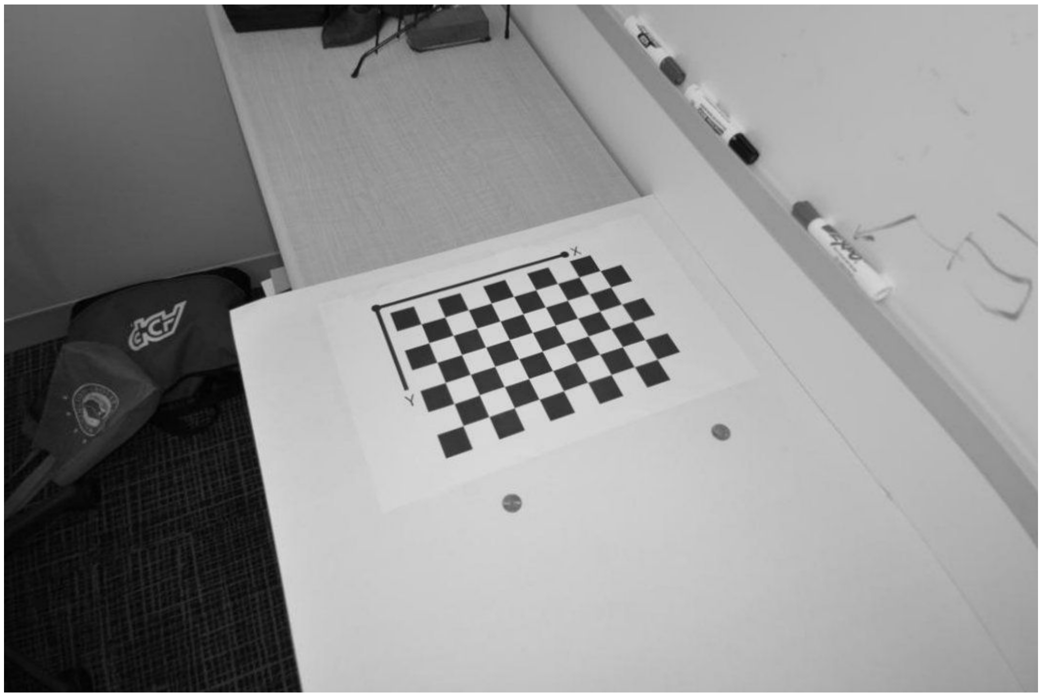
$$x_d = \mathrm{x}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_d = \mathrm{y}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

   Where r is the distance of (x,y) from centre of distorted image, (x,y) are distorted image coefficients and k are distortion coefficients defined by the lens used. Here,
   k1= -0.3536, k2= 0.1730, k3=0
4. Now, get image coordinates of these (x_d, y_d) pairs
5. Create a new image by writing original distorted image using these indices into this new image.
6. Clip the indices out of range. Also, Image coordinates are fractional, use bilinear interpolation explained in first part.

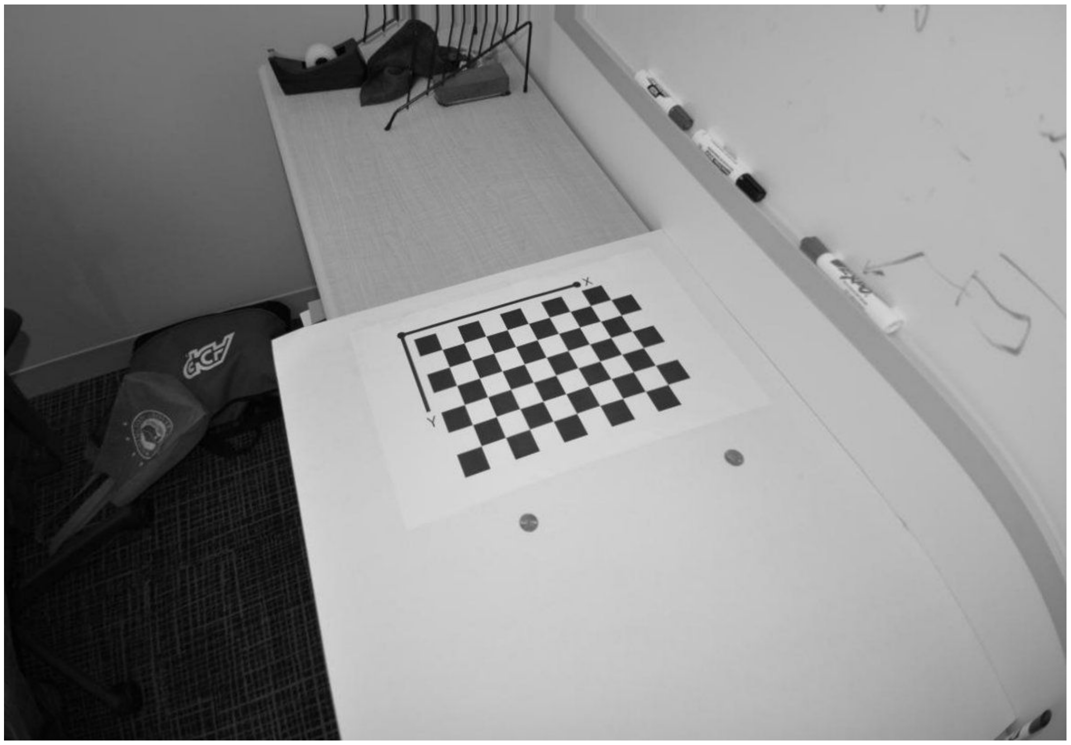
Experimental Results:



