EE569 Project 3

Name: Shreyas Dinesh Patil
Email: shreyasp@usc.edu

Problem 1: Geometric modification

(a) Geometric transformation

I. Motivation and Abstract:

- Geometric transformations are necessary if the imaging process suffers from some inherent geometric distortions.
- Sometimes we want to change the shape and size of the image instead of its color or pixel values. Geometric transformations change the image coordinates to create geometrically modified images.
- Geometric transformations transforms image to another by modifying input image coordinates to output image coordinates such that pixel value at u, v in input image is mapped to x, y coordinates in the output image.
- Generally, the geometric transformations of images are carried in the cartesian plane. Therefore, we first convert image coordinates to cartesian coordinates.
- Any geometric transformation can be implemented as forward or backward mapping. In forward mapping new coordinates are calculated for each pixel in the input image and values are copied to the new location. In backward mapping each pixel of the output image is inverse transformed to original image pixel location using inverse mapping function.

II. Approach:

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \boldsymbol{c_x} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{c_y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Translation matrix

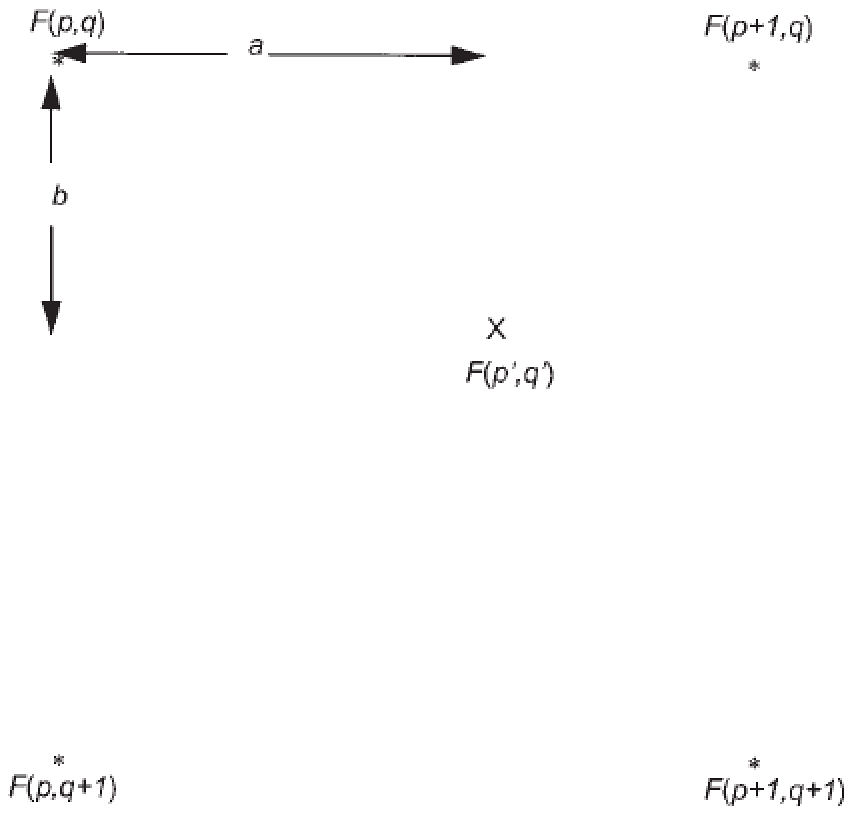
$$\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation matrix

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & -\boldsymbol{c_x} \\ \mathbf{0} & \mathbf{1} & -\boldsymbol{c_y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Translation back matrix

Bilinear Interpolation:
In bilinear interpolation, interpolation is carried linearly along each row of the image and then the result is interpolated along the column direction. The following and equation shows the interpolation,



Source: William K. Pratt: Digital Image Processing, 4th Edition, John Wiley & Sons Inc., 2007. (ISBN 9780471767770).

$F(p', q') = (1 - b)[(1 - a)F(p, q) + aF(p + 1, q)] + b[(1 - a)F(p, q + 1) + aF(p + 1, q + 1)]$
$F(p', q')$ is the estimated pixel value at location (p', q')

Procedure:
1) Find the 4 corner points for each lightouse1, lighthouse2 and lighthouse3 images. This was done by traversing each sub-image from top, bottom, left and right hand sides until non-white pixel was encountered. The encountered points are the 4 corner points of the subimage.
2) The 4 corner points were then converted from image to cartesian coordinates where center of the image was the origin in cartesian coordinate system.
3) The center of the interest region in the image was found by taking the midpoint of any 2 opposite corner points. Angle of rotation theta is the angle between the segment joining right corner point and bottom point with the horizontal axis.
4) Inverse of translation back matrix, is applied to each pixel location of the interest region by first converting the pixel location to cartesian coordinates.

5) Inverse of rotation matrix, is then applied to result of step 4).
6) Inverse of translation matrix, is applied to the result of step 5) to get coordinates of the original sub-image in cartesian coordinates.
7) The cartesian coordinates are converted back to image coordinates.
8) Nearest-neighbor interpolation was used after step 7 to floating point coordinates.
9) Bilinear Interpolation method is used after resizing the image to 160x160 to assign pixel values to floating point coordinates after scaling.
10) The image obtained is scaled to 160x160 and bilinear interpolation is used then.
11) Now, the corner points of the white patches are found using the masks like,

$$\begin{bmatrix} x & x & x \\ x & 255 & 255 \\ x & 255 & 255 \end{bmatrix} \begin{bmatrix} x & x & x \\ 255 & 255 & x \\ 255 & 255 & x \end{bmatrix} \begin{bmatrix} 255 & 255 & x \\ 255 & 255 & x \\ x & x & x \end{bmatrix} \begin{bmatrix} x & 255 & 255 \\ x & 255 & 255 \\ x & x & x \end{bmatrix}$$

Here x=don’t care

12) After getting corner points of the white patches the resized sub-images are filled in it.

III. Experimental results:

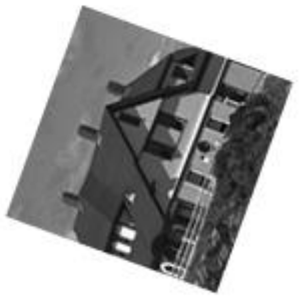


Figure1: Given lighthouse1.raw image

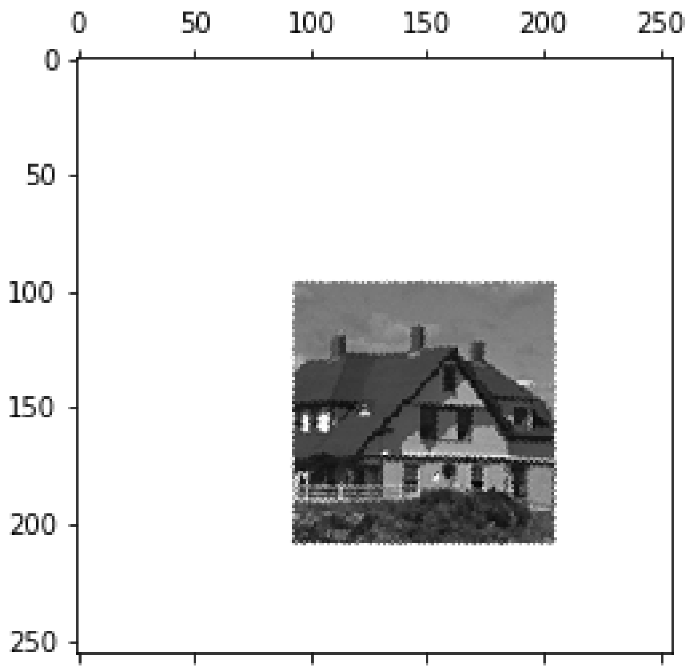


Figure2: After applying inverse translation, inverse rotation and inverse translation back transformations to the given lighthouse1.raw image

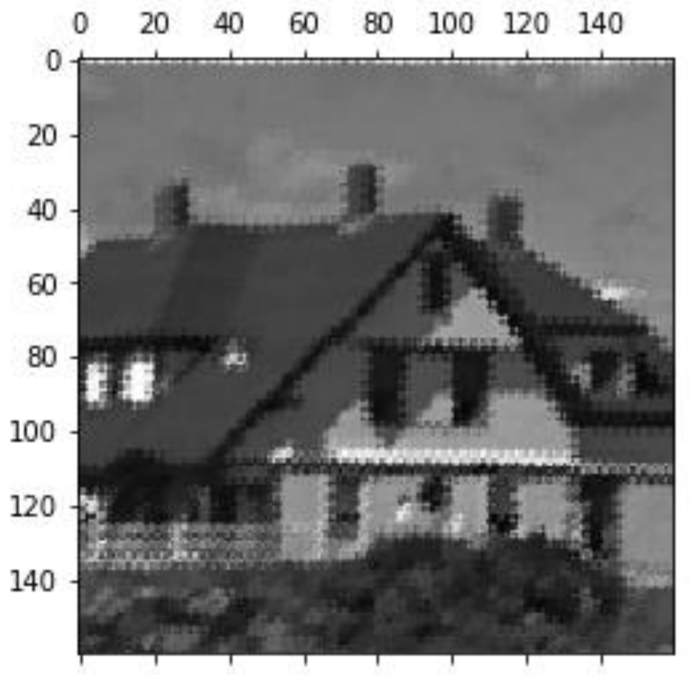


Figure3: After applying interpolation to resize above image to 160x160

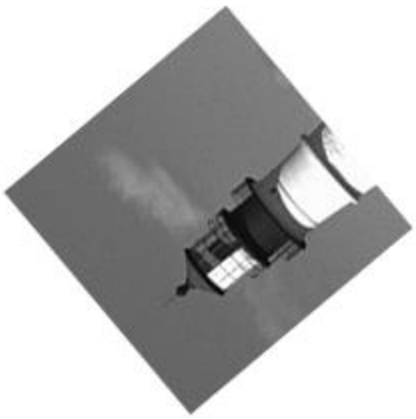


Figure4: Given lighthouse2.raw image

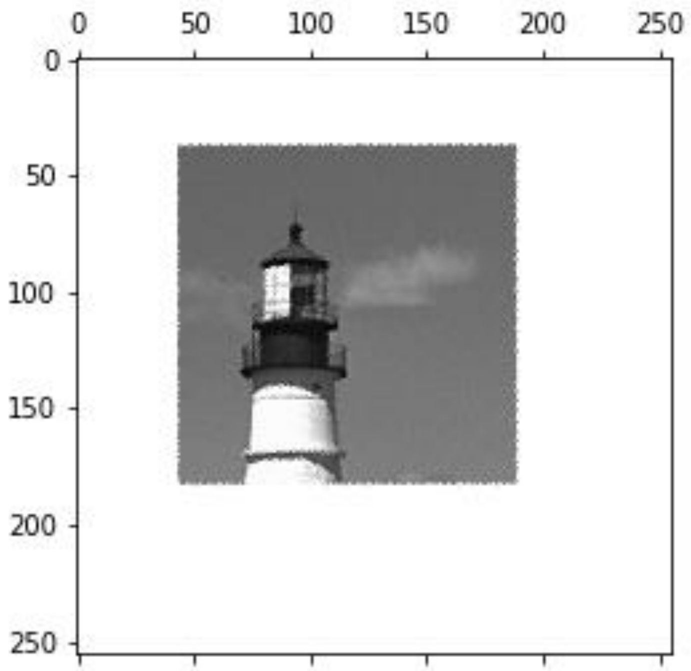


Figure5: After applying inverse translation, inverse rotation and inverse translation back transformations to the given lighthouse2.raw image

