HOMEWORK #3

Issued: 02/19/2020 **Due: 03/03/2020**

Problem 1: Geometric Image Modification

I. **Abstract and Motivation:** Till now, we dealt with Image processing algorithms which modifies the pixel values (Intensity values). These find applications in several fields. But there is an interesting operation where we can change the Image geometry without touching the pixel intensity values. This exciting operation is called Geometric Image Modification. It finds extensive applications in Computer Graphics and Image Registration. In this assignment, we discuss a few Geometric Image Modification Algorithms and implement them to visualize the results and discuss them.

Image Coordinates (Matrix representation) vs Cartesian Coordinates:
In Image processing we often encounter two coordinate systems which vary in the manner we express the pixel locations.

Image Coordinate system (Matrix representation): In this coordinate system, pixel coordinates are integer values with x-axis (row) from top to bottom and y-axis (columns) from left to right. Figure 1. Shows the representation.

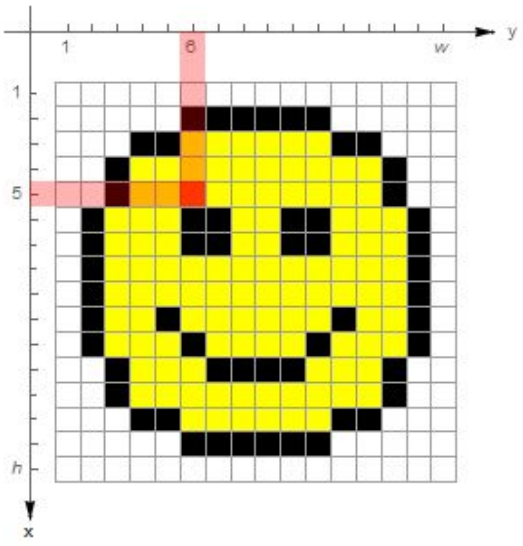


Figure 1. Image Coordinate system

Cartesian Coordinates: We do all the affine transformations in the coordinates. The pixel values are arranged from bottom to top (rows) and right to left (columns)[1]. Figure 2 shows the representation of Catersian coordinates

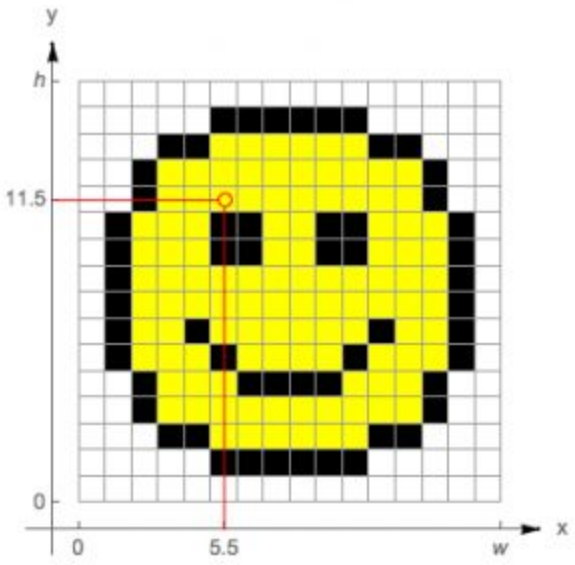


Figure 2. Cartesian Coordinate system

We can see from the above image that the coordinates in Image representation (5,6) are represented as (5.5, 11.5) in cartesian coordinate system. Below equation shows the conversion from Image coordinates to Cartesian coordinates.

$$X_k = k - 1/2$$
$$Y_k = HEIGHT + 1/2 - j$$
$$where,\ k\ and\ j\ rows\ and\ columns\ in\ Image\ coordinate\ system$$

The matrix representation of the above equations is shown below.

$$\begin{pmatrix} Xk \\ Yk \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & -1/2 \\ -1 & 0 & H+1/2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \\ 1 \end{pmatrix}$$

We do all geometric modification techniques in cartesian coordinate system and map it back to the Image coordinate system for visualization.

Affine Transformation: An affine transformation takes the pixel values to different location such that,

I. Linearity is preserved.
II. Origin shouldn't be distorted.

Figure 3. shows an example of Affine transformation.

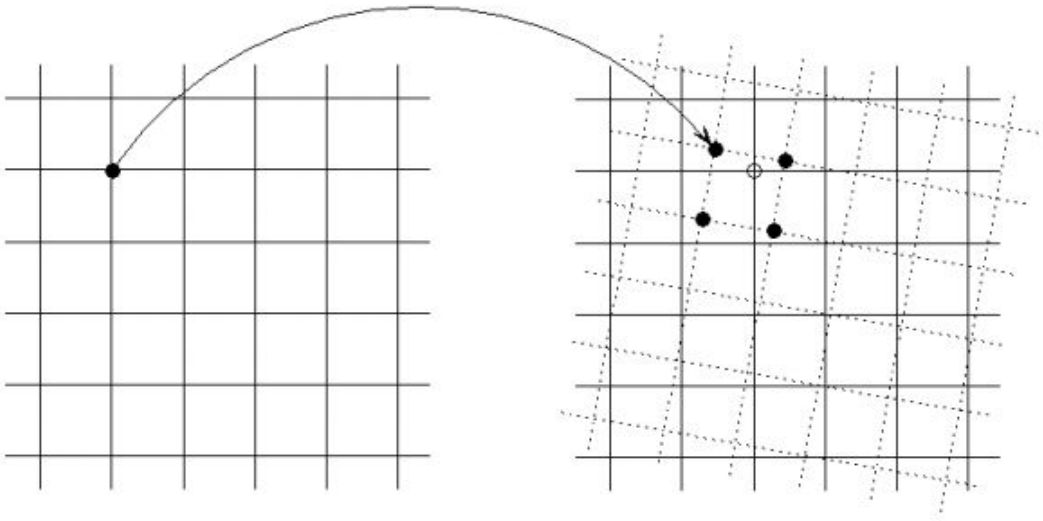


Figure 3. Affine (Spatial) Transformation of pixels

There are mainly three transformation operations. They are,

- ➔ Translation.
- ➔ Scaling (Zoom in and Zoom out).
- ➔ Rotation and Reflection.

The affine transformations are one-one mapping so there exists inverse transformation which maps the pixel back to the original location.

Suppose, we have (x,y) in the original image and it transformed to (u,v).

$$u = f_1(x,y)$$
$$v = f_2(x,y)$$

Now, inverse mapping is done from the equations below.

$$x = f_1^{-1}(u,v)$$
$$y = f_2^{-1}(u,v)$$

Figure 4. Shows the Inverse transforms.

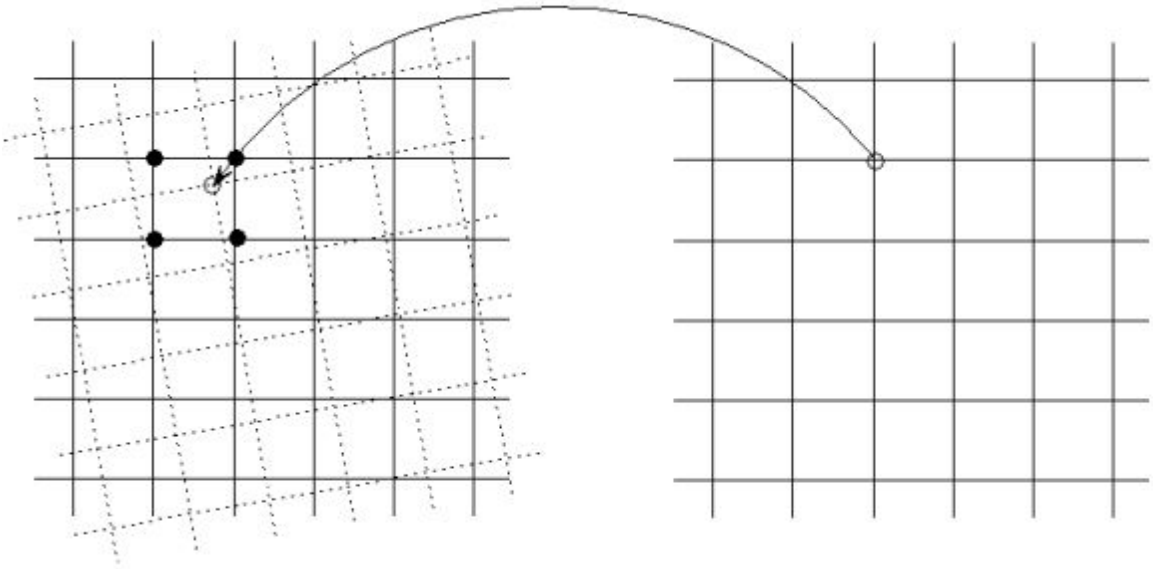


Figure 4. Inverse Mapping

Translation: Translation is an operation in which we take coordinates to some other position. Below equation represents the translation using translation parameters (tx, ty).

$$x = u + t_x$$
$$y = v + t_y$$

The matrix representation of the equation is given below.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

The inverse mapping is given by the following equation.

$$u = x - t_x$$
$$v = y - t_y$$

Note: We need to make sure that tx and ty should be integers because while mapping back to image coordinates it should remain integers. To achieve this we often employ **"Inverse address mapping."** In inverse address mapping we need to render the output image first (in integer locations). So, we trace back the corresponding input pixel locations, which can be fractional.

Figure 5. shows the translation in cartesian coordinate system.

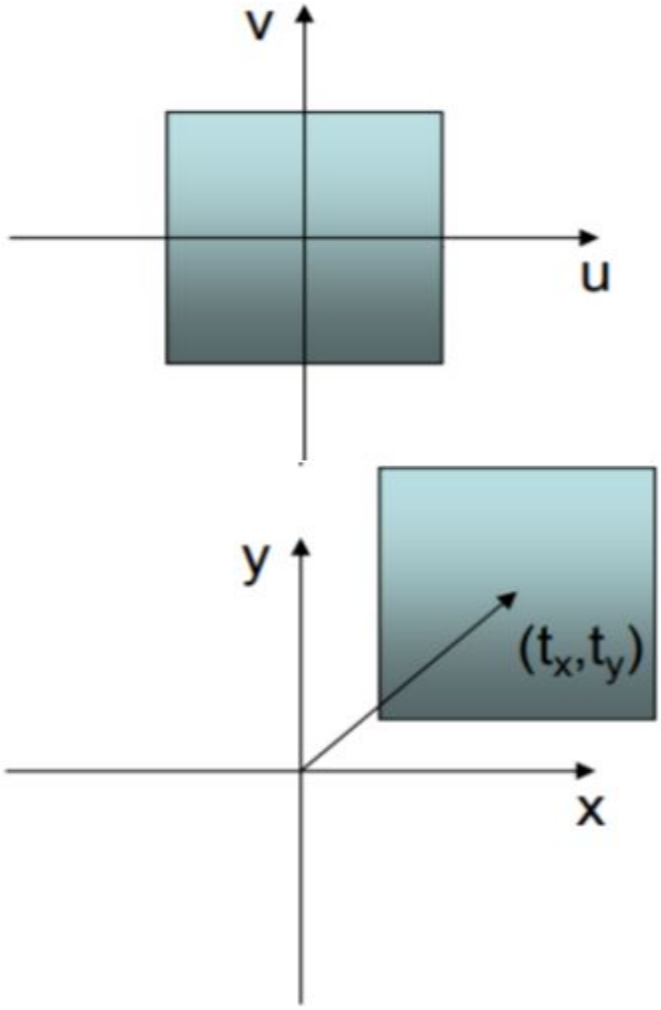


Figure 5. Translation

Scaling: In scaling we modify the size of the image. The scaling parameters decides the amount of scaling. Scaling should be done with respect to origin i.e, we need to get the center pixel to the origin, do scaling and then move back the pixel to the original location. Equations below show the scaling operation.

$$x = S_x u$$
$$y = S_y v$$

The matrix representation of this is shown in the equation below.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Figure 6. Shows scaling operation.

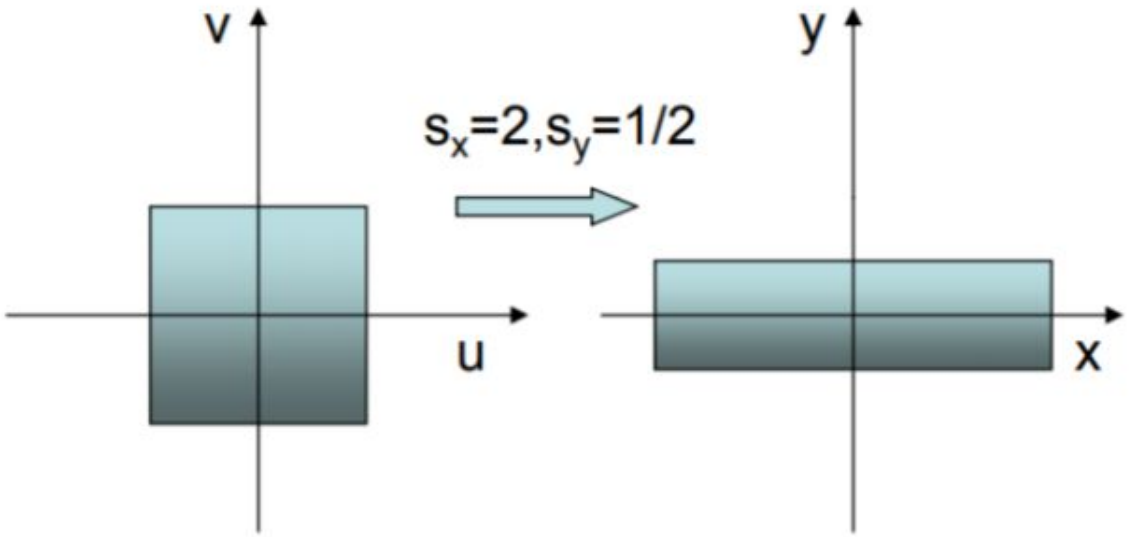


Figure 6. Scaling

In the above figure we can observe that, the image is scaled by a factor of 2 in x-direction and scaled by a factor of ½ in y-direction.

Rotation: Like scaling, rotation should also be performed with respect to origin. The rotation is an image with respect to origin by an angle θ is shown in the below matrix equation[2].

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

Figure 7. shows the step by step procedure to perform rotation.

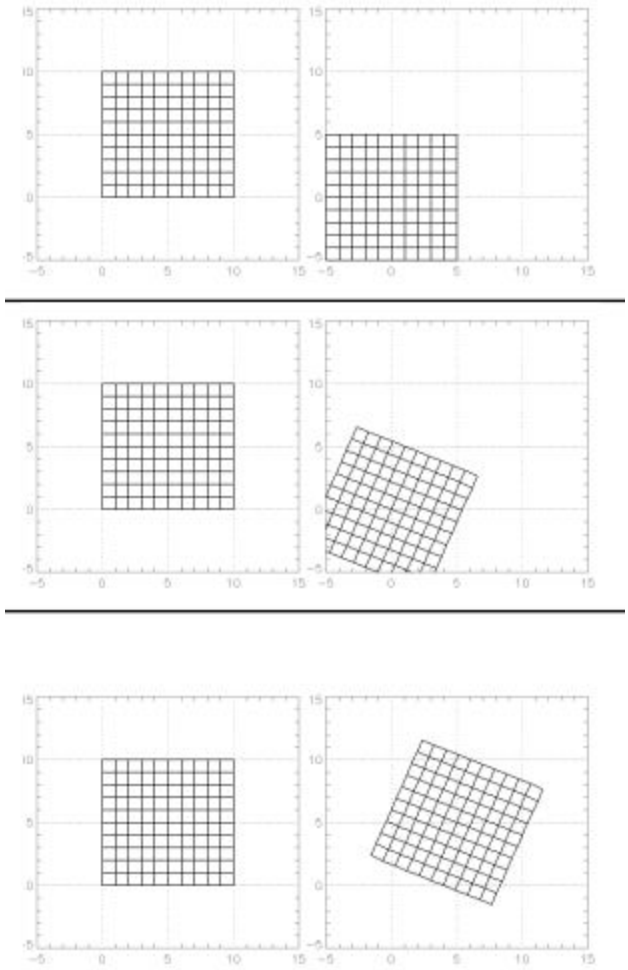


Figure 7. Rotation Operation - Step by Step.

Composite Transformation: We can simultaneously perform translation, scaling and rotation of an image. Let T1 represent the transformation matrix for translation, T2 represent the transformation matrix for scaling and T3 represent the transformation matrix for rotation. The compound operation is given by,

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

$$X = (T_1 T_2 T_3) U$$

The inverse transformation is given by,

$$U = (T_3^{-1} T_2^{-1} T_1^{-1}) X$$

a) Geometric Warping: The main objective of Image warping is to change the spatial orientation of the image by applying geometric transformations discussed in the previous section. The whole idea of changing the pixel location is inherited here [3]. Figure 8. shows an example of Image Warping.

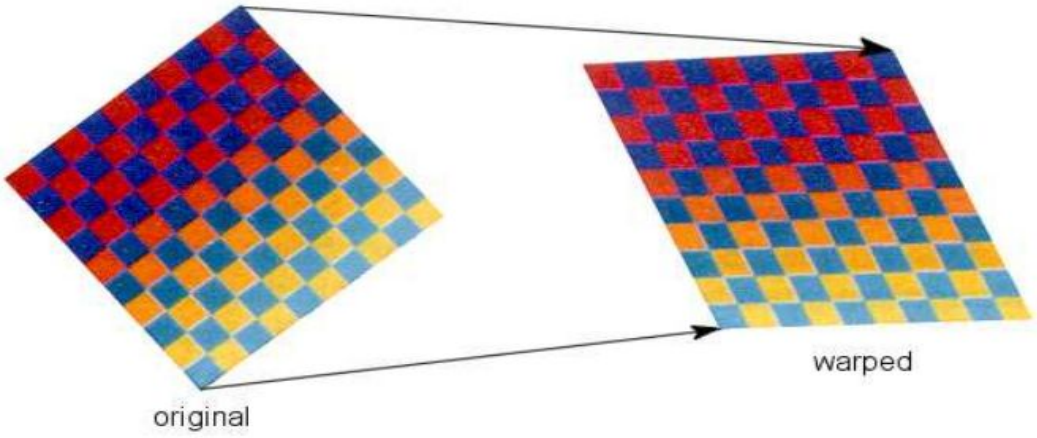


Figure 8. Image Warping.

The main idea of Image warping is, given an original image I and it's original coordinates (u, v), we need to find a transformation function between the new position (x, y) and (u, v). The derived transformation function is applied to the remaining pixels in the image.

Polar coordinates: Some geometric modification require polar coordinates to perform warping. Figure 9 shows an example of such an instance.

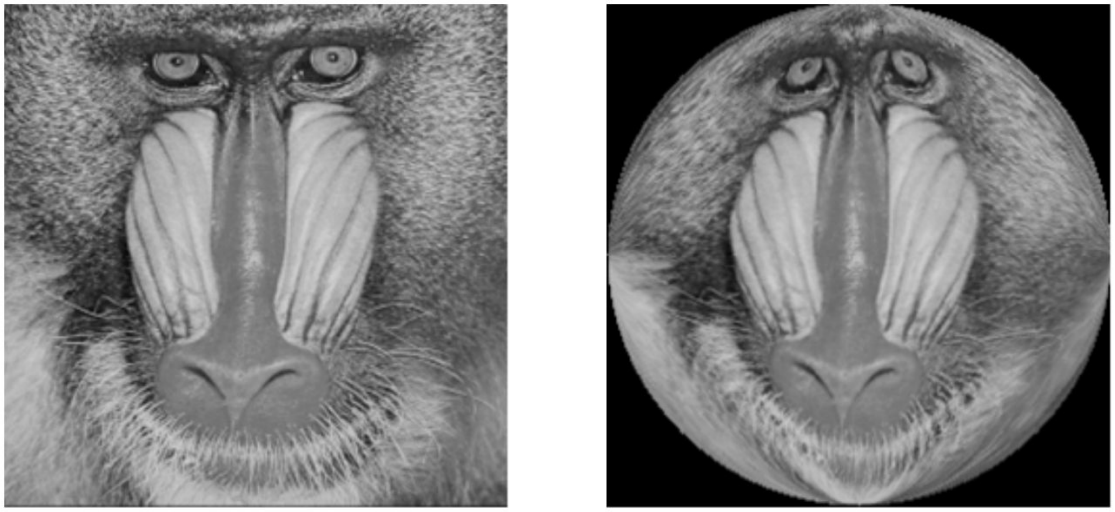


Figure 9. Warping an Image inside a circle/Disc

In this type of problems, we first convert the cartesian coordinates to polar coordinates, perform required operations and then bring back the output to cartesian coordinates. Figure 10. Shows the mapping from cartesian to polar coordinates.