Distorted Image



Undistorted image without padding



Undistorted Image with padding

Discussion:
Without padding,
As we can see, the curve of whiteboard and table are fixed. They appear almost straight. Border values are clipped. Reason being, lens are curved and hence forward mapping gives us undistorted image indices outside the range of actual image.
With padding,
The indices which go out of range (or the values which are clipped) can be taken into a picture by increasing its dimension. Thus, nothing is clipped anymore. Ie. all data is retained. But we can see distortion in the corners of these images due to non-linear mapping.
So there's a tradeoff between clipping and distorted corners. We can choose according to our application.

Morphological Processing

(a) Basic Morphological Process Implementation:

Shrinking:
1. Binarize the image. ie. Convert all pixel values to 0 or 1.
2. Write down all the conditional and unconditional masks for Shrinking.
3. Extend boundary of the image by 1 pixel and check if every pixel in image "hits" any of the conditional mask. If it does, assign that pixel as 1, else 0.

4. Extend the boundary again by 1 pixel again and check if every pixel in image “hits” any of the unconditional mask. If it does, assign that pixel as 1, else 0.
5. Output image pixel value can be calculated as
   G(j,k)= X $\cap [\overline{M} \cup$P($[\overline{M}$, $[\overline{M0} \ldots$ $[\overline{M7}$)]
   Here X is input image, $\overline{M}$ is conjugate of M we got after applying conditional masks and P is image we got after applying unconditional masks.
6. Now use this output image as input to next iteration and repeat from step 3.
7. Stop when input and output to one loop re the same images. ie. when there is no further shrinking.
8. Replace all 1s by 255 and write in output image.

Thinning:

1. Follow the same procedure as shrinking with conditional and unconditional masks for thinning.

Skeletonizing:

1. Follow the same procedure as shrinking with conditional and unconditional masks for skeletonizing.
2. After we find the output, we do bridging:
   G(j,k)= X $\cup$[P1$\cup$P2$\cup$P3 $\cup$P4 $\cup$ P5 $\cup$P6]

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

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

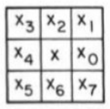
$$L_1 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap \bar{X}_7$$

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

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

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

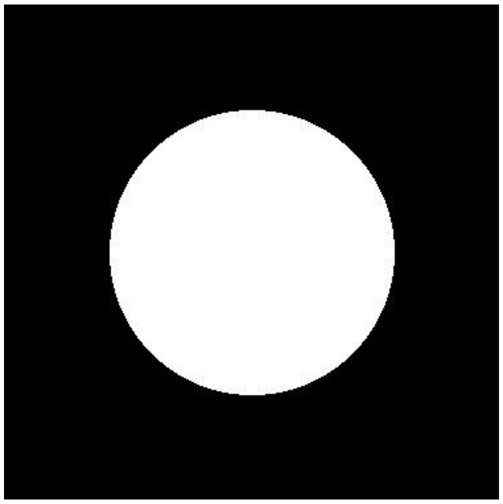
Pixel neighborhood is defined by:

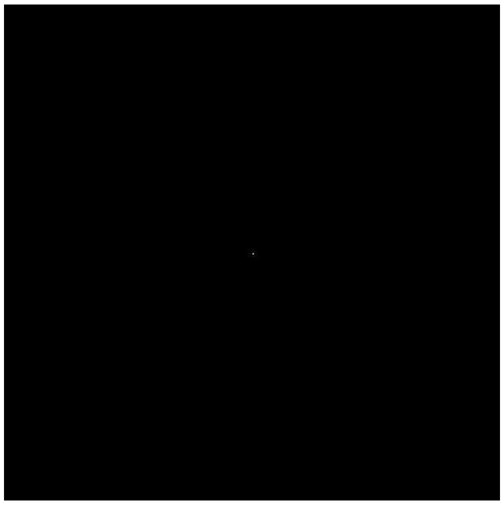


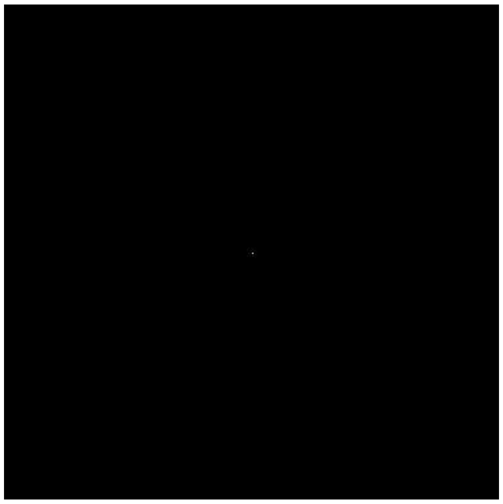
PIXEL NEIGHBORHOOD

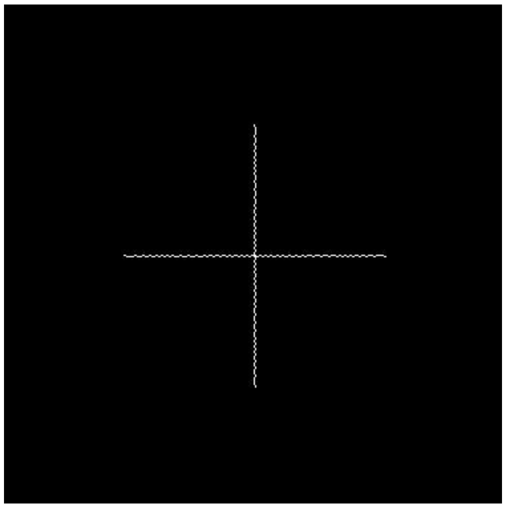
Experimental Results:

Pattern 1:

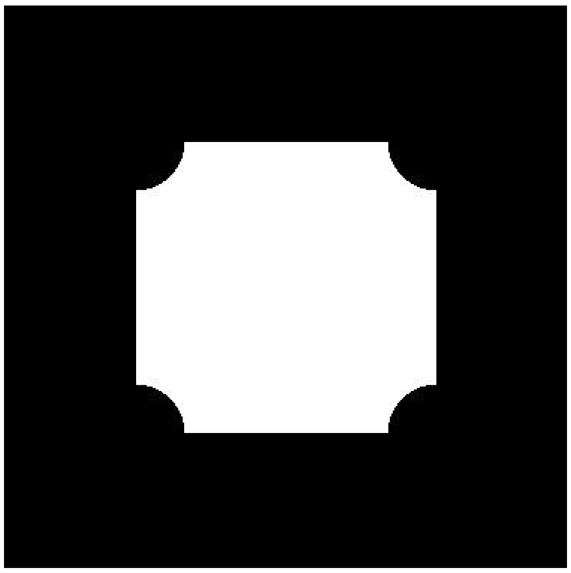

Original

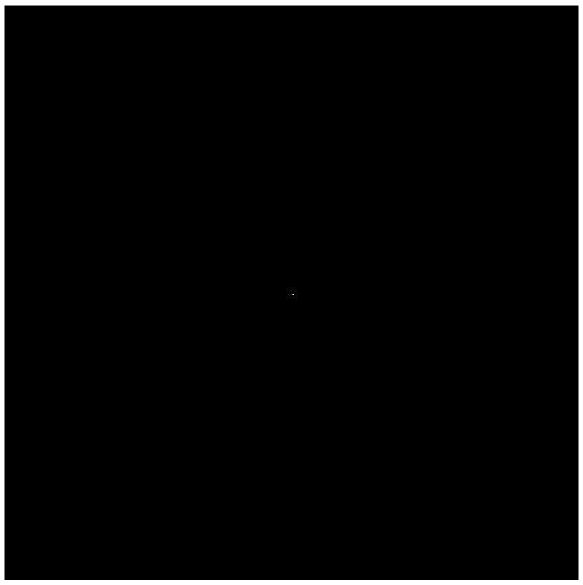

Shrinking

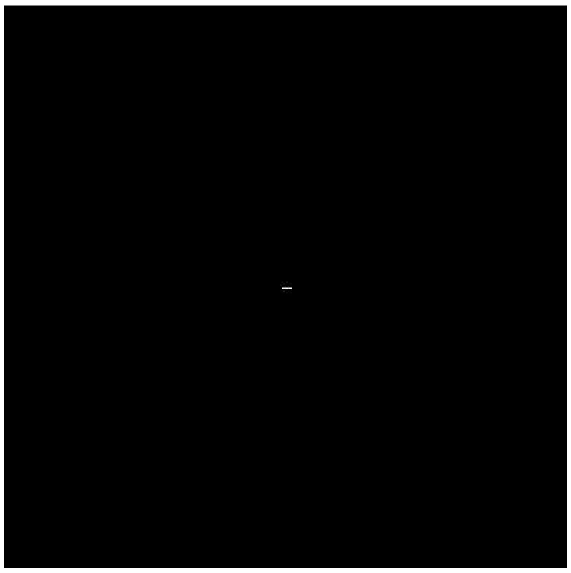

Thinning

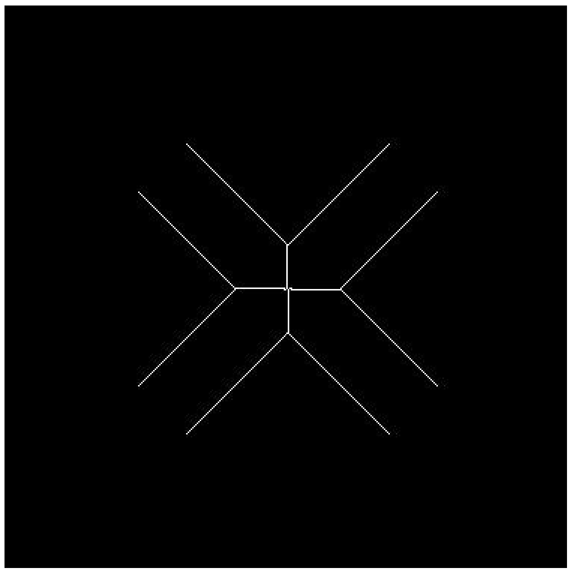

Skeletonizing

Pattern 2:

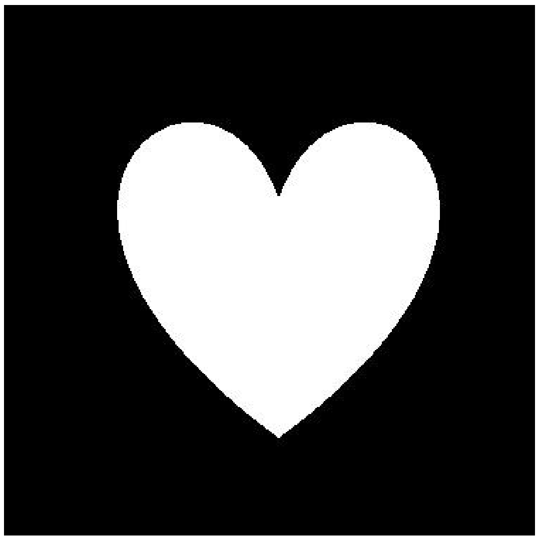

Original


Shrinking

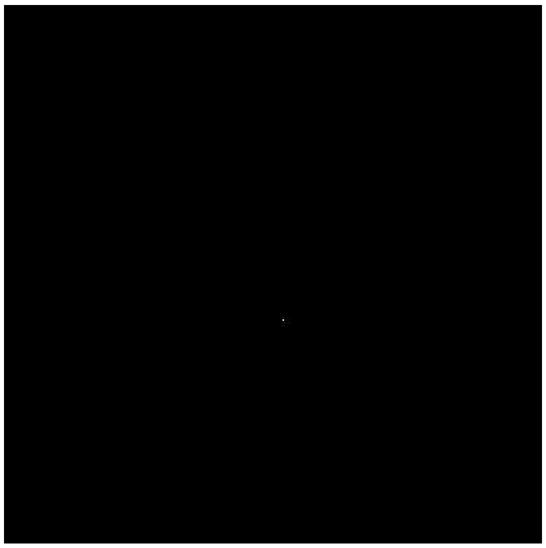

Thinning


Skeletonizing

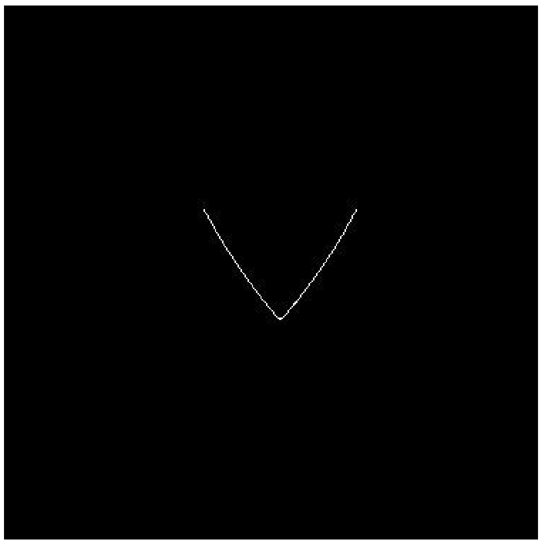
Pattern 3:



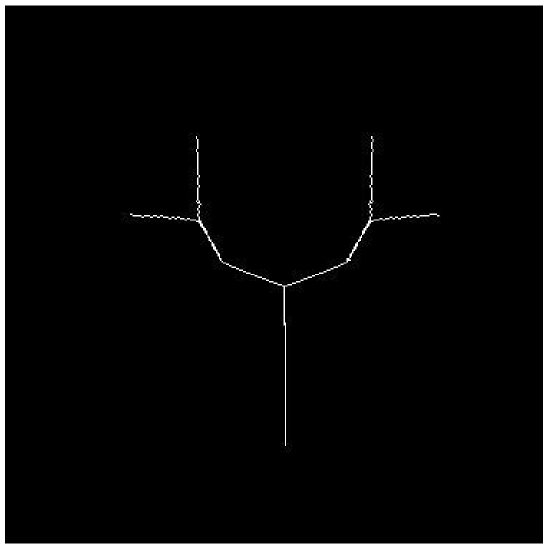
Original



Shrinking

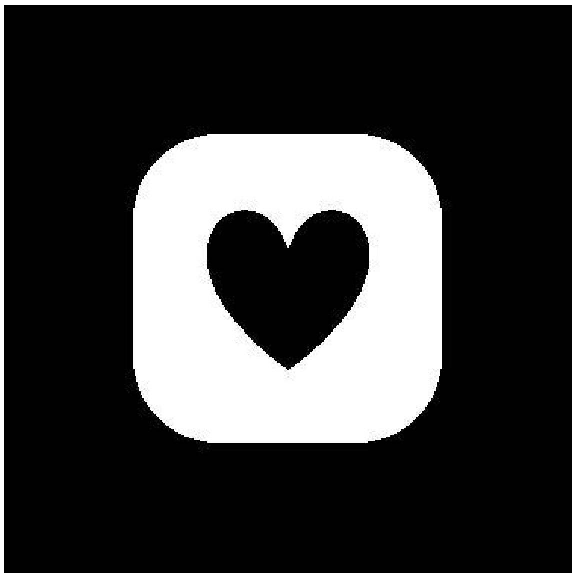
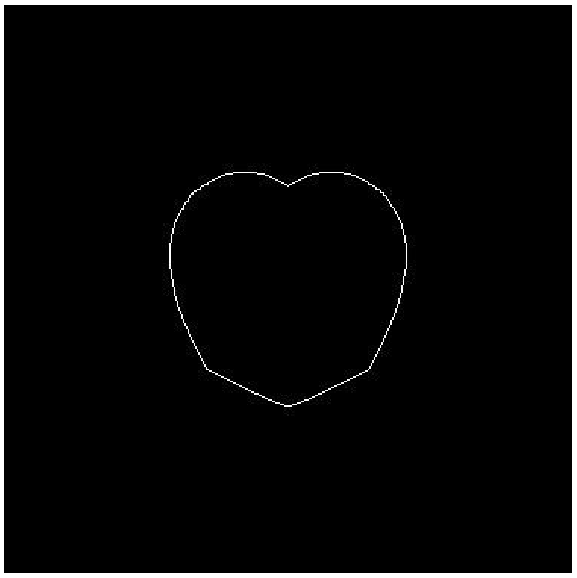