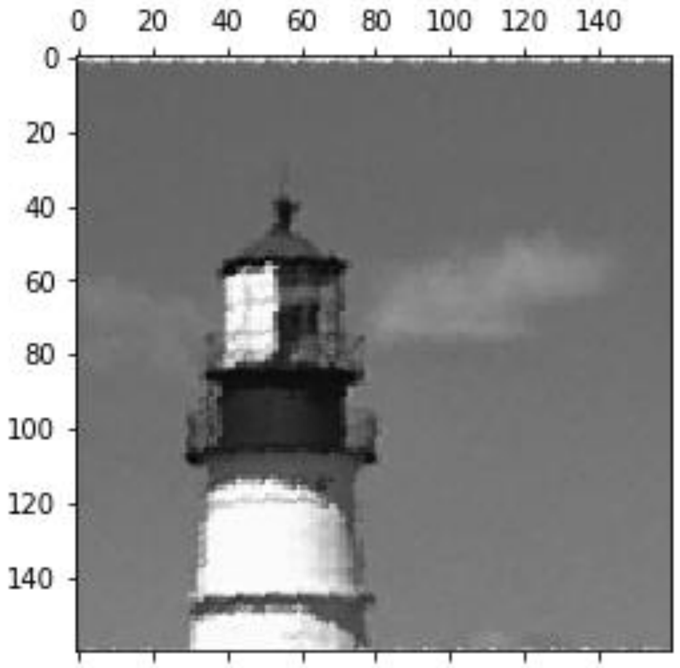


Figure6: After applying interpolation to resize above image to 160x160

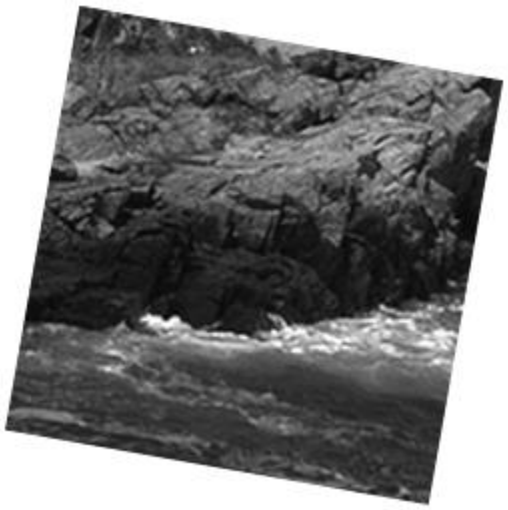


Figure7: Given lighthouse3.raw image

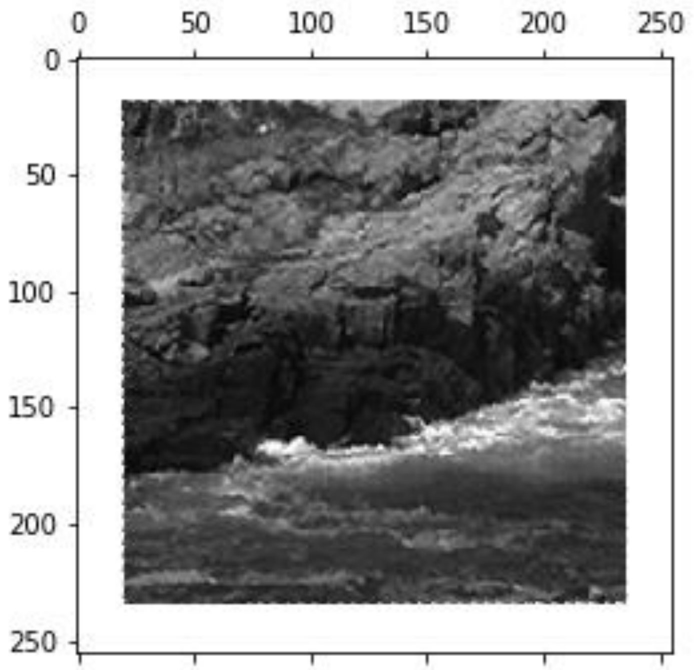


Figure8: After applying inverse translation, inverse rotation and inverse translation back transformations to the given lighthouse3.raw image

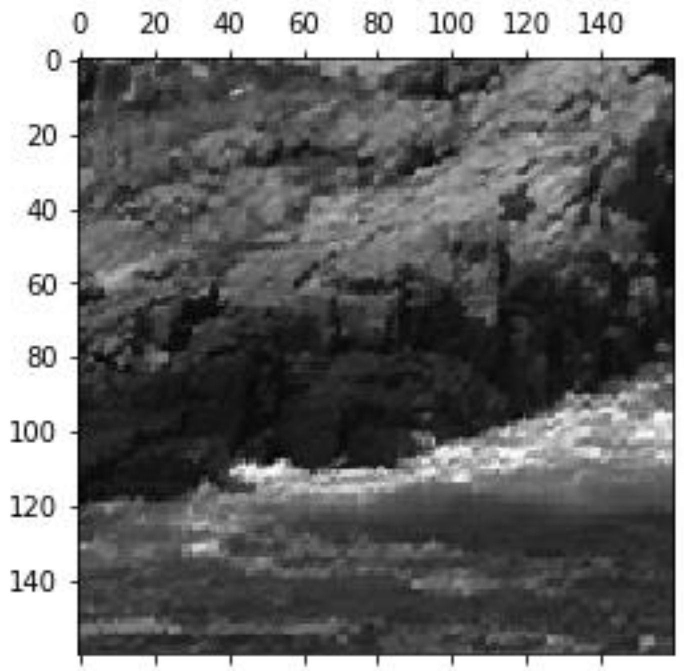


Figure9: After applying interpolation to resize above image to 160x160

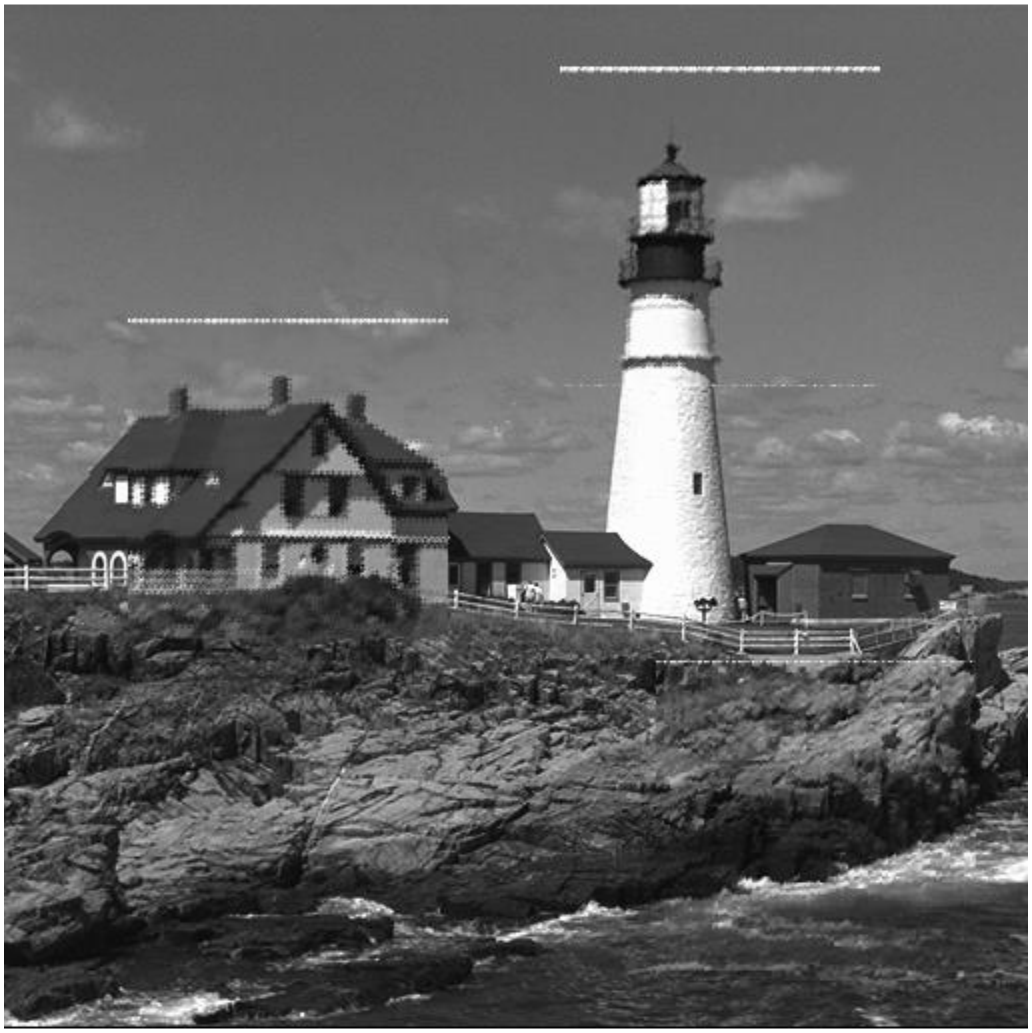


Figure10: Final lighthouse image

III. Discussion

- The experiment results are obtained using reverse address mapping technique for geometric transformation. Here the coordinates of output image are calculated using input image coordinates using matrix operations performing translation, rotation and scaling.
- The obtained input image coordinates can be float, therefore bilinear interpolation technique is used to approximate pixel values for that coordinate.
- Forward technique can also be used, but it has the disadvantage that if an input coordinate value is mapped to float value, then such a pixel cannot be approximated by bilinear interpolation as there are no neighboring pixel known in the output image.

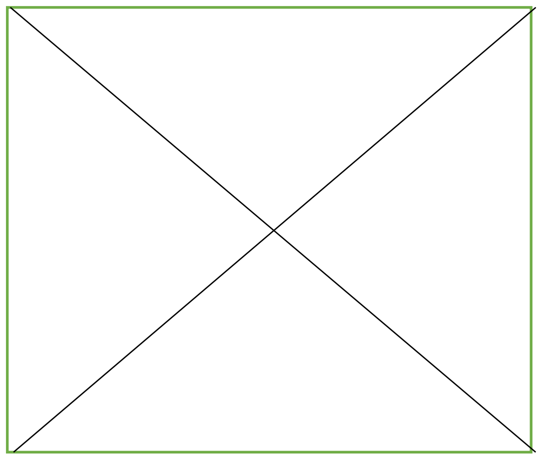
(b) Spatial Warping

I. Motivation and abstract:

Warping of images is primarily used for lens distortion correction in camera or to create distorted and creative images. Image can be warped using forward or reverse address mapping. In this approach I have used reverse address mapping.

II. Approach

1. The image is conceptually divided into 4 triangles as follows,



2. The image coordinates are then converted to cartesian coordinates where center of the image corresponds to (0, 0) in cartesian plane. The conversion formula is,
   X = I – 255, Y = 255 – J where (I, J) are the image coordinates.
3. The six corner points considered for each triangle are the vertices of it and midpoints of the edges. For warped image 5 points are same as original image corner points only one is on the curve at height of 128 from the side of the triangle as shown.
4. The coefficients of the curves for each triangle are found by reverse 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 the u,v coordinates corresponds to the input image corner points and x, y correspond to warped image corner points.

5. After obtaining the 6 coefficients for each of the 4 triangles coordinates reverse address mapping is used to obtain warped coordinates using,

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

6. Coordinates obtained from step 5. are converted back to image coordinates using the equations,
   i = u+255
   j = 255-v
7. The warped coordinate value at i, j are obtained by interpolating obtained coordinates u, v from step 6.

