EE569 Digital Image Processing

HOMEWORK #3

Issued: 02/17/2020 Due: 03/03/2020

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #3
March 3, 2020

Problem 1: Geometric Image Modification (50%)

(a) Geometric warping (Basic: 20%)

(b) Homographic Transformation and Image Stitching (Basic: 20%)

1.1 Abstract and Motivation

When digitally manipulating an image, it's nature for us to do image warping, by which any shapes that are portrayed in a certain image have been significantly distorted. Warping can render individuals a great many useful applications ranging from correcting image distortion to achieving creative purposes.

Image stitching plays an essential role in photography, which enables humans to combine multiple existing images with overlapping areas of view to produce a segmented image that is desirable for watching. People commonly perform that operation with the help of computer software, which usually leads to an almost completely stitched panorama.

In order to understand the basic mechanism of geometric warping and image stitching, in this report, I am about to implement algorithms to achieve them. For the reason that I have to get solutions of a system of linear equations, I will use MATLAB 2019b to finish this homework. Necessary resulting images will be presented later in my report.

1.2 Approach and Procedures

(a) Geometric warping

In this part, I will create and realize a spatial warping technique that transforms an square image into an output image that is disk-shaped. My approach is stated as follows:
Step1: Transform the input image whose size is 512 by 512 into 513 by 513 by padding zeros.
Step2: Get the location of the center pixel of the modified image from the step1.
Step3: Split the system into four parts having the same size in term of the center pixel: A, B, C, and D as shown below:

$$image = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Step4: Count the number of pixels whose distance from the center pixel is less than the radius of the final wrapped image in every row for four parts.
Step5: Mapping pixels outside the circle into their specific locations of the inner circle in every row of four parts according to the results from the step4.

Step6: Converge four parts (A, B, C, and D) into one image and transform the image into the original size 512 by 512. The resulting images are shown in Fig. 1.4, Fig. 1.5 and Fig. 1.6 respectively.

What's more, in order to recover the original images, I will implement an algorithm to realize the reverse spatial warping to each resulting image.

Step1: Transform the input image whose size is 512 by 512 into 513 by 513 by padding zeros.

Step2: Get the location of the center pixel of the modified image from the step1.

Step3: Split the system into four parts having the same size in term of the center pixel: A, B, C, and D as shown below:

$$image = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Step4: Count the number of pixels whose distance from the center pixel is less than the radius of the final wrapped image in every row for four parts.

Step5: Make pixels outside the circle get their specific intensity values from the inner circle in every row of four parts according to the results from the step4. Then, do the bilinear interpolation to make images look better.

Step6: Converge four parts (A, B, C, and D) into one image and transform the image into the original size 512 by 512. The resulting images are shown in Fig. 1.7, Fig. 1.8 and Fig. 1.9 respectively.

It can't be denied that there some slight differences between resulting reverse images and original images. Later I will discuss about it.

(b) Homographic Transformation and Image Stitching

Generally speaking, the process of homographic transformation and image stitching can be summed up to the following 5 steps:

Step 1: Select 4 pairs of control points from the left image with middle image, and the middle image with the right image respectively by using the built-in functions from [1]:

detectHarrisFeatures, *extractFeatures*, and *matchFeatures*.

Step 2: Find a homograph mapping matrix by applying homographic transformation, which is stated below.

The 4 pairs of control points, which stem from two different camera viewpoints, are under a potential projection that is realized by a linear transformation:

$$P_{middle} = H_1 P_{left}$$

$$P_{middle} = H_2 P_{right}$$

where H_i is a 3x3 homographic transformation matrix, P_{left}, P_{middle}, and P_{right} denote the left image, the middle image and the right image respectively. Generally, we have

$$P_2 = HP_1$$

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

In order to estimate matrix H, I will do the following steps:

• First of all, fix $h_{33} = 1$ so that I can only determine 8 parameters.

- Utilize 4 pairs of control points in two images to build 8 linear equations.
- Solve the 8 equations to get the 8 parameters by MATLAB.
- Get the transform matrix H.

Step 3: Wrap one image onto the other by projecting all points from one image to another with the help of the transform matrix H.

Step 4: applying the interpolation technique to make the resulting images look better.

Step 5: Create a new image, which should be big enough to hold the panorama, and composite the two wrapped images into it properly.

I will use the MATLAB to implement the homographic transformation and image stitching.

1.3 Experimental Results

Fig. 1.1: The original “hedwig” image

Fig. 1.2: The original “raccoon” image

Fig. 1.3: The original “bb8” image

Fig. 1.4: The resulting “hedwig” image by geometric mapping

Fig. 1.5: The resulting “raccoon” image by geometric mapping

Fig. 1.6: The resulting “bb8” image by geometric mapping

Fig. 1.7: The resulting “hedwig” image by inverse geometric mapping

Fig. 1.8: The resulting “raccoon” image by inverse geometric mapping

Fig. 1.9: The resulting “bb8” image by inverse geometric mapping

Fig. 1.10: The original left image

Fig. 1.11: The original middle image

Fig. 1.12: The original right image

Fig. 1.13: The resulting stitched image

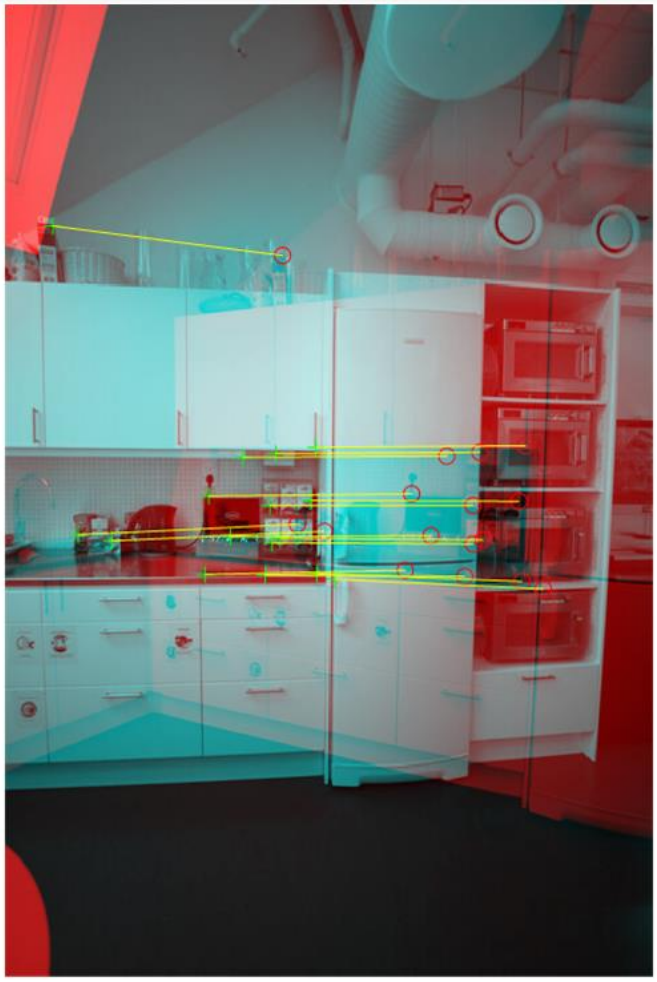


Fig. 1.14: The result of feature match between the left image and the middle image

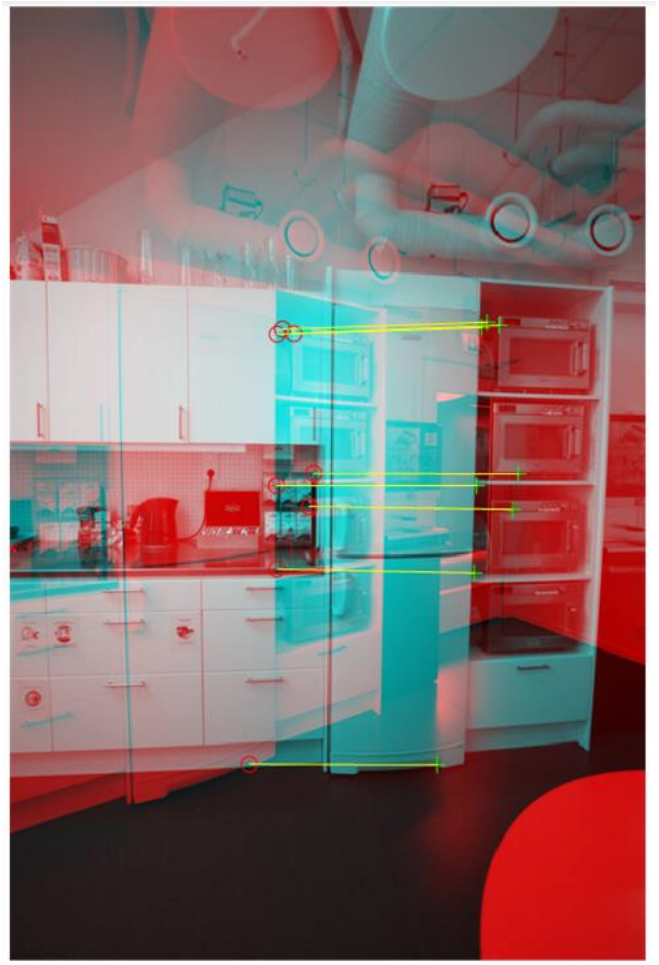


Fig. 1.15: The result of feature match between the middle image and the right image

1.4 Discussion

As I have mentioned in the previous report, how to evaluate images' visual quality is always a huge challenge to people, even to image scholars themselves, because the process of that evaluation is so subjective that it's almost impossible for individuals to reach a consensus about whether certain images' visual quality is good enough. It's not unapparent that in order to evaluate an image's quality objectively, we have to ask a number of people's opinions.

However, in this report, I will merely use my own judgement to evaluate the visual quality of resulting images. My discussion of experiment images of Problem 1 is stated as below.

In part (a), I use MATLAB 2019b to implement the geometric warping. I have already discussed about the approach to realizing forward and reverse spatial warping. From the Fig. 1.4, the Fig. 1.5, and the Fig. 1.6, it's not hard for us to reach a conclusion that my geometric warping algorithm is good enough to meet the requirement. From the Fig. 1.7, the Fig. 1.8, and the Fig. 1.9, at first glance, my reverse geometric warping is perfect. But after a careful watch, there are still some differences between resulting images and original images. For the "hedwig" image, the resulting one has some little artifacts around the edge of the bird's head. For the "raccoon", the resulting one's bottom boundary is different from the original one. For the "bb8", the resulting one has some white dots around the edge boundary of the robot's head. From my point of view,

that distortion is largely due to the process of interpolation. There is no denying that images will lose information when operating the forward geometric warping. The weakness of my algorithm is that the top and the bottom edges will lose most information and rows closer to the middle will lose less information. In that situation, it's inevitable for images to get black dots when having inverse warping. Hence, I use the nearest interpolation to fix that problem, which means that I try to make the black pixel get a value from its nearest 8 pixels having a nonzero value until the images have no black dots as shown in the Fig. 1.7, the Fig. 1.8, and the Fig. 1.9. But this interpolation method has some randomness, thus making it impossible for us to get images that exactly the same as the original ones.

In part (b), I use MATLAB 2019b to implement the homographic transformation and image stitching. The resulting stitched image is shown in Fig. 1.13. According to the TA's sample result, I believe my result is acceptable. The result of feature match between the left image and the middle image is shown in Fig. 1.14 and the one between the middle image and the right image is shown in Fig. 1.15. During the course of my implementation, first of all, I use the built-in function *detectHarrisFeatures* to make two object images get their points respectively. Then I use the built-in function *extractFeatures*, whose input arguments are object images and points, to get final features and valid points. Later, I use the built-in function *extractFeatures*, which I set the "MaxRatio" as 0.4 in order to get more salient points, to get output "indexPairs". Finally, with the help of valid points and "indexPairs", I am able to get four pairs of control points by randomly selecting them. After I have got control points, I am likely to create the panorama with the procedure that I have already clearly stated. The chosen 4 pairs of control points between the left image with the middle image and the right image with the middle image are shown in the Table 1 and Table 2 respectively.