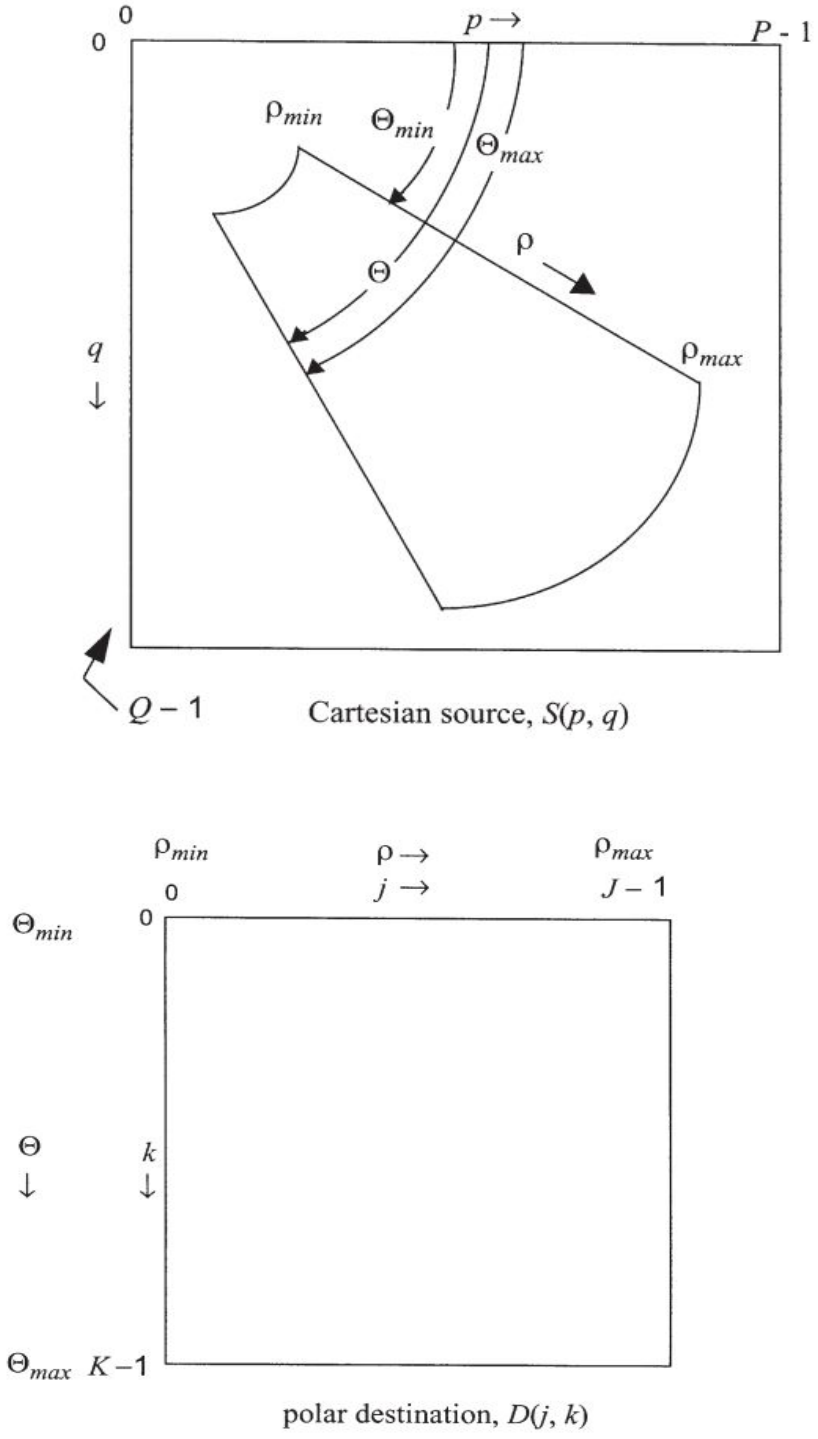


Figure 10. Mapping of pixels from Cartesian to Polar coordinates

The conversion equations are shown below.

Cartesian to Polar: We employ inverse address mapping to map each pixel from cartesian to polar coordinates.

$$p(j) = \frac{j[p_{max} - p_{min}]}{J-1} + p_{min}$$

$$\theta(k) = k\frac{[\theta_{max} - \theta_{min}]}{K-1} + \theta_{min}$$

$$For\ all\ (j,\ k) \in image$$

For each $p(j)\ and\ \theta(k)$ we calculate,

$$p' = [p(j)]cos\{\theta(k)\}$$

$$q' = [p(j)]sin\{\theta(k)\}]$$

Now for each (p', q') the address might be fractional, so we interpolate the values using bilinear interpolation and map to the polar coordinates.

Polar to Cartesian: We come back from polar to cartesian coordinate system using the below set of equations.

$$p(j) = \sqrt{j^2 + k^2}$$
$$\theta(k) = tan^{-1}(\frac{k}{j})$$

Now for each p(j) and θ (k) we compute,

$$p' = \frac{[p(j) - p_{min}][P-1]}{p_{max} - p_{min}}$$

$$q' = \frac{[\theta(k) - \theta_{min}][Q-1]}{\theta_{max} - \theta_{min}}$$

We discuss the implementation approach in detail in the Implementation section.

b) **Holographic transformation and Image Stitching:** Image Stitching is one of the most interesting topics in perspectives computer vision. In this section we dive into the details of Image Stitching a.k.a, Panorama Image generation [5]. We explore each topic in detail and discuss the implemented result in detail.

The main steps in Image Stitching are as follows,

1. Feature Matching between images (SIFT)
2. Calculating Homography Matrix to induce logical consistency between the images.
3. Warping.

To implement Image stitching we need images which are taken from the optical center of the camera. We then overlap one image onto the other by calculating the homography matrix and blend the overlapping area.

Scale Invariant Feature Transform: SIFT is the most sophisticated and robust algorithm in computer vision developed by David Lowe of University of British Columbia [6]. Unlike data driven approaches or deep learning (back propagation) methods to find features, SIFT robustly finds and describes the local features of an image.

The main idea behind SIFT is to find the important local features in an image and store it in a data set. Then we compare the new image with the stored features. Later, the matches are filtered out to obtain good matches. The features are compared using the euclidean distance between the two feature vectors[7]. SIFT is considered to be efficient because it extracts and matches the features under any change in conditions i.e., rotated, noisy, change in illumination etc. It lives up to it's name **"Scale Invariant".**

Following are the main steps in SIFT,

1. **Scale space extrema detection:** This step explains how SIFT is scale invariant. Usually in images we have different kinds of keypoints (features). For example we can have blunt corners or we can have huge corners. We definitely can't use the same window to detect both the corners (features). We need bigger windows for huge corners while we need small windows for blunt corners. In order to overcome this, we find Laplacian of Gaussian (LoG) with different values of σ. Here σ is our scaling parameter because LoG acts as a blob detector. Since LoG is computationally inefficient, SIFT uses Difference of Gaussian which closely approximates with the LoG. Figure 11 shows the scale space extrema detection [8].

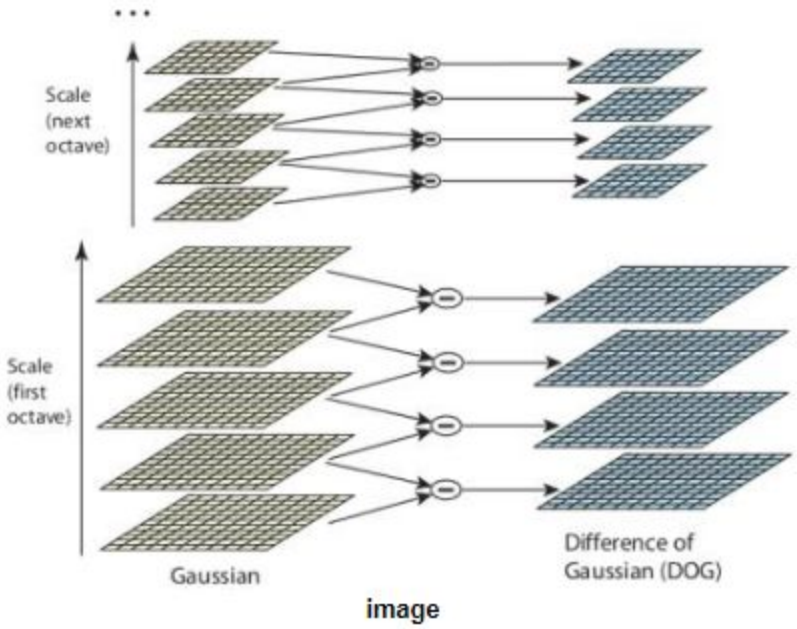


Figure 10. LoG to DoG

2. **Keypoint localization:** After getting the key points, we need to refine them to obtain only the important ones. Since DoG is sensitive to edges, we get unnecessary edges and noises as keypoints. It's better to filter them all before finalizing the keypoints. The original implementation used Taylor series expansion to obtain

the important extrema and thresholding is done after this to filter out the key points.

3. **Orientation Assignment**: We also know that one of the important criteria to consider is **rotation invariant**. Harris corner detector is rotation invariant but it's not scale invariant. However, SIFT is both Scale and Rotation invariant. In this step we calculate gradient magnitude and gradient direction of each detected keypoints and then histogram is plotted for neighboring pixels and the maximum 80% of it is considered. This induces stability in matching.

4. **Keypoint descriptor:** We pick 16*16 neighborhood of each key points and consider 16 subblocks of 4*4 sizes each. For each subblock we collect 128 bins and represent it as a keypoint descriptor vector.

5. **Keypoint matching:** We use keypoint matching to match between the keypoints between two images. The original implementation uses the nearest neighbor approach to achieve this.
   Figure 11-12 shows the implementation result of SIFT.

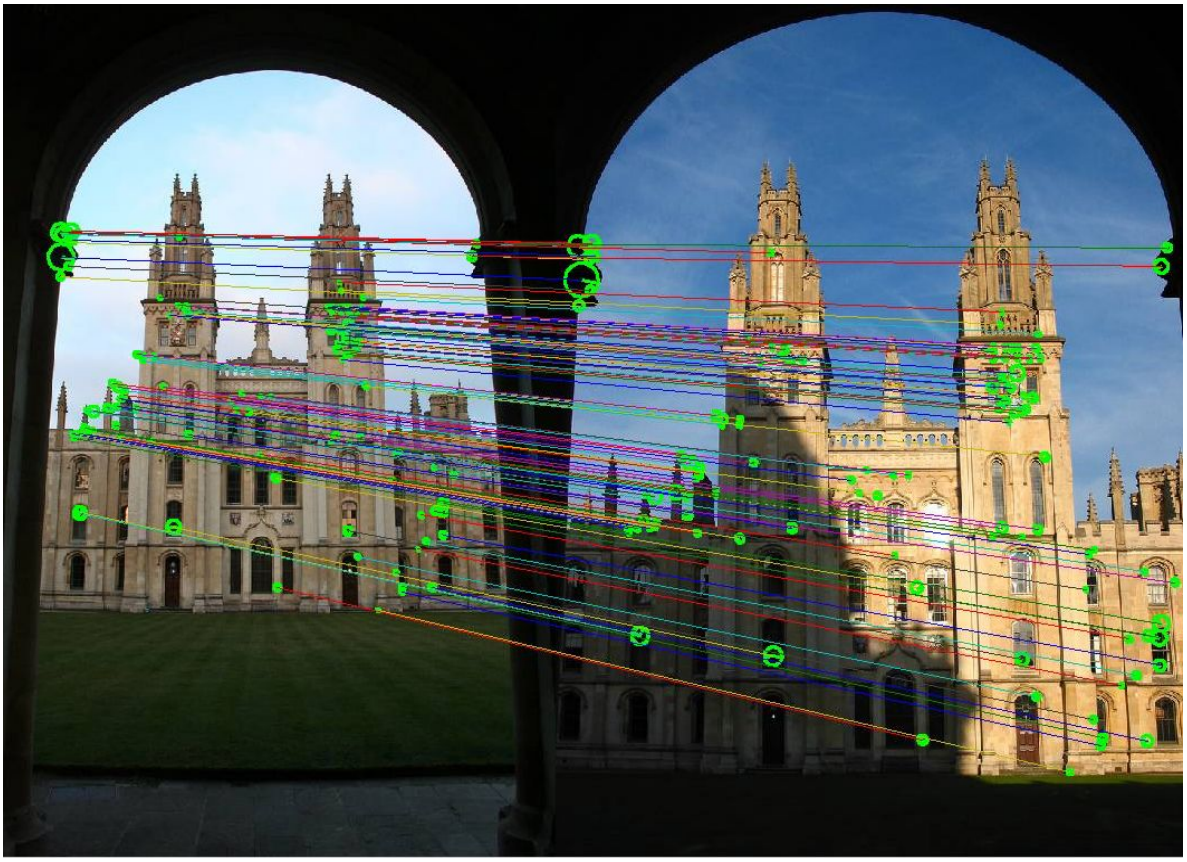


Figure 10. SIFT Feature Matching.

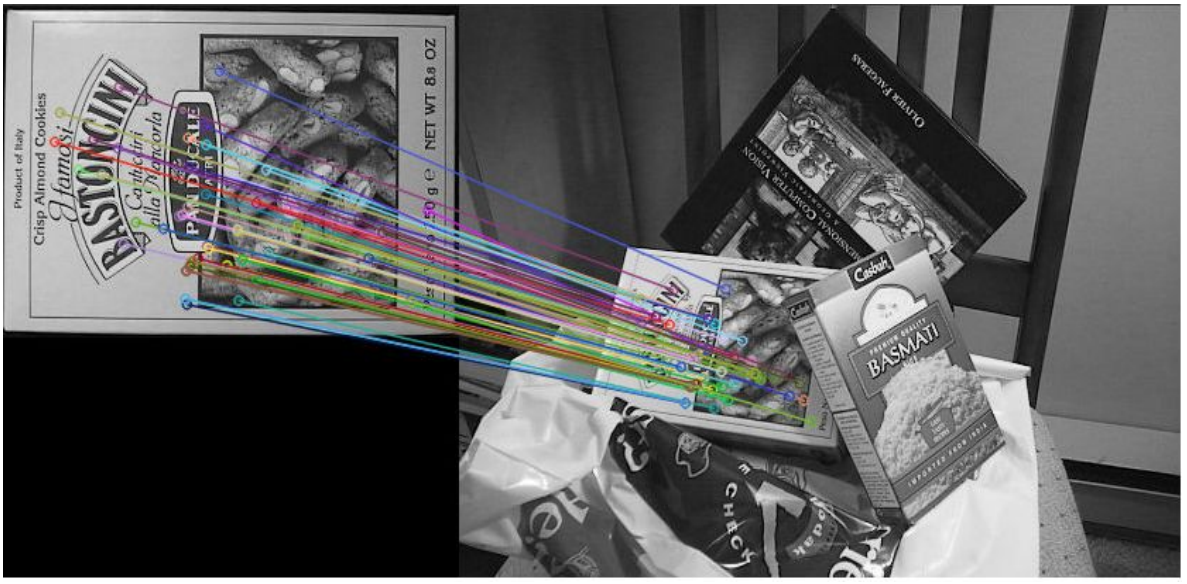


Figure 11. SIFT Feature Matching

Homography: Hands down, homography is the most interesting topic explored in this course. Homography is a perspective transformation between two images taken in different orientation with fixed center. Homography is a projective transformation which **preserves straight lines.**

In the context of Image Stitching, after obtaining key points in two different images, we calculate homography matrix such that any pixel in the image applied to the homography matrix will yield projection of the points which lies in both the images. Projective transforms are just like translation, scaling, rotation and other affine transforms. Figure 12 shows the comparison of different transforms.

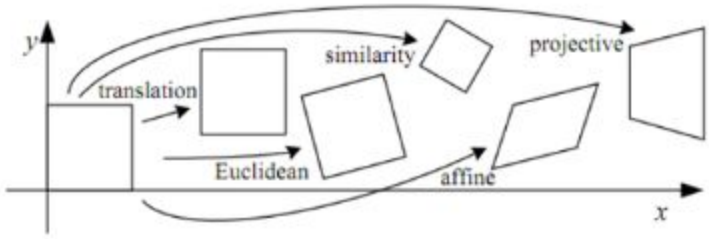


Figure 12. Types of Transforms.

The projection of points using Homography matrix is shown in Figure 13.

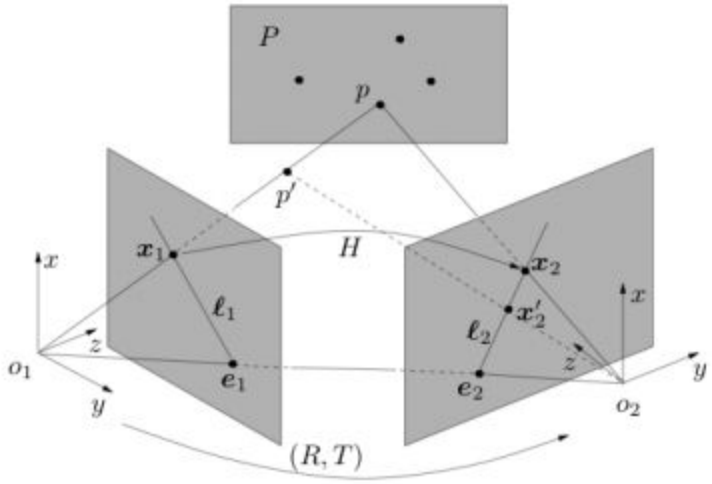


Figure 13. Homography Matrix Projection

After feature matching, we find the key-points between two images and generate a Homography Matrix. The homography matrix is calculated using the below equations.

$$\begin{bmatrix} x_2' \\ y_2' \\ w_2' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{x_2'}{w_2'} \\ \frac{y_2'}{w_2'} \end{bmatrix}$$

$$x_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13}$$
$$y_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23}$$

We set H_{33} to 1 and generate 2 equations for every pair of points. So, we take 4 pairs of points and generate 8 equations. Now we have 8 degrees of freedom ($H_{11} - H_{32}$) and 8 equations and we solve them to get the required matrix [9].

Figure 14 shows the result of applying homography matrix to points in the images.

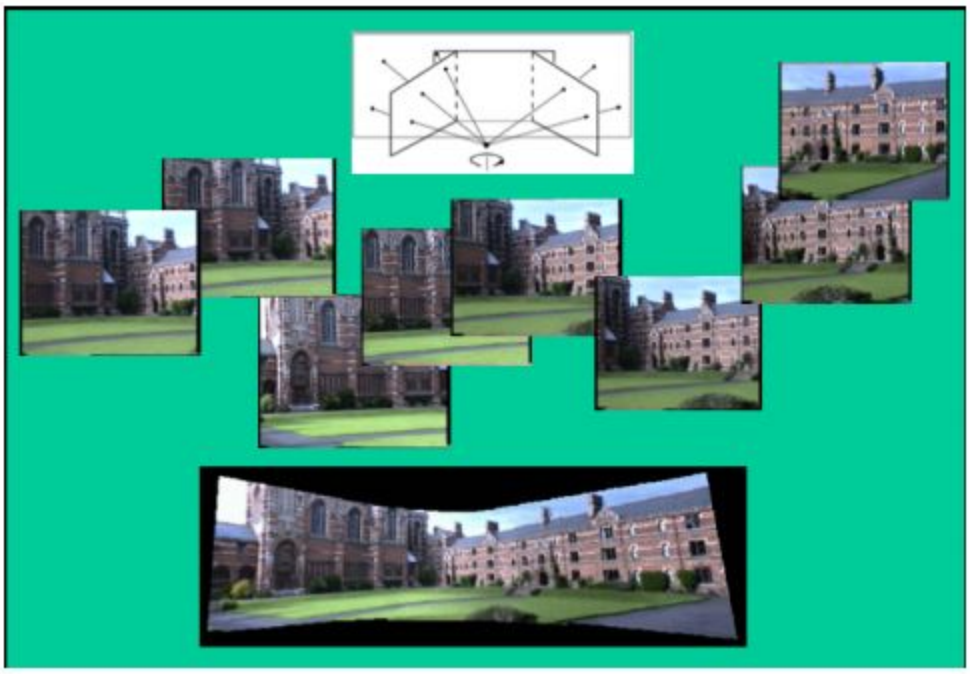
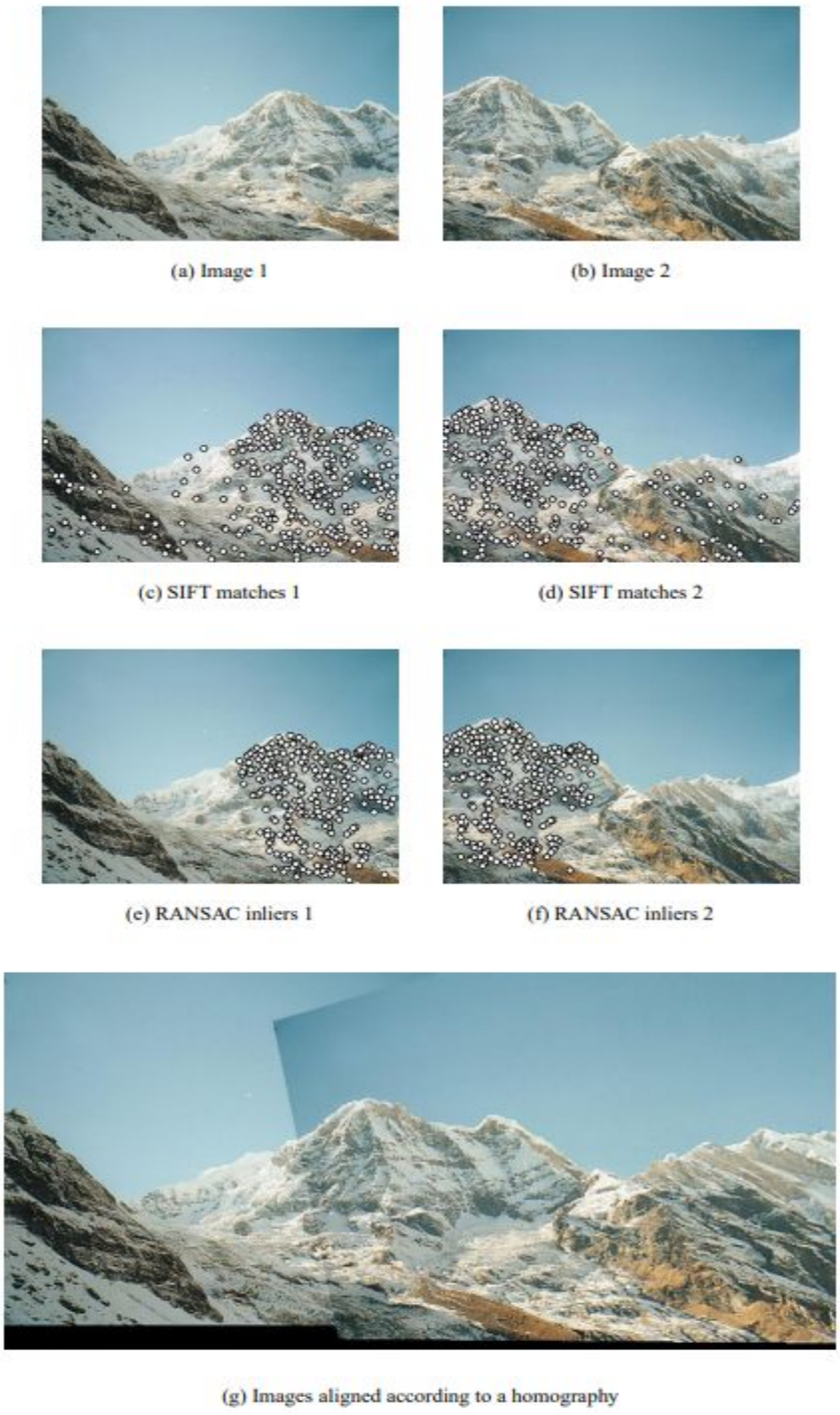


Figure 14. Result of Homography

In this assignment we calculated the homography matrix by hand. But in automated panorama generation softwares, we use an efficient and robust algorithm called RANSAC. Let us explore this algorithm briefly.