III. Experimental results

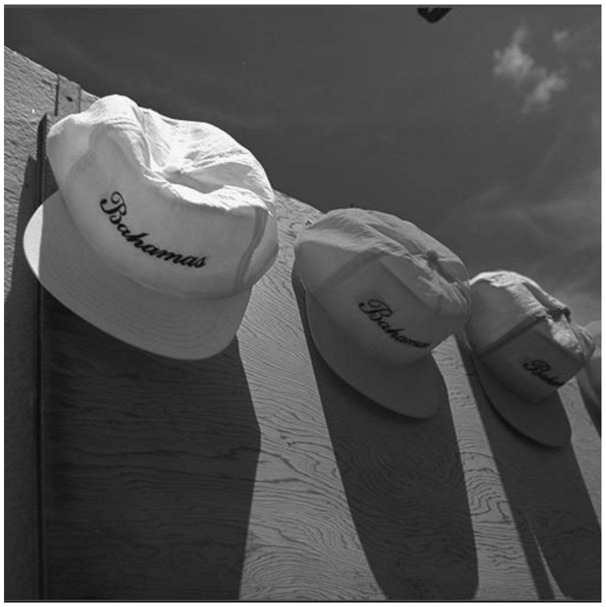


Figure1: hat.raw

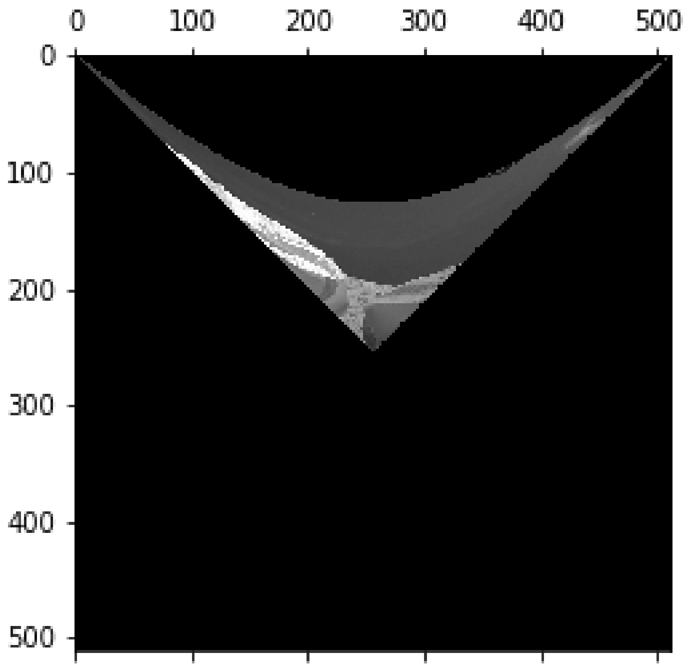


Figure2: top triangle part warped

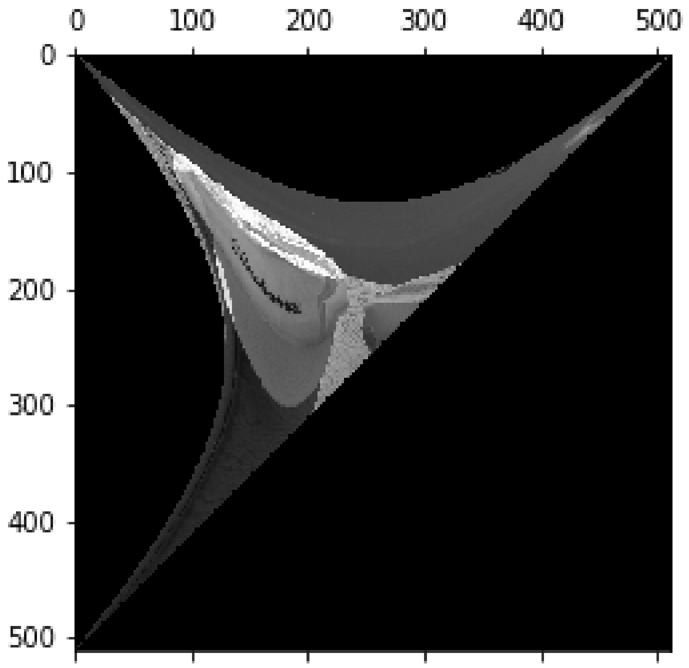


Figure3: top and left triangle part warped

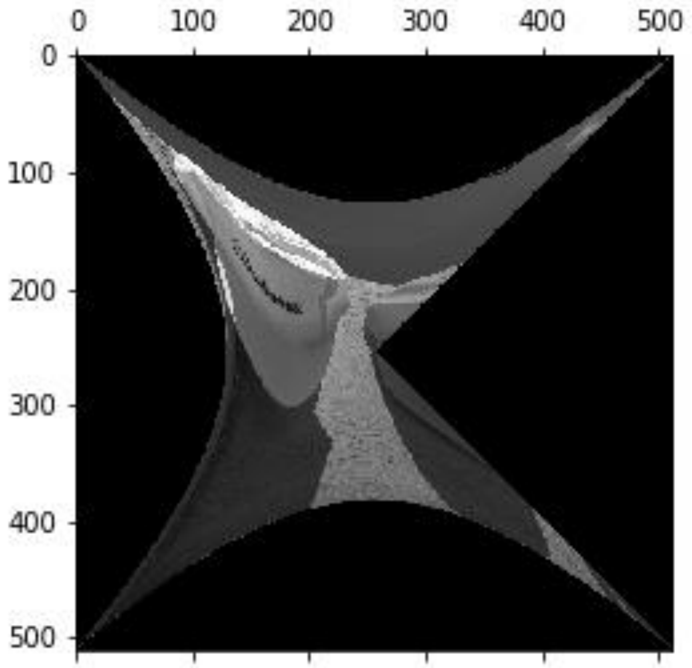


Figure4: top, left and bottom triangle part warped

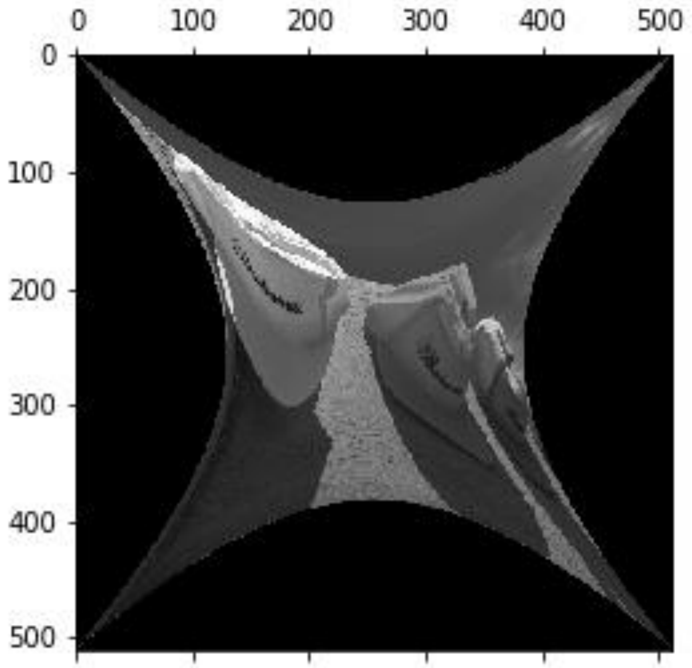


Figure5: warped image

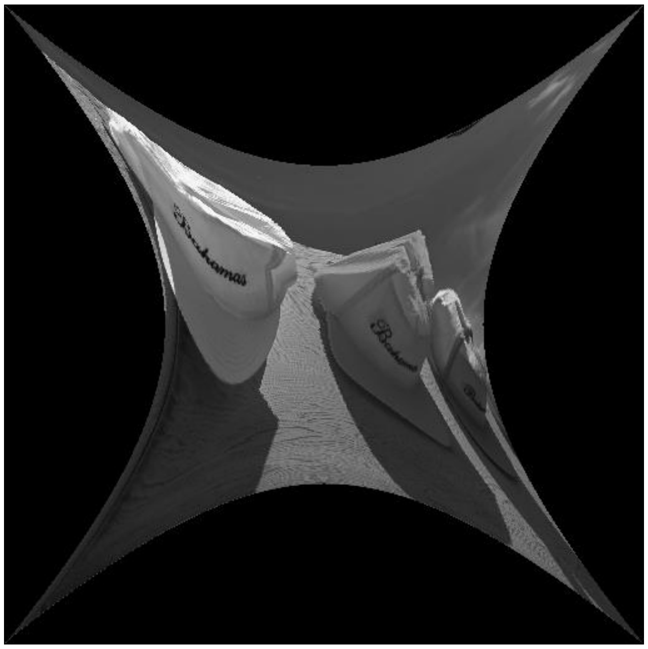


Figure6: warped image.raw

IV. Discussion

- The warping of the image has caused sharpening of the image edges at the cost of center of image.
- Here reverse mapping was used for warping instead of forward, because in the forward mapping pixels from the source image may be mapped to positions beyond the boundary of the image. Also, many pixels mapped to the destination may not have any values assigned creating holes in the image. This is the disadvantage of using forward mapping.

(c) Lens distortion and correction:

I. Abstract and motivation:

Distortion in images can occur in many different patterns. Radial distortion often happens when an image is captured on a non-flat camera's focal plane. Radial distortions are usually classed into 2: pincushion and barrel. In this experiment the image given has barrel distortion. Distortion in images due to camera lenses is dependent on radial distortion coefficients which are camera specific. In this approach I have removed the distortion in the given image by backward mapping the coordinate in the undistorted image to the coordinate in the distorted image.

II. Approach

1) The given image coordinates are converted into camera coordinates using the relation,
   $x = (u - uc)/fx$ and $y = (v - vc)/fy$
   Where (uc, vc) is the center of the image and $fx = fy = 600$
2) For each coordinate (x, y) of the undistorted image
   1) Change the coordinates from camera to image.
   2) Calculate its distorted coordinates using,
      x_d = x*(1 + K1*(x*x + y*y) + K2*(x*x + y*y) **2)
      y_d = y*(1 + K1*(x*x + y*y) + K2*(x*x + y*y) **2)
      where x, y are the undistorted coordinates and x_d, y_d are distorted.
   3) Use bilinear interpolation to find the pixel value of the coordinate obtained in 3). Clip the coordinate values out of grid
   4) Assign the result to the point (x, y) in the undistorted image.

III. Experimental results

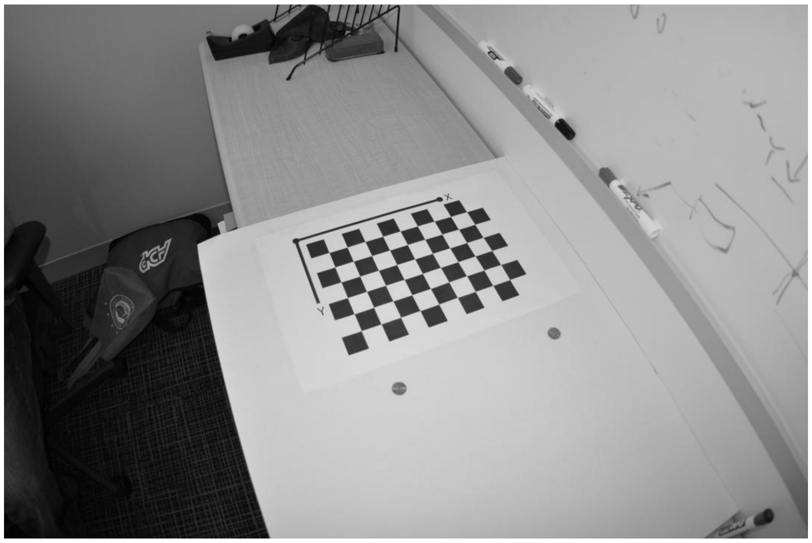


Figure1: classroom.raw

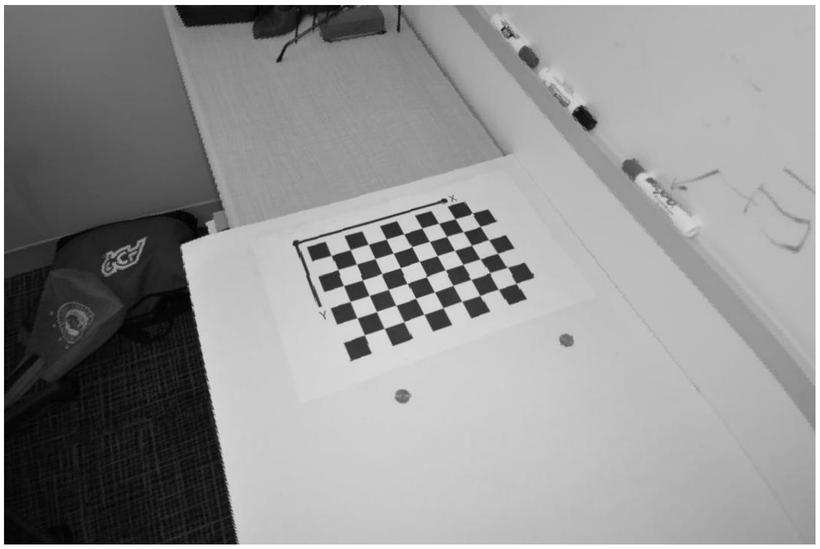


Figure2: undistorted image

IV. Discussion

- The result I have got is by reverse mapping the undistorted image coordinate to distorted image coordinate. This maintains the known values on the grid and helps for interpolation.
- As we can see the distortion of white chart containing chessboard pattern is removed and now it appears straight. Similarly, the white board bottom edge has also been approximately straightened.
- The disadvantage of image distortion correction is that the edges of the image have been cropped out. This can influence the composition of image.
- Distortion correction has redistributed the image’s resolution. Now the center of the image appears slightly sharper than the edges.

Problem 2: Morphological Processing

(a) Basic Morphological Process Implementation

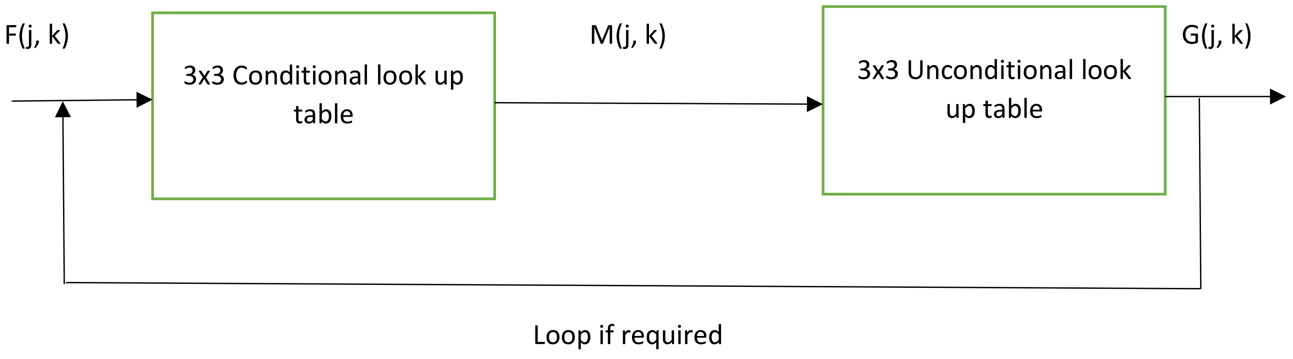
I. Abstract

Morphological operators take an image and structuring element and combine them using set operators to process the shape of the objects. Here I have implemented shrinking, thinning and skeletonizing morphological operators on 4 different patterns.
Shrinking, thinning and skeletonizing are forms of conditional erosion where the erosion is controlled to prevent total erasure and ensure connectivity. These morphological operations are implemented using hit-or-miss transforms. In hit-or-miss transform an odd size mask like 3x3 is scanned through the image and whenever pixels under mask match, the mask is said to hit the center pixel location. The center pixel value is then changed to desired value and remains unchanged when the mask misses.
In this approach I have used conditional and unconditional hit-or-miss masks for shrinking, thinning and skeletonizing.