Thinning

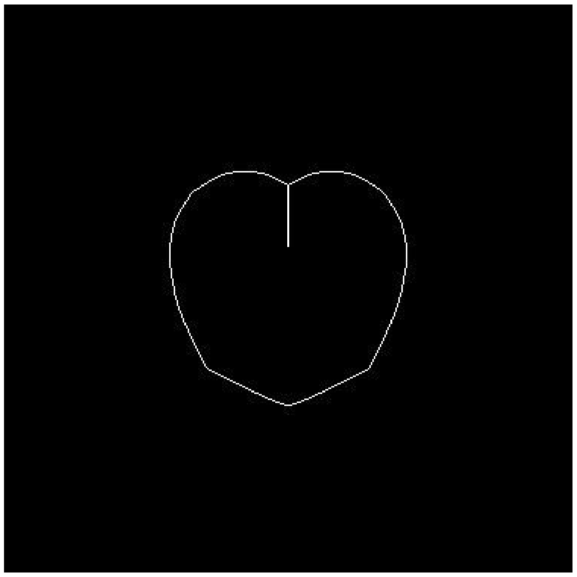
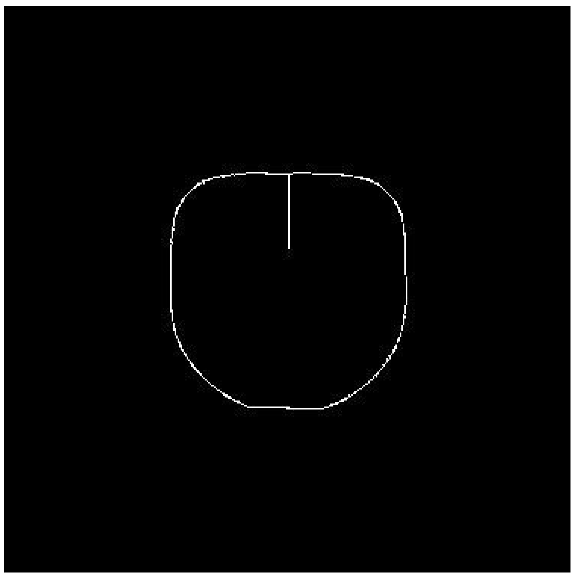


Skeletonizing

Pattern 4:

Original **Shrinking**

Thinning **Skeletonizing**

Discussion:
A single 3x3 mask does not give desired shrinking. A 5x5 can be used. But it is too computationally expensive. Hence we use the algorithm above with one 3x3 mask followed by another 3x3 mask. Masks are designed so that some prevent shrinking of a single pixel into to being erased completely while some break the connectivity to shrink the image. Same works for thinning and skeletonizing.

From the above results we can observe:

Shrinking of an image without holes leads to a single dot. The location of dot depends on centroid of the image. ie. centre of mass. It's usually at or around the centroid. Hence, for pattern 1,2,3 we see dots. 1,2 are perfectly symmetrical in all dimensions. Hence, dot is in centre. Pattern 3 is symmetrical along y axis. Thus shrinking lies on y axis but x coefficient is not in centre due to asymmetry. Shrinking of image with holes shrinks the image to a connected ring which lies in the midway of nearest hole and nearest boundary.

Thinning of image without holes leads to minimally connected stroke which are midway from its nearest boundaries. With holes leads to a minimally connected ring which lies in the midway of nearest hole and nearest boundary.

We use different masks like Corner cluster, Tee branch, Vee branch, Diagonal branch. These masks lead to stick like or skeleton like image with different structures. Eg. v-like(pattern 3), T-like(pattern 4) ,etc. While erosion in this technique continuity of the skeleton is not always observed. Hence, we do bridging to restore the connectivity lost while erosion for skeletonizing. Unlike shrinking and thinning, boundaries of image can be restored from skeletonized output.

(b) Defect Detection and Correction

Motivation:
Digital Images may have some defects. These can be noise introduced due to any Image Processing techniques. We have to remove these noises to acquire a clean image. Following is one of the simplest approach to remove point defects.

Approach:

1. Load the image.
2. If the image is not binarized, binarize it to 0 and 255.
3. Make a 3x3 mask with all white(255) and centre pixel as dark(0)
4. Check if the mask hits of every pixel in input image, when it does, increase count of defect and replace it by 255. Save this location as location with defected pixel.