Random Sample Consensus (RANSAC): After feature matching we get several key-points. This algorithm is all about generating the homography matrix using the best possible 4 matching points. We randomly select 4 pairs of points and solve the linear equations to obtain the parameters. We repeat this process 500 times and find the pair that is consistent with the homography. So, RANSAC is a probabilistic method to find the best pair of points which can fetch the required homography matrix.

Figure 15 shows all the steps explained till now



(a) Image 1

(b) Image 2

(c) SIFT matches 1

(d) SIFT matches 2

(e) RANSAC inliers 1

(f) RANSAC inliers 2

(g) Images aligned according to a homography

Figure 15. Image Stitching

II. Approach and Implementation

1. Geometric Image Modification:

Goal: Our task is to implement Geometric Modification and fit the given images into a disc such that,

a) Boundary pixels should be on the boundaries of the circle.
b) The center pixel should be mapped to the center.
c) The mapping should be one-one i.e., inverse transformation should exist.

Math: To map all the points in a square to circle in the range,

$$\begin{aligned} -1 \leq x \leq 1 \\ -1 \leq y \leq 1 \end{aligned}$$

The formula to map the points is given by,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{x}{\sqrt{2}} \\ \sqrt{1 - \frac{x^2}{2}} \end{bmatrix}$$

This equation uses the general equation of ellipse and put constraints to obtain the mapping in a circle [10]. Figure 16 shows the mapping.

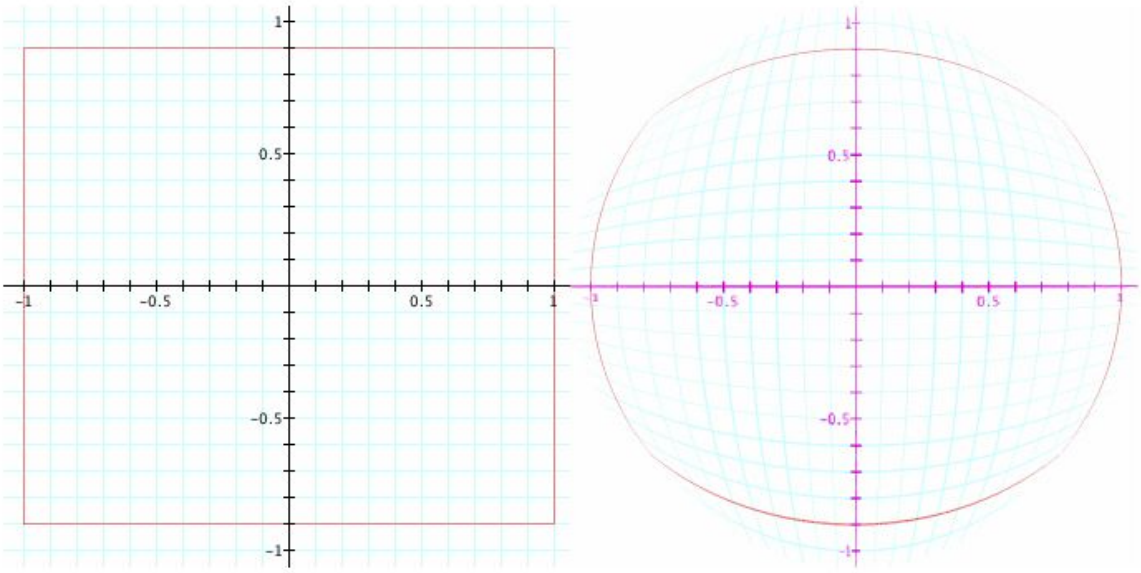


Figure 16. Mapping from Square to Circle

Logic: Mapping all the points of a square to a circle.

1) We convert from Image coordinates to Cartesian coordinates using the formula.

$$x_k = cols - 0.5$$

$$y_k = HEIGHT + 0.5 - rows$$

2) We normalize the pixel coordinates to wrap this in between -1 and 1 by subtracting the coordinates by center value (256.5, 256.5) and dividing it by 256.5.
3) We apply inverse address mapping to the formula shown above i.e., instead of mapping the points in square to circle we find points in square corresponding to the pixel value in circle. Doing this, we can avoid black dots in the image.
4) We convert back from cartesian coordinates to Image coordinates.
5) The values in image coordinates may be fractional. So we use bilinear interpolation to get the integer pixel values.

Logic: Reconstructing the original Image (Inverse Transformation):

1) The technique is similar to the forward mapping but here we use the reverse of the formula used in forward mapping.
2) The normalization, interpolation, center shifting are all employed in Inverse Transformation also.
3) The reconstructed image is observed and compared.

2. Homographic Transformation and Image Stitching:

Goal: To create a panorama image using multiple images taken from different orientation but with same center.

We are given 3 images (middle, left and right) corresponding to each direction.

Steps:

a) To obtain the key-points we used OpenCV-Python to get the match pairs.
   Note: Python code is submitted which is in **.ipynb format.**
b) Using the inbuilt function to calculate SURF feature matching, we got keypoints and descriptors.
c) We get several key points for each of the middle, left and right images.
d) We use Opencv's knnmatch function. We find similar feature points between middle-left and middle-right images.
e) We get several matches using the function. In order to filter only good matches, we use thresholding. After this we get 588 and 411 matches respectively for middle-left and middle-right images.
f) To calculate the homography matrix, we can either randomly choose 4 points or hand pick points from the image. Handpicked points from the image give better results, so I chose points from the image directly.
g) Following points are chosen between the middle and left images.
   {**middle:** (228, 248), (574, 242), (249, 296), (414, 249)}
   {**left:** (217, 405),(593, 385), (236, 468), (421, 401)}

h) We built a large canvas of size (1500, 2000) and we map all the points from the original image to canvas.
i) We use an offset of 280 pixel width between images.
j) The corresponding pixels in the canvas will be,
{**left:** (608, 405),(984, 385),(630, 468),(812, 401)}
{**middle:** (619,1009),(965,1003),(640,1057),(805, 1010)}
k) We convert these points to a cartesian coordinate system.
l) Now, we obtain 8 equations to solve for h11-h32.

```
[[-10.9983136   0.267510394   1553.27437]
 [-8.60849818 -2.54597342   2101.15424]
 [-0.0161620692   0.000864037213   1.0]]
```

m) We apply this transformation to the image to get the projection. We again use inverse address mapping to do this.
n) Same procedure is followed between the right and middle image.
o) Note: *I wasn't able to get the output in c++. The results attached here are obtained from opencv's warpPerspective function. The opencv code will be attached in the .zip file.*

III. Results and Discussion

1) Geometric Image Modifications:

We applied Geometric modification technique to fit the image into a disk to BB8, Hedwig and Raccoon. Figure 17-25 shows the original image, warped image and reconstructed image for each of the given images respectively.

Figure 17. BB8 - Original Image

Figure 18. BB8 - Warped in Disc (Not exactly Disk)

Figure 19. BB8 Reconstructed using Inverse Transformation

Figure 20. Hedwig Original Image

Figure 21. Hedwig Warped in a Disc (Not exactly Disc)

Figure 22. Hedwig Reconstructed Image using Inverse Transformation

Figure 23. Raccoon Original Image

Figure 24. Raccoon Warped in a disc (Not Exactly Disc)

Figure 25. Hedwig Reconstructed Image using Inverse Transformation

Discussion:

1) The resulting image is not completely warped in the disc. The curves in the corners have become blunt. This is because, as discussed in the above section I am using ellipse derivation to get the circle evolved. In this approach, we need to normalize the image and set it in between -1 to 1 and bring the center to the origin. As I am converting the image to cartesian coordinate system, my center is not exactly placed on origin. This is the reason I am not getting an exact circle. Due to time constraints I couldn't debug and correct it.
2) We can clearly see that the center is mapped to the center. Borders are mapped to the borders of the warped image.
3) Since the transformation function is 1-1, we can reconstruct the image.
4) I am using **Horizontal Squeezing** in the implementation.

5) As we can see in the BB8 image the doom shaped body has become rhombus, in the Hedwig Image the eyes and nose of hedwig is squeezed indicating horizontal squeezing.
6) There are no black holes or lines in the warped Image. Thanks to inverse address mapping and interpolation.
7) However, there are steps like patterns in the borders of the image.
8) **Reconstruction:** Reconstructed image is almost the same as original image. There are no distortions. However, if we clearly observe we can find that the reconstructed image is **smoothened** (low pass filtering). The contrast is decreased and sharp edges are smoothened. This can be clearly observed in Raccoon image - the hairs in the original image were clear and distinct, but in the reconstructed image the image is smoothened. We can also see some distortions in the edges.
9) **The reason for the above distortion is bilinear interpolation**. We do bilinear interpolation while mapping, this averages the surrounding pixels.

2) Homographic Transformations and Image Stitching:

Step1: Used opencv's inbuilt surf function to get the keypoints and descriptors.

```
detector = cv2.xfeatures2d_SURF.create(hessianThreshold =minHessian)
keypoints1, descriptors1 = detector.detectAndCompute(middle_np,None)
keypoints2, descriptors2 = detector.detectAndCompute(left_np,None)
keypoints2, descriptors3 = detector.detectAndCompute(right_np,None)
```

Step2: After obtaining the keypoints and descriptors for each image, we find the matches between the two images.

```
matcher=cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBAS)
knn_matches = matcher.knnMatch(descriptors1, descriptors3, 2).
```

Step3: We used thresholding to obtain only the good matches.

Step4: Drew lines between the matched points. Figure 26-27 shows the similar features between two images.

Figure 26: Matched features between Middle and left Images.

Figure 27: Matched features between Middle and Right Images.

Step6: We have two choices at this stage, we can select random points obtained from opencv to calculate the transformation or we can manually select the points from the image to calculate the transformation matrix.

Step7: Create a large canvas and add images so that it should fit the entire image. Figure 28 shows the image of it.

Figure 28: Canvas

Step8: Selecting Control points: I selected the fridge as the reference object and picked points from the 4 corners of the image from middle, left and right images.

Following control points were chosen,

Control Points between Middle and Left Image

Control Points	1	2	3	4
Middle	(228, 248)	(574, 242)	(249, 296)	(414, 249)
Left	(217, 405)	(593, 385)	(236, 468)	(421, 401)

Control Points between Middle and Right Image

Control Points	1	2	3	4
Middle	(229, 246)	(230, 350)	(577, 241)	(570, 341)
Right	(217, 84)	(237, 196)	(601, 93)	(564, 200)

Step9: Following transformation matrix is derived using the above control points.

$H_{left-middle}$

```
[[-1.09983136e+01  2.67510394e-01  1.55327437e+03]
[-8.60849818e+00 -2.54597342e+00  2.10115424e+03]
[-1.61620692e-02  8.64037213e-04  1.00000000e+00]]
```

$H_{right-middle}$

```
[[ 5.40711593e-01 -2.51687941e-02  1.85914301e+02]
[-3.14973150e-01  8.43225933e-01  5.19069626e+01]
[-9.41839566e-04  4.37683871e-06  1.00000000e+00]]
```

Step10: Using these transformation matrix we calculate projections and map that to the canvas.

Step11: The result of projection of right and left images are shown in Figure 29-30.

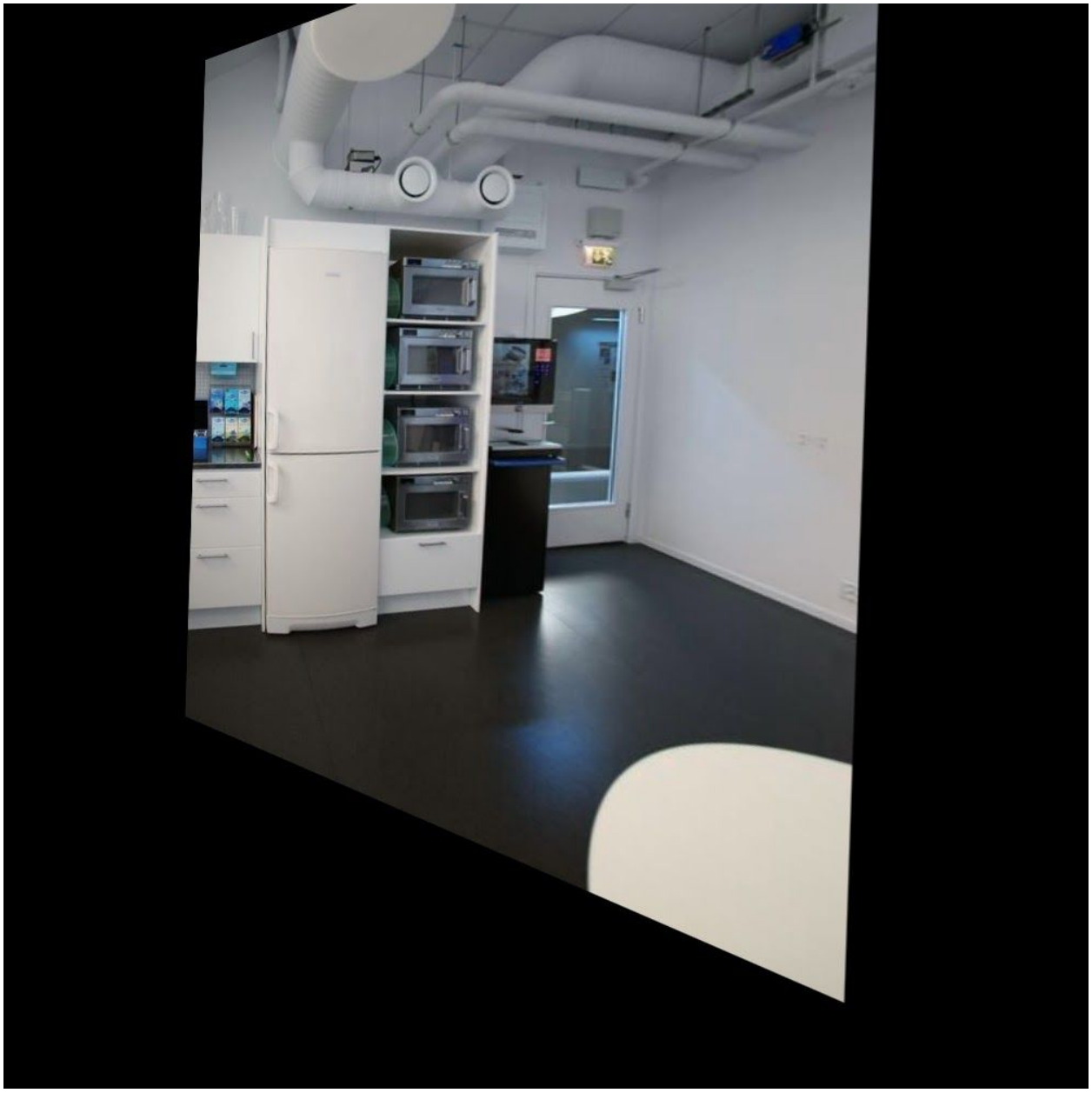


Figure 29: Projection of Right Image on the canvas.

Figure 30: Projection of Left image on the canvas.

Due to time constraints, I wasn't able to stitch the images and my logic was failing terribly. The C++ code and Python code for this implementation is attached in the .zip file.

Discussions:

1) 4 control points were chosen between the middle - left image and middle - right image. Since there are 8 degrees of freedom, we need 4 pairs of control points (8 points) to construct a linear system of equations and to solve the matrix. The control points are shown in the above tables in step 8.
2) Control points were chosen manually from the middle-left image and middle-right image. We can also pick 4 matched points from the matched pair array obtained from the opencv function. The step by step approach is explained in the Abstract and motivation section.

Problem 2: Morphological Processing

I. Abstract and Motivation:

Till now we came across abstract mathematics, linear algebra and a few parts of coordinate geometry in Image processing. But in this section we use Set theory extensively for morphological image processing. Most oftenly we deal with binary images. A set in binary represents an object in an image and it is 2D vector i.e., (x,y) coordinates. In gray scale images, it will be a 3D vector (x,y, val).

Image connectivity: We use 4 and 8 connectivity du define the morphological operations. Figure 1 shows 4 and 8 connectivity.

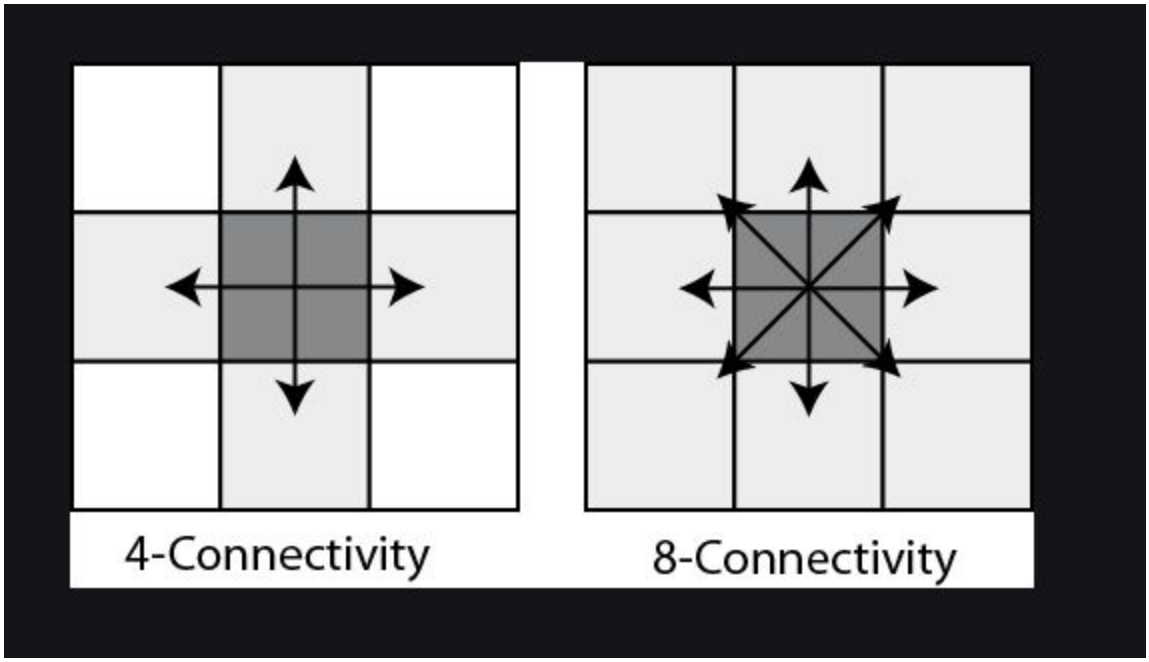


Figure 1. Connectivity between pixels

Binary Hit or Miss Transformation: This operation is the most fundamental and rudimentary operation in Morphology. We use this for all other morphological operations. The operation is straight forward. We use an odd size structuring element (mask) and scan the image against this mask. If the pattern in the image matches with the mask, it's a hit else it's a miss. We assign a binary value for two of these results.

Additive Morphological Operations: Additive hit or miss transformation changes the center pixel value against the scanning structuring element from 0 to 1. Let us define basic operations using the given logical statements under additive operations.