(1) Shrinking, thinning, Skeletonizing

II. Approach:

It is implemented in 2 stages


F(j, k)
3x3 Conditional look up table
M(j, k)
3x3 Unconditional look up table
G(j, k)
Loop if required

Here F or X – input image

M – image array with hits on conditional look up table masks

G – output image of morphological process

P-image array with hits of M on unconditional masks

Approach: (Shrinking and thinning)

1) Write conditional and unconditional look up tables for shrinking and thinning.
2) Apply each conditional mask on the extended input image and see if the 3x3 input image region matches any of the 3x3 masks in the conditional masks. If there is a hit then the center pixel in the input image is updated into array M with value 1.

3) After iterating through entire input image and getting M. M's boundaries are extended and every 3x3 pixel region is checked against unconditional masks. If there is a hit then the center pixel value 1 is stored in P.
4) The final pixel values in the output image are calculated using the logical expression, $G(j,k)=X\cap[\bar{M}\cup P(\bar{M},\bar{M}0,\ldots,\bar{M}7)]$
5) This process continues until convergence when there is no more change in the output image (This is implemented using a while loop).

Approach: Skeletonizing

1) Write conditional and unconditional look up tables for skeletonizing.
2) Apply each conditional mask on the extended input image and see if the 3x3 input image region matches any of the 3x3 masks in the conditional masks. If there is a hit then the center pixel in the input image is updated into array M with value 1 else 0.
3) After iterating through entire input image and getting M. M's boundaries are extended and every 3x3 pixel region is checked against unconditional masks. If there is a hit then the center pixel value 1 is stored in P else 0.
4) The final pixel values in the output image are calculated using the logical expression,
$$G(j,k)=X\cap[\bar{M}\cup P(\bar{M},\bar{M}0,\ldots,\bar{M}7)]$$
5) Steps 2-4 are iterated until the algorithm converges and no further change can be observed in the output image. (This is implemented using a while loop).
6) Bridging procedure is applied to every pixel value obtained on result obtained in step 5 using the logical expression,

   $G(j,k)=X\cup$ [P1 ∪ P2 ∪ P3 ∪ P4 ∪ P5 ∪ P6]

   Where,

   L1 = X ∩ X0 ∩ X1 ∩ X2 ∩ X3 ∩ X4 ∩ X5 ∩ X6 ∩ X7
   L2 = X ∩ X0 ∩ X1 ∩ X2 ∩ X3 ∩ X4 ∩ X5 ∩ X6 ∩ X7
   L3 = X ∩ X0 ∩ X1 ∩ X2 ∩ X3 ∩ X4 ∩ X5 ∩ X6 ∩ X7
   L4 = X ∩ X0 ∩ X1 ∩ X2 ∩ X3 ∩ X4 ∩ X5 ∩ X6 ∩ X7

   PQ = L1 ∪ L2 ∪ L3 ∪ L4

   P1 = X2 ∩ X6 ∩ [X3 ∪ X4 ∪ X5] ∩ [X0 ∪ X1 ∪ X7] ∩ PQ;
   P2 = X0 ∩ X4 ∩ [X1 ∪ X2 ∪ X3] ∩ [X5 ∪ X6 ∪ X7] ∩ PQ
   P3 = X0 ∩ X6 ∩ X7 ∩ [X2 ∪ X3 ∪ X4]
   P5 = X2 ∩ X4 ∩ X3 ∩ [X0 ∪ X6 ∪ X7]
   P6 = X4 ∩ X6 ∩ X5 ∩ [X0 ∪ X1 ∪ X2]
7) After applying bridging to every single pixel in G the output is stored in output image array.