Table 1: The 4 pairs of control points between the left image with the middle image

	X_1	Y_1	X_2	Y_2	X_3	Y_3	X_4	Y_4
Left	236.991	390.917	301.577	363.239	327.598	335.289	346.996	371.878
Middle	79.792	398.021	151.444	364.921	176.946	336.200	196.725	371.928

Table 2: The 4 pairs of control points between the right image with the middle image

	X_1	Y_1	X_2	Y_2	X_3	Y_3	X_4	Y_4
Right	202.441	247.686	202.111	425.042	214.817	247.230	229.904	352.926
Middle	356.465	242.626	350.960	427.924	369.824	240.862	384.043	353.194

Problem 2: Morphological processing (60%)

(a) Basic morphological process implementation (Basic: 18%)

(b) Counting games (Basic: 15%)

(c) PCB analysis (Advanced: 15%)

(d) Defect detection (Advanced: 12%)

2.1. Abstract and Motivation

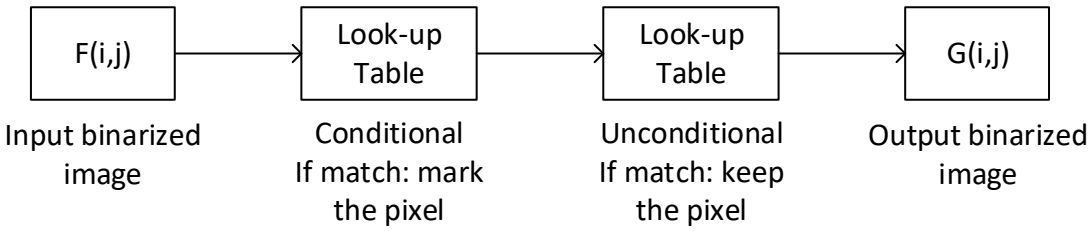
Morphological processing, one branch of digital image processing, contains a collection of non-linear operations that have relations with the shape or morphology of features of a given image. According to the definition of morphology, morphological operations only rely on the ordering of pixel values rather than their numerical values, thus meaning that individuals normally do the processing of binary images.

In this report, firstly I will use MATLAB 2019b realize 3 basic morphological processing implementations ("shrinking", "thinning" and "skeletonizing"). Then I do some applications such as counting games, PCB analysis, defect detection by using the morphological techniques.

2.2. Approach and Procedures

(a) Basic morphological process implementation

In this part, I will realize "shrinking", "thinning" and "skeletonizing" respectively. With the help of pattern tables given by the TA, I will create an algorithm whose thought is briefly summed up as follows:



Note: The given pattern tables are as follows:

Table 14.3-1: Shrink, Thin and Skeletonize Conditional Mark Patterns (M=1 if hit)

Table 14.3-2: Shrink and Thin and Unconditional Mark Patterns

Table 14.3-3: Skeletonize Unconditional Mark Patterns

The intermediate and final results with reasonable explanation will be stated later.

(b) Counting games

First of all, in order to count the total stars of the image, I will firstly shrink the image to make it only have a number of white dots. Then I will design a simple algorithm to calculate the number of white dots.

Secondly, in order to count how many different sizes are present in the image, according to the TA's hint, I will count the shrinking times for each star as its size approximately.

Thirdly, according the TA's hint in the discussion, I will use the connected-component-labeling method to solve the first two questions. The thought of that method is summed up as follows:

Step1: Input a binarized image.
Step2: On the first pass:

Iterate through each pixel of the image by column and then by row (Raster Scanning). If the pixel value is not zero, let the 4 neighboring pixels, which belongs to the current pixel, (left, left top, top, and right top) with the minimum value except 0 render to that current pixel. If there are no wanting neighbors, uniquely label the current element and move forward.

Step3: On the second pass:

Iterate through each pixel of the image by column, then by row. If the element is not the background, relabel the element with the lowest equivalence label.

Finally, the number of unique values is the number of stars and the number of each unique is the size of every star.

Later I will show my results and images.

(c) PCB analysis

First of all, in order to count the number of holes of the image, I will firstly shrink the image to make it only have a great many white dots and several contour lines. Then I will design a simple algorithm to calculate the number of white dots, thus discovering the number of holes.

What's more, in order to count how many pathways are present in the given PCB, I will firstly reverse the image and thin it, during which I have to choose the threshold value properly in order to make holes and lines not overlap each other. Then I utilize the connected-component-labeling method to get each marker's size, therefore making it possible to remove all the markers whose sizes are less than a threshold value, which means removing all the single holes. Finally, after the removing all the single holes, I use the connected-component-labeling method again to get the number of unique values, which can mean the number of pathways present in the given PCB.

Later I will show my results and images.

(d) Defect detection

Step1: Shrink the image and find 4 white dots' locations that are centers of black circle holes. Then use those four locations to find the center of this gear.

$$X_{center} = \frac{X_1 + X_2 + X_3 + X_4}{4}$$

$$Y_{center} = \frac{Y_1 + Y_2 + Y_3 + Y_4}{4}$$

Step2: Estimate the outside radius of this gear by finding the distance between the center to the top boundary in terms of the number of pixels. It's not difficult for us to find that on the outline of the gear there should be a tooth at every 30 degrees.

Step3: Find the positions of the gear teeth that are intact in this gear: firstly, split the image into four parts. Then constrain the distance and the tangent values toward the center with some fluctuations. If the 8 neighborhood pixels are all 1, that position must be a complete tooth. So, in this way we can find all the good teeth.

Step4: Find the positions of the gear teeth that are missing in this gear with the same method as the Step3. If the 8 neighborhood pixels are all 0, that position must be a missing tooth. So, in this way we can find all the missing teeth.