Interior Fill: The equation is given by,

$$G(j,k) = X \cup [X_0 \cap X_2 \cap X_4 \cap X_6]$$

Diagonal Fill: The equation is given by,

$$G(j,k) = X \cup [P_1 \cup P_2 \cup P_3 \cup P_4]$$

where,

$$P_1 = \bar{X} \cap X_0 \cap \bar{X}_1 \cap X_2$$

$$P_2 = \bar{X} \cap X_2 \cap \bar{X}_3 \cap X_4$$

$$P_3 = \bar{X} \cap X_4 \cap \bar{X}_5 \cap X_6$$

$$P_4 = \bar{X} \cap X_6 \cap \bar{X}_7 \cap X_0$$

Bridge: The equation is given by,

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

where,

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

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

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

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

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

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

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

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

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

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

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

Subtractive Morphological Operations: In subtractive Morphological operations we do the exact reverse of Additive operation.
Some of the common subtractive operations are discussed below,

a) **Isolated Pixel Removal:** The equation is given below,

$$G(j,k) = X \cap [X_0 \cup X_1 \cup \cdots \cup X_7]$$

b) **Interior Pixel Removal:** The equation is given below,

$$G(j,k) = X \cap [\bar{X}_0 \cup \bar{X}_2 \cup \bar{X}_4 \cup \bar{X}_6]$$

c) **Eight Neighbor erode:** The equation is given below,

$$G(j,k) = X \cap X_0 \cap \cdots \cap X_7$$

Morphological Shrinking: Shrinking is a 2 parse morphological process in which we end up with a single pixel in the center if the Image has no holes. But we end up with a ring shared structured if we have holes around it. The logical expression of Shrinking is given in the equation below,

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

We can implement shrinking in a single stage by using 5*5 filter masks. But the number of patterns will go up to 2^{25}. So, in order to reduce the number of patterns and break the symmetry we use 2 stage processing with 3*3 patterns. The block diagram of shrinking is shown in Figure 2.

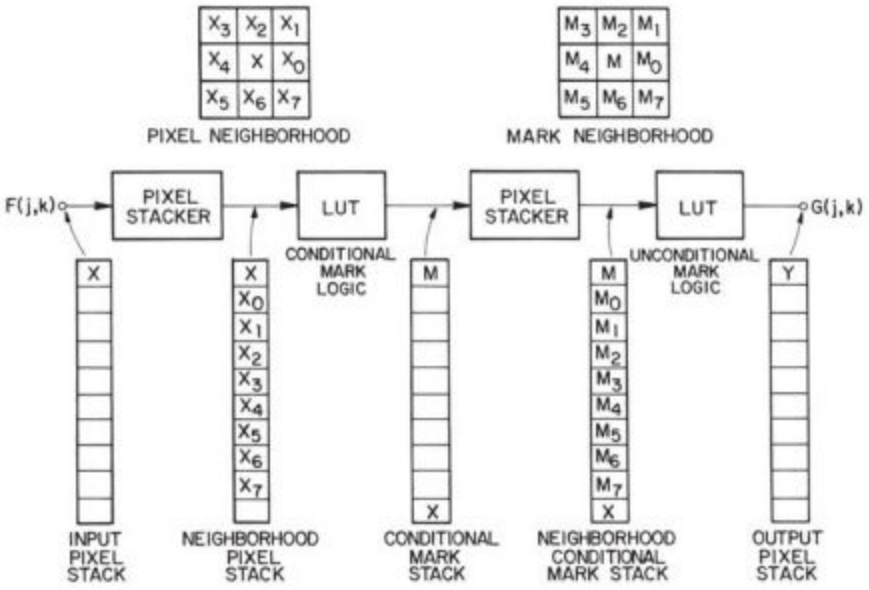
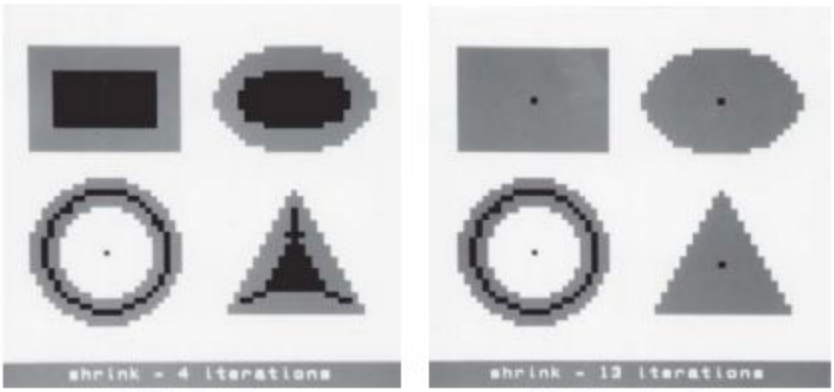


Figure 2. Block diagram

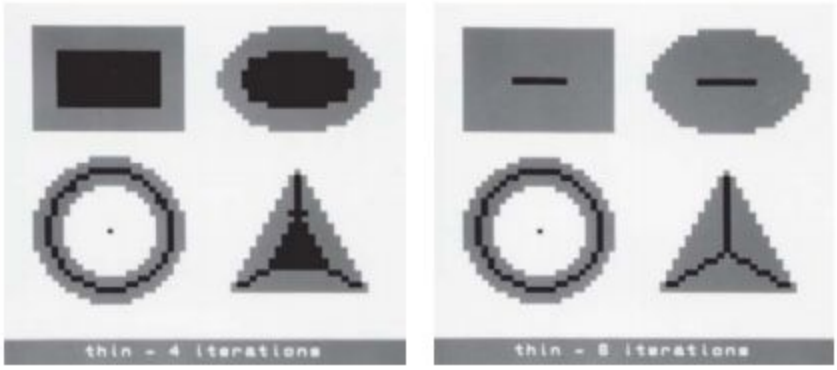
Figure 3 shows some of the sample results of shrinking.



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

Figure 3. Results of Shrinking

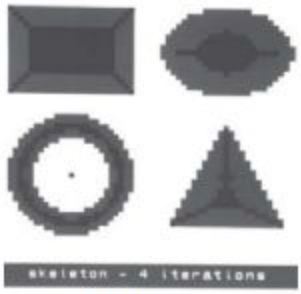
Morphological Thinning: Thinning is similar to shrinking but in thinning instead of a single dot/pixel we get a small line in the center. Like Shrinking, thinning can also be implemented in a 2 stage process with conditional and unconditional masks. Figure 4. Shows the results of thinning.



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

Figure 4. Results of Thinning

Morphological Skeletonizing: This operation is used to describe a structure. It gives a stick diagram of the object. The results are similar to thinning but varies in different scenarios. Figure 5 shows the results of thinning.



(*a*) Four iterations

(*b*) Ten iterations

Figure 5. Skeletonizing

The filter table for Shrinking, Thinning and Skeletonizing is shown in Figure 6.

Table	Bond	Pattern							
		0 0 1	1 0 0	0 0 0	0 0 0				
S	1	0 1 0	0 1 0	0 1 0	0 1 0				
		0 0 0	0 0 0	1 0 0	0 0 1				
		0 0 0	0 1 0	0 0 0	0 0 0				
S	2	0 1 1	0 1 0	1 1 0	0 1 0				
		0 0 0	0 0 0	0 0 0	0 1 0				
		0 0 1	0 1 1	1 1 0	1 0 0	0 0 0	0 0 0	0 0 0	0 0 0
S	3	0 1 1	0 1 0	0 1 0	1 1 0	1 1 0	0 1 0	0 1 0	0 1 1
		0 0 0	0 0 0	0 0 0	0 0 0	1 0 0	1 1 0	0 1 1	0 0 1
		0 1 0	0 1 0	0 0 0	0 0 0				
TK	4	0 1 1	1 1 0	1 1 0	0 1 1				
		0 0 0	0 0 0	0 1 0	0 1 0				
		0 0 1	1 1 1	1 0 0	0 0 0				
STK	4	0 1 1	0 1 0	1 1 0	0 1 0				
		0 0 1	0 0 0	1 0 0	1 1 1				
		1 1 0	0 1 0	0 1 1	0 0 1				
ST	5	0 1 1	0 1 1	1 1 0	0 1 1				
		0 0 0	0 0 1	0 0 0	0 1 0				
		0 1 1	1 1 0	0 0 0	0 0 0				
ST	5	0 1 1	1 1 0	1 1 0	0 1 1				
		0 0 0	0 0 0	1 1 0	0 1 1				
		1 1 0	0 1 1						
ST	6	0 1 1	1 1 0						
		0 0 1	1 0 0						
		1 1 1	0 1 1	1 1 1	1 1 0	1 0 0	0 0 0	0 0 0	0 0 1
STK	6	0 1 1	0 1 1	1 1 0	1 1 0	1 1 0	1 1 0	0 1 1	0 1 1
		0 0 0	0 0 1	0 0 0	1 0 0	1 1 0	1 1 1	1 1 1	0 1 1

Table	Bond	Pattern							
		1 1 1	1 1 1	1 0 0	0 0 1				
STK	7	0 1 1	1 1 0	1 1 0	0 1 1				
		0 0 1	1 0 0	1 1 1	1 1 1				
		0 1 1	1 1 1	1 1 0	0 0 0				
STK	8	0 1 1	1 1 1	1 1 0	1 1 1				
		0 1 1	0 0 0	1 1 0	1 1 1				
		1 1 1	0 1 1	1 1 1	1 1 1	1 1 1	1 1 0	1 0 0	0 0 1
STK	9	0 1 1	0 1 1	1 1 1	1 1 1	1 1 0	1 1 0	1 1 1	1 1 1
		0 1 1	1 1 1	1 0 0	0 0 1	1 1 0	1 1 1	1 1 1	1 1 1
		1 1 1	1 1 1	1 1 1	1 0 1				
STK	10	0 1 1	1 1 1	1 1 0	1 1 1				
		1 1 1	1 0 1	1 1 1	1 1 1				
		1 1 1	1 1 1	1 1 0	0 1 1				
K	11	1 1 1	1 1 1	1 1 1	1 1 1				
		0 1 1	1 1 0	1 1 1	1 1 1				

Figure 6. Pattern table for conditional stage (Shrinking, Thinning and Skeletonization)

The pattern table for unconditional stage is given in Figure 7.

Pattern							
Spur		Single 4-connection					
0 0 *M*	*M*0 0	0 0 0	0 0 0				
0 *M*0	0 *M*0	0 *M*0	0 *MM*				
0 0 0	0 0 0	0 *M*0	0 0 0				
L Cluster							
0 0 *M*	0 *MM*	*MM*0	*M*0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 *MM*	0 *M*0	0 *M*0	*MM*0	*MM*0	0 *M*0	0 *M*0	0 *MM*
0 0 0	0 0 0	0 0 0	0 0 0	*M*0 0	*MM*0	0 *MM*	0 0 *M*
4-Connected offset							
0 *MM*	*MM*0	0 *M*0	0 0 *M*				
*MM*0	0 *MM*	0 *MM*	0 *MM*				
0 0 0	0 0 0	0 0 *M*	0 *M*0				
Spur corner cluster							
0 *A M*	*MB* 0	0 0 *M*	*M*0 0				
0 *MB*	*A M*0	*A M*0	0 *MB*				
*M*0 0	0 0 *M*	*MB* 0	0 *A M*				
Corner cluster							
MMD							
MMD							
DDD							
Tee branch							
*DM*0	0 *MD*	0 0 *D*	*D*0 0	*DMD*	0 *M*0	0 *M*0	*DMD*
MMM	*MMM*	*MMM*	*MMM*	*MM*0	*MM*0	0 *MM*	0 *MM*
*D*0 0	0 0 *D*	0 *MD*	*DM*0	0 *M*0	*DMD*	*DMD*	0 *M*0
Vee branch							
MDM	*MDC*	*CBA*	*A DM*				
DMD	*DMB*	*DMD*	*B MD*				
A B C	*MDA*	*MDM*	*C DM*				
Diagonal branch							
*DM*0	0 *MD*	*D*0 *M*	*M*0 *D*				
0 *MM*	*MM*0	*MM*0	0 *MM*				
*M*0 *D*	*D*0 *M*	0 *MD*	*DM*0				

[a] $A \cup B \cup C = 1 \quad D = 0 \cup 1 \quad A \cup B = 1.$

Figure 7. Unconditional stage (Shrinking and Thinning)

Pattern											
Spur											
0	0	0	0	0	0	0	0	*M*	*M*	0	0
0	*M*	0	*0*	*M*	0	0	*M*	0	0	*M*	0
0	0	*M*	*M*	0	0	0	0	0	0	0	0
Single 4-connection											
0	0	0	0	0	0	0	0	0	0	*M*	0
0	*M*	0	0	*M*	*M*	*M*	*M*	0	0	*M*	0
0	*M*	0	0	0	0	0	0	0	0	0	0
L corner											
0	*M*	0	0	*M*	0	0	0	0	0	0	0
0	*M*	*M*	*M*	*M*	0	0	*M*	*M*	*M*	*M*	0
0	0	0	0	0	0	0	M	0	0	*M*	0
Corner cluster											
M	*M*	*D*	*D*	*D*	*D*						
M	*M*	*D*	*D*	*M*	*M*						
D	*D*	*D*	*D*	*M*	*M*						
Tee branch											
D	*M*	*D*	*D*	*M*	*D*	*D*	*D*	*D*	*D*	*M*	*D*
M	*M*	*M*	*M*	*M*	*D*	*M*	*M*	*M*	*D*	*M*	*M*
D	*D*	*D*	*D*	*M*	*D*	*D*	*M*	*D*	*D*	*M*	*D*
Vee branch											
M	*D*	*M*	*M*	*D*	*C*	*C*	*B*	*A*	*A*	*D*	*M*
D	*M*	*D*	*D*	*M*	*B*	*D*	*M*	*D*	*B*	*M*	*D*
A	*B*	*C*	*M*	*D*	*A*	*M*	*D*	*M*	*C*	*D*	*M*
Diagonal branch											
D	*M*	0	0	*M*	*D*	*D*	0	*M*	*M*	0	*D*
0	*M*	*M*	*M*	*M*	0	M	M	0	0	*M*	*M*
M	0	*D*	*D*	0	*M*	0	*M*	*D*	*D*	*M*	0

[a] $A \cup B \cup C = 1 \quad D = 0 \cup 1$.

Figure 8. Unconditional masks for Skeletonizing.

The applications of these basic morphological operations include Object Analysis, Defect detection and for some low level computer vision problems. A few of them will be discussed and implemented.

II. Approach and Implementation:

1) Shrinking, Thinning and Skeletonization:

a) We convert the filter table to 8bit binary number manually and we calculate decimal value corresponding to each of the filter masks.
b) We store these decimal numbers in an array.
c) We calculate decimal values corresponding to each filter separately for conditional and unconditional masks.
d) We iterate through the Image and convert the neighboring 8bits to decimal value and check whether the value is present in the array or not.
e) We repeat the same for stage 2 or unconditional stage.
f) We loop these steps back to back until there are no changes in the image.
g) This method is faster compared to storing each filter in a 2D array.
Note: *This method didn't provide the required output. So a header file is created with filter array and the images patches are compared against it.*

2) Counting the stars:

a) We can use 2 approaches for this method. One is Shrinking and another is Connected component analysis.
b) Using Shrinking we shrink the image to a dot and we count the isolated dots at the end to count the number of stars.
c) To find the star size, we need to keep track of shrinking steps.
d) Using connected component analysis, we can do it more efficiently.
e) This is a 2 parse algorithm. We iterate through the image as a **raster** pattern.
f) Initialize the object label to 1
g) If we find 1 in any of the pixel location,

i) Check the already scanned neighbors and if the neighbors are all 0's, assign a new object label to the pixel and increment the object label
ii) If there is only one non zero value, assign the same value to the location
iii) If we have multiple non zero values, find the minimum value and assign. The rest of the values are stored in the map table to change the values in 2nd parse.

h) In the second parse, change the keys in the map table with the stored values.
i) After 2 parses, we will have different labels to each connected component in the image.
j) If we find the frequency of each connected component, we get the size of the star.

3) PCB Analysis:

a) Similar to counting stars, we can also exploit Shrinking properties.
b) Shrinking provides a dot for the objects with no holes. So, after applying shrinking we get an isolated point at the hole location.
c) We iterate through the image to find a white pixel with it's 8 neighborhood being (a.k.a isolated white dot).
d) We count the number of such white dots to get the number of holes.
e) To calculate the pathways, we use connected component analysis.
f) We invert the image so that pathways become white and apply connected component analysis. The total number of connected components will be the pathways.

4) Defect Detection:

a) We find the center of the 4 black holes.
b) Also, we find the center of the gear.
c) We should then implement the **Morphological thinning** operation.
d) We get the skeleton of the image with a line near the gear tooth.
e) Since there should be 12 gear teeth, there should be a gear tooth for every 30 degrees.
f) Using the center coordinate value of the gear, we sweep every 30 degrees and try to find the line.
g) We mark the missing line as defective.

III. Results and Discussion:

1) **Morphological Shrinking:** Shrinking operation is performed on fan, cup and maze images and results are obtained:

   Figures 9-13 shows the results of shrinking on Fan Image.