III. Experimental Results:

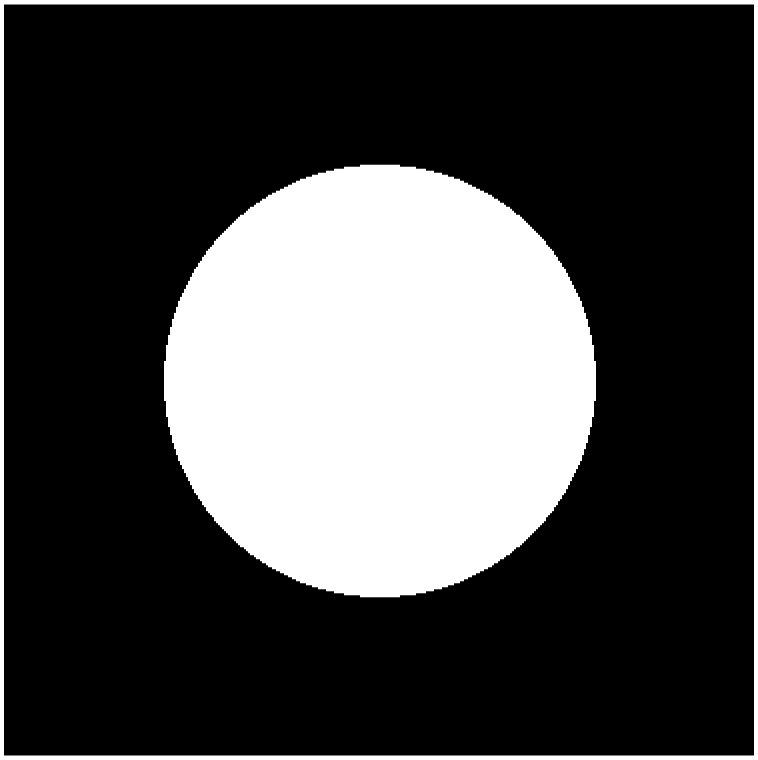


Figure1: pattern1.raw

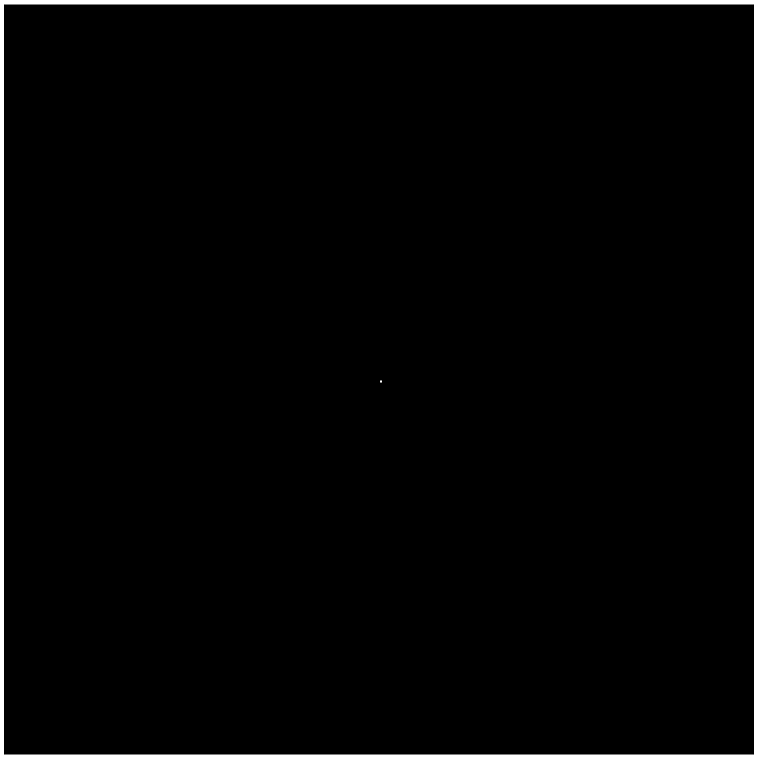


Figure2: shrinking pattern1 output

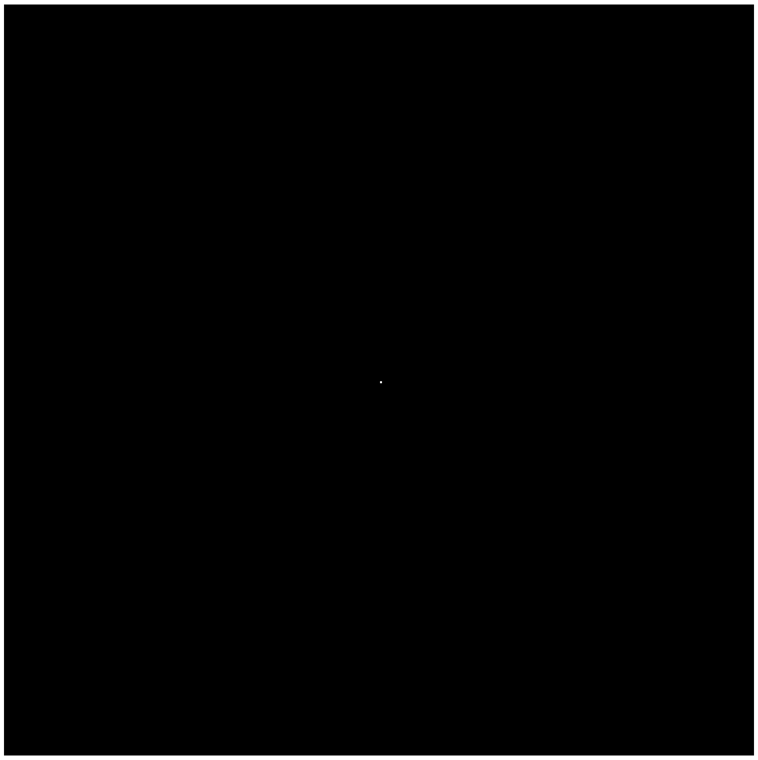


Figure3: thinning pattern1 output

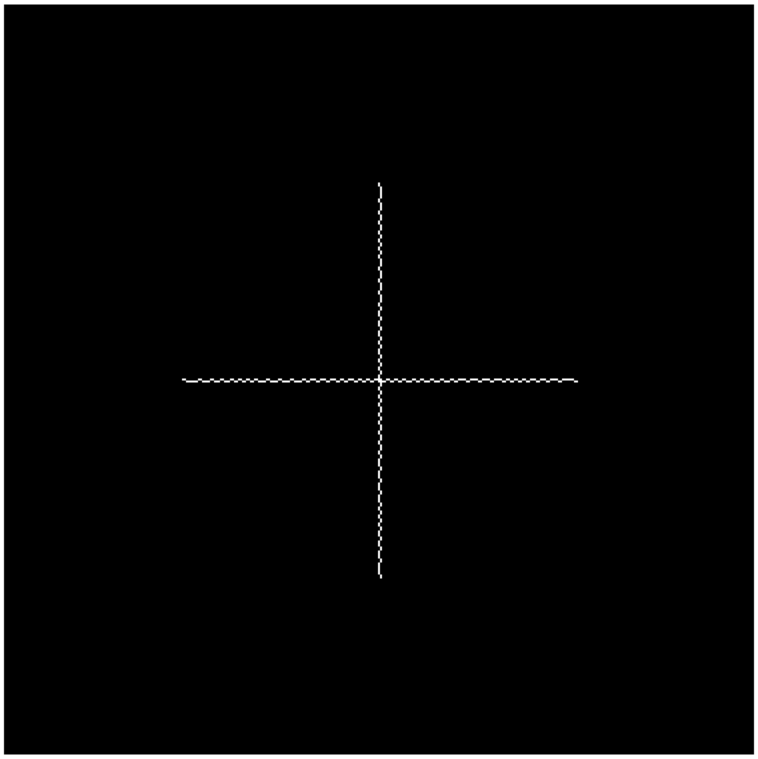


Figure4: skeletonizing pattern1 output

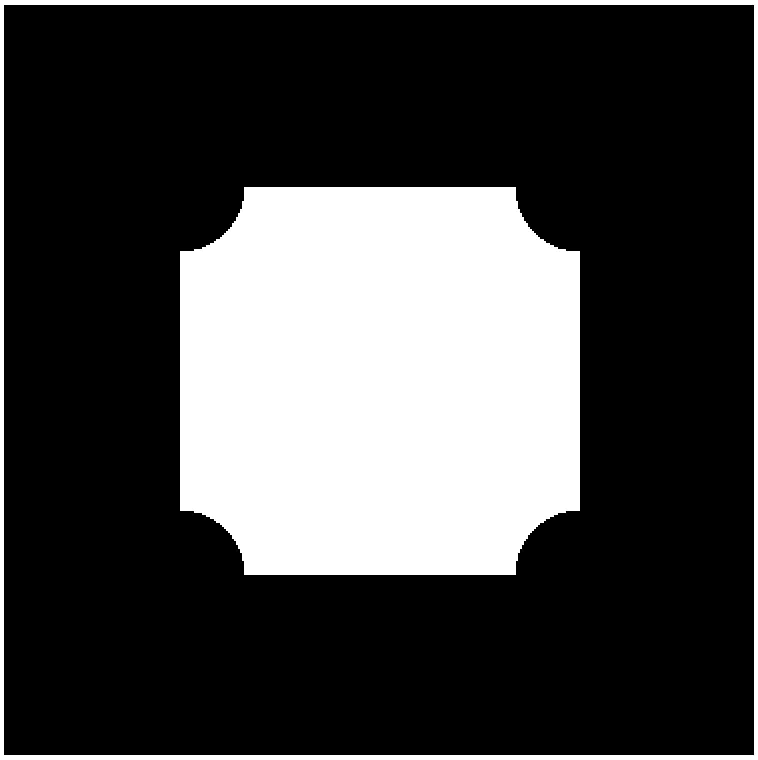


Figure5: pattern2.raw

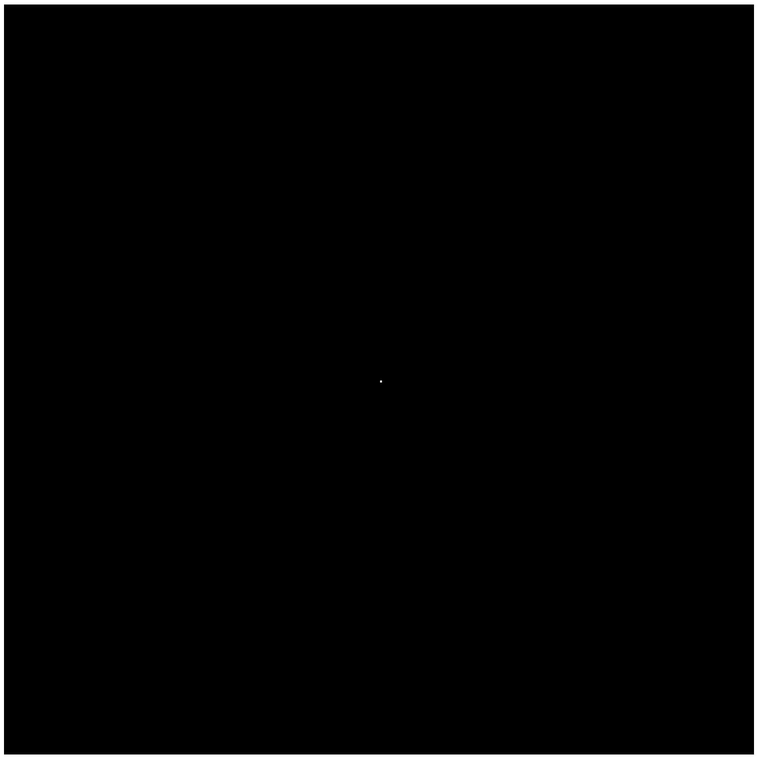


Figure6: shrinking pattern2 output

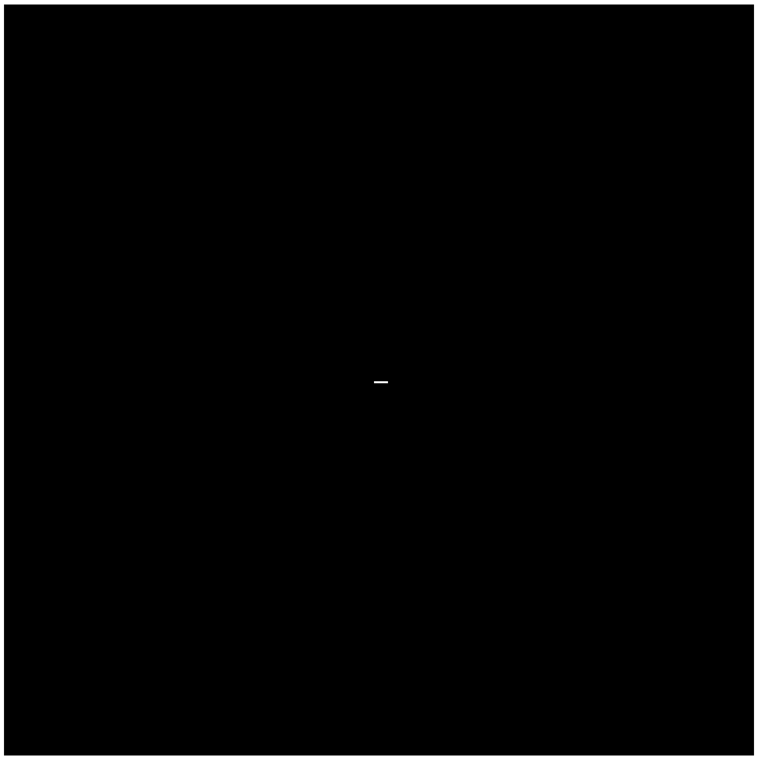


Figure7: thinning pattern2 output

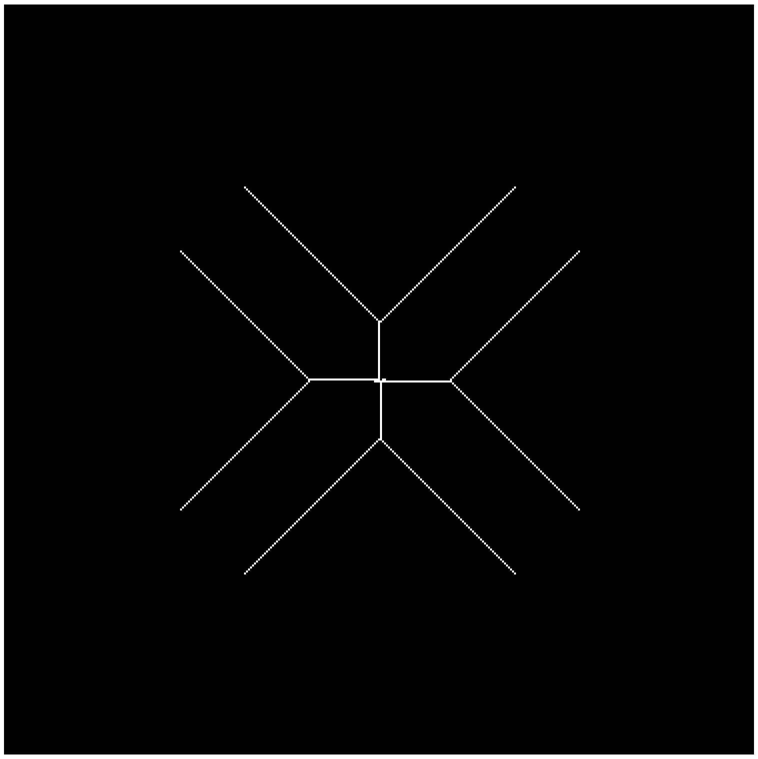


Figure8: skeletonizing pattern2 output

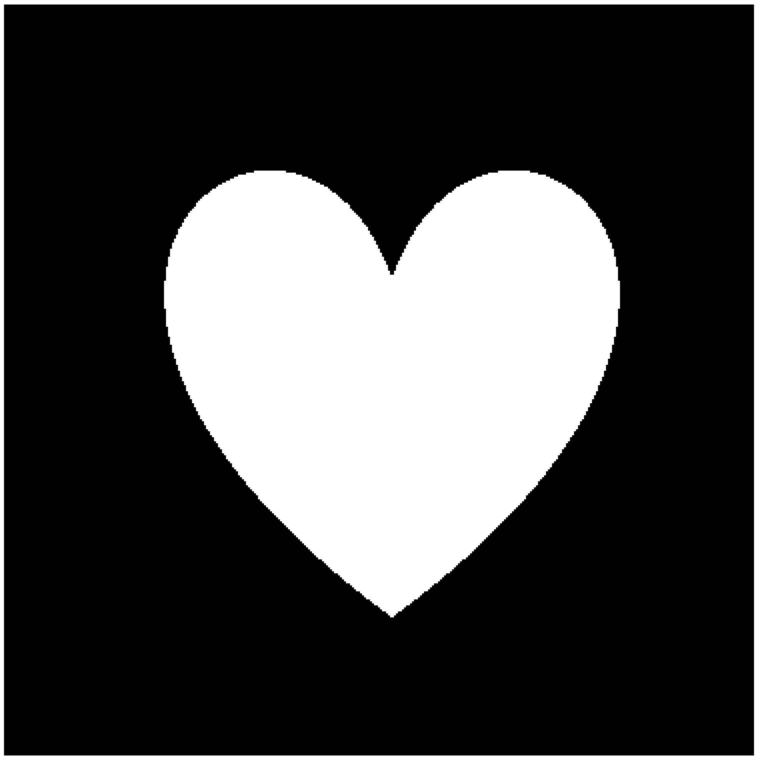


Figure9: pattern3.raw

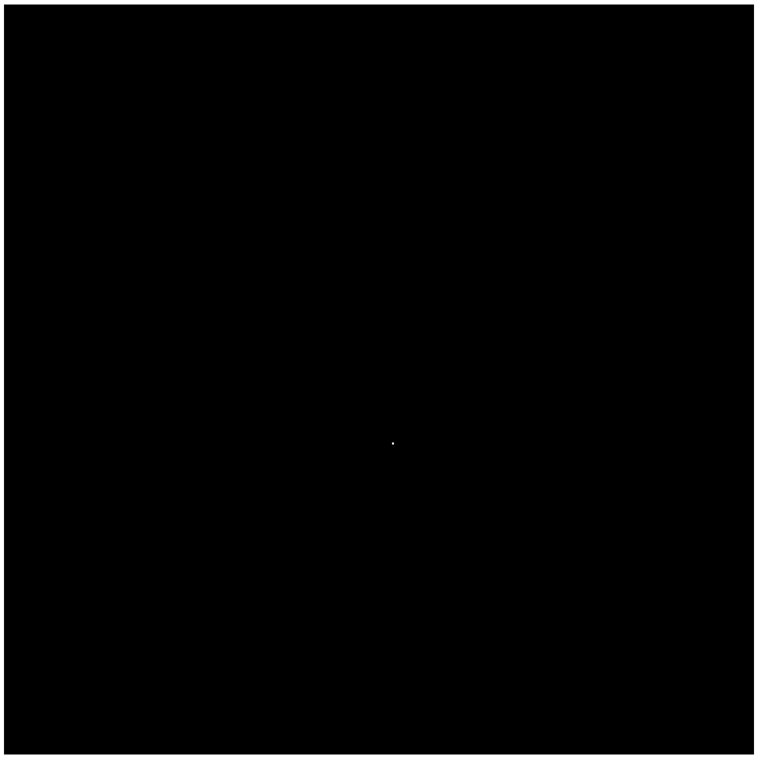


Figure10: shrinking pattern3 output

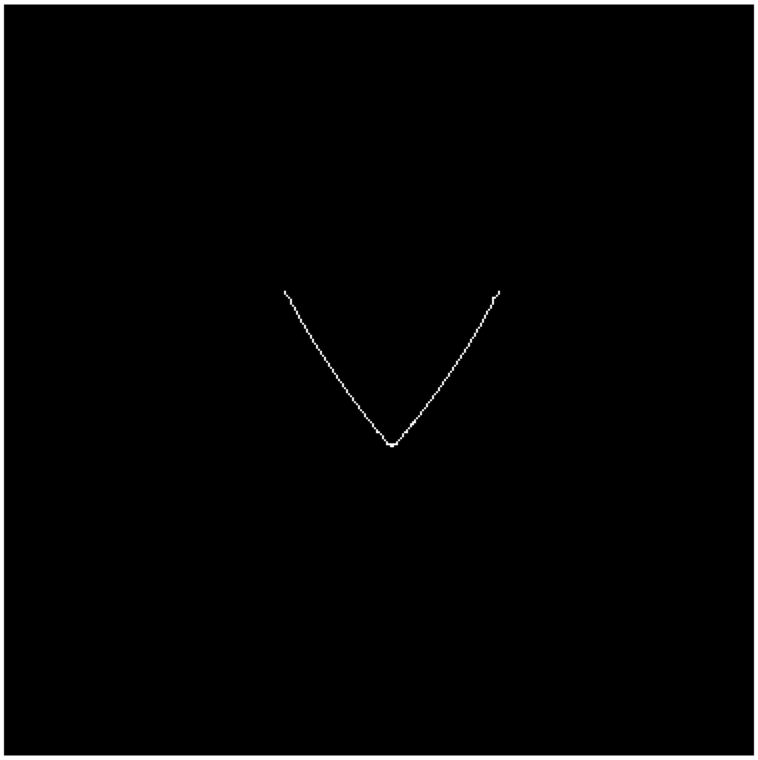


Figure11: thinning pattern3 output

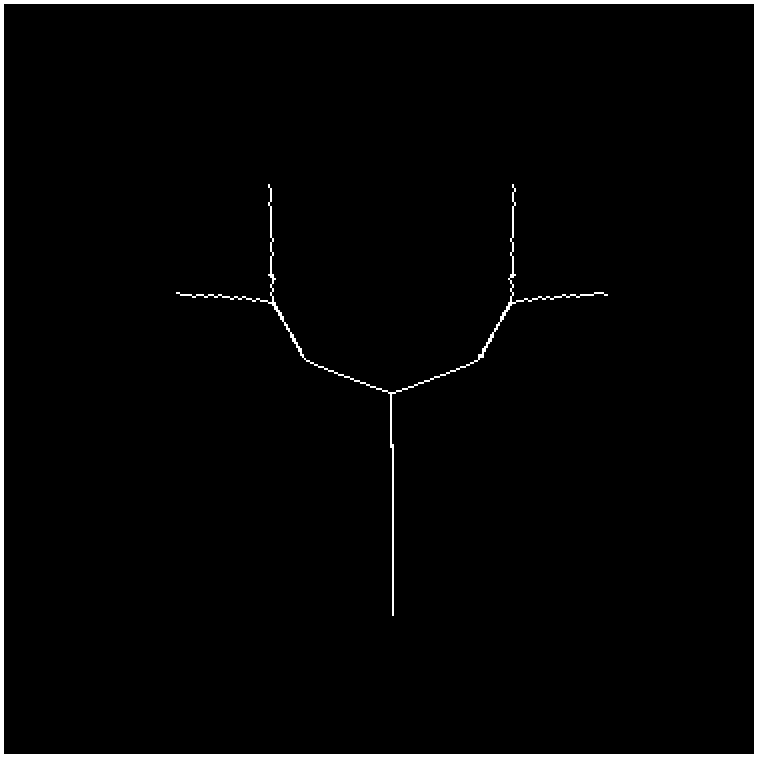


Figure12: skeletonizing pattern3 output

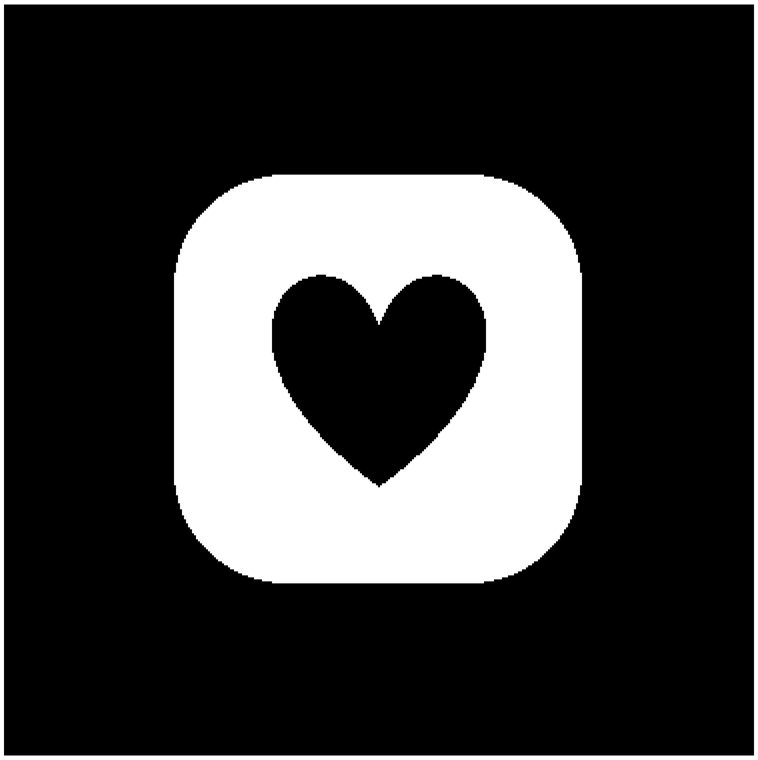


Figure13: pattern4.raw

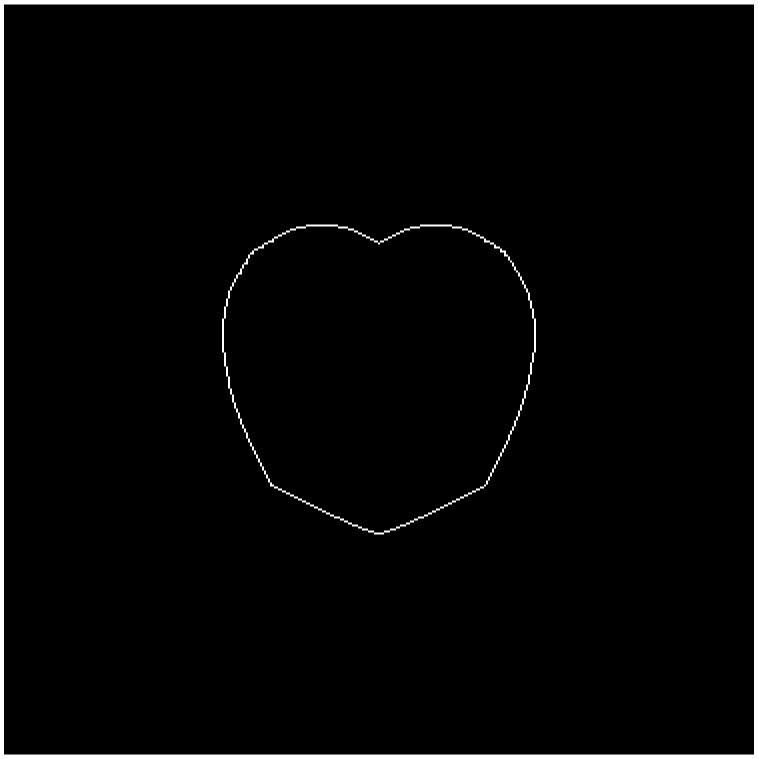


Figure14: shrinking pattern4 output

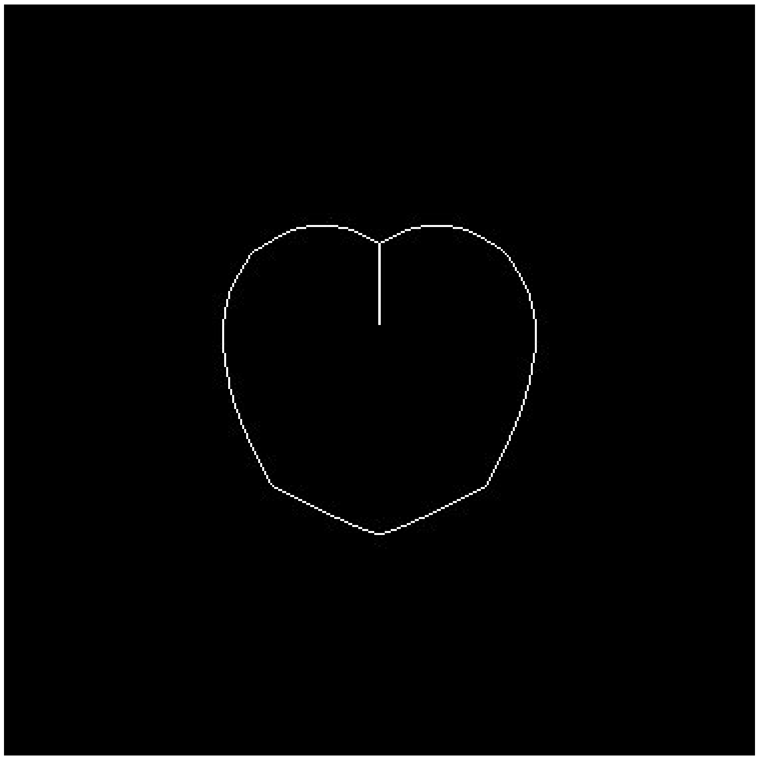


Figure15: thinning pattern4 output

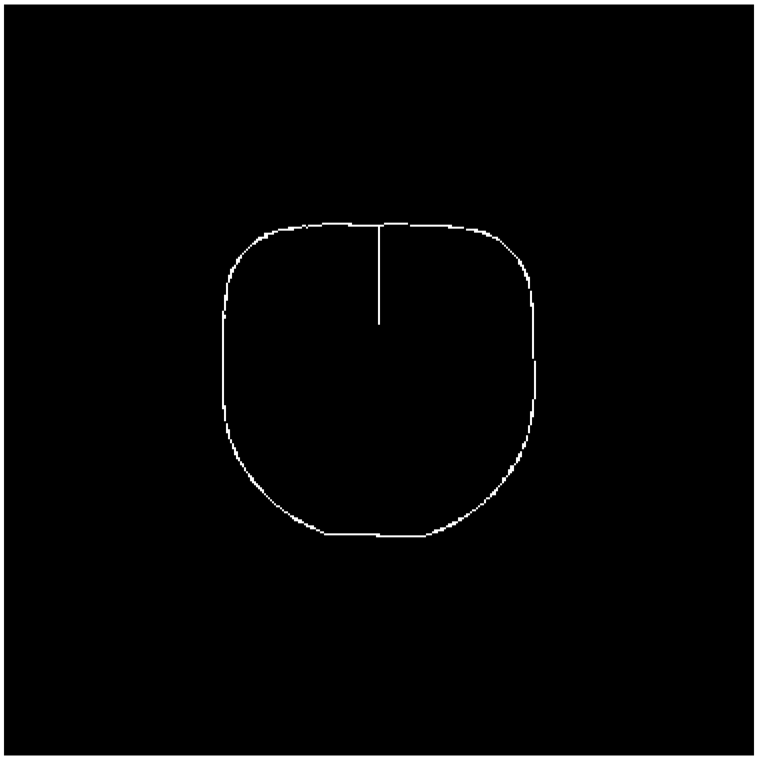


Figure16: skeletonizing pattern4 output

IV. Discussion:

- Skeletonizing is computationally more expensive than thinning and shrinking.
- We can see from skeletonizing outputs the skeleton looks to be well connected. This happens due to the bridging procedure applied after skeletonizing procedure which is a additive morphological operator. Bridging creates white pixel if introducing the white pixel results in connectivity of previously unconnected pixels in the neighborhood.
- For symmetric objects in images shrinking reduces the object to a single point pixel. This happens due to particular structure of conditional masks for shrinking.