2.3. Experimental Results

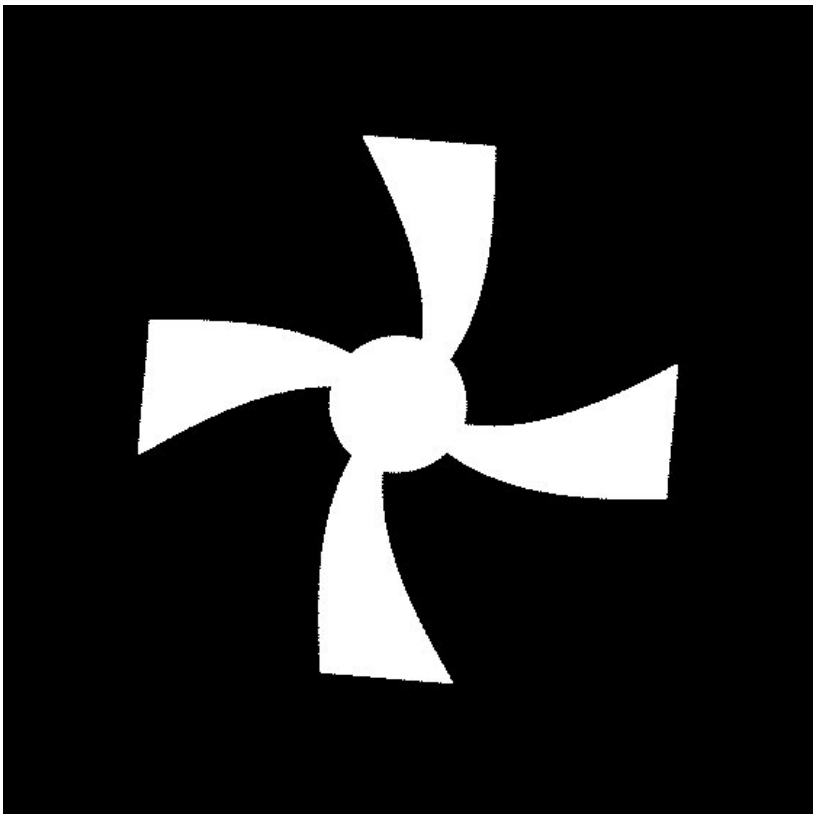


Fig. 2.1: The original “fan” image

Fig. 2.2: The original “cup” image

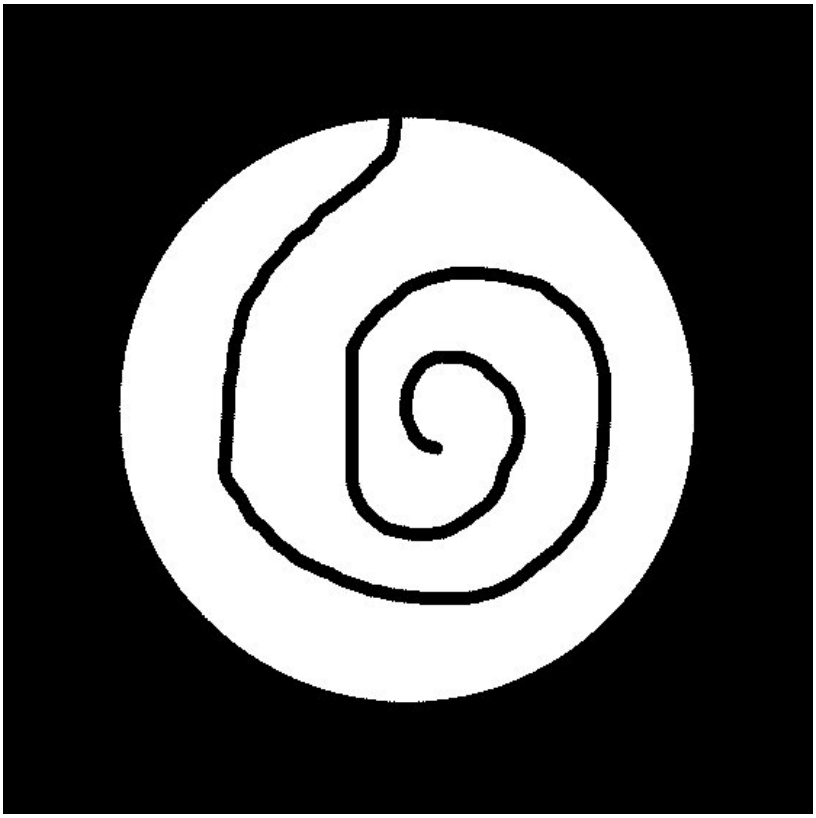


Fig. 2.3: The original “cup” image

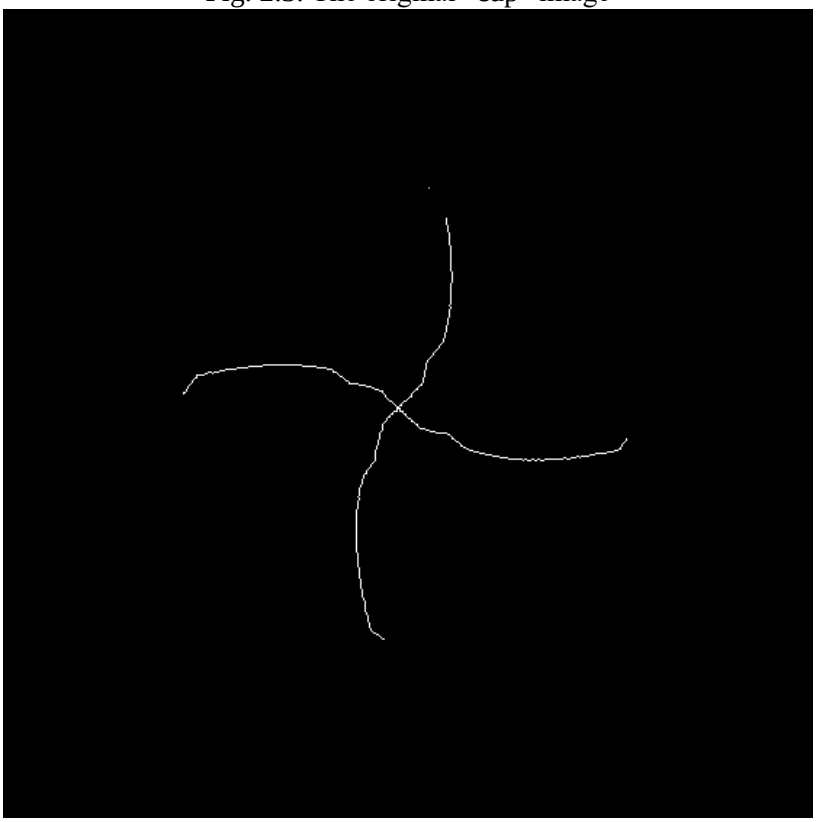


Fig. 2.4: The 1st intermediate “fan” image by shrinking

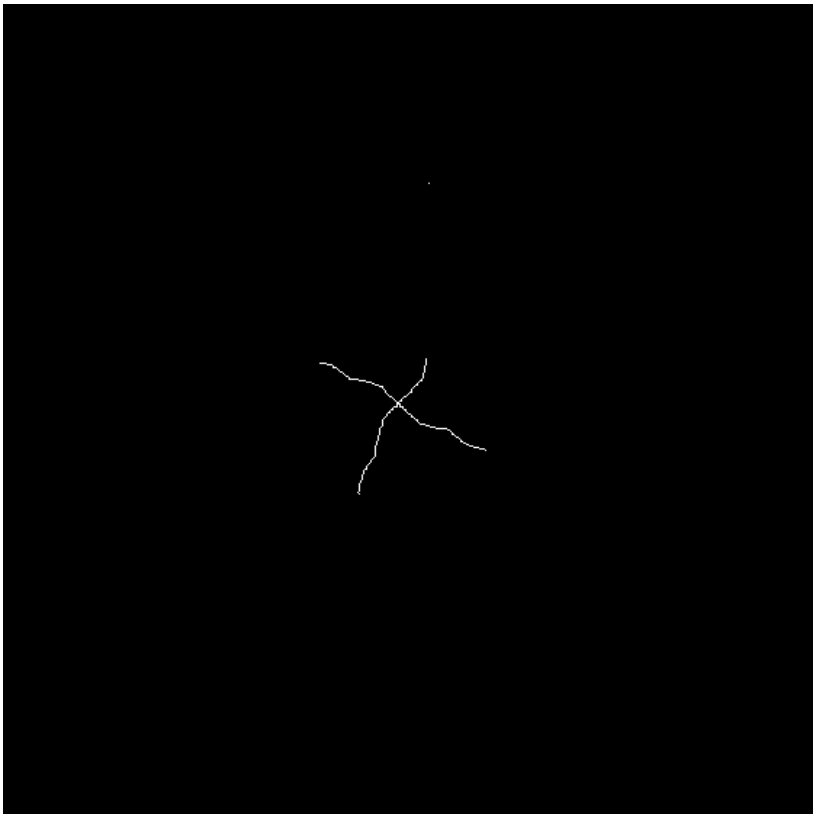


Fig. 2.5: The 2nd intermediate “fan” image by shrinking

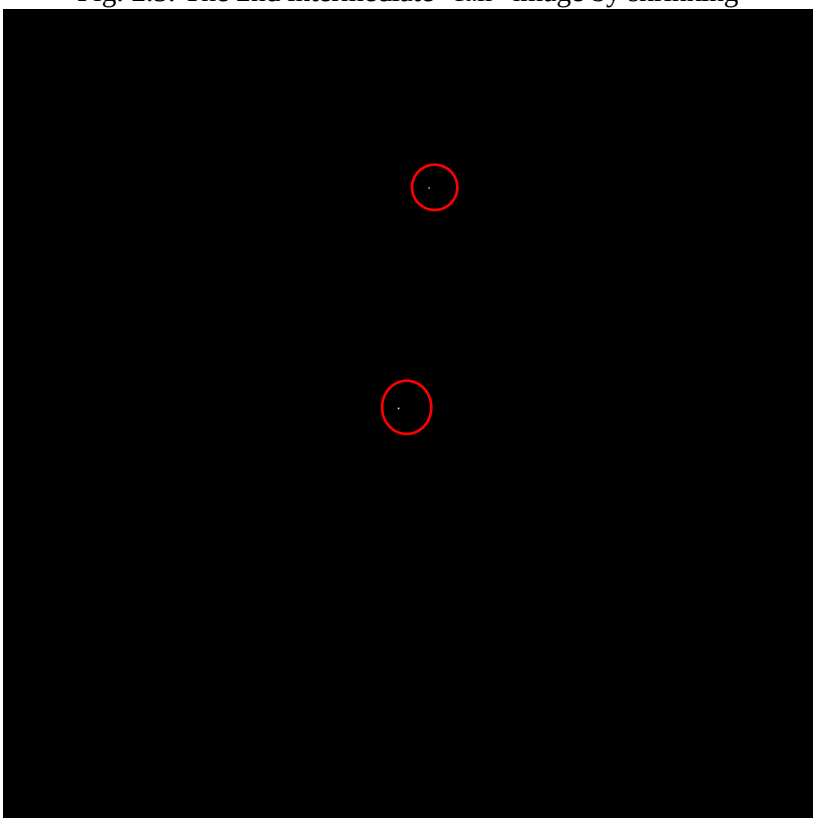


Fig. 2.6: The final “fan” image by shrinking

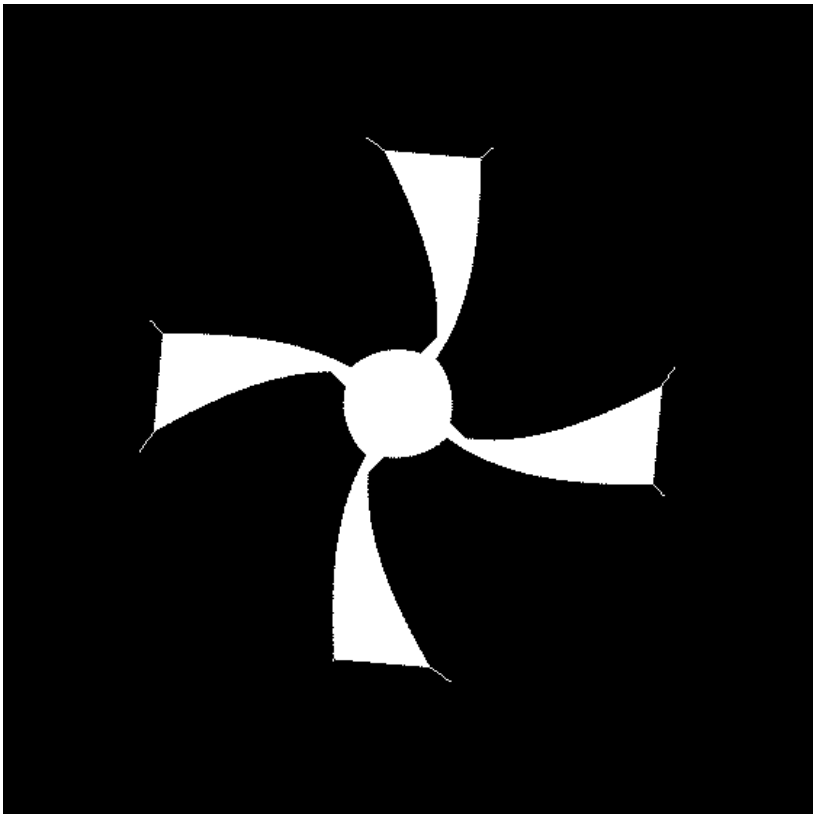


Fig. 2.7: The 1st intermediate “fan” image by thinning

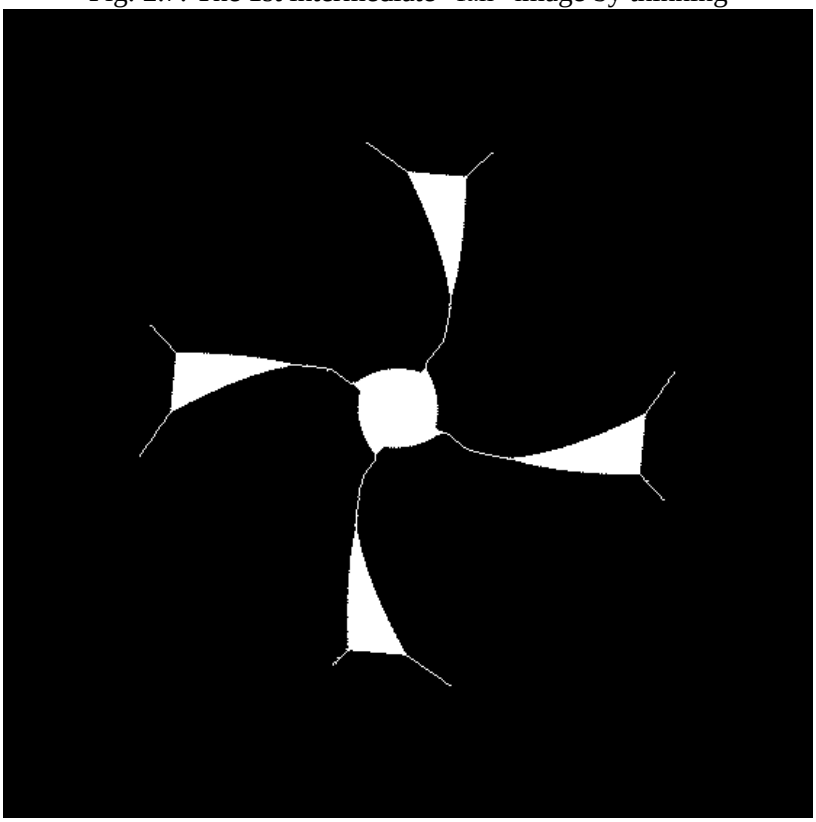


Fig. 2.8: The 2nd intermediate “fan” image by thinning