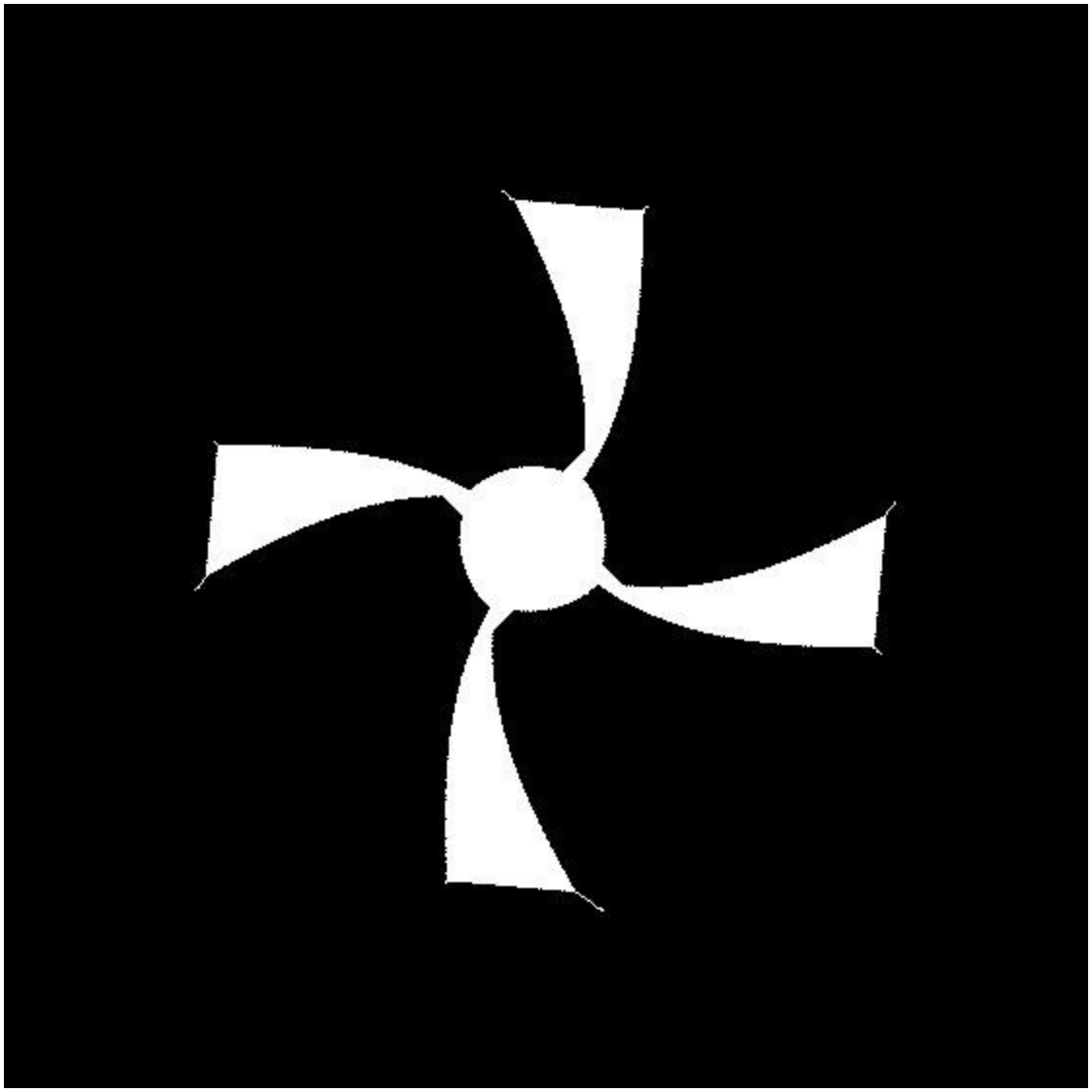


Figure 9: Shrinking of Fan after 10 iterations

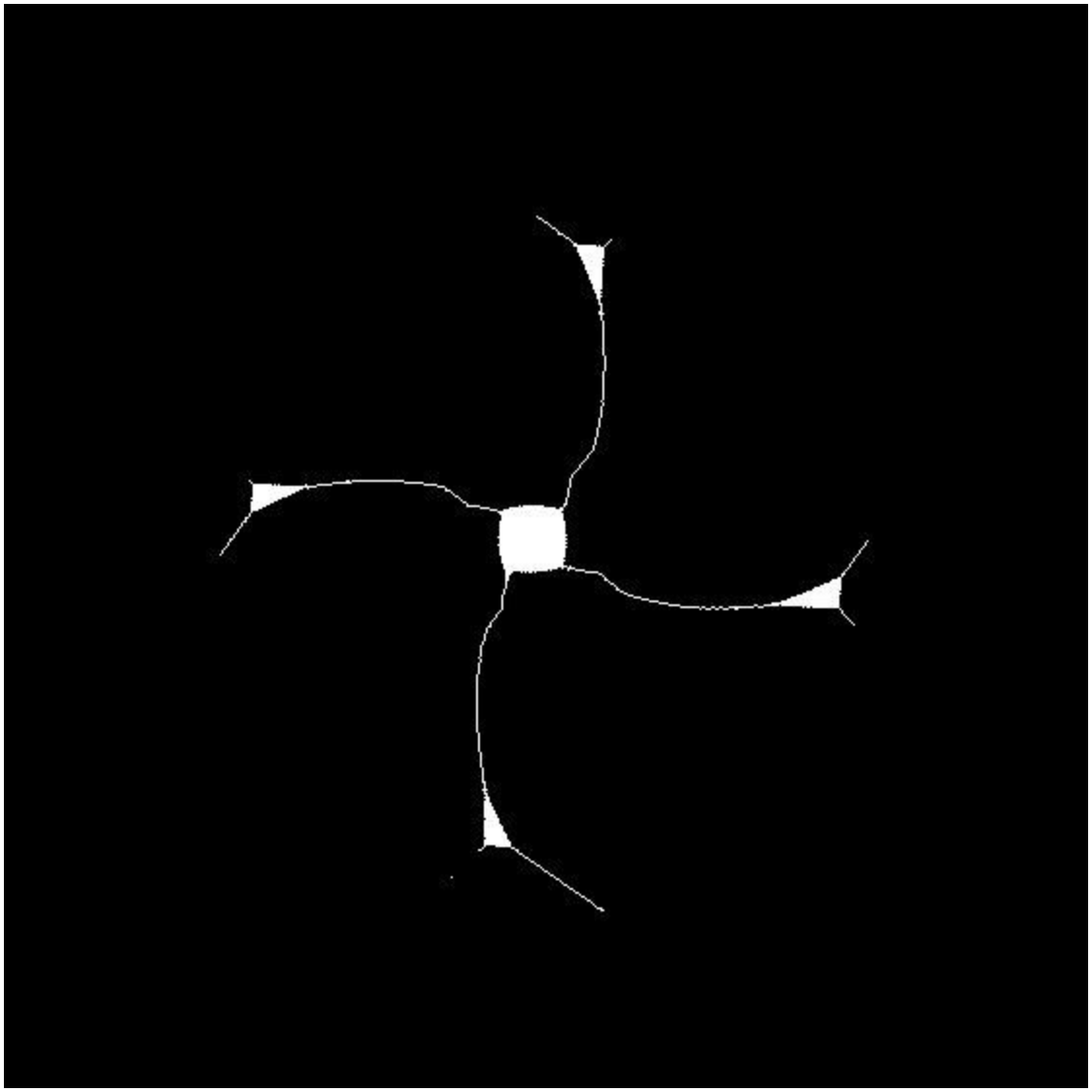


Figure 10: Shrinking of Fan after 30 iterations

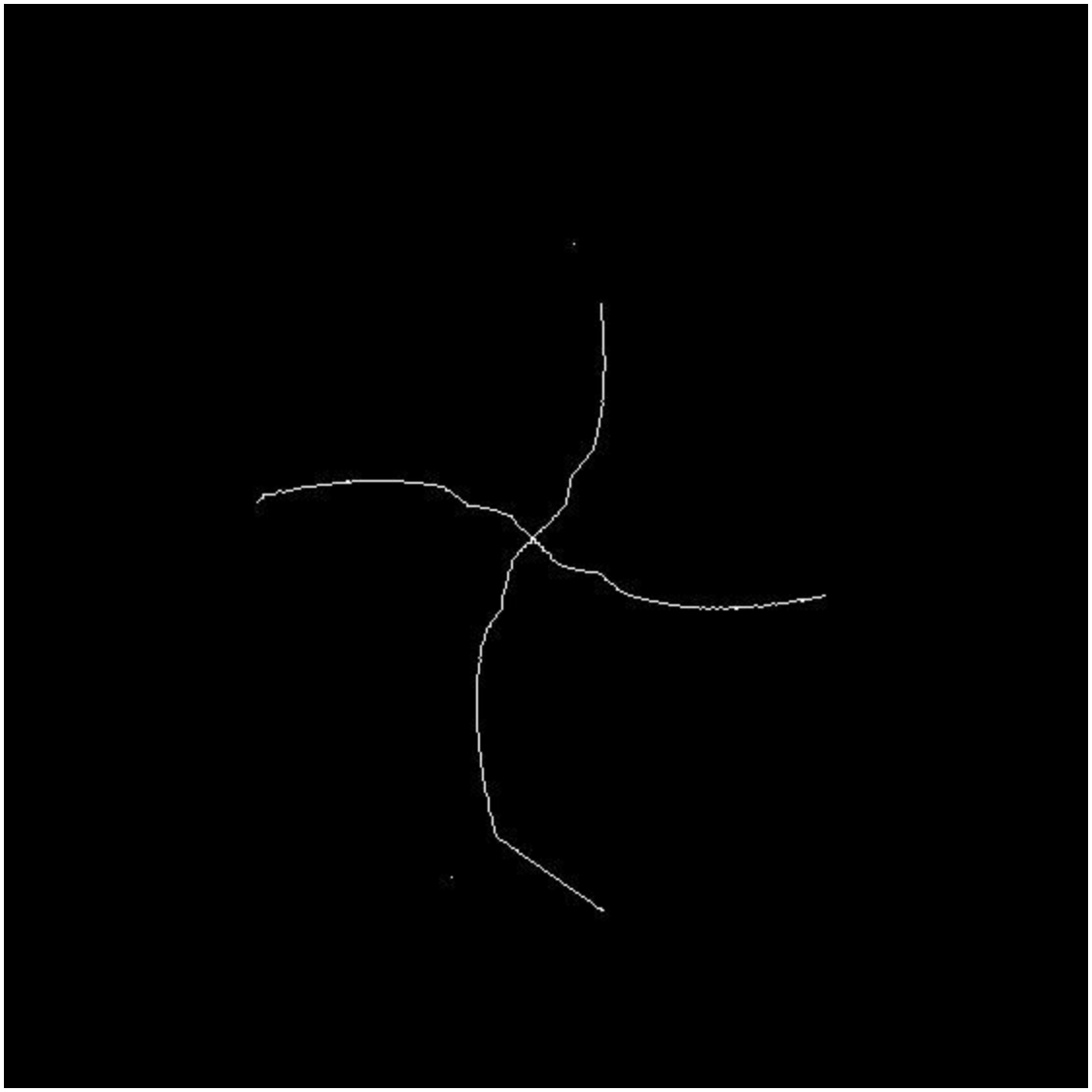


Figure 11: Shrinking of Fan after 60 iterations

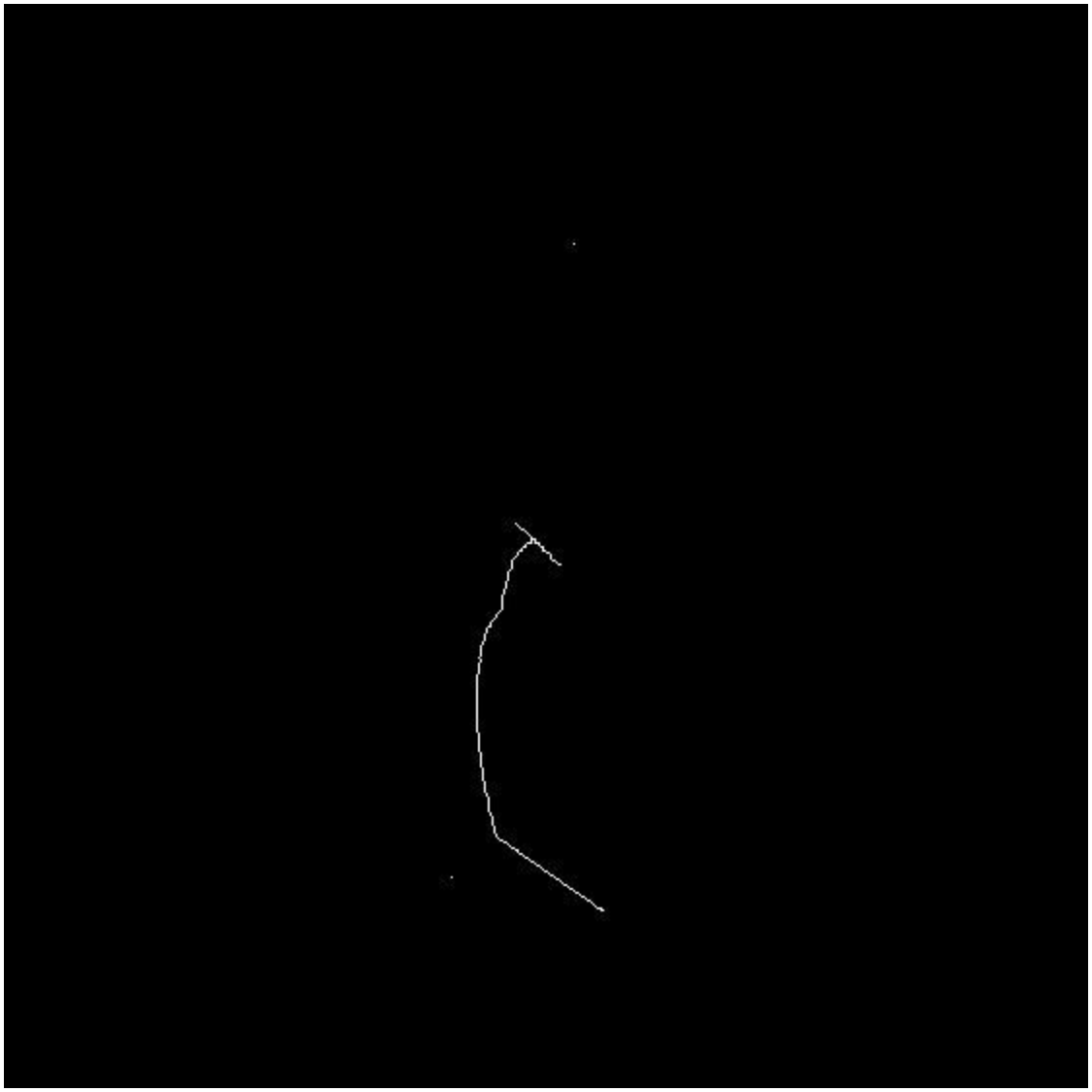


Figure 12: Shrinking of Fan after 200 iterations

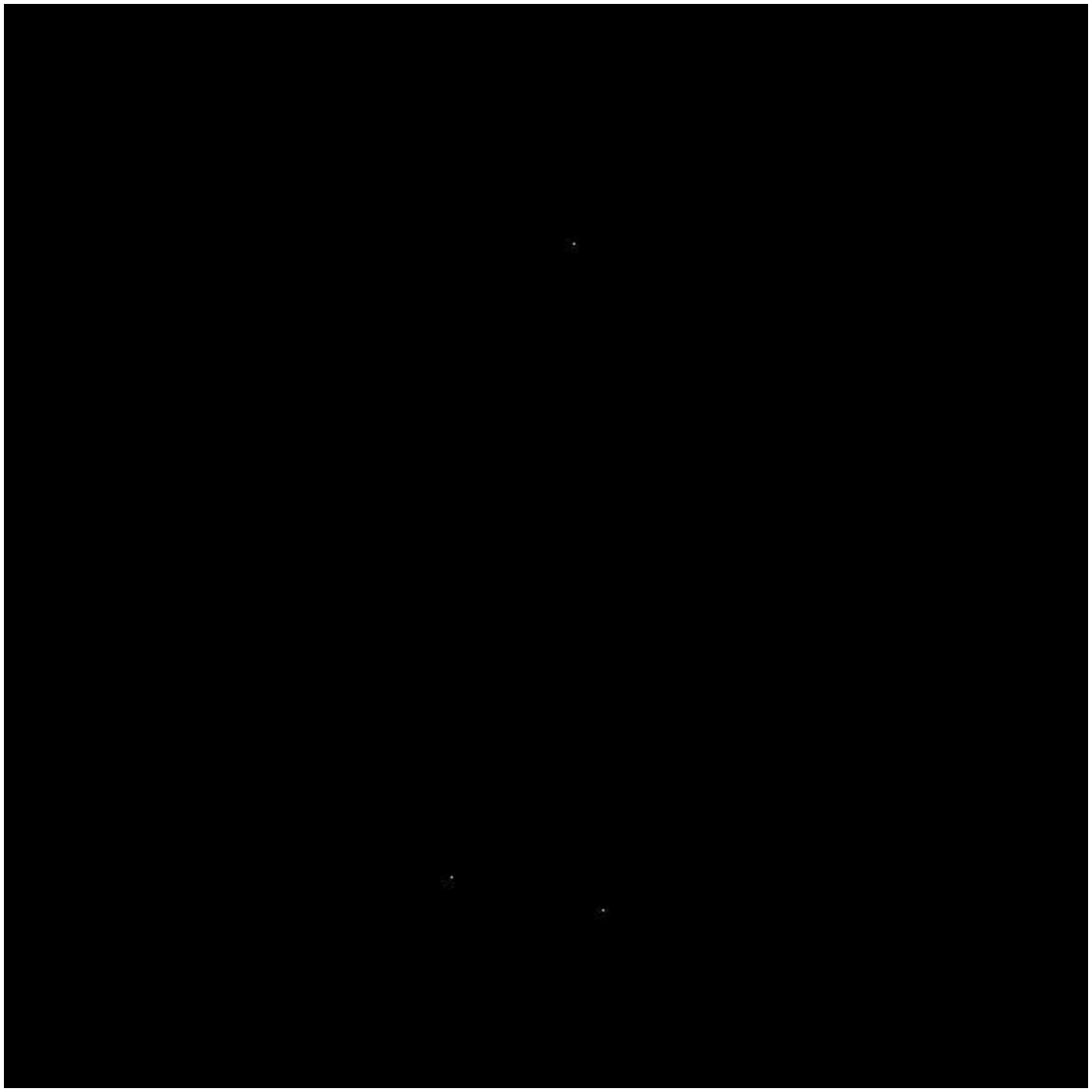


Figure 13: Shrinking of Fan after 500 iterations

Figures 14-16 shows the results of shrinking on cup image.