- Thinning operation has reduced the objects to single pixel width.

(b) Defect detection and correction

I. Motivation and abstract

Sometimes defects are introduced in the printed materials due to the low precision of the printing machine, material fault during image production and inconsistency between image and the printing machine. These defects can cause bad visual effect on the printed material. From earlier times these defects are identified visually and handled. Here I have removed defects which are black dots on the deer image using isolated pixel remove subtractive operator. Subtractive hit-or-miss operators cause the center pixel of the 3x3 mask to be converted from white to black depending on some predetermined conditions of its neighboring pixels.

II. Approach

X =

$$\begin{bmatrix} 255 & 255 & 255 \\ 255 & 0 & 255 \\ 255 & 255 & 255 \end{bmatrix}$$

1. Extend the input image boundaries by 1 pixel.
2. Filter X was scanned through entire image to identify the location of the dots on the body of the deer.
3. Wherever there was hit the center pixel value was updated with pixel value 255.

III. Experimental results

Pixel locations of defects

Location x-498 y-207
Location x-93 y-280
Location x-275 y-284
Location x-334 y-335
Location x-331 y-352
Total defects 5

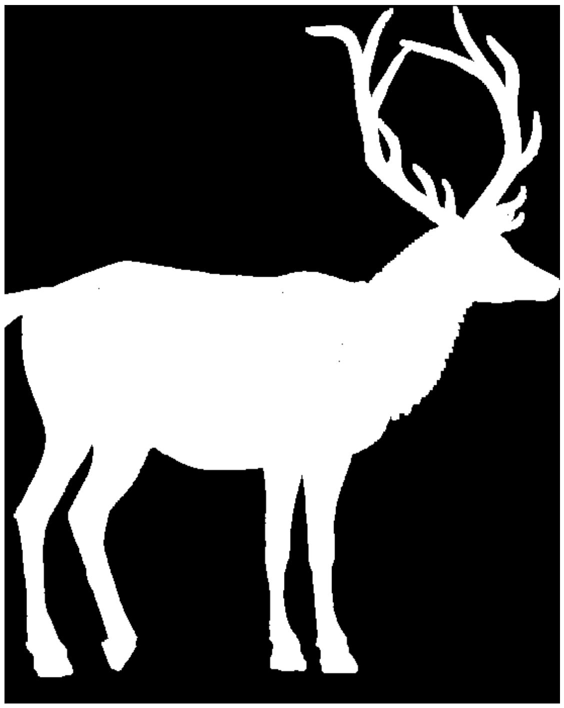


Figure1: deer.raw

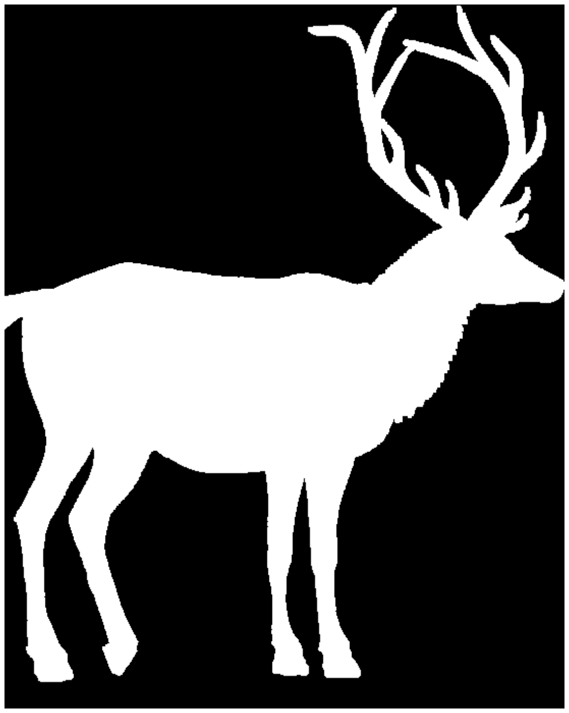


Figure2: Image after removing defects

IV. Discussion

- The image has 5 black spots as can be observed from the above experimental results.
- As the defects were given to be dots, isolated pixel removal morphological operator was used. Other subtractive operators such as spur remove, H-break, eight neighbor erode and interior pixel remove can be used depending on the morphological characteristics of the defect. Also generalized erosion techniques can be used for any defect removal.