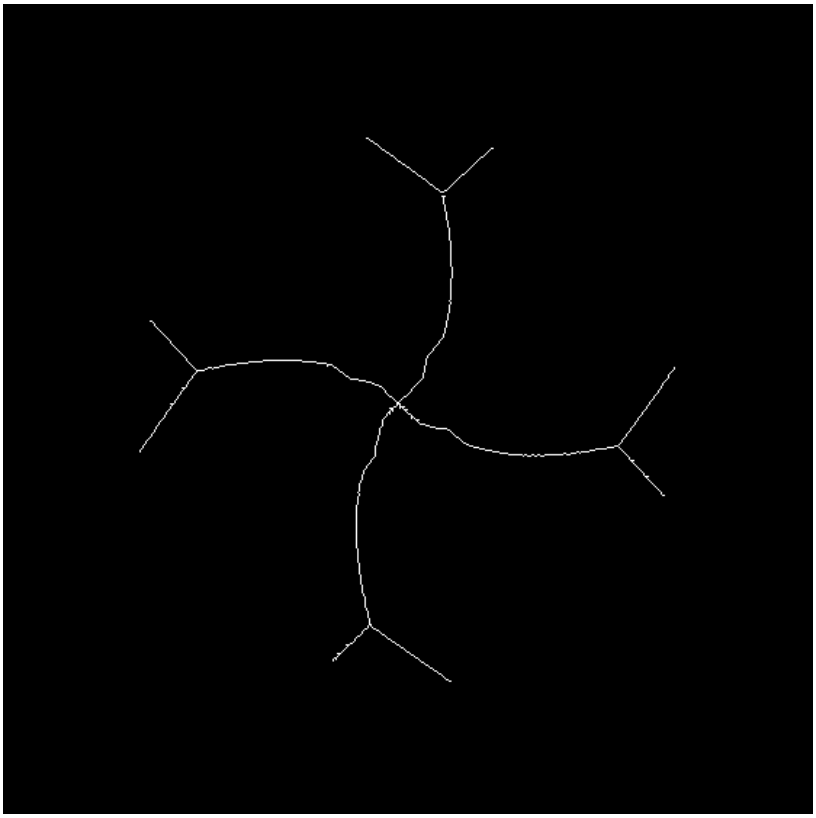


Fig. 2.9: The final “fan” image by thinning

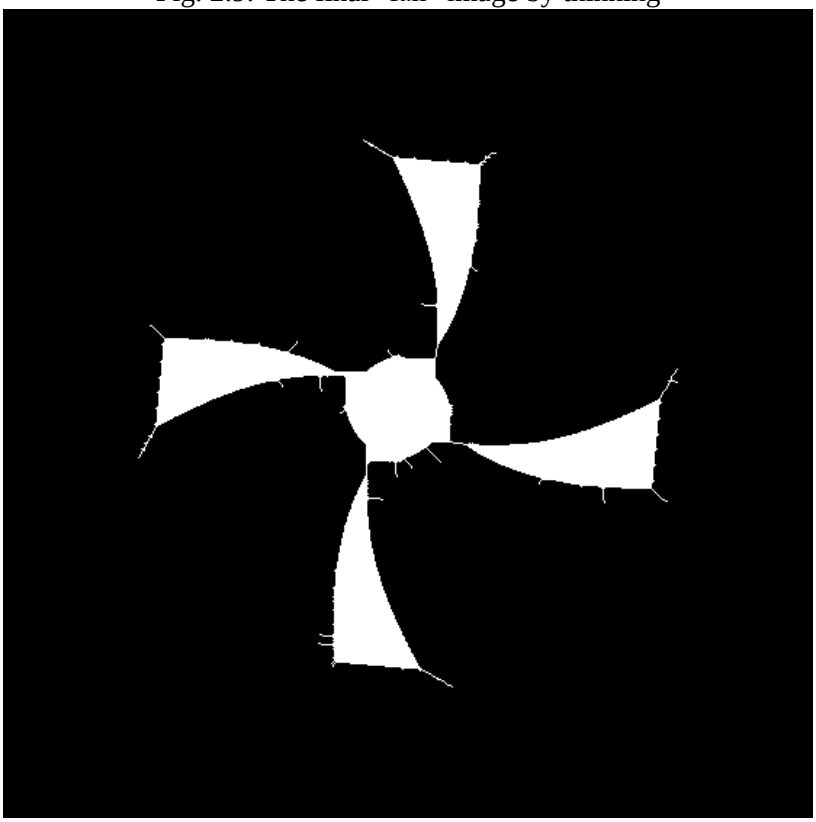


Fig. 2.10: The 1st intermediate “fan” image by skeletonizing

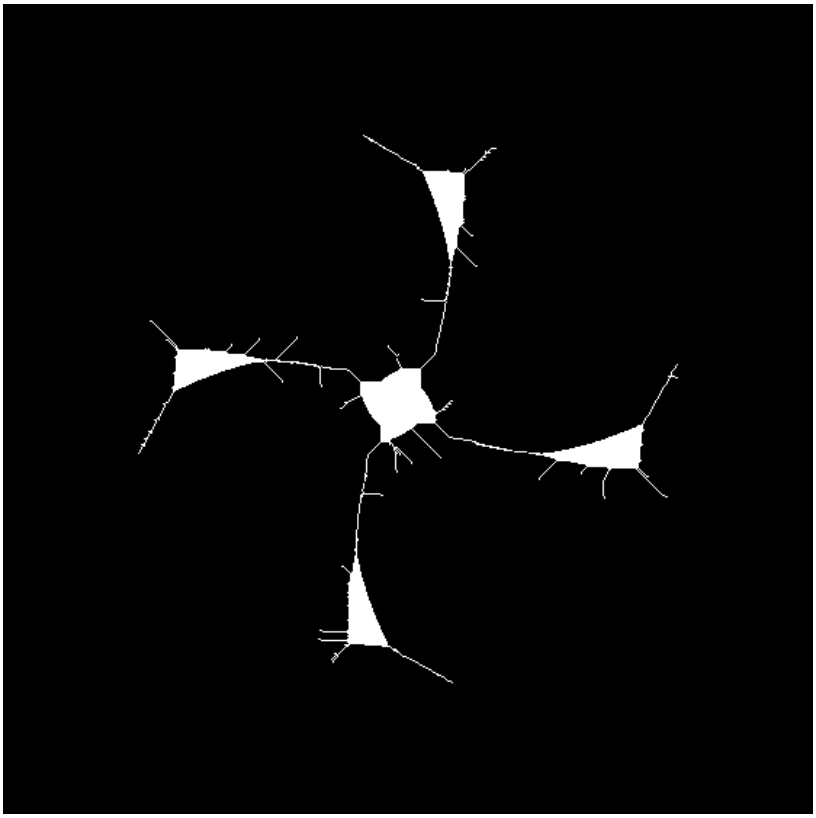


Fig. 2.11: The 2nd intermediate “fan” image by skeletonizing

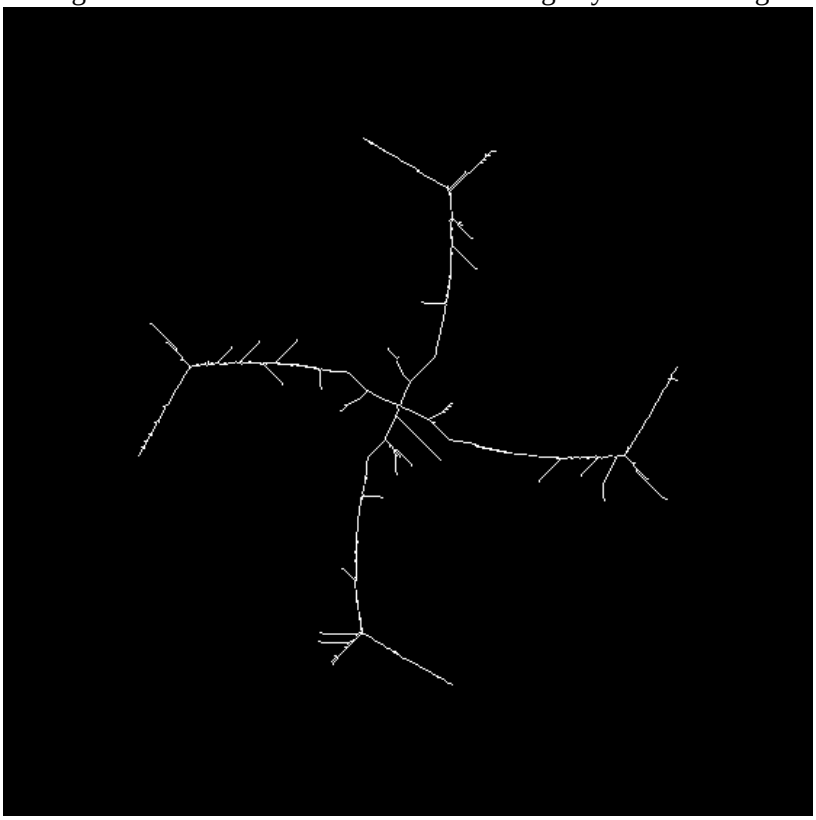


Fig. 2.12: The final “fan” image by skeletonizing

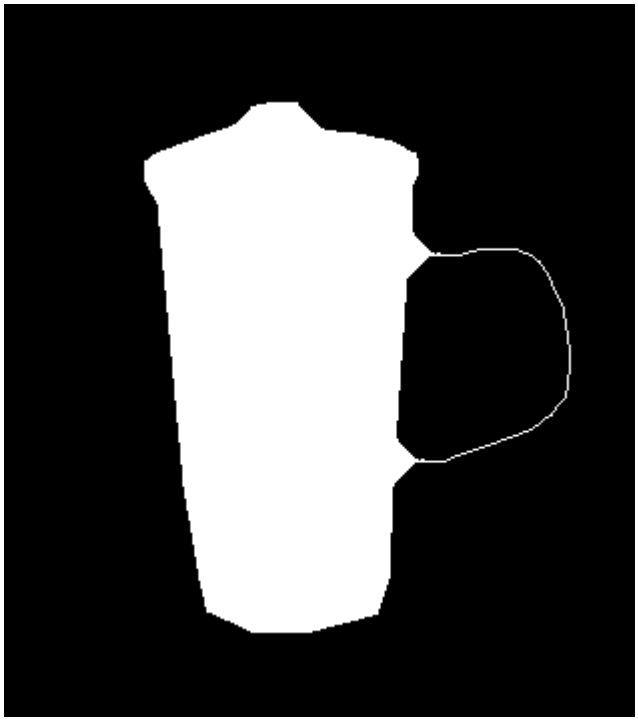


Fig. 2.13: The 1st intermediate “cup” image by shrinking

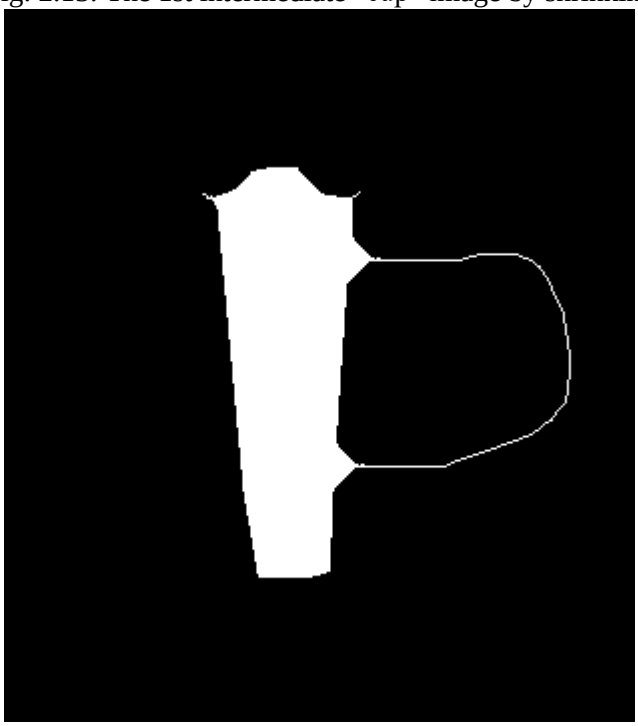


Fig. 2.14: The 2nd intermediate “cup” image by shrinking

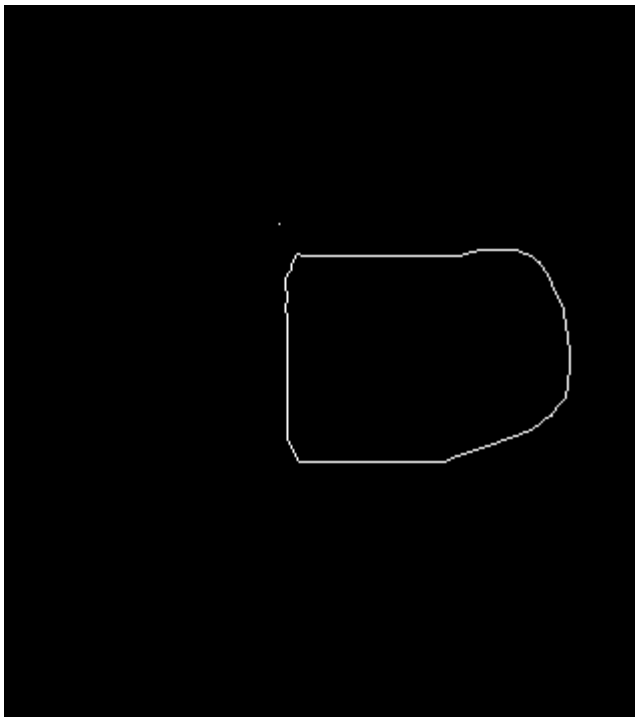


Fig. 2.15: The final “cup” image by shrinking