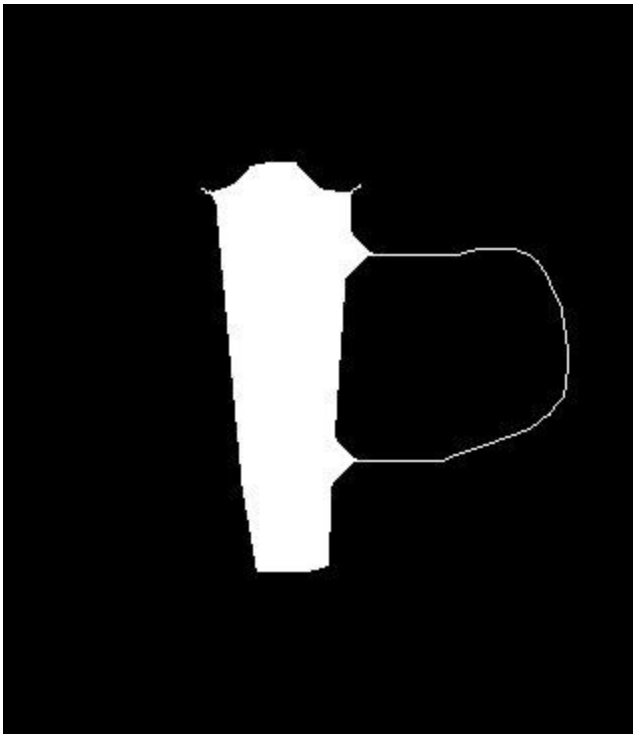


Figure 14: Shrinking of cup after 50 iterations

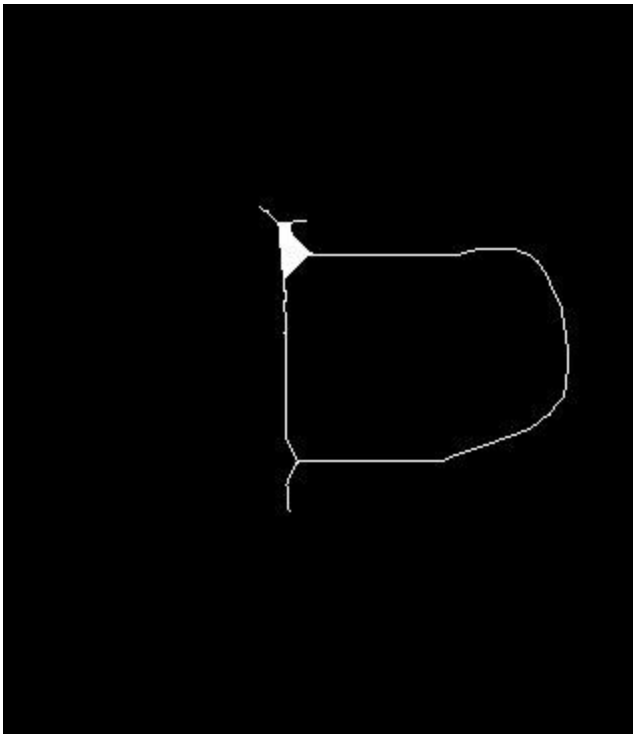


Figure 15: Shrinking of cup after 80 iterations

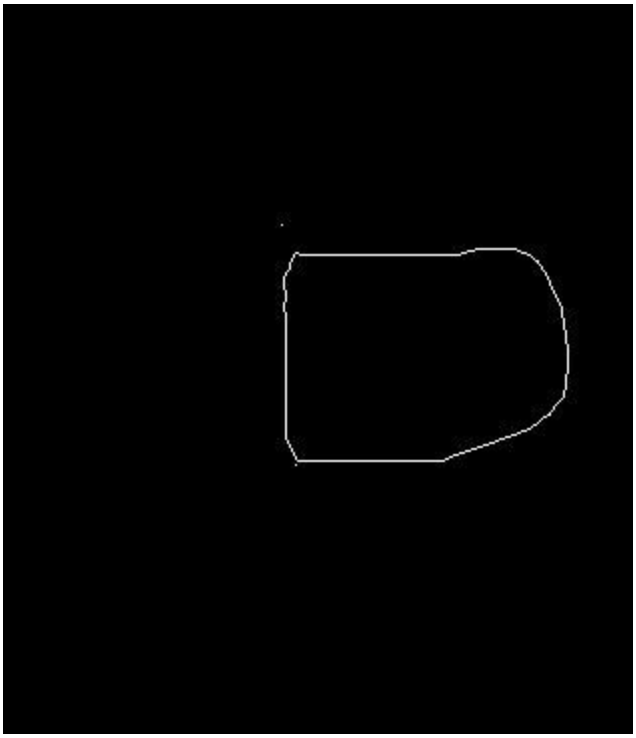


Figure 16: Shrinking of cup after 250 iterations

Figures 17-20 shows the results of shrinking of maze

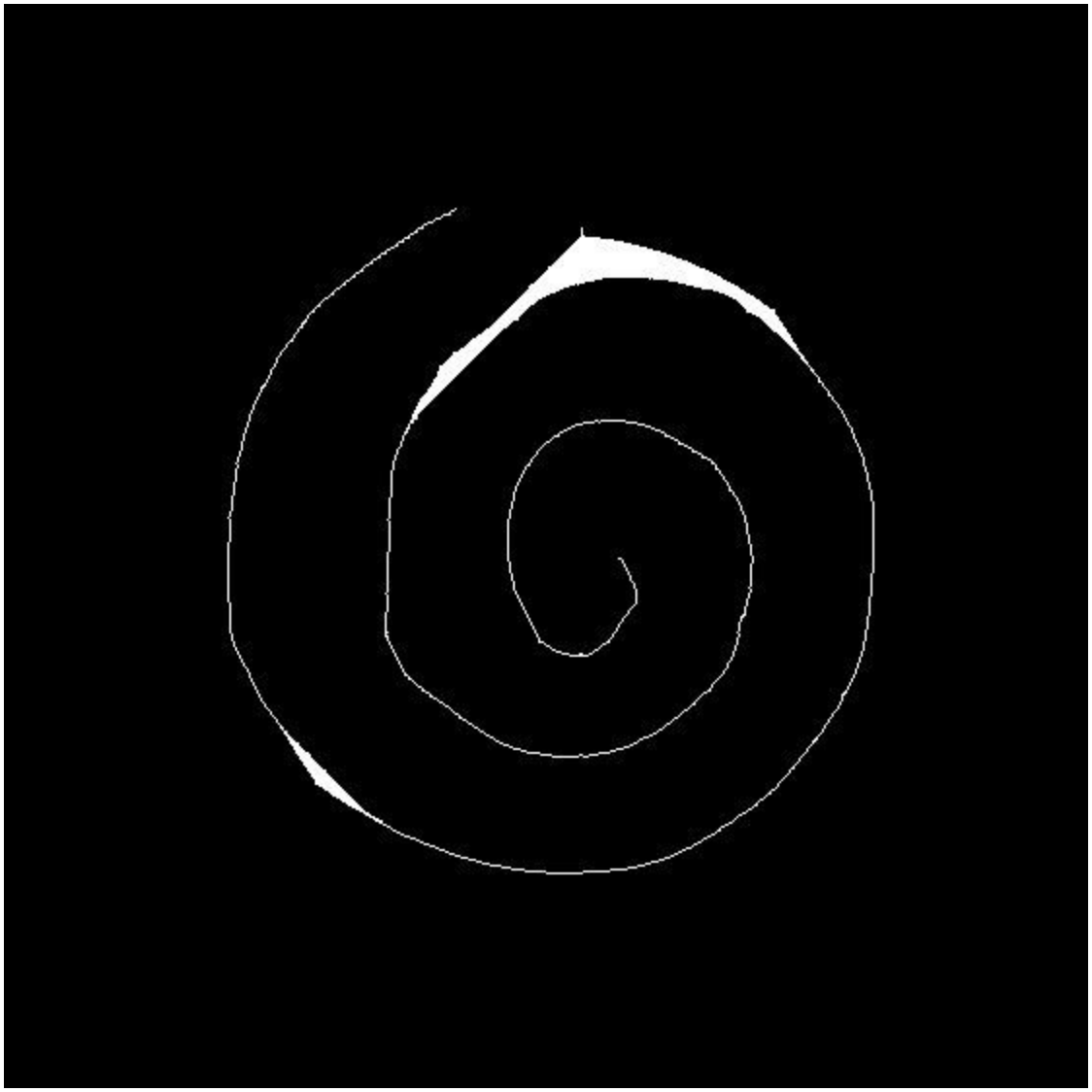


Figure 17: Shrinking of maze after 40 iterations

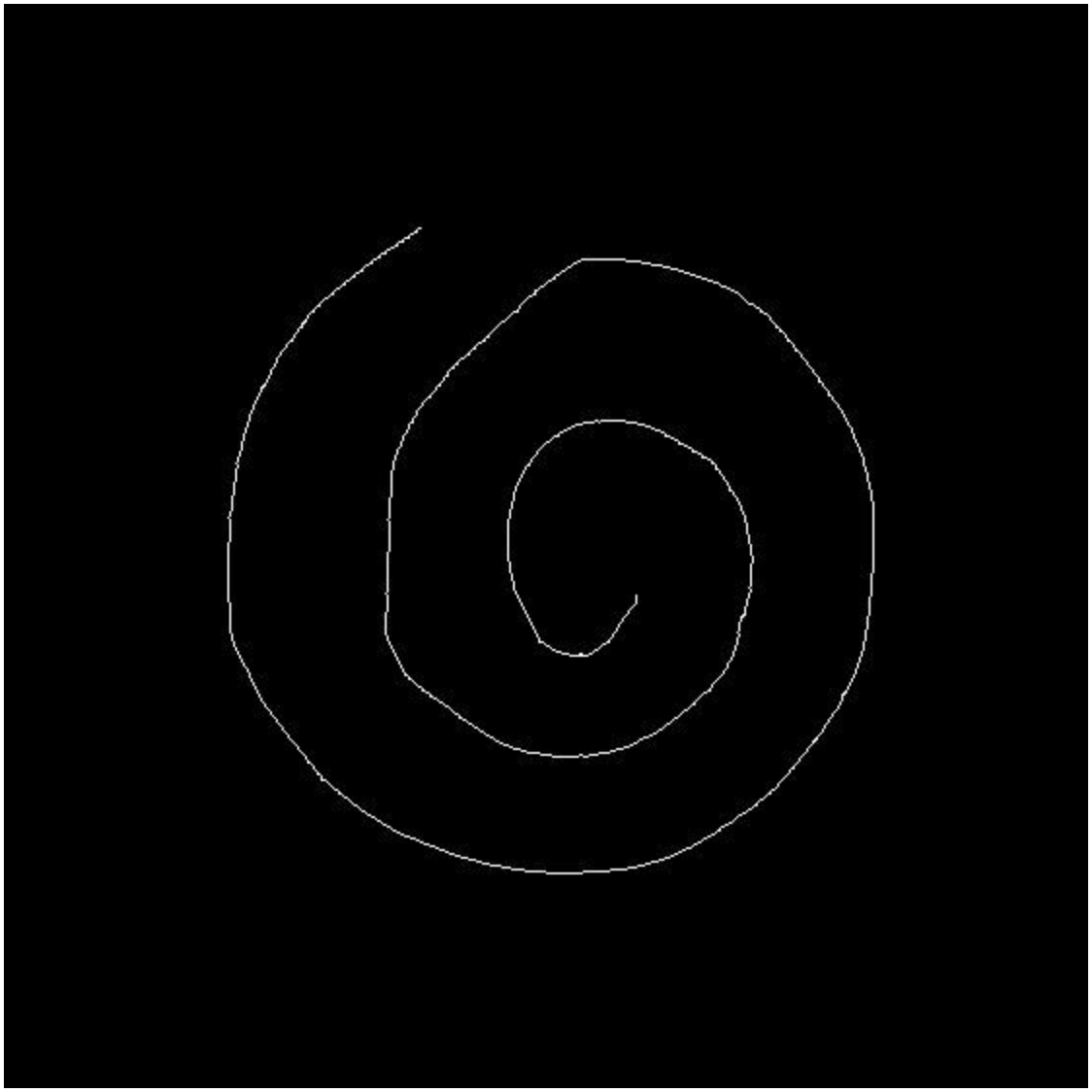


Figure 18: Shrinking of maze after 60 iterations

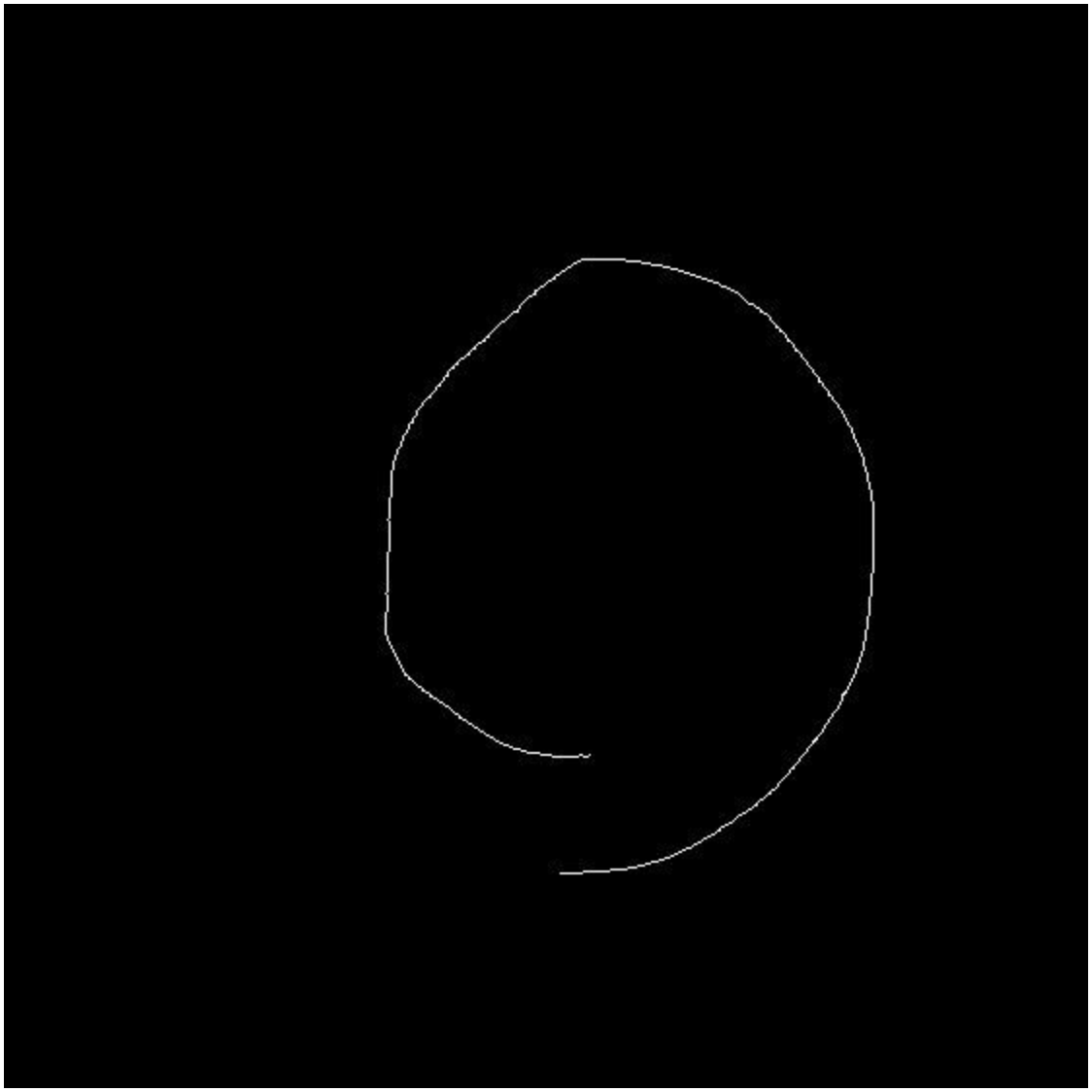


Figure 19: Shrinking of maze after 500 iterations

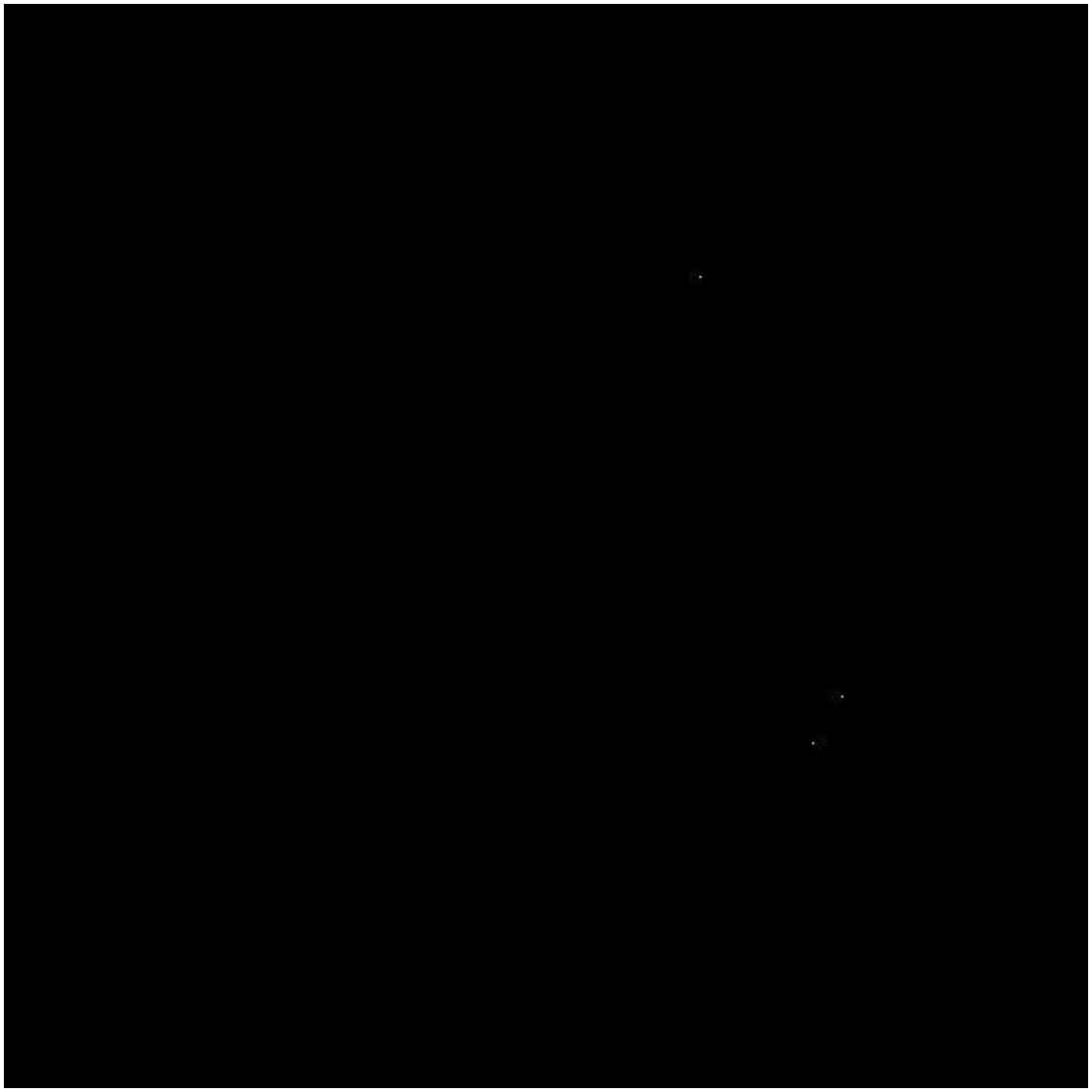


Figure 20: Shrinking of maze after 2000 iterations

2) **Morphological Thinning:**

Thinning is implemented using the lookup table and tested on fan, cup and maze image.

Figures 21-23 shows the thinning of the fan.

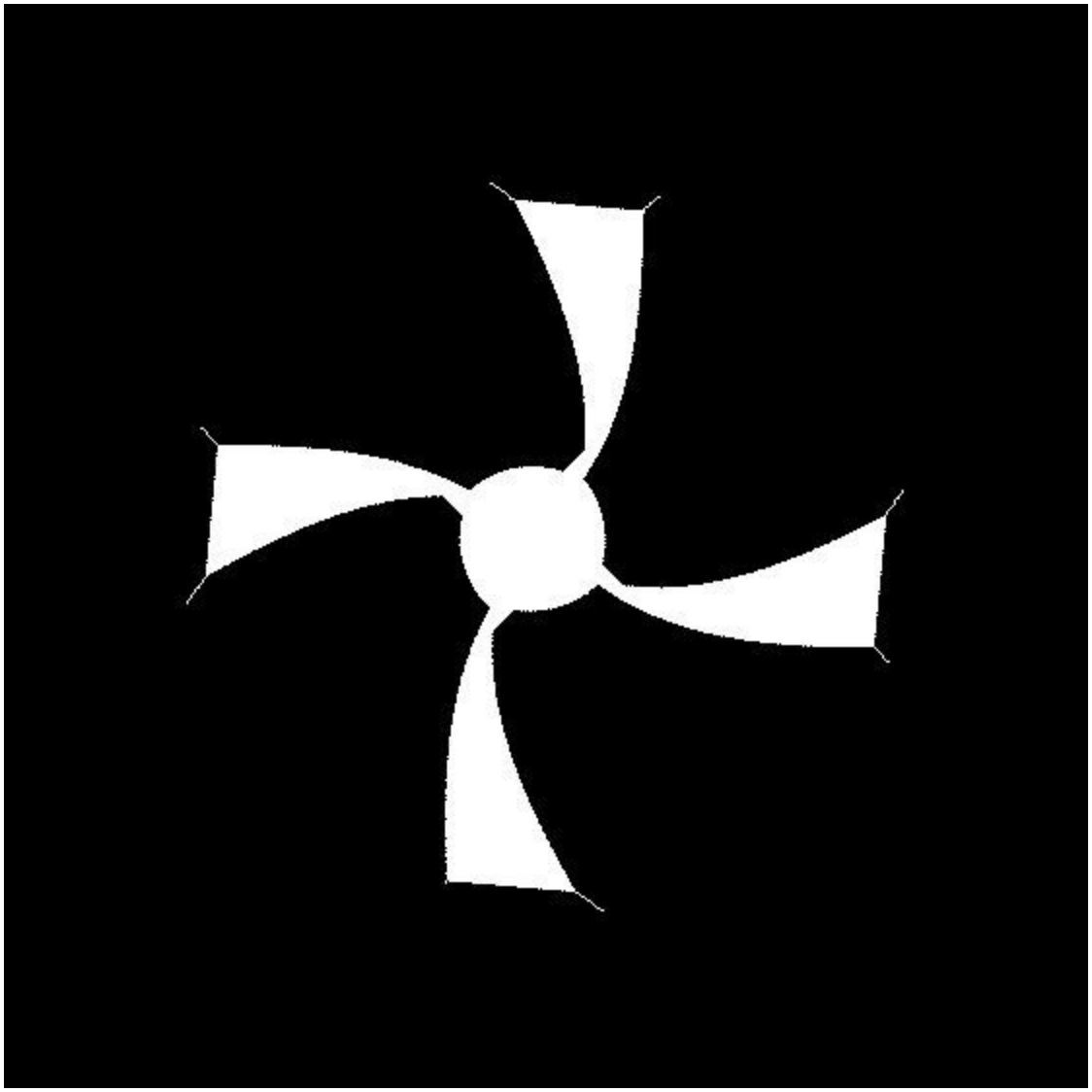


Figure 21: Thinning of fan after 10 iterations

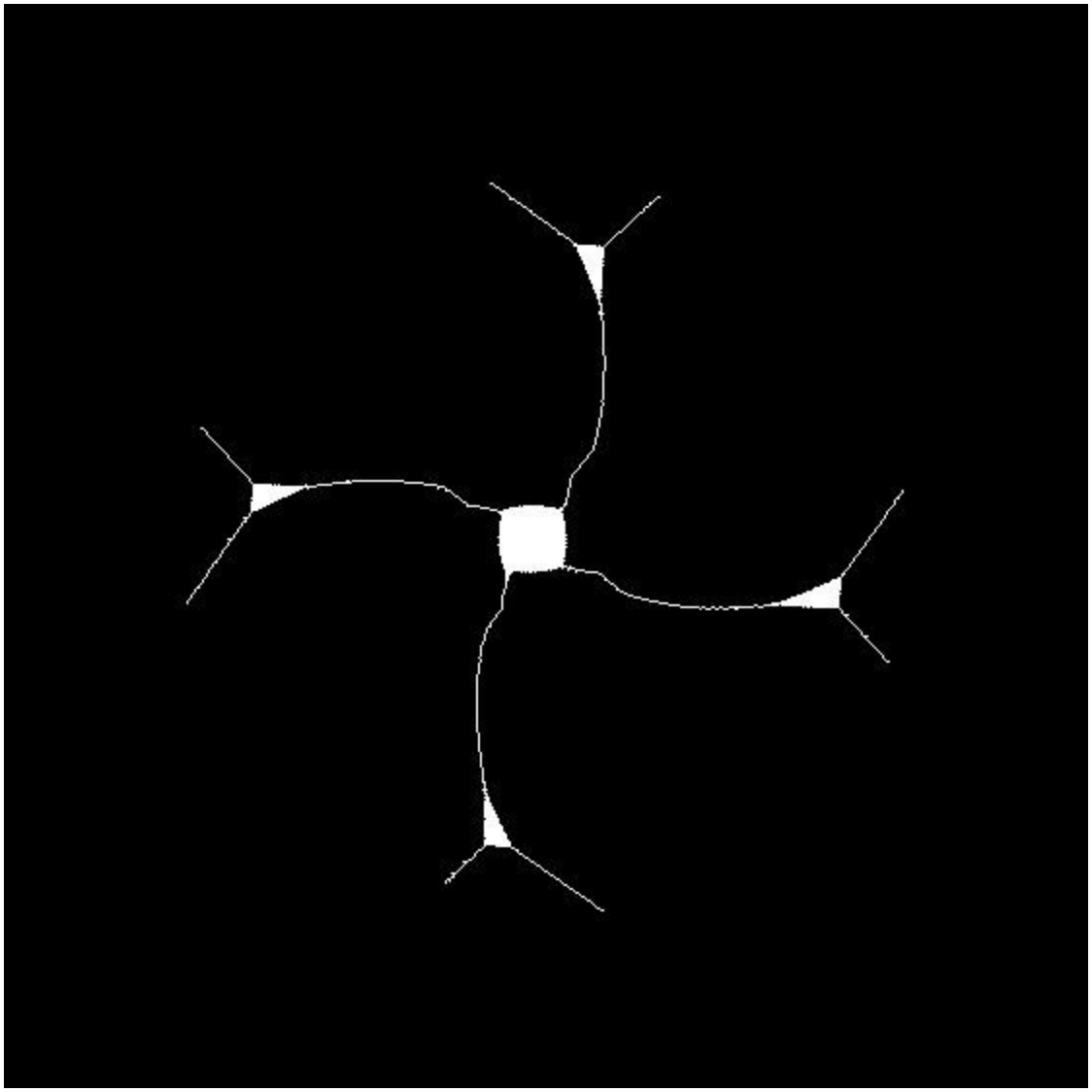


Figure 22: Thinning of fan after 30 iterations

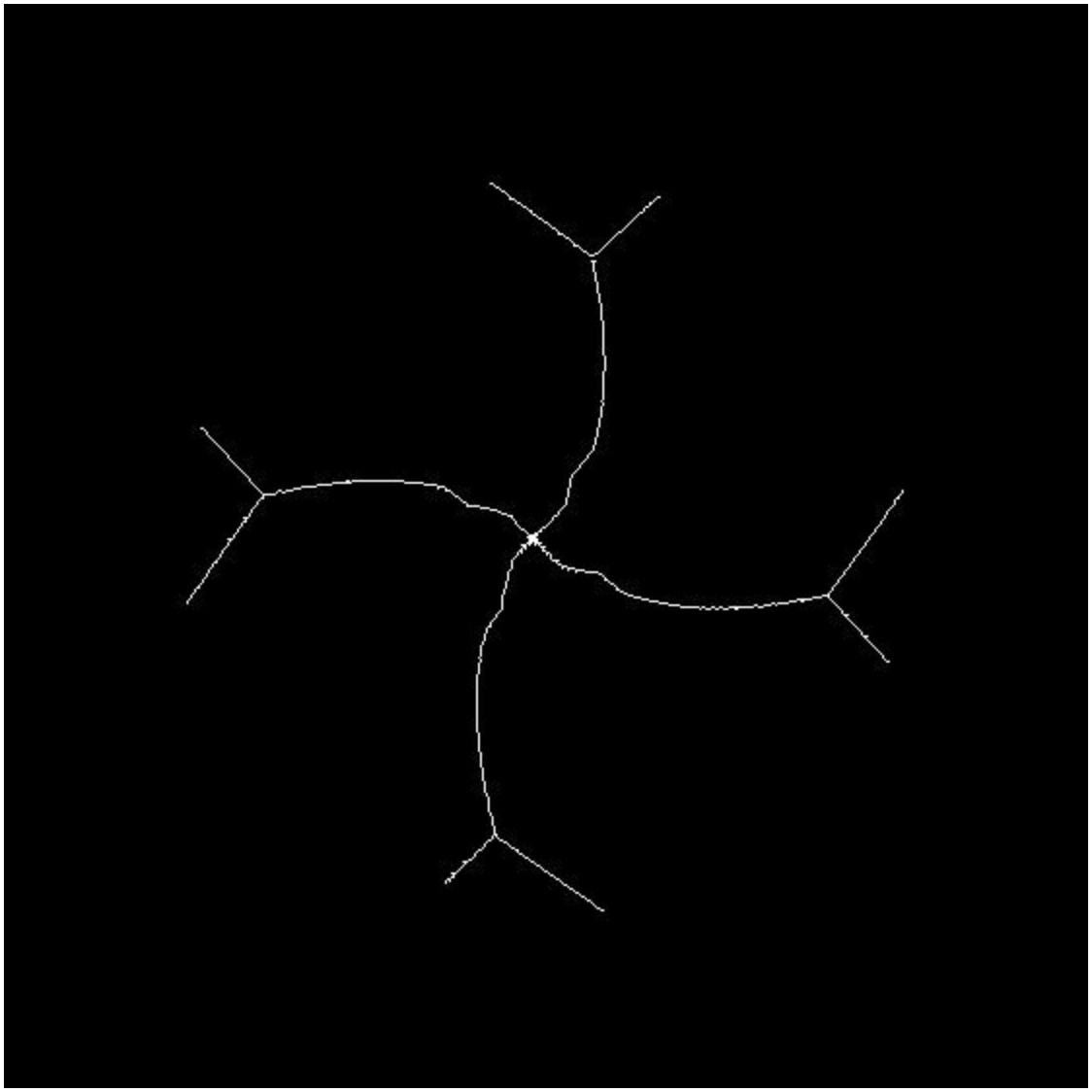


Figure 23: Thinning of fan after 45 iterations

Figures 24-26 shows the result of thinning on cup image.