(c) Object Analysis

I. Abstract and motivation

Morphological procedures are often used for identifying objects in the image and analyzing their properties. Here I have used morphological operations like dilation, erosion, floodfill along with canny edge detector to determine total number of rice grains, compare their sizes and rank them from small to large in terms of their type.

II. Approach (Matlab inbuilt functions used for the implementation)

1) First convert the given image to gray scale
2) Use canny edge detector with threshold value 0.1 to detect edges of rice grains in the obtained grayscale image.
3) Use dilation operator with mask as,

$$\begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix}$$

4) Floodfill the result obtained from 3) with holes.
5) Use erode operation on the image obtained from 4)
6) Shrinking operation was used on eroded output to get dots corresponding to each grains.
7) Find area of each grain on the eroded output (“regionprops” matlab function was used to get this).
8) Average area of each rice grain type is calculated by dividing the image into 11 regions each belonging to one type starting from top left corner and traversing horizontally.
9) These averages were sorted in ascending order to get grain size from small to large in terms of type.

III. Experimental results

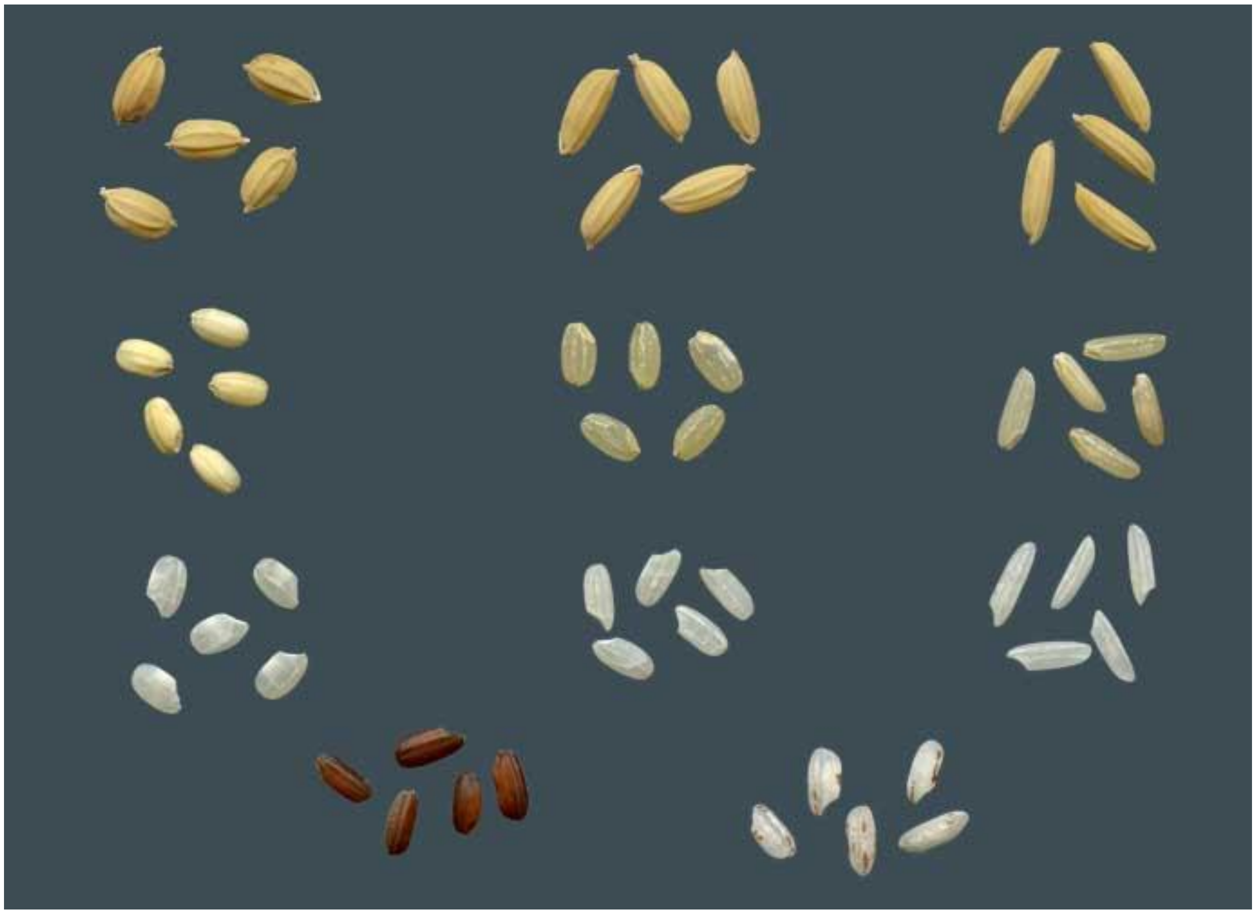


Figure1: rice.raw

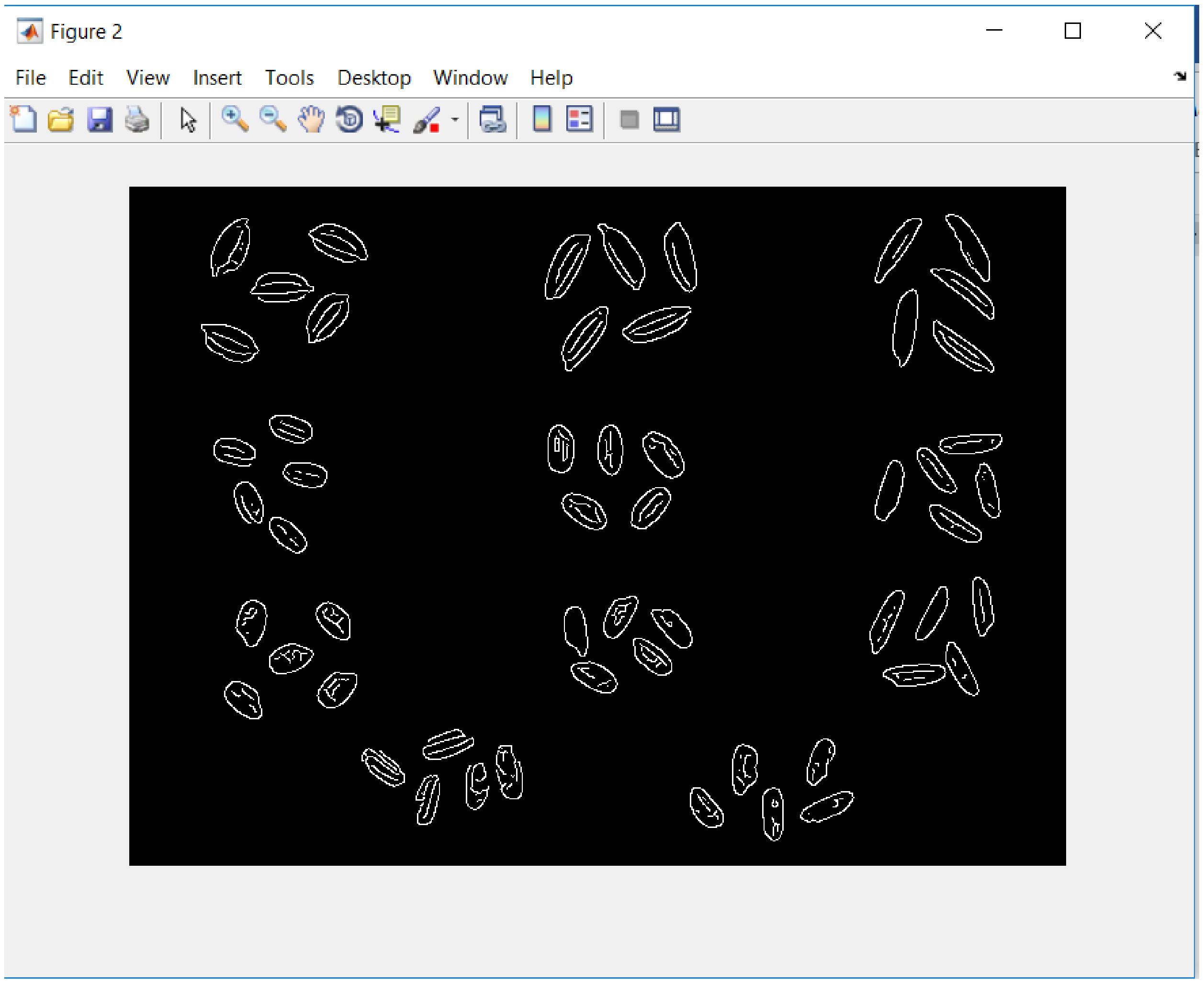


Figure2: canny edge detector output with high threshold as 0.1

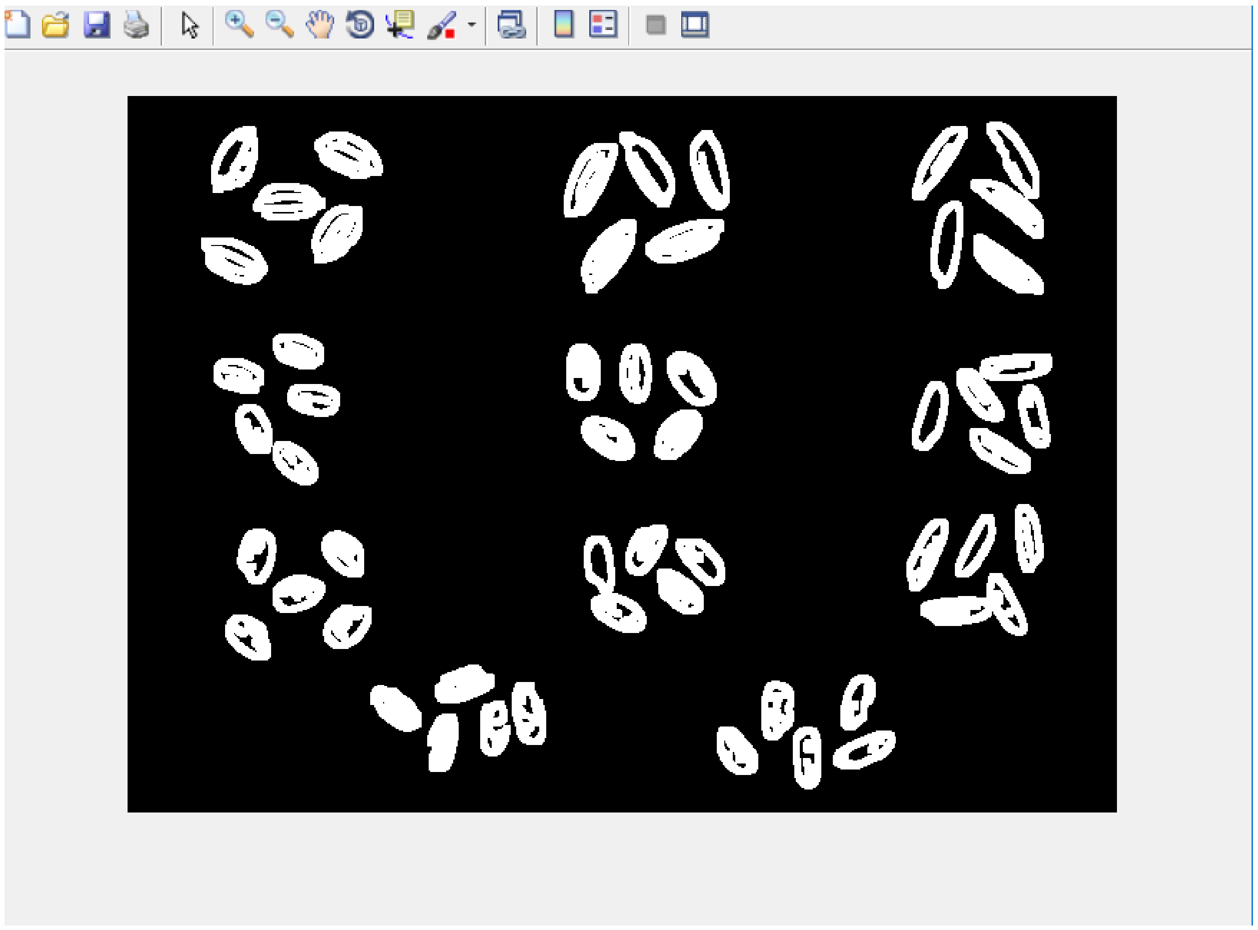


Figure3: After applying dilation operation on canny output.