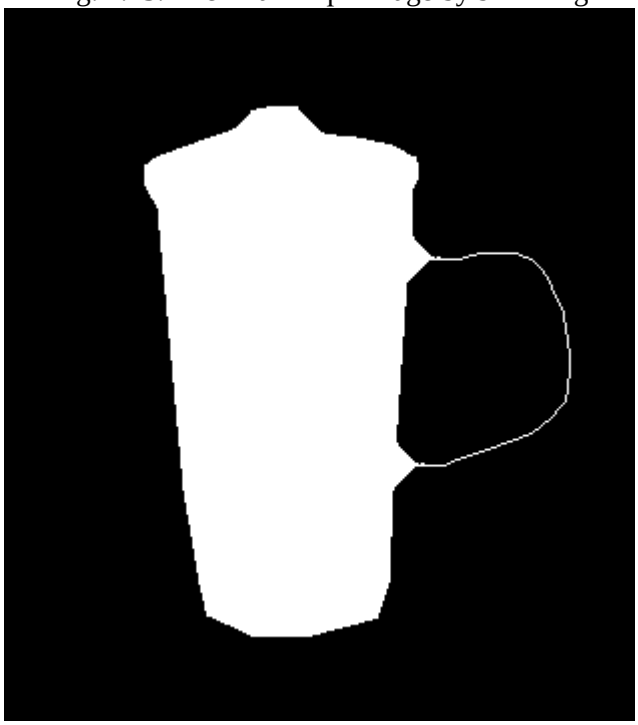


Fig. 2.16: The 1st intermediate “cup” image by thinning

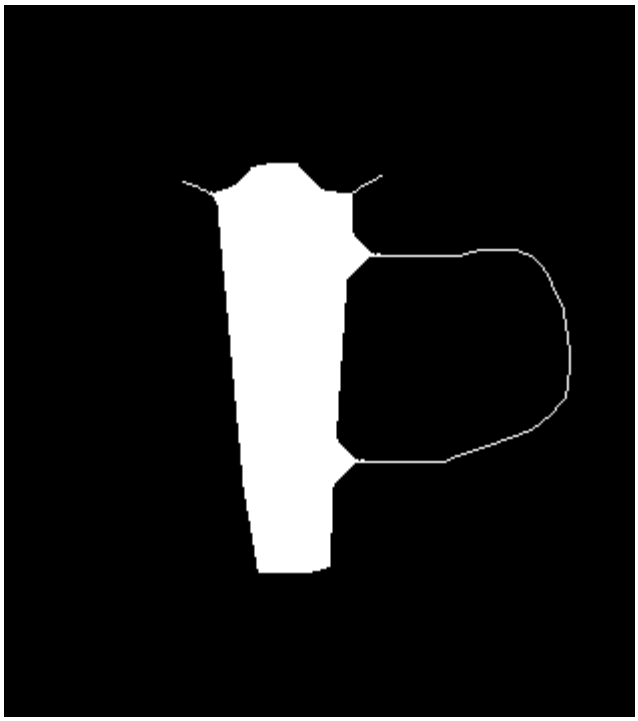


Fig. 2.17: The 2nd intermediate “cup” image by thinning

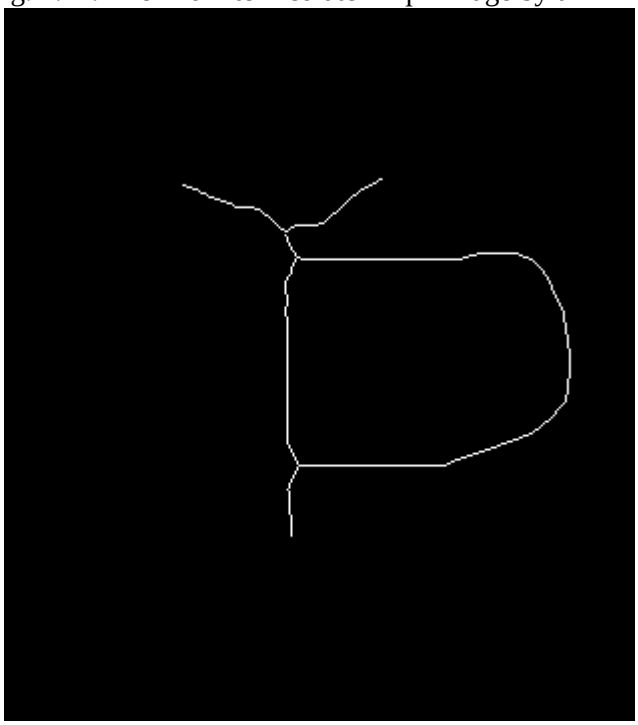


Fig. 2.18: The final “cup” image by thinning

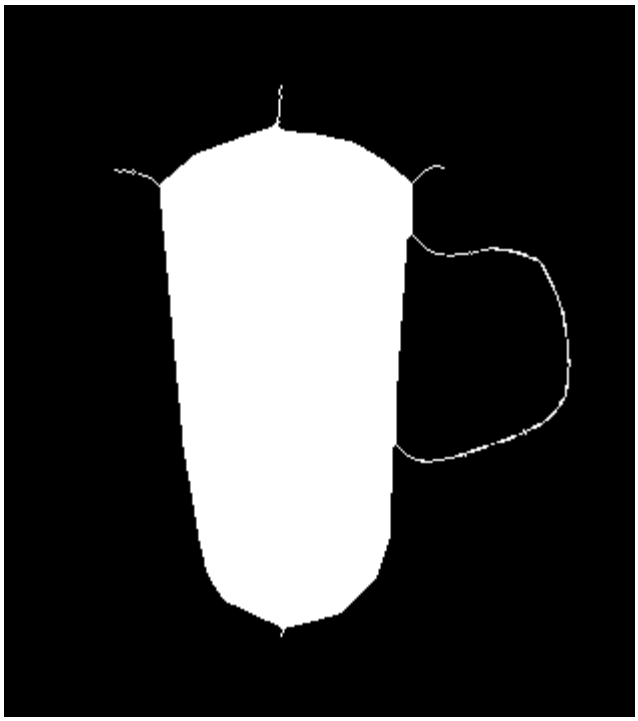


Fig. 2.19: The 1st intermediate “cup” image by skeletonizing

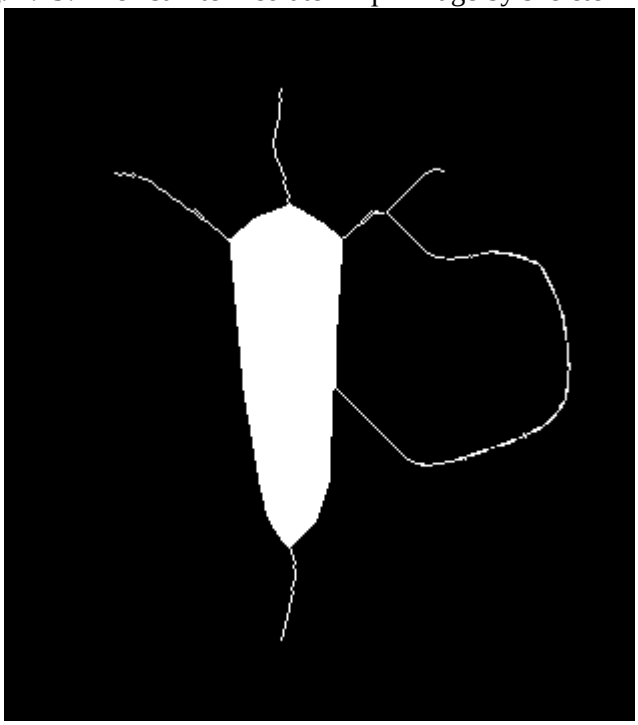


Fig. 2.20: The 2nd intermediate “cup” image by skeletonizing

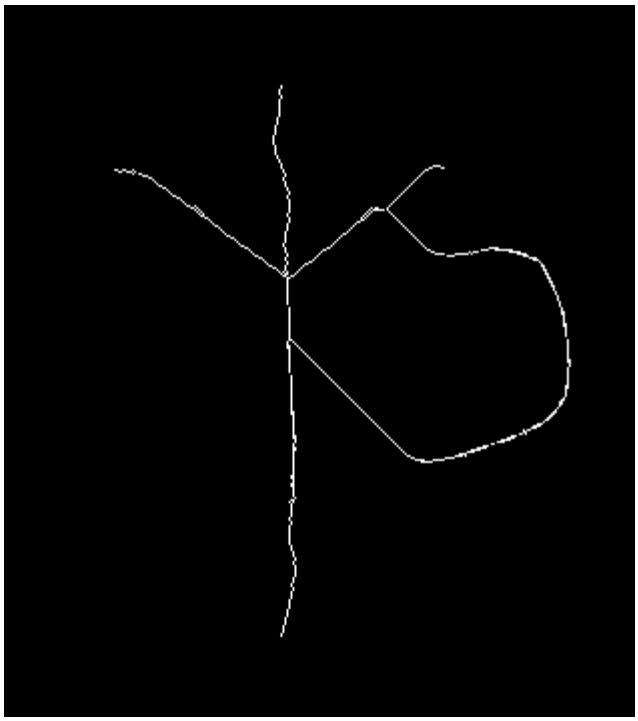


Fig. 2.21: The final “cup” image by skeletonizing

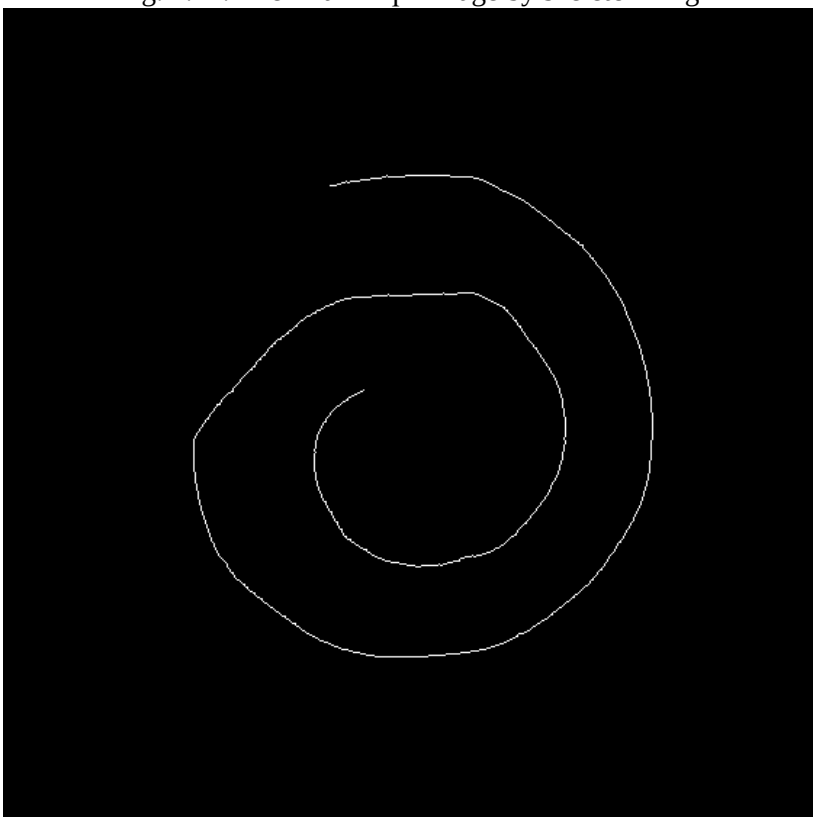


Fig. 2.22: The 1st intermediate “maze” image by shrinking