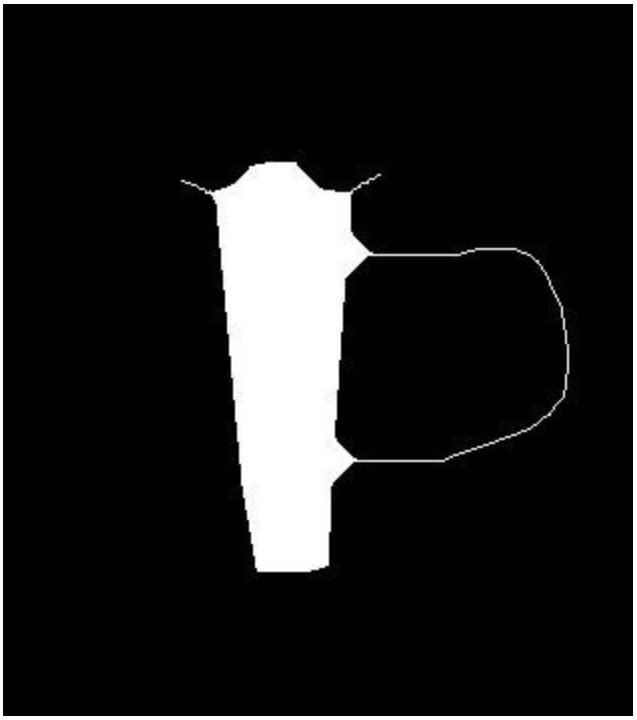


Figure 24: Thinning of cup after 50 iterations

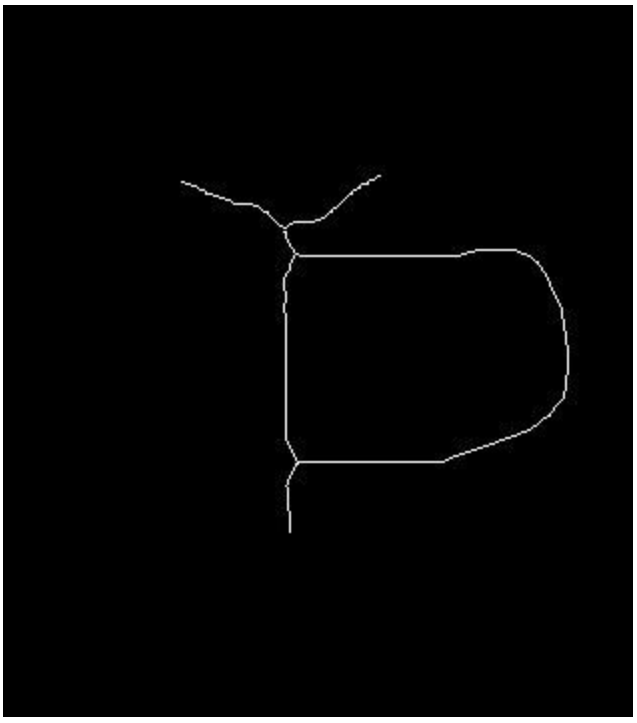


Figure 25: Thinning of cup after 100 iterations

Figure 26-27 shows the results of thinning of the maze image.

Figure 26: Thinning of maze after 20 iterations

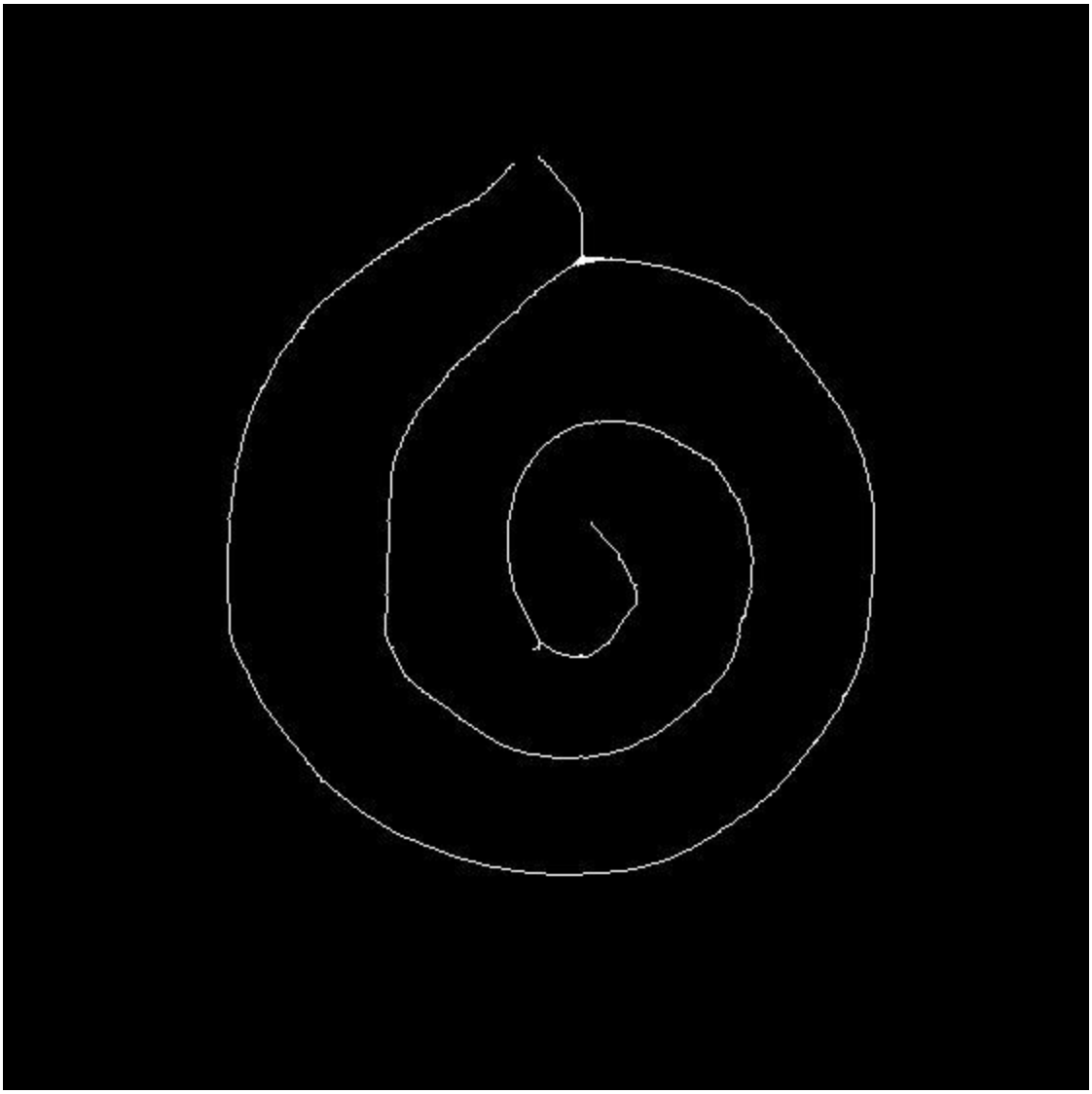


Figure 27: Thinning of cup after 50 iterations

3) **Morphological Skeletonizing:** Skeletonizing is implemented using the given pattern table and results are observed on fan, cup and maze image.

   Figures 28-30 shows the skeletonizing of the fan.

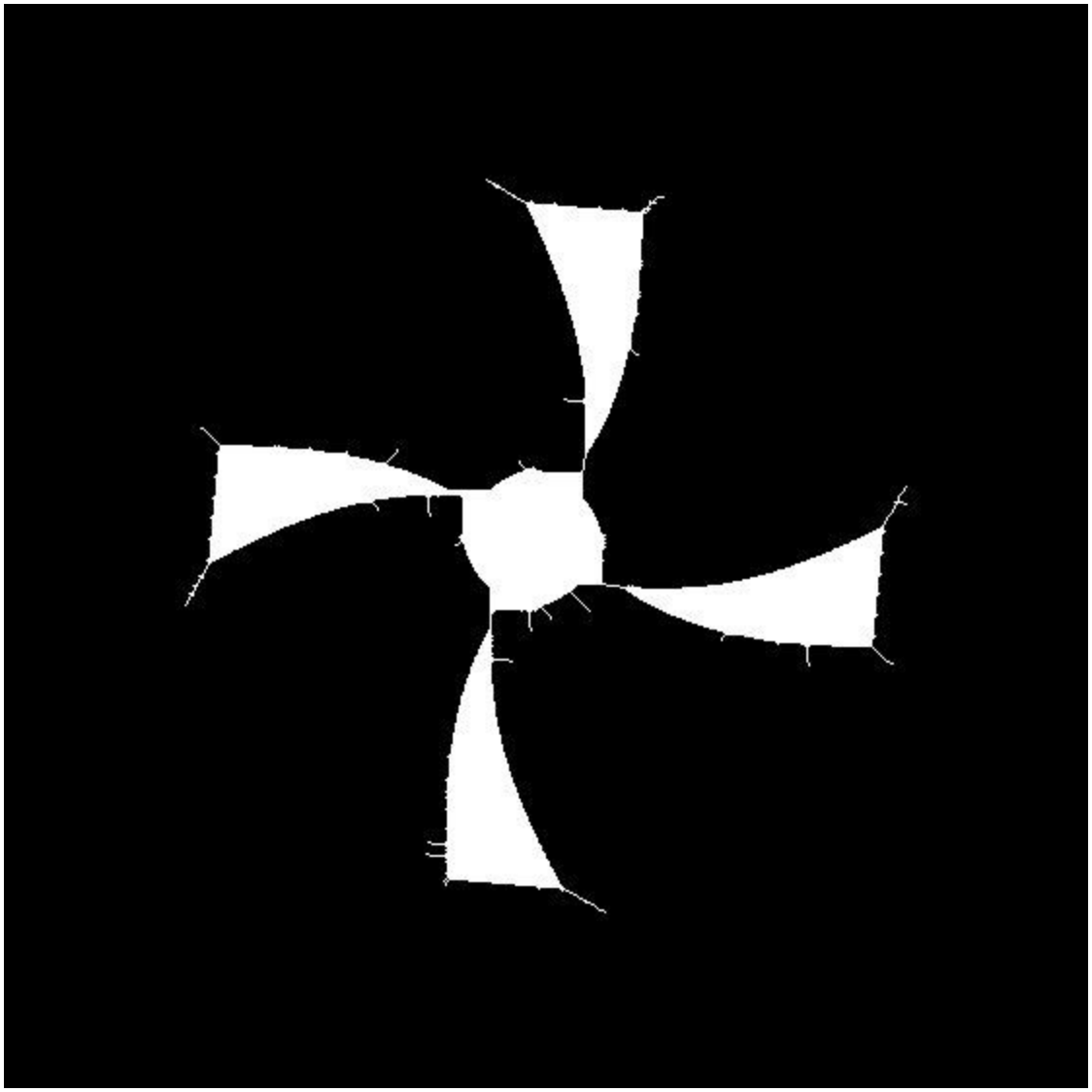


Figure 28: Skeletonizing of fan after 10 iterations

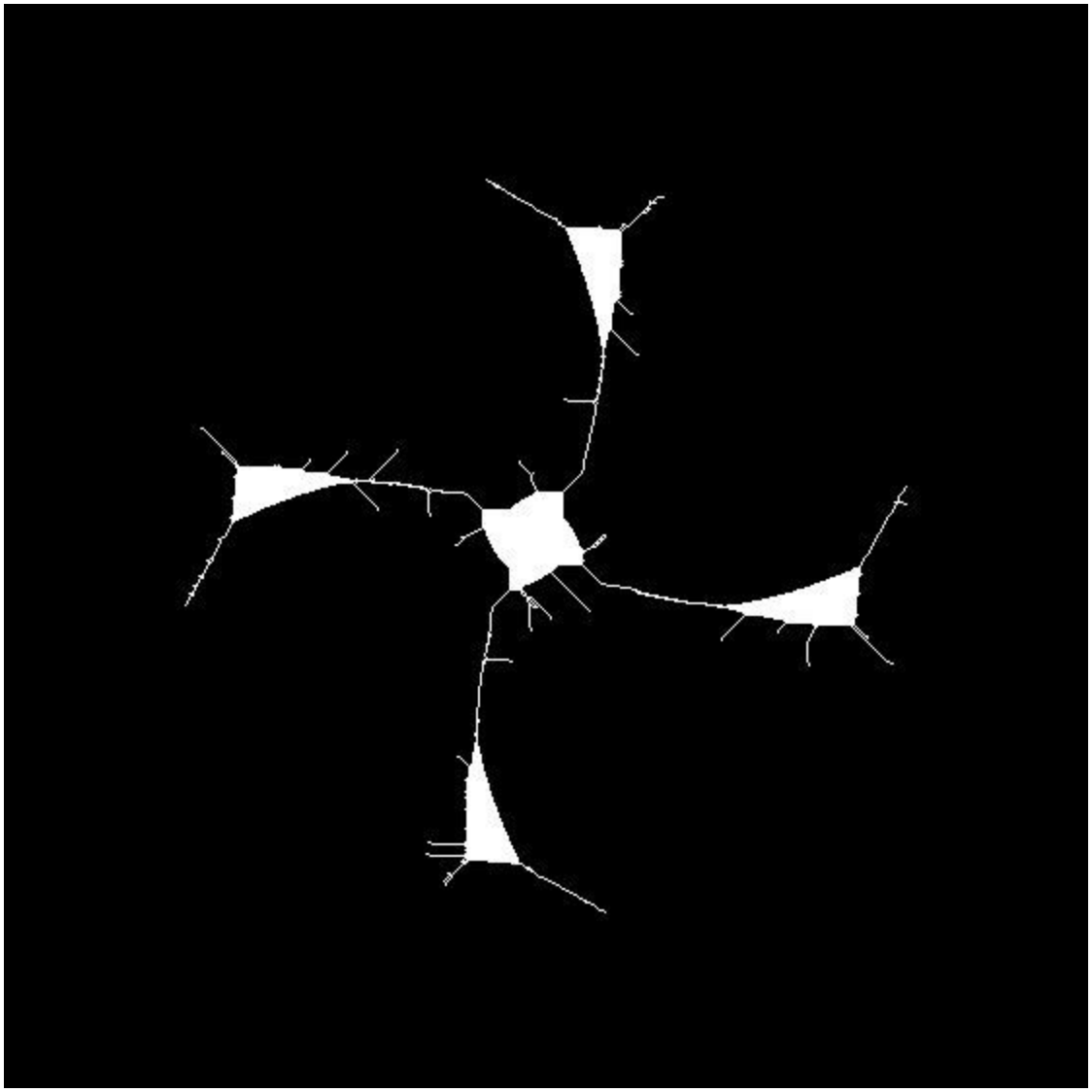


Figure 29: Skeletonizing of cup after 20 iterations

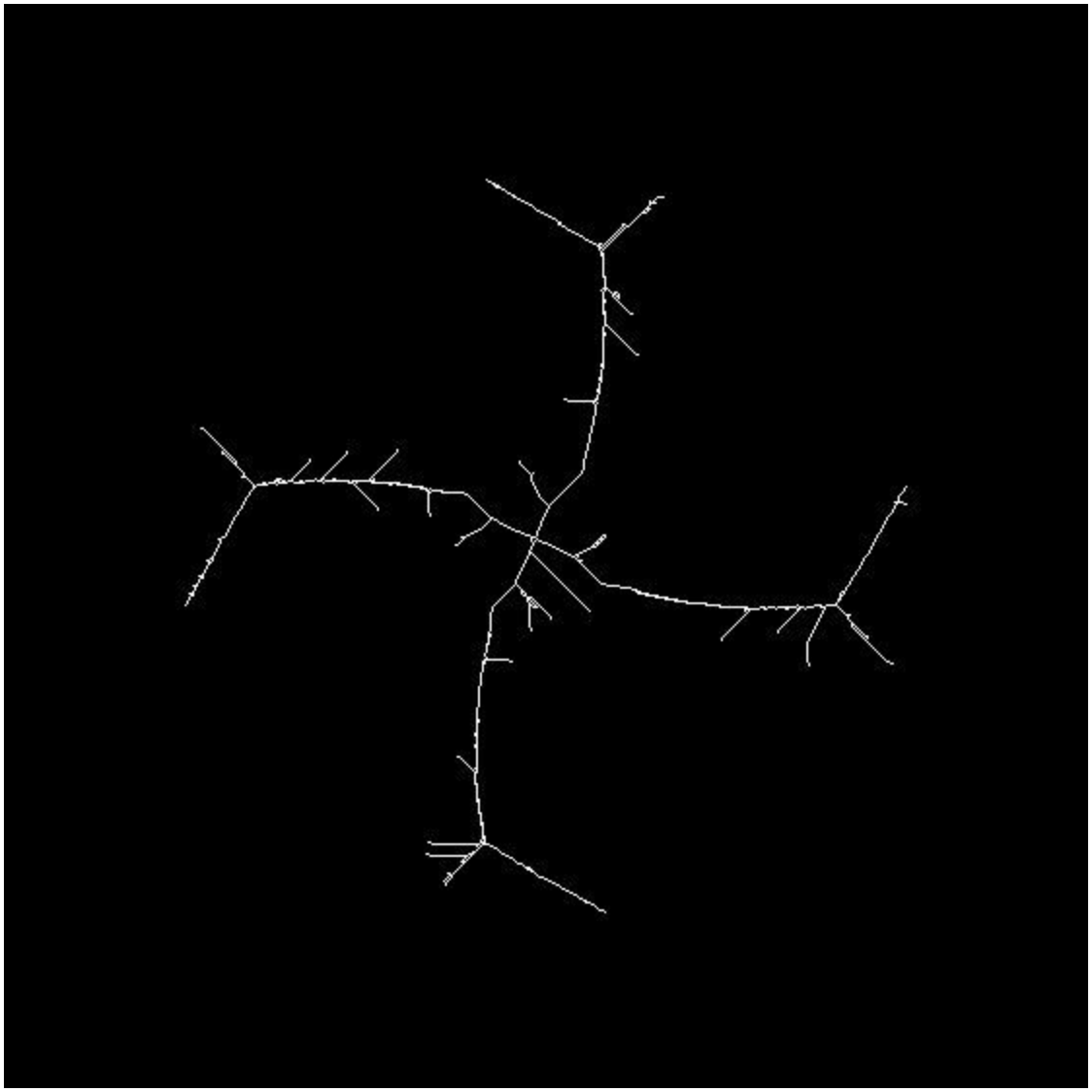


Figure 30: Thinning of cup after 40 iterations.

Figures 31-33 shows the result of thinning on cup image.