Experimental Results:

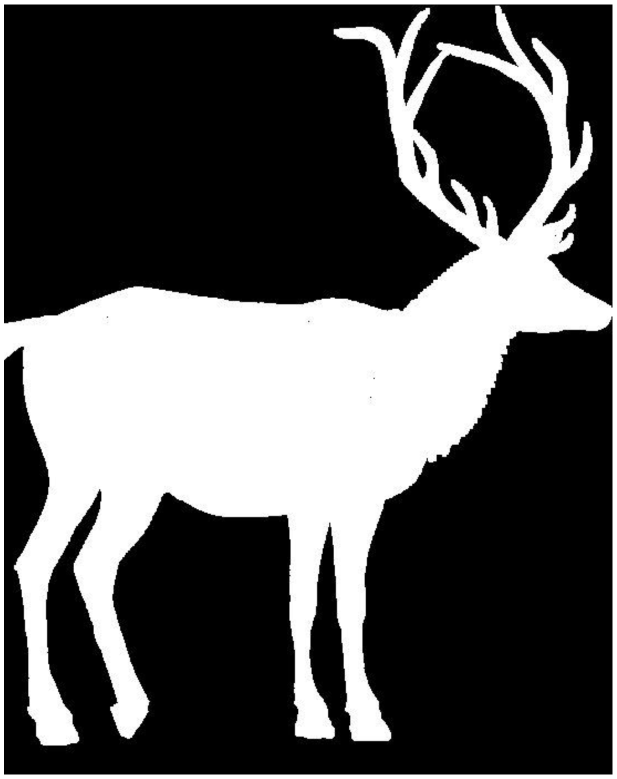


Image with defects

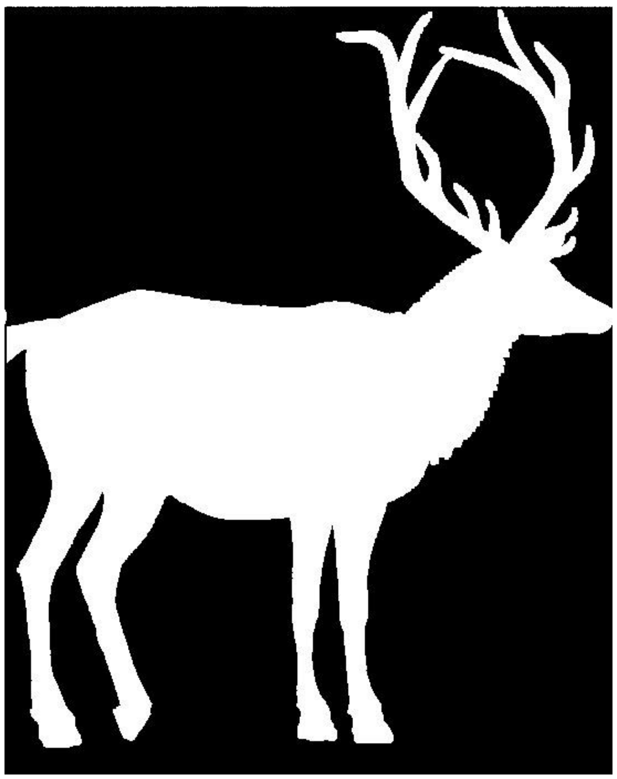


Image without defects

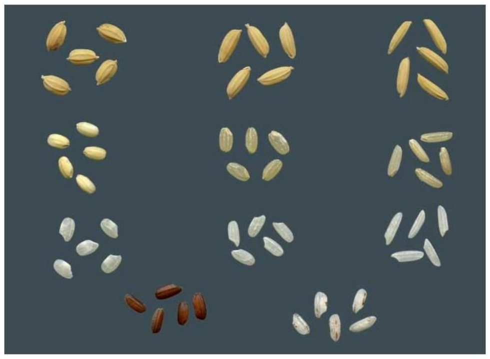
Discussion:
No of defects in above image= 5
In the given figure, we can see that all defects are point defects. Hence, 3x3 marks perfectly well to remove defects from this image. It is a very simple method which gives us output without defects. For a different problem with defects other than single dots, other detect detection methods can be used. Advanced methods like clustering and image segmentation need to be used for complex defects.

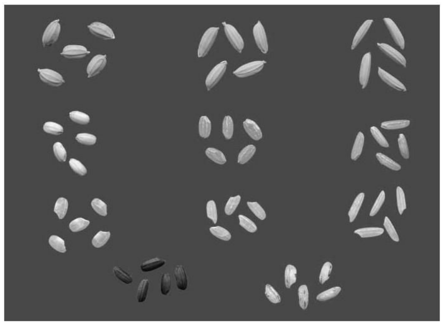
(c) Object Analysis

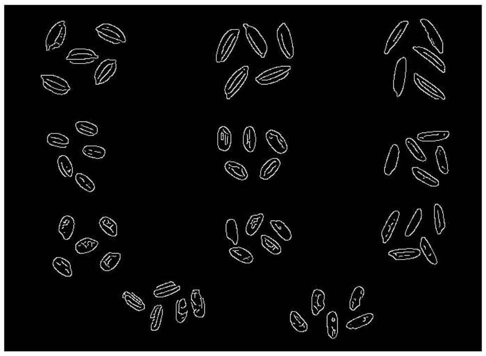
Motivation:
Many applications now a days need object detection and tagging. Eg. Facebook. But first we need to detect the number of objects, and its properties to correctly detect it as a distinct object. In this problem we are doing some analysis on the rice image.