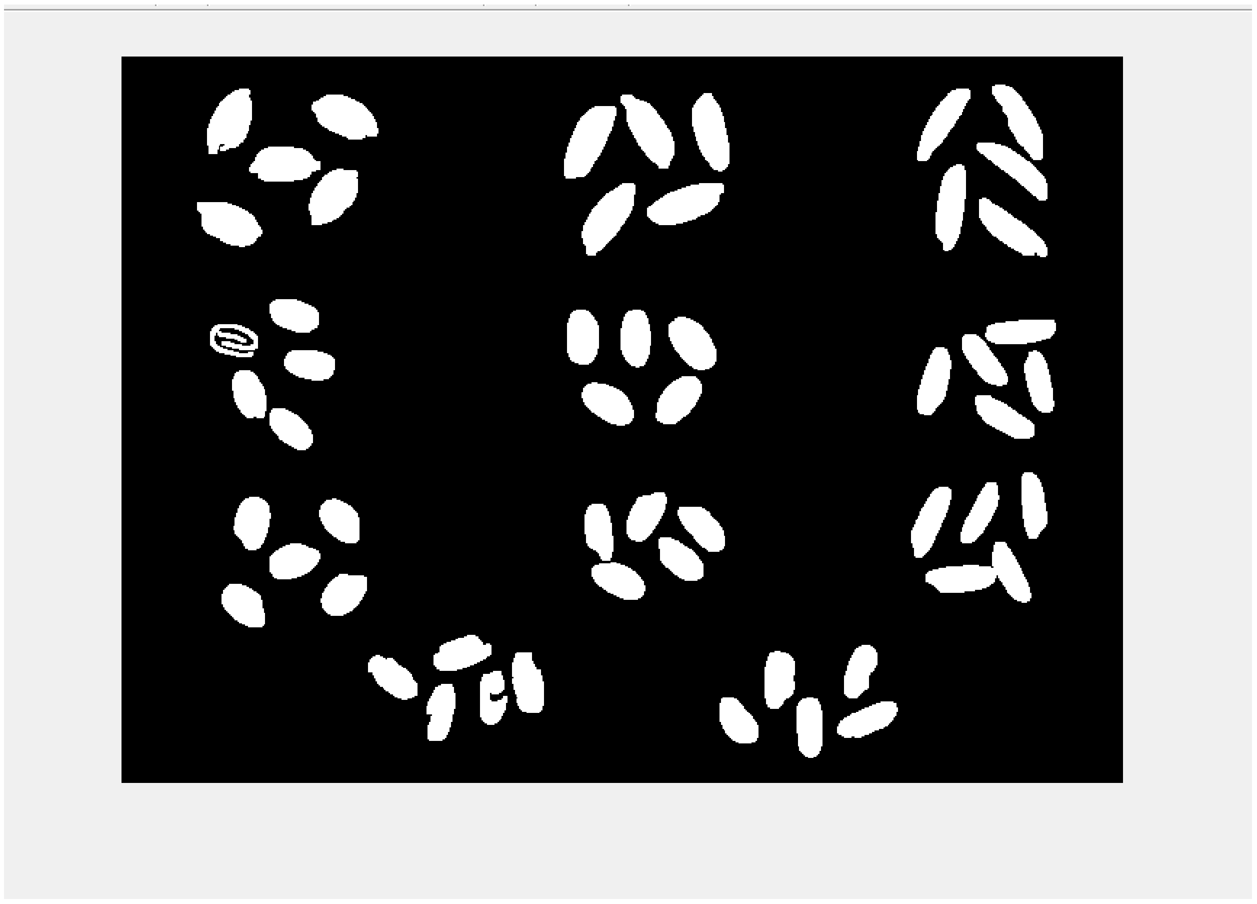


Figure4: After filling operation on dilation

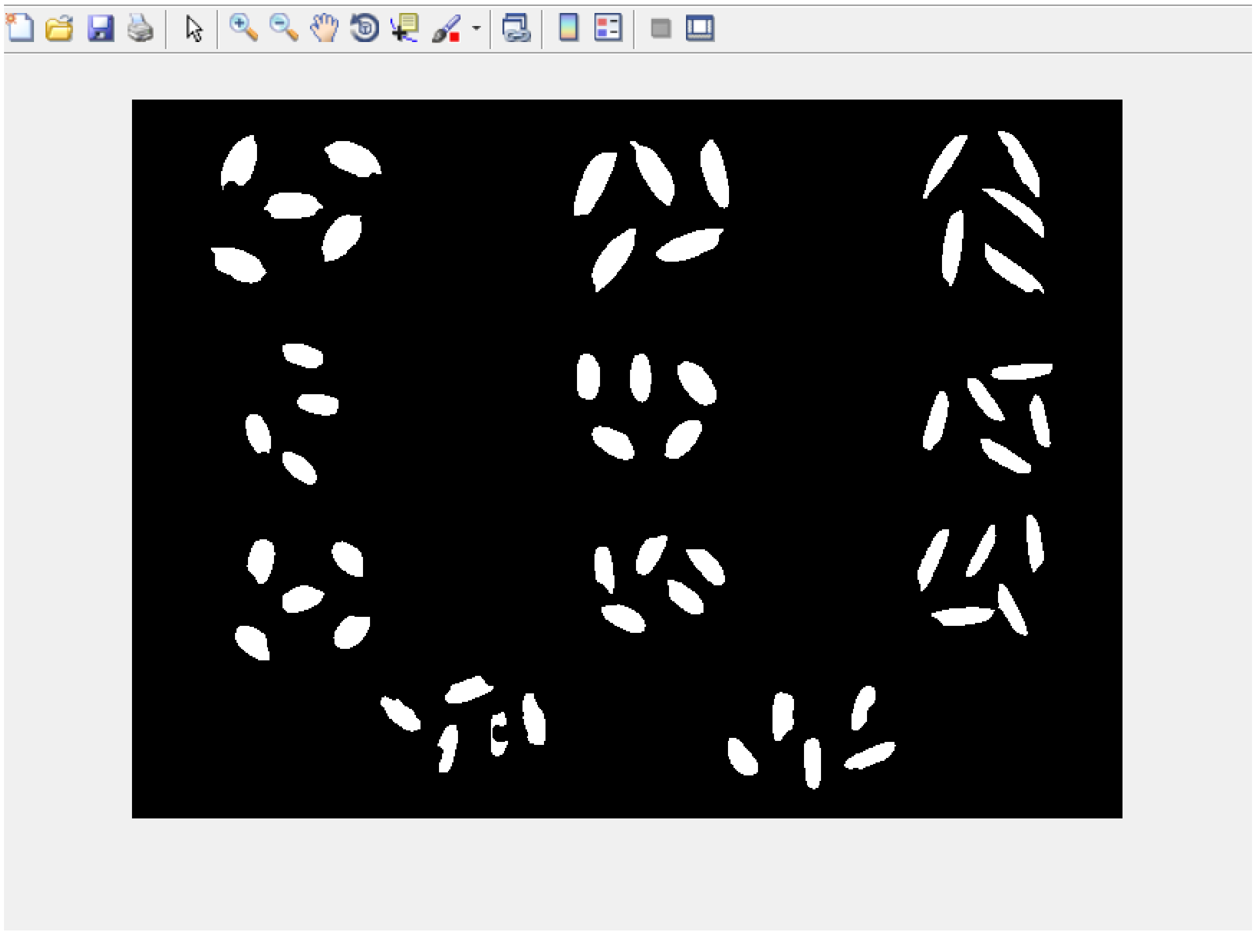


Figure5: After erosion operation

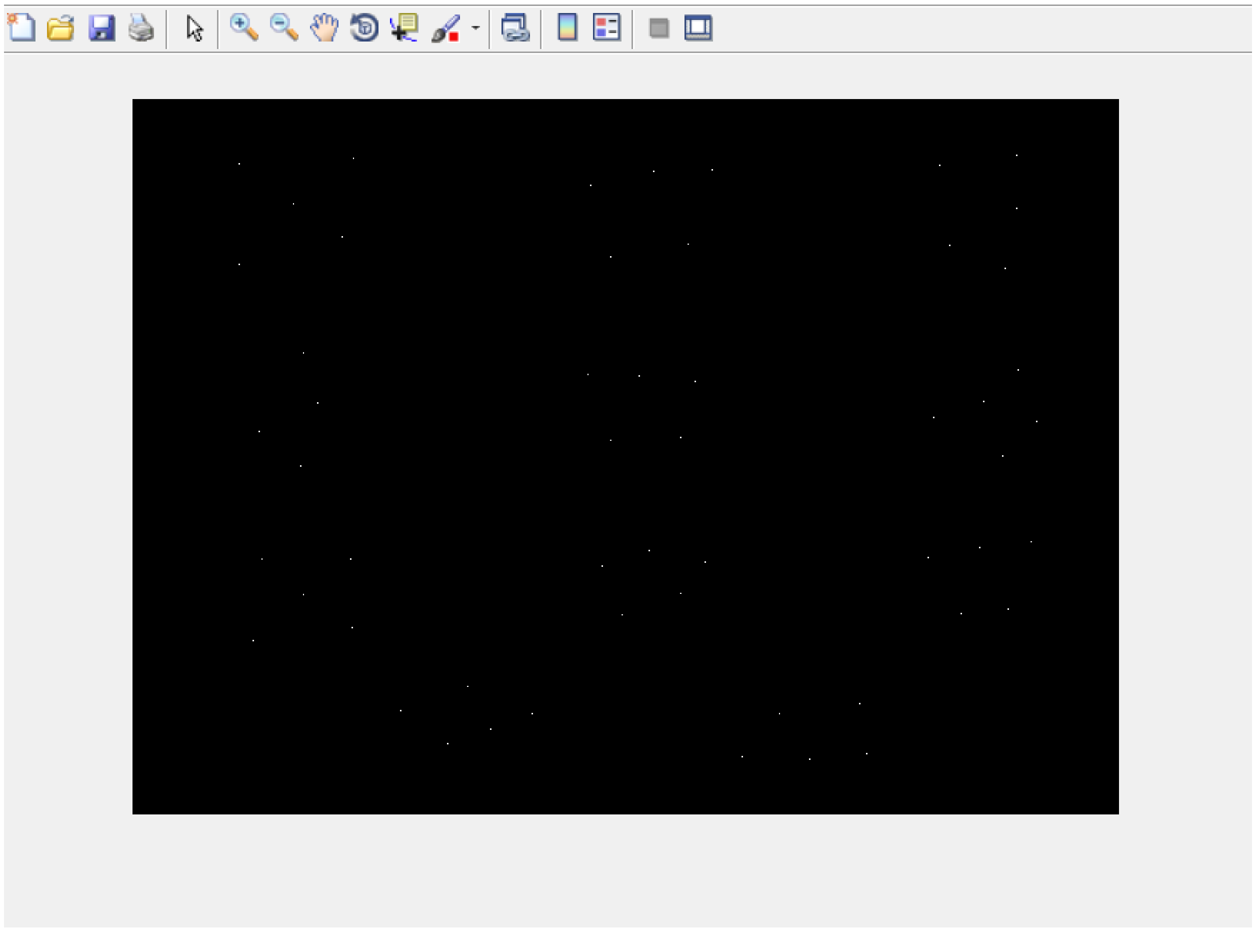


Figure6: output after applying shrinking to get count of grains in the image.

Here the Types of rice grains are defined starting from top left corner of the image and traversing horizontally.

```
 Columns 1 through 9
‘Type10’   ‘Type4’   ‘Type9’   ‘Type11’   ‘Type6’   ‘Type8’   ‘Type7’   ‘Type5’

‘Type3’

Columns 10 through 11
‘Type1’   ‘Type2’
```

Corresponding average areas:

```
Columns 1 through 7

  [215.4000]    [243.2000]    [305.6000]    [312.6000]    [315.6000]    [323.4000]    [362]

 Columns 8 through 11

  [404.8000]    [426.6000]    [553]    [592.6000]
```

Comparing size of rice grains in terms of "Area"

Type1	Type2	Type3	Type4	Type5	Type6	Type7	Type8	Type9	Type10	Type11
569	641	382	312	379	365	354	278	361	335	321
549	609	458	312	406	305	387	351	346	257	352
506	594	378	300	344	344	353	329	256	263	306
553	577	482	292	405	296	339	321	295	278	311
588	542	433		490	268	377	338	270	361	273

IV. Discussion

- The image was first converted in grayscale for morphological operations of matlab to function. Canny edge detector was used to detect and connect edges of each of the grains.
- The image was then dilated to fill any gaps in connectivites so that it can be used for fill operation which fills the object until it encounters white edges.
- The filled image was eroded to reduce the size of the grains and remove any unwanted connectivities. Finally shrinking operation was used to identify points corresponding to each shrinked grain. The count was then obtained using for loop.
- Different kind of morphological operator combinations were tried out to get the final result.

References

1) **http://www.jestr.org/downloads/Volume11Issue1/fulltext221112018.pdf**
2) William K. Pratt: Digital Image Processing, 4th Edition, John Wiley & Sons Inc., 2007. (ISBN 9780471767770).
3) https://zenodo.org/record/1322720#.XHrQdIhKhPY
4) https://www.cambridgeincolour.com/tutorials/lens-corrections.htm