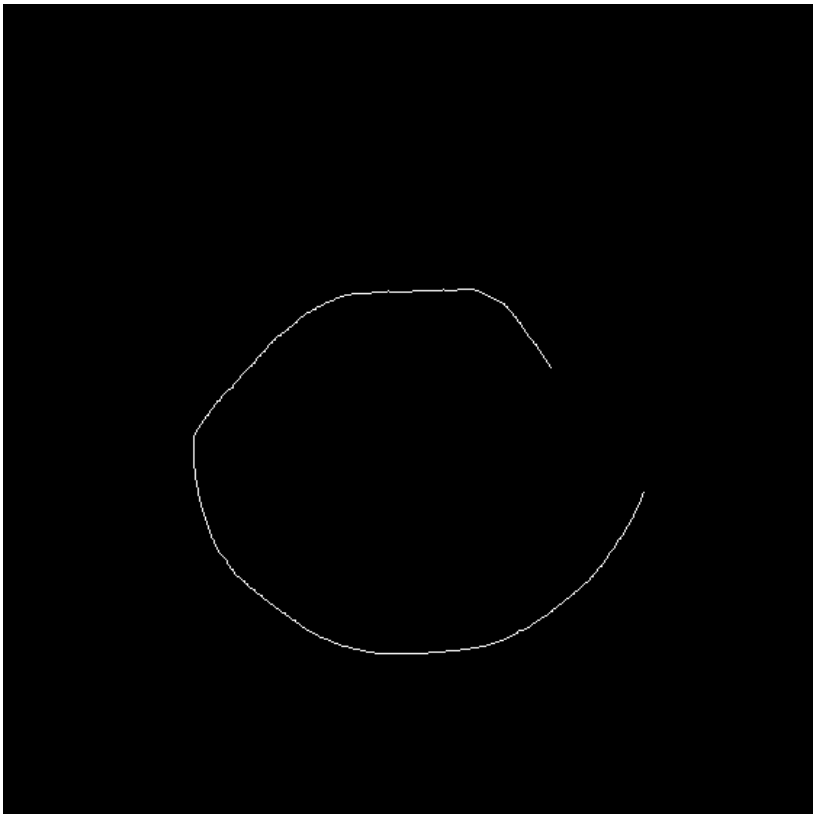


Fig. 2.23: The 2nd intermediate “maze” image by shrinking

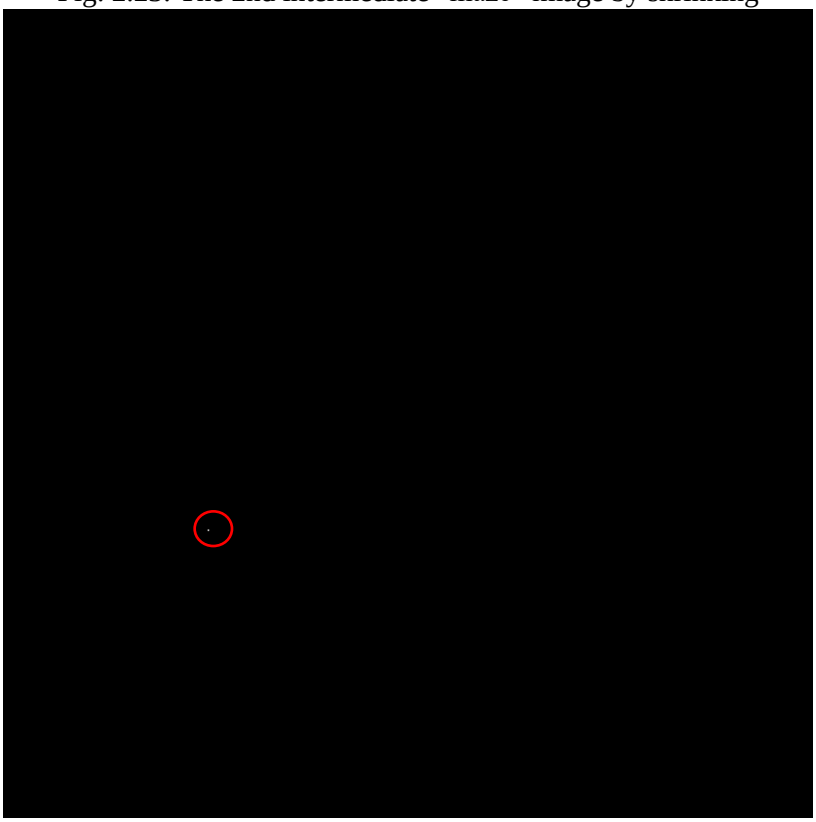


Fig. 2.24: The final “maze” image by shrinking

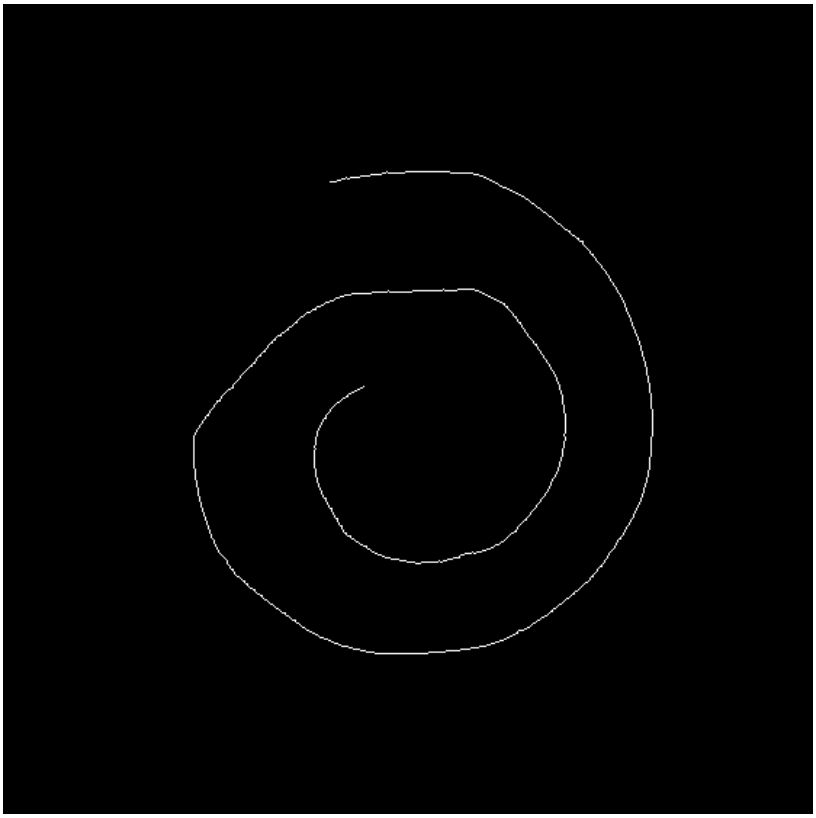


Fig. 2.25: The 1st intermediate “maze” image by thinning

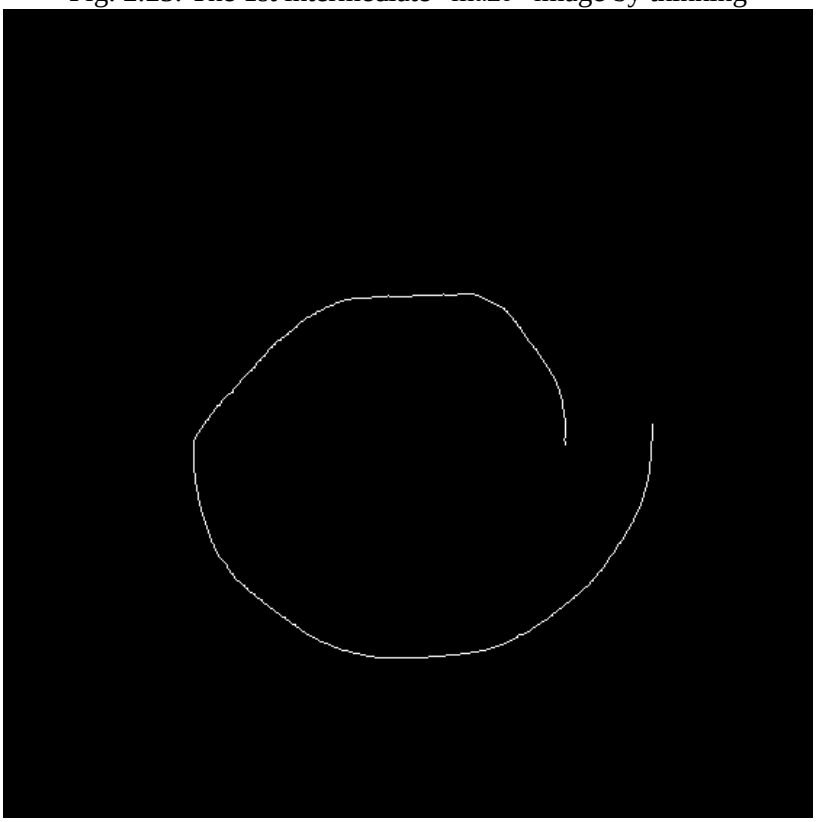


Fig. 2.26: The 2nd intermediate “maze” image by thinning

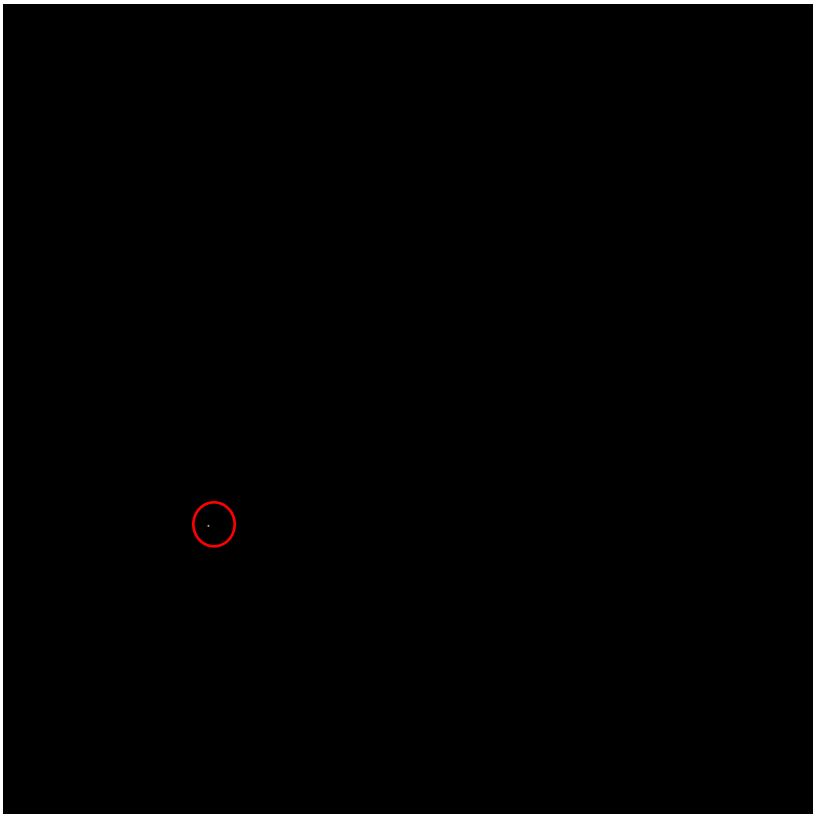


Fig. 2.27: The final “maze” image by thinning

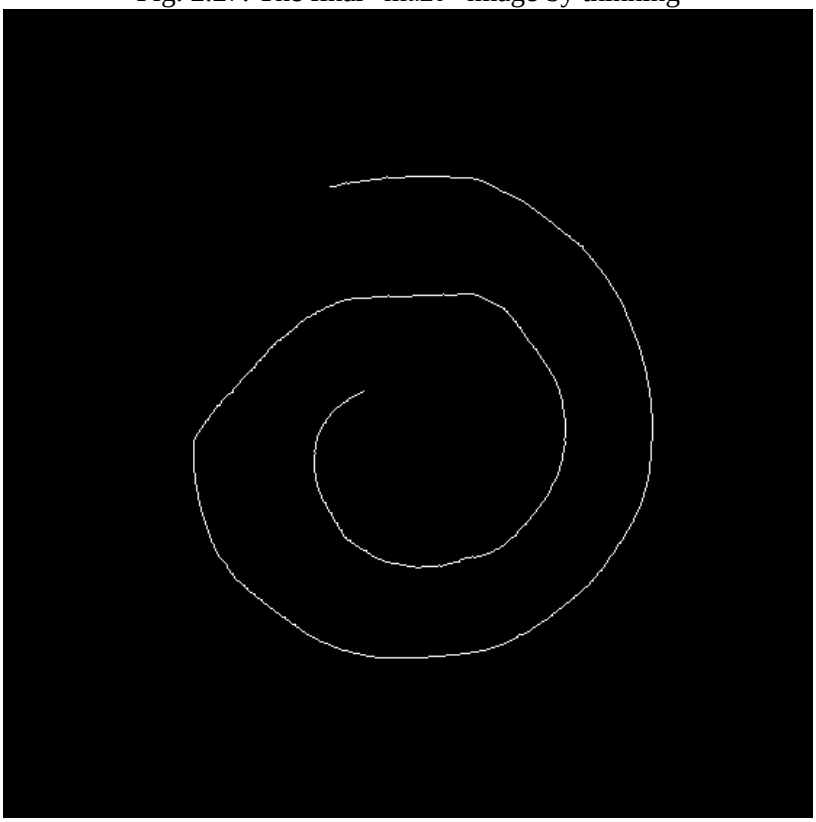


Fig. 2.28: The 1st intermediate “maze” image by skeletonizing

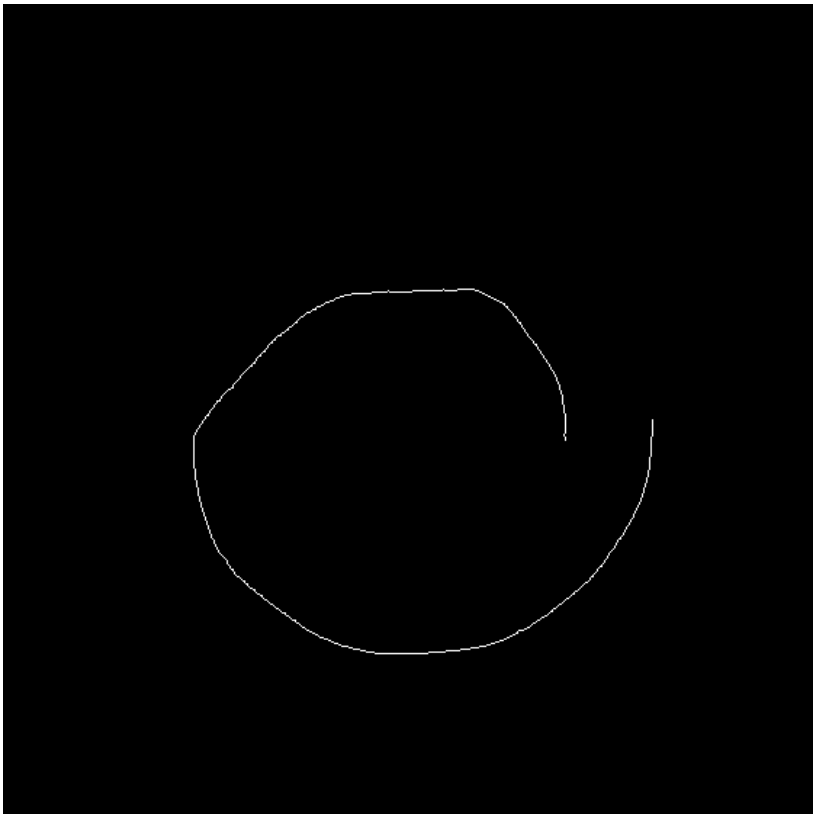