Approach and Experimental Results:

First I converted RGB image to grayscale image. And did the operations in the order shown to get one dot for each grain. All in MATLAB


RGB

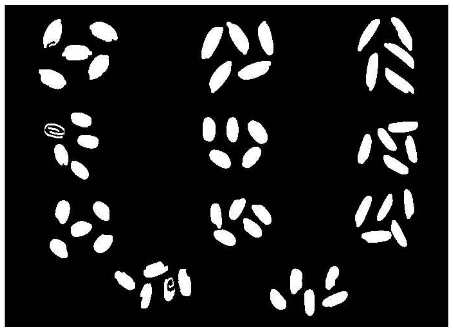

Grayscale

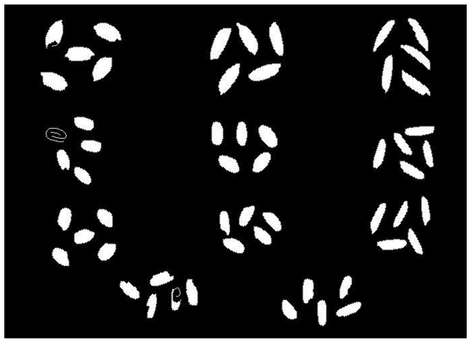

Canny

Bridge

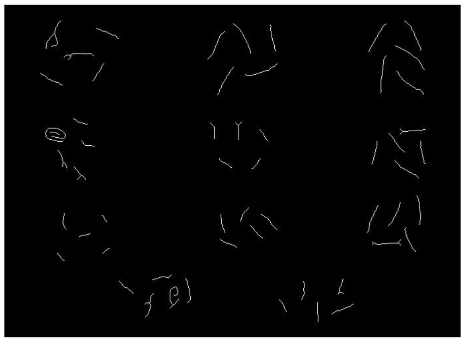
Bridge

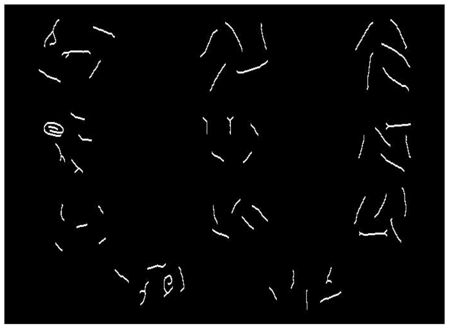
Dilate



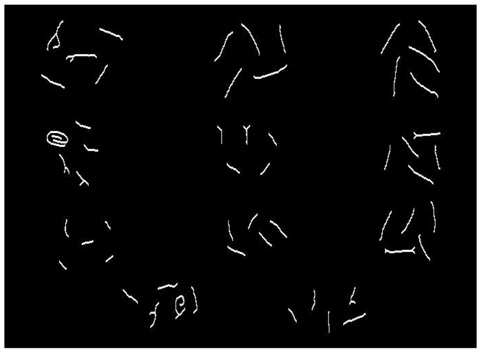


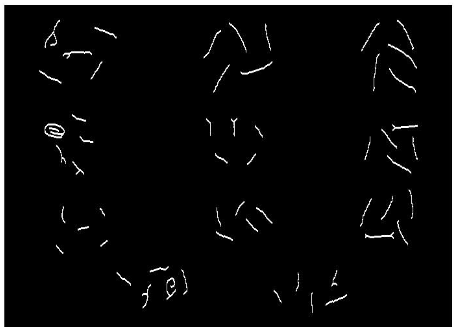
Fill the holes **Shrink**



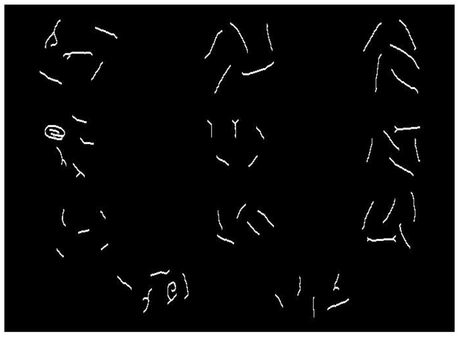


Thin **Dilate**

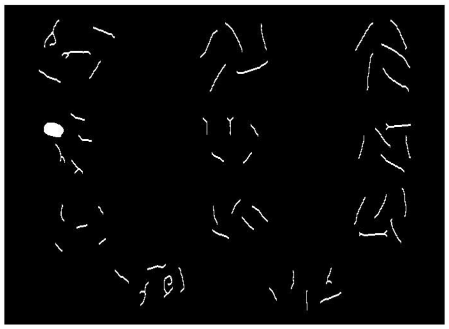




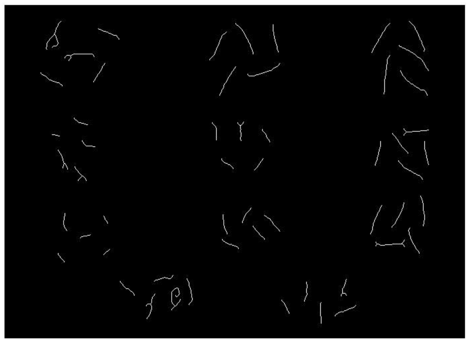
Fill the hole **Bridge**



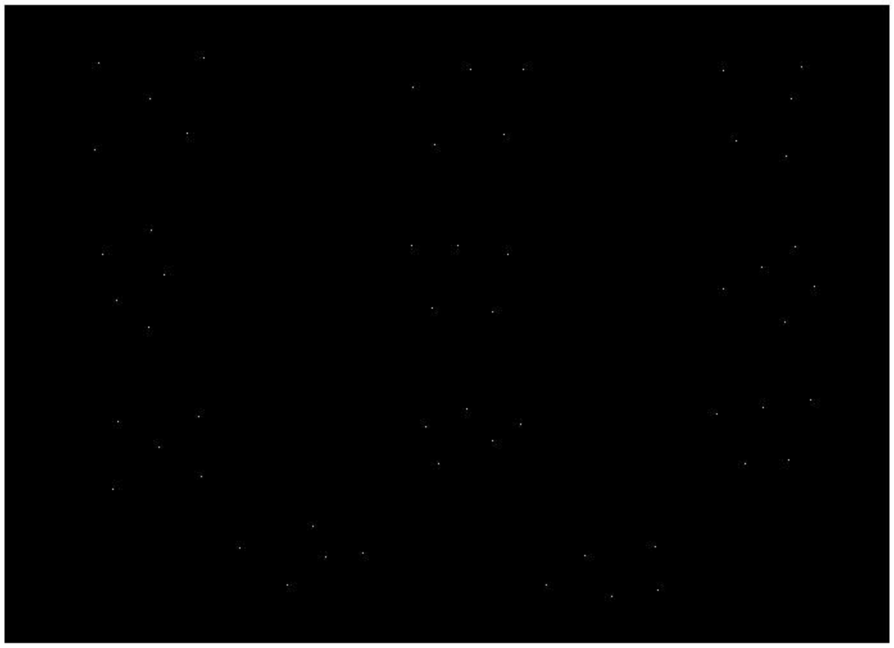
Bridge



Fill the holes



Thin

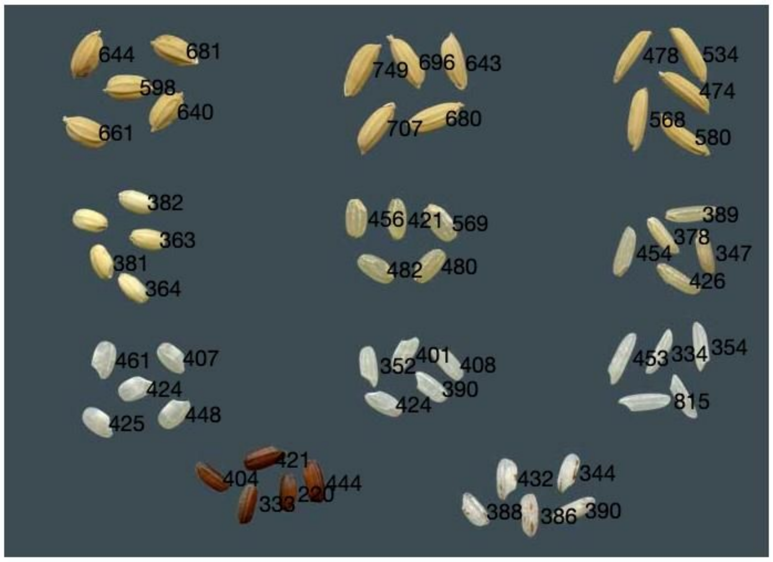


Shrink

Once I found one dot for each grain, I just counted the no of dots for counting the no of grains.

For the ranking of rice grains by size,

1. For the canny output, first determine the area and center of each rice piece.
2. Determine the regions of each type of rice grains from the centre locations above and number them from 1 to 11 from left to right and top to bottom.
3. Find the average rice size of each region
4. Sort them according to ascending order and output the sorted indices.



Area of rice grains

Discussion:
No of rice grains: 55
First after converting the RGB image to grayscale. I did canny to detect edges. Next task to join some edges which had broken in the process. So I did bridging. Now I used Dilate to thicken these boundaries and strengthen the connections done by bridging. Parameters were chosen by ad-hoc method. Now, fill the holes. We can see that two grains didn't get filled. Then I did shrink as the grain boundaries of certain grains were very close to merging. Next, I did thinning for one grain to get converted into one single line. I had to dilate for them for each grain to have one fully connected surface. Next I did filling for the grain which recently became a closed object followed by thinning to reduce the last object to a single line. And ended with shrink, which gave me 55 dots for 55 distinct lines.

Order : [10, 4, 11, 9, 8, 6, 7, 5, 3, 1, 2].
Rice grains have been localized and segregated into 11 regions based on their positions and sizes.

Reference:
Discussions,
https://www.codementor.io/tips/7814203938/resize-an-image-with-bilinear-interpolation-without-imresize
https://en.wikipedia.org/wiki/Image_warping
https://zenodo.org/record/1322720#.XHrQdIhKhPY