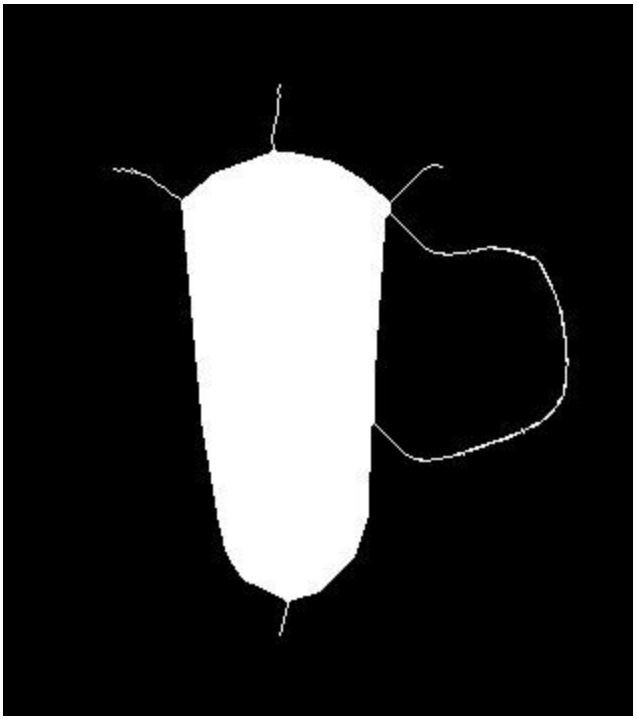


Figure 31: Skeletonizing of cup after 30 iterations

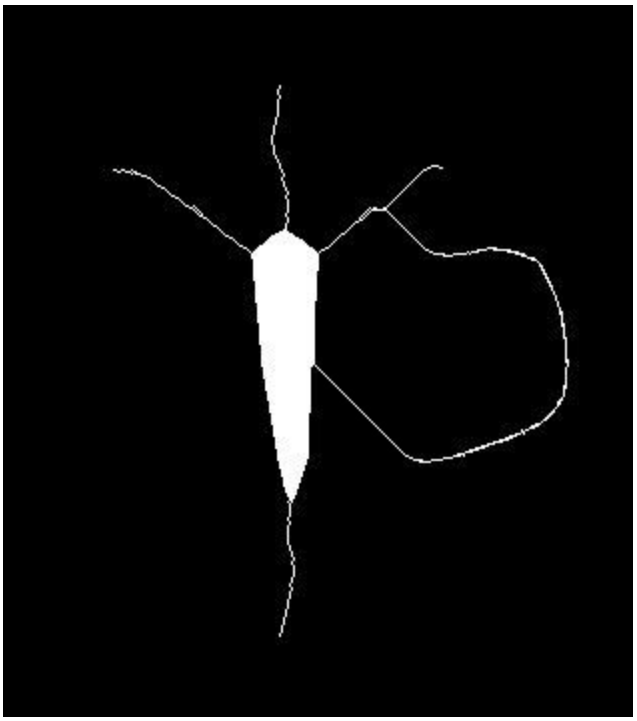


Figure 32: Skeletonizing of cup after 60 iterations

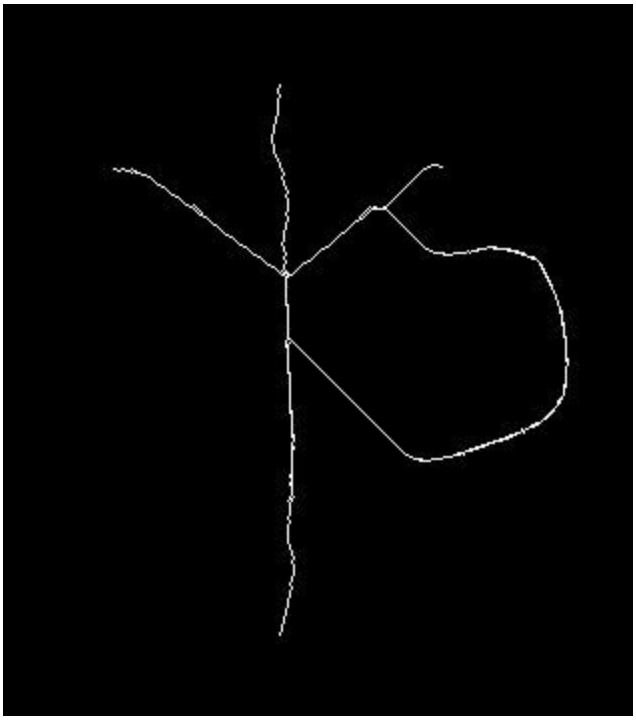


Figure 33: Skeletonizing of cup after 100 iterations

Figures 34-35 shows the skeletonizing of maze

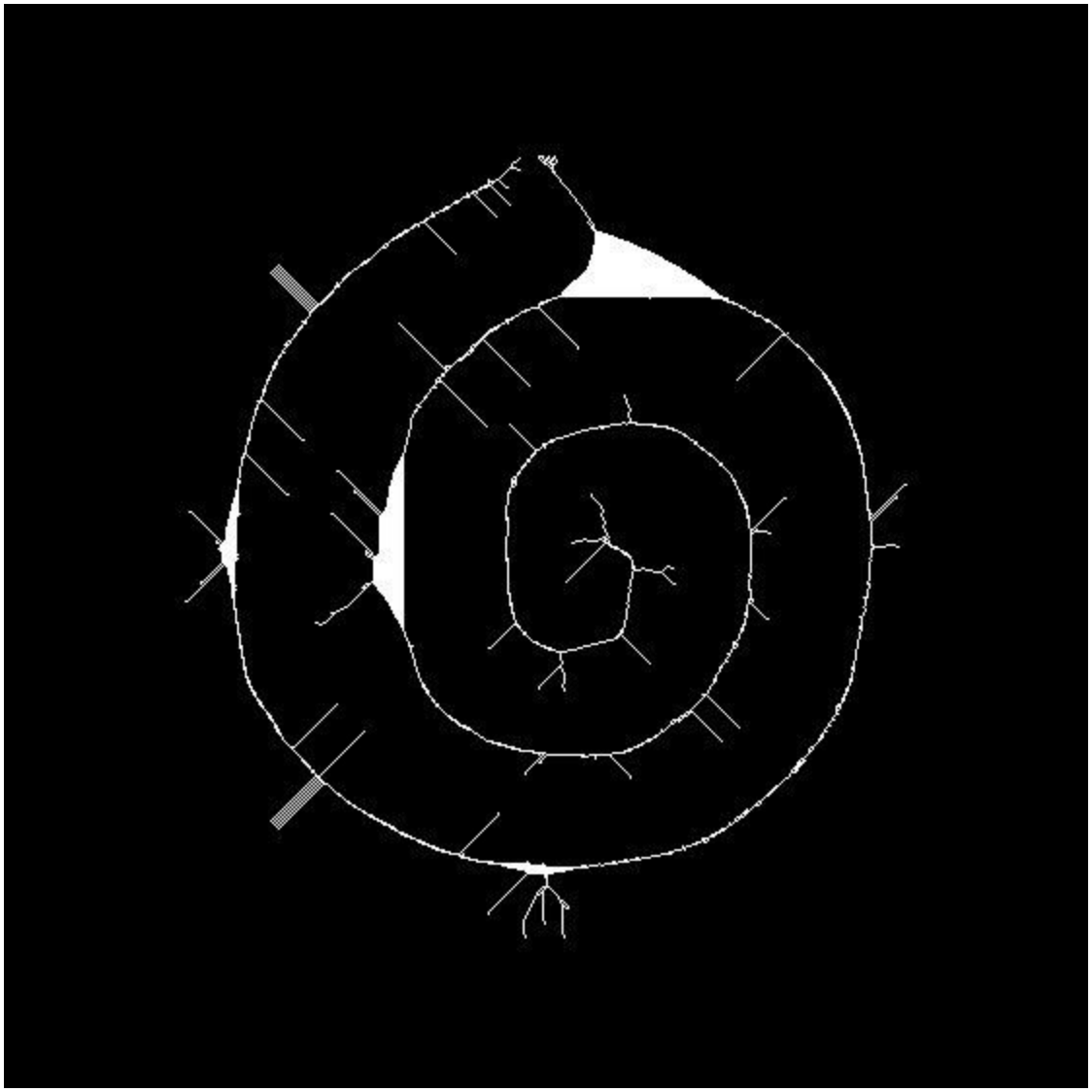


Figure 34: Skeletonizing of maze after 30 iterations

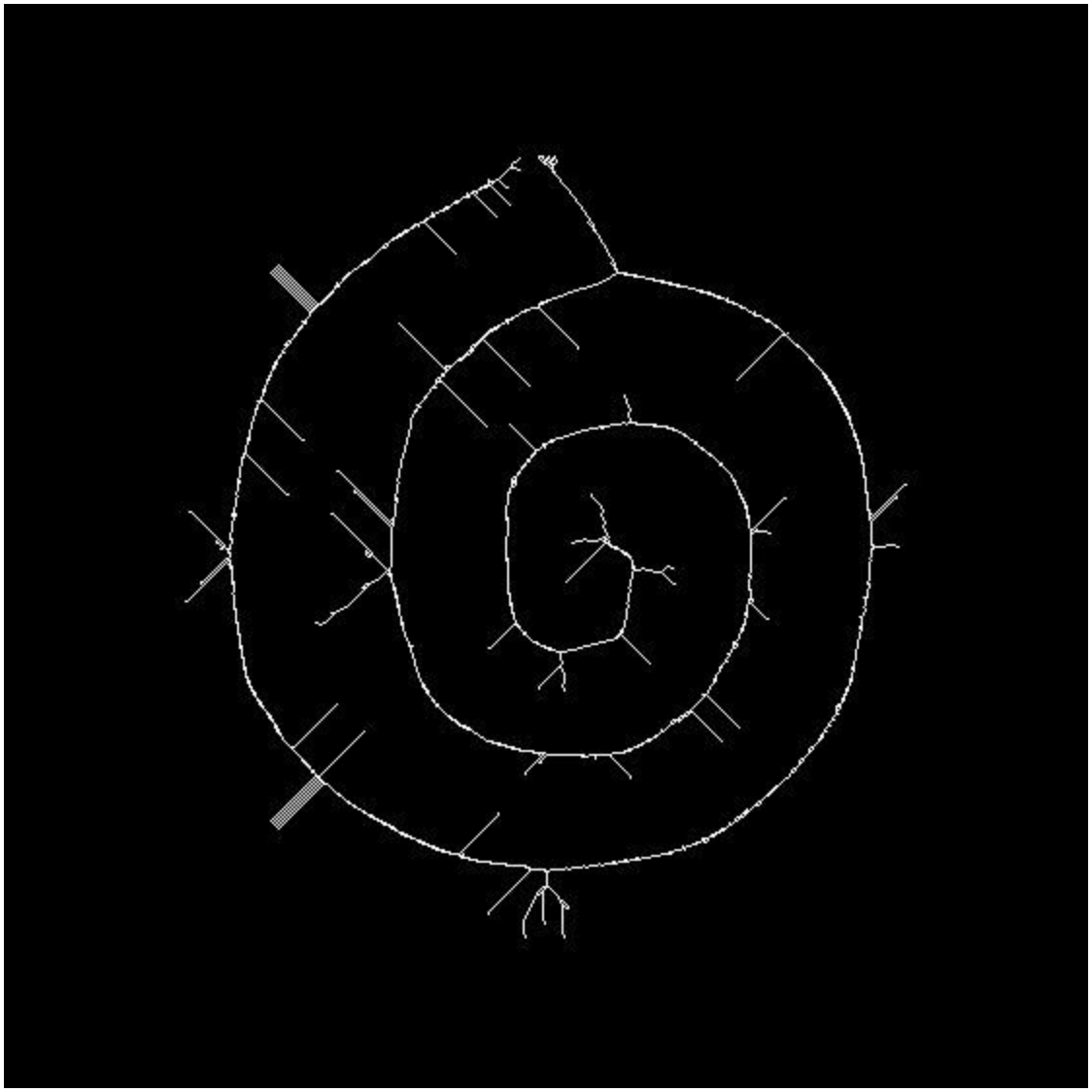


Figure 35: Skeletonizing of cup after 50 iterations

Discussion:

1) **Shrinking:**
    a) The fan image shrinks to 3 points after over **450** iterations. We can see from the intermediate results, shrinking usually takes more time. Ideally we should get a single point but due to some problem in the pattern table, we have got 3 isolated points.
    b) The cup gets shrunk to 2 components. One is a single isolated point. Another is a connected ring. We got a connected ring because we had hole in the cup image. It took 250 iterations for the cup image to shrink.

c) Compared to cup and fan, maze took a lot of iterations to shrink. It took almost 2000 iterations to complete. Even the Maze image got shrunk to 3 isolated points. But ideally, it should be a single isolated point. The reason for more iterations is that the image has spiral shapes which usually won't erase in the first stage.

2) Thinning:

a) Thinning operation on fan image resulted in mesh shape which corresponds to the fan image. It took 100 iterations to get the final result.
b) Thinning operation of the cup image also resulted in a mesh shape where we can see the structure of the object is preserved. It took 100 iterations to get the final result.
c) Thinning of the Maze image resulted in a thing spider web. It took 100 iterations.
d) We can see that Thinning took fewer iterations than Shrinking. This is because we don't have many filter patterns for less bond numbers in thinning unlike in shrinking.

3) Skeletonizing:

a) Skeletonizing operation on fan image gave the actual skeleton structure of the fan image. It took only 50 iterations.
b) The result of skeletonizing is almost similar to thinning. But we have extra branches in skeletonizing. It took 100 iterations to get the result.
c) Skeletoning on the maze image gave thin spider web along with additional radial lines. It took 50 iterations.

2) Counting of Stars: There are a total of 114 connected components in the image. So, the total number of stars is **114.**

Different star sizes can be calculated by plotting the histogram of the star sizes.

Figure 36 shows the histogram of the star size.

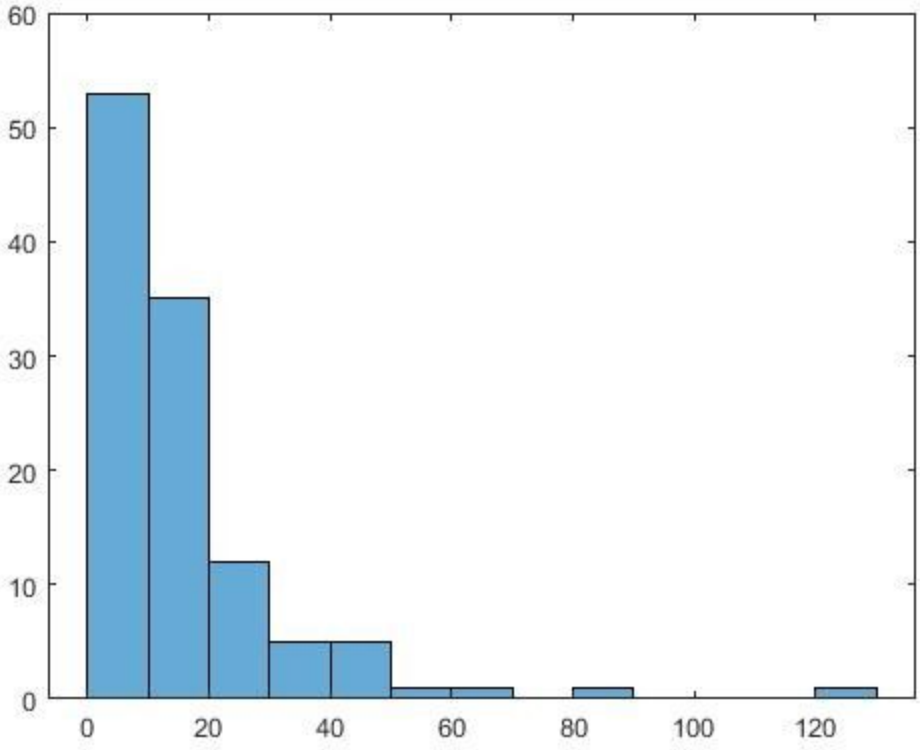


Figure 36: Histogram of star sizes.

So from the above histogram, we can infer that the total number of different star sizes is **9**.

The frequency plot of these stars sizes is shown in Figure 37.

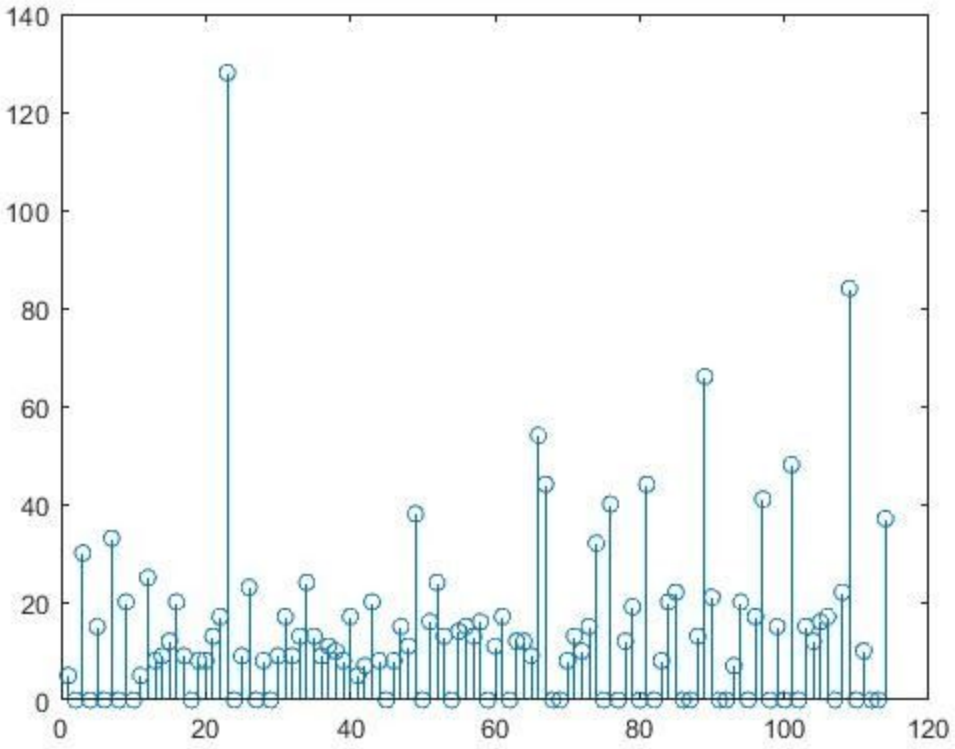


Figure 37. Frequency of star sizes

Alternate Approach (Other than Morphological Operations):

We can use **Connected Component Analysis** to solve the same problem.

Algorithm:

1) This is a 2 parse algorithm. We iterate through the image as a **raster** pattern.
2) Initialize the object label to 1
3) If we find 1 in any of the pixel location,
    a) Check the already scanned neighbors and if the neighbors are all 0's, assign a new object label to the pixel and increment the object label
    b) If there is only one non zero value, assign the same value to the location

c) If we have multiple non zero values, find the minimum value and assign. The rest of the values are stored in the map table to change the values in 2nd parse.

4) In the second parse, change the keys in the map table with the stored values.
5) After 2 parses, we will have different labels to each connected component in the image.
6) If we find the frequency of each connected component, we get the size of the star.

3) PCB Analysis:

a) Counting the number of holes: Figure 38 shows the shrinked image of the PCB with 1 white dot in places of holes.

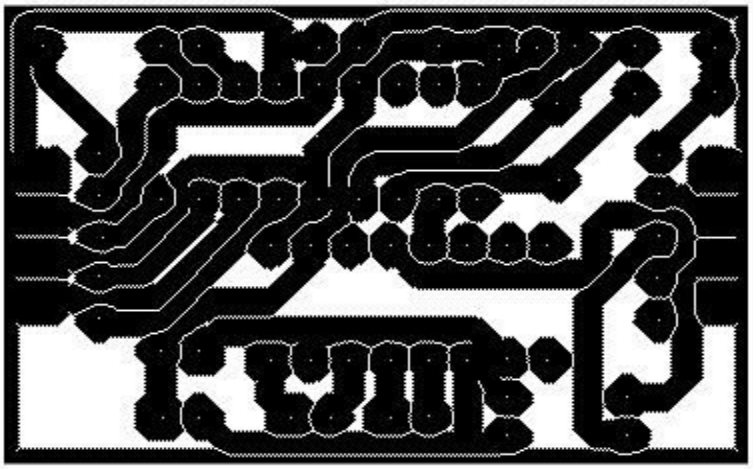


Figure 38: Shrinking on PCB Image

Answer: After shrinking, we iterated through the image to find white dots with 8 neighbors being 0. The answer will be **72.**

b) Counting the number of pathways in the PCB: Figure 39 shows the inverted image of the PCB

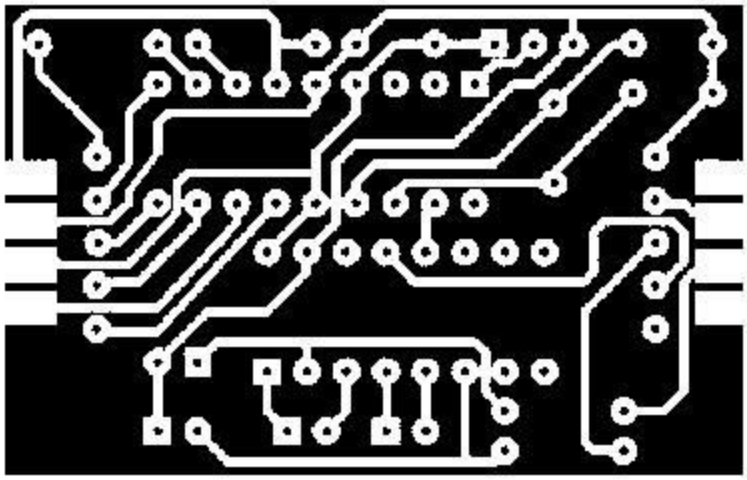


Figure 39: Inverted Image of PCB

Answer: We found connected components on the inverted image to find the number of pathways. The answer is **24.**

4) Defect Detection:

The approach is discussed in the previous section. I Wasn't able to implement it due to time constraints.

References:

1. https://support.wolfram.com/25330?src=mathematica.
2. https://www.cis.rit.edu/class/simg782/lectures/lecture_02/lec782_05_02.pdf.
3. https://www.comp.nus.edu.sg/~cs4340/lecture/imorph.pdf
4. https://courses.cs.washington.edu/courses/cse576/16sp/Slides/10_ImageStitching.pdf
5. http://matthewalunbrown.com/papers/ijcv2007.pdf
6. https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.
7. https://en.wikipedia.org/w/index.php?title=Scale-invariant_feature_transform&oldid=929569116
8. https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html.
9. https://cseweb.ucsd.edu/classes/sp04/cse252b/notes/lec04/lec4.pdf.
10. http://mathproofs.blogspot.com/2005/07/mapping-square-to-circle.html.
11. http://www0.cs.ucl.ac.uk/staff/G.Brostow/classes/IP2008/L3_Morphology.pdf