Fig. 2.29: The 2nd intermediate “maze” image by skeletonizing

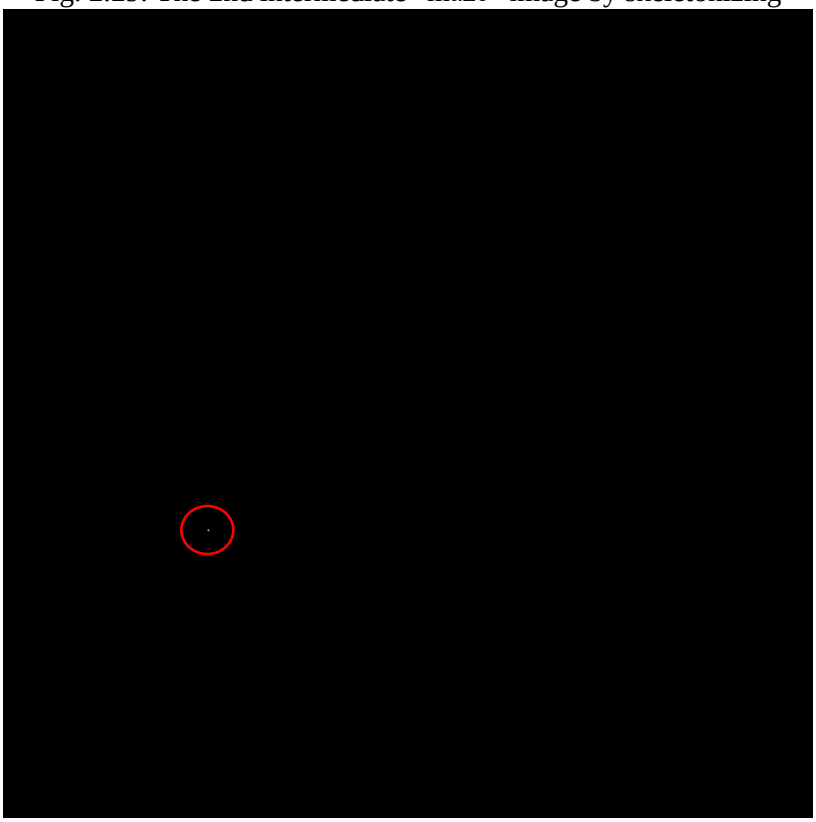


Fig. 2.30: The final “maze” image by skeletonizing

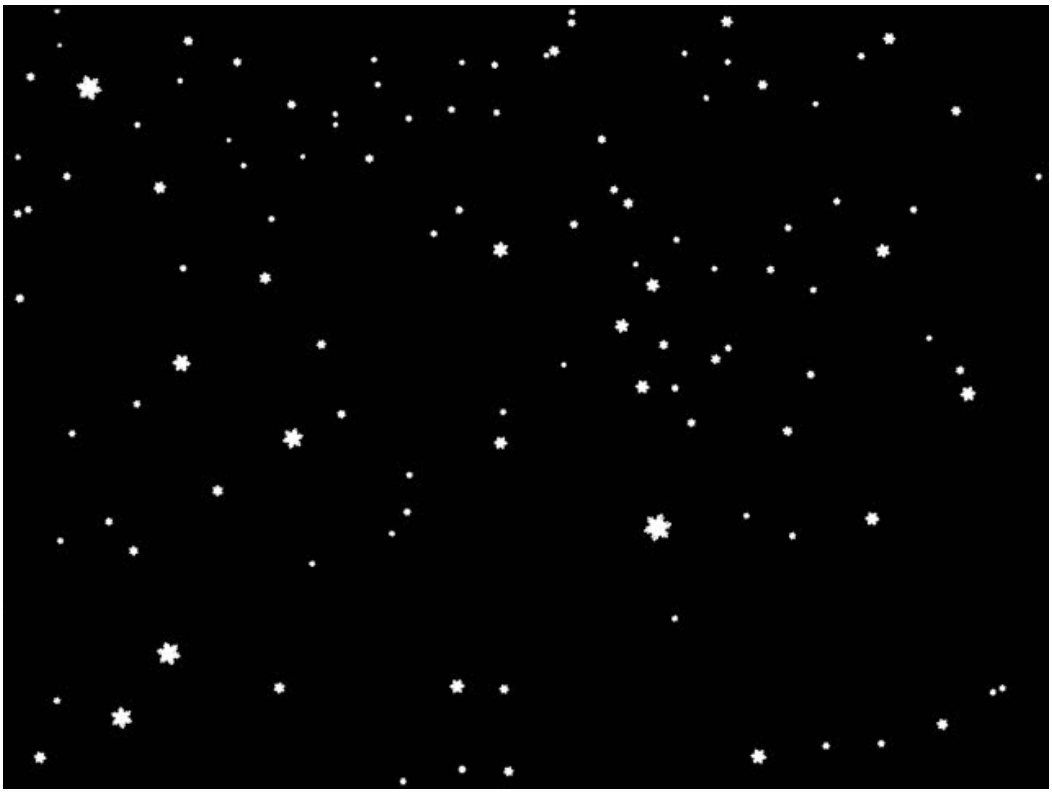


Fig. 2.31: The original “stars” image

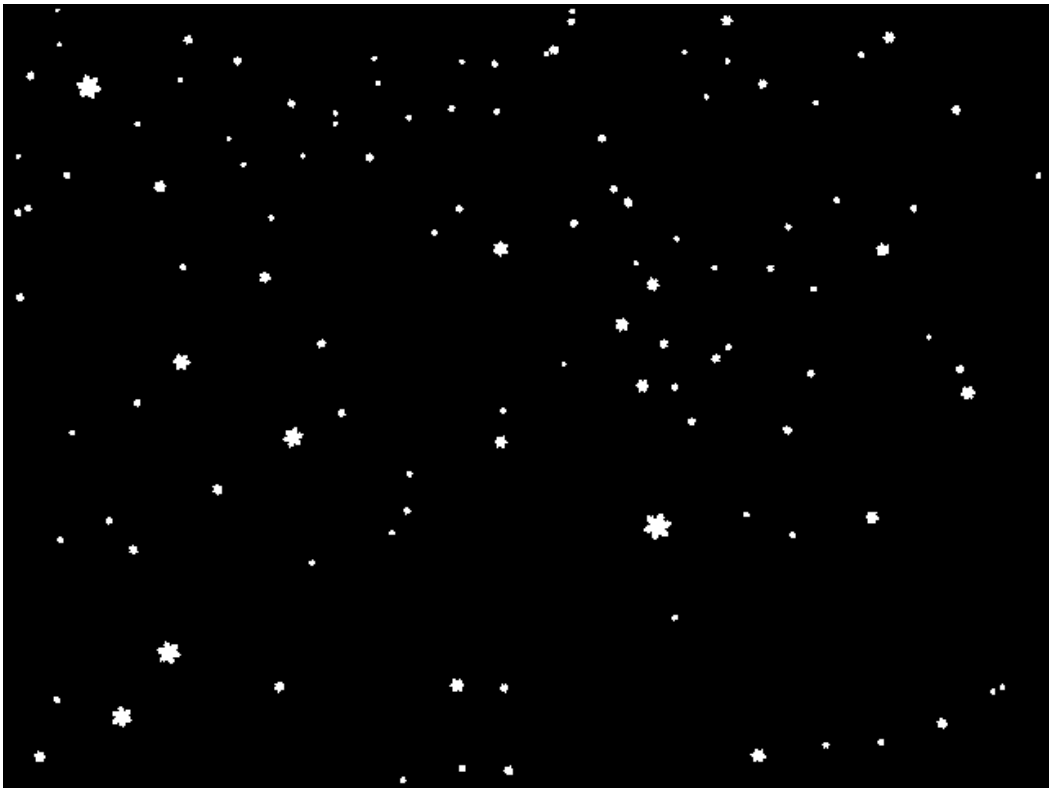


Fig. 2.32: The binarized “stars” image when the threshold value is 120

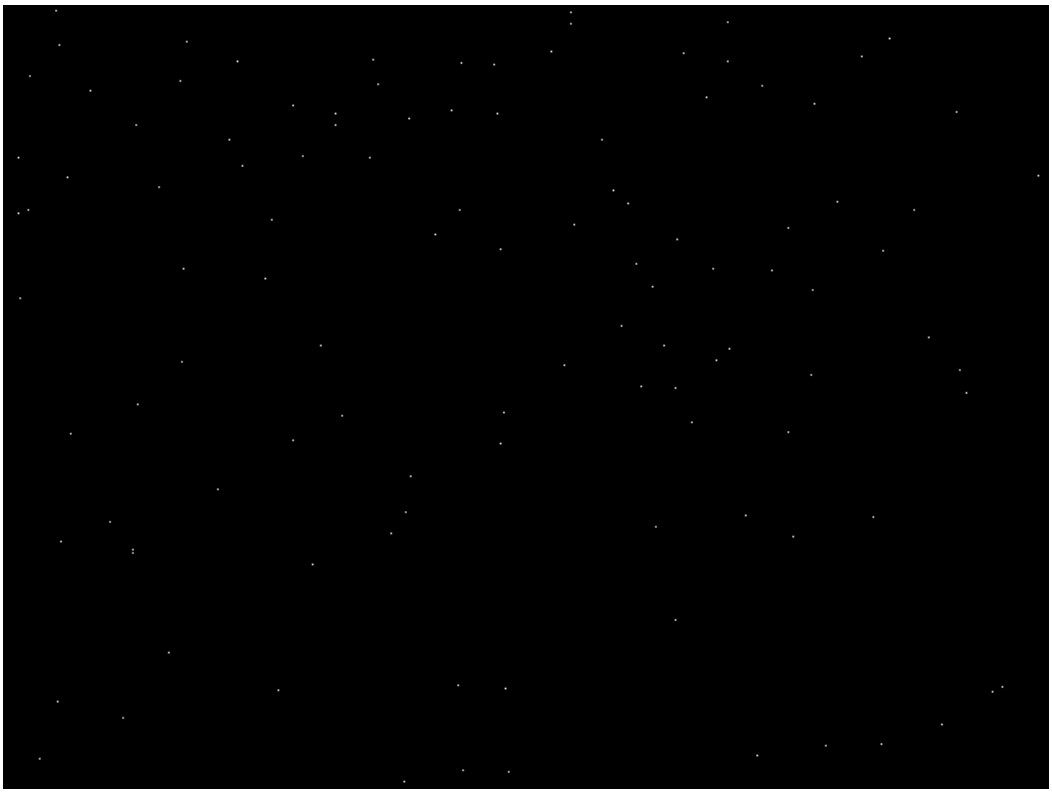


Fig. 2.33: The resulting “stars” image after shrinking

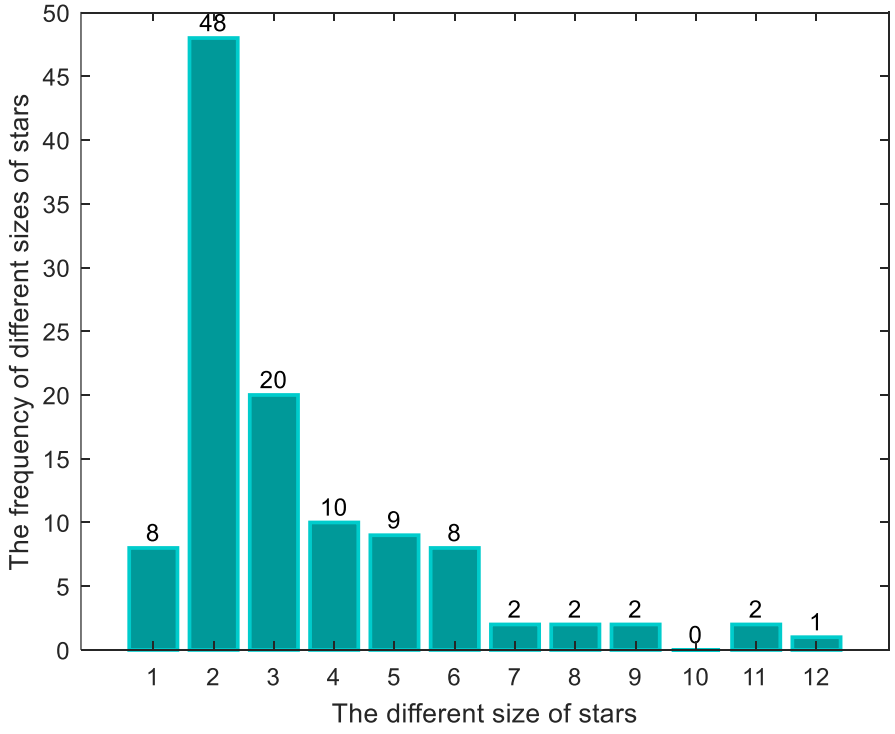


Fig. 2.34: The histogram of star sizes with respect to iterations

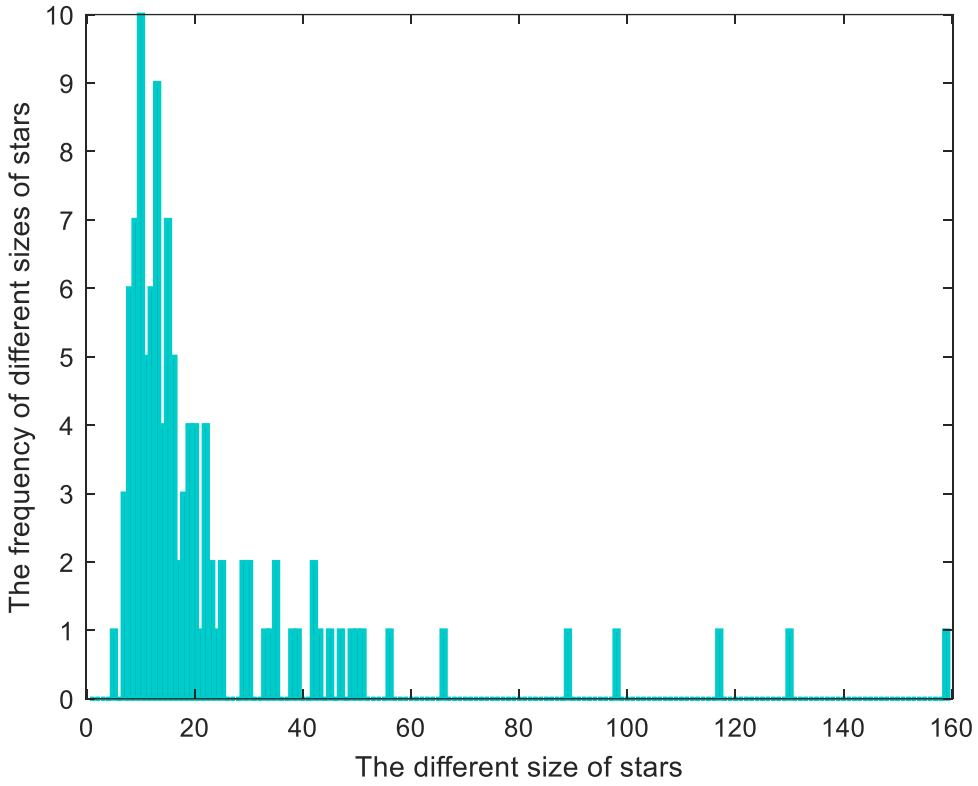


Fig. 2.35: The histogram of star sizes with respect to areas by using the connected-component-labeling method

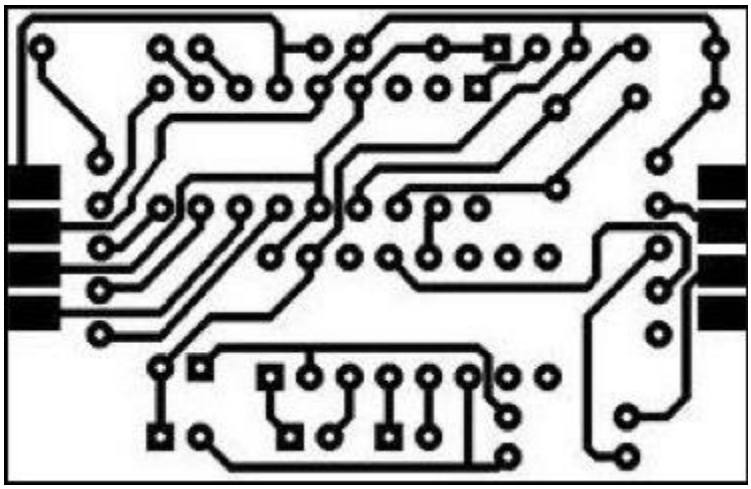


Fig. 2.36: The original “PCB” image

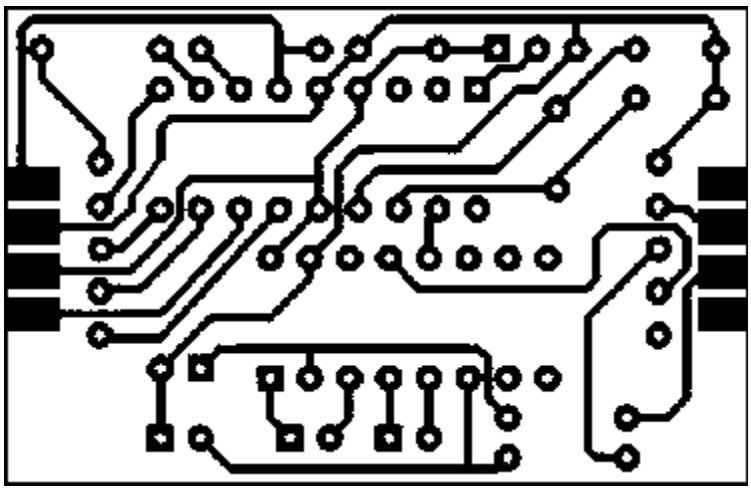


Fig. 2.37: The binarized “PCB” image when the threshold value is 100

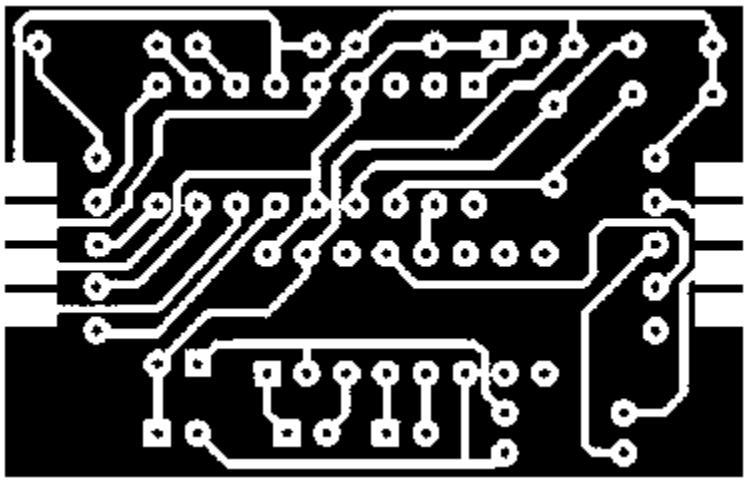


Fig. 2.38: The resulting “PCB” image after reversing the Fig. 2.37

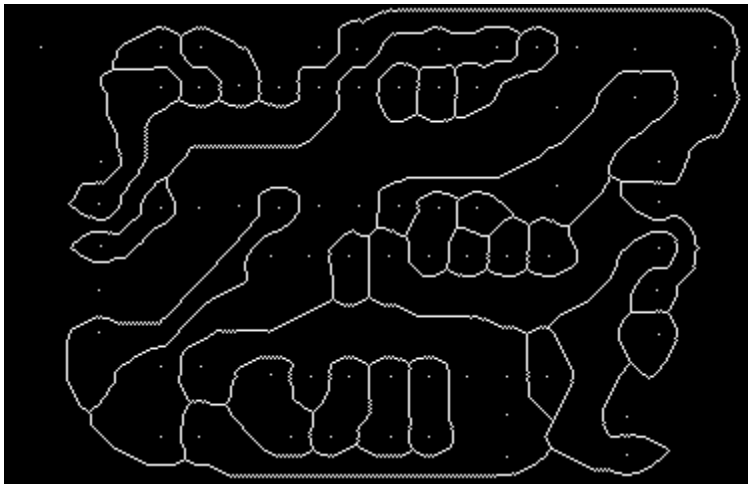


Fig. 2.39: The resulting “PCB” image after shrinking the Fig. 2.38

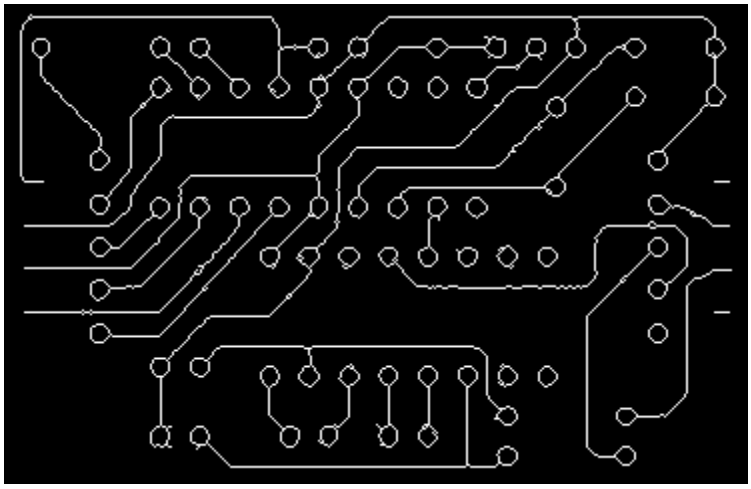


Fig. 2.40: The resulting “PCB” image after thinning the Fig. 2.38 but by setting the threshold value as 30 instead

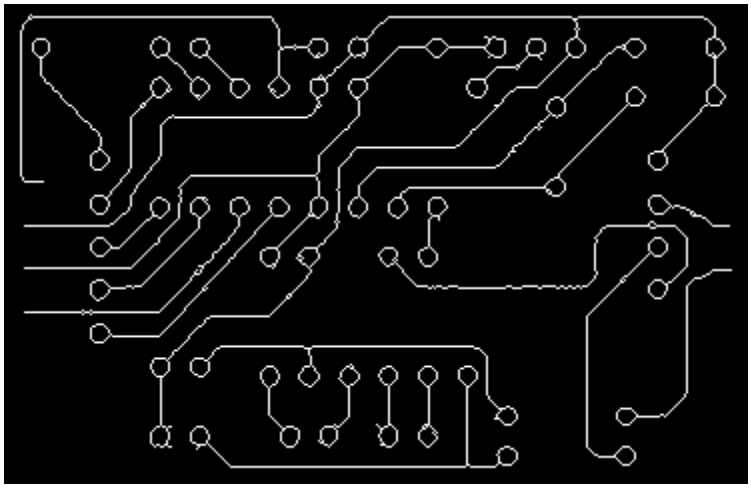


Fig. 2.41: The resulting “PCB” image after removing objects with small areas in the Fig. 2.40

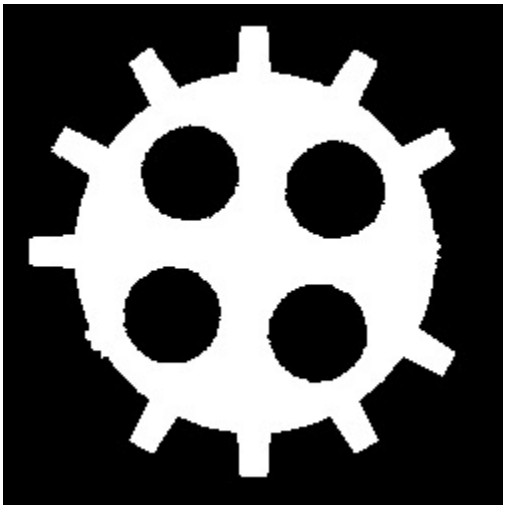


Fig. 2.42: The original “GeerTooth” image

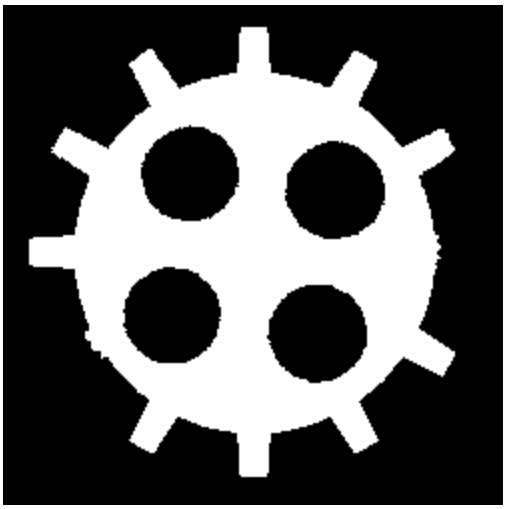


Fig. 2.43: The binarized “GeerTooth” image when the threshold value is 30

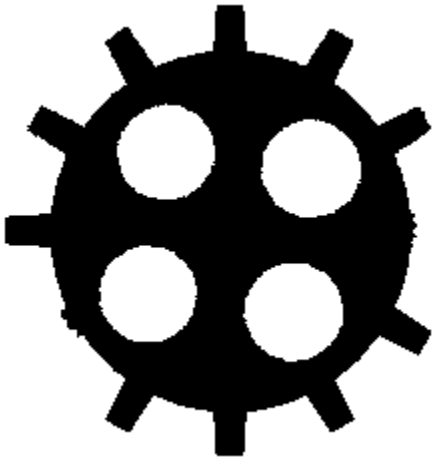


Fig. 2.44: The resulting “GeerTooth” image after reversing the Fig. 2.43

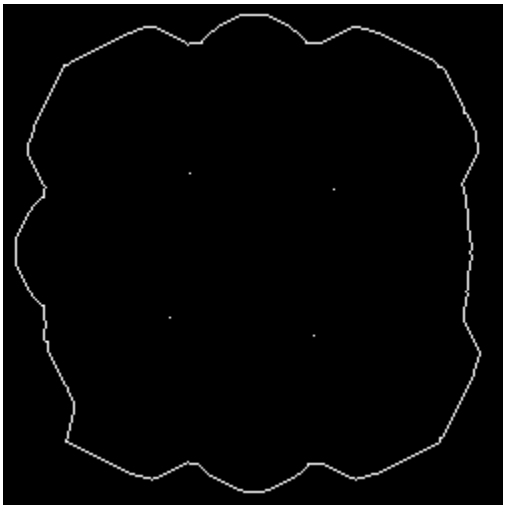


Fig. 2.45: The resulting “GeerTooth” image after shrinking the Fig. 2.44

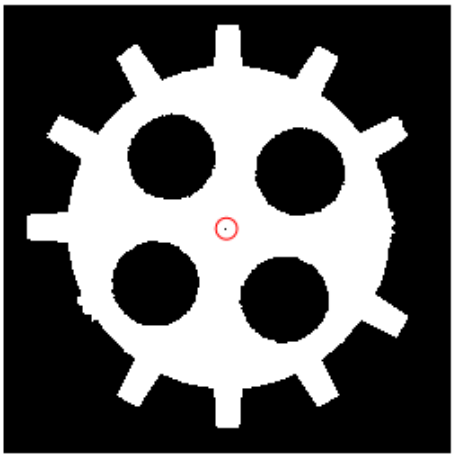


Fig. 2.46: The center of this gear

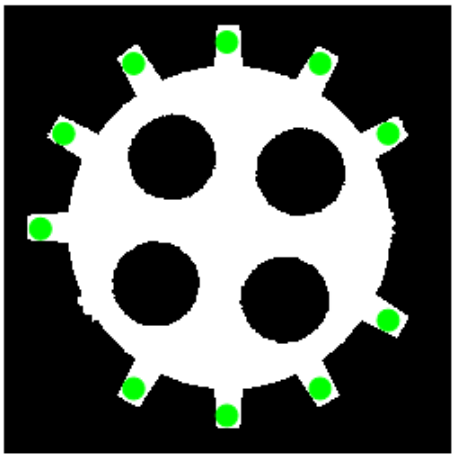


Fig. 2.47: The positions of the gear teeth in this gear

Fig. 2.48: The positions of the missing teeth in this gear

2.4. Discussion

As stated before, in this report, I will merely use my own judgement to evaluate the visual quality of resulting images. My discussion of experimental results of Problem 2 is stated as below.

In part (a), first of all, for the "fan" image, according to the Fig. 2.4, the Fig. 2.5, and the Fig. 2.6, the final image becomes two single white dots, which is a right answer due to something wrong with the pattern tables, because shrinking can remove pixels so that objects without holes become a point. According to the Fig. 2.7, the Fig. 2.8, and the Fig. 2.9, the final image becomes several straight lines because thinning can remove pixels so that an object without holes become a minimally connected stroke. According to the Fig. 2.10, the Fig. 2.11, and the Fig. 2.12, the final image roughly becomes a skeleton because skeletonizing can remove pixels on the boundaries of objects but isn't likely to allow objects to break apart, meaning that the remaining pixels can make up a simple image skeleton.

What's more, for the "cup" image, according to the Fig. 2.13, the Fig. 2.14, and the Fig. 2.15, the final image becomes a single white dot and a contour line because shrinking can remove pixels so that objects without holes become a point and objects with holes become a connected ring halfway between each hole and its outer boundary. According to the Fig. 2.16, the Fig. 2.17, and the Fig. 2.18, the final image becomes several straight lines and a contour line because thinning can remove pixels so that objects without holes become a minimally connected stroke and objects with holes become a connected ring halfway between each hole and its outer boundary. According to the Fig. 2.19, the Fig. 2.20, and the Fig. 2.21, the final image roughly becomes a skeleton because skeletonizing can remove pixels on the boundaries of objects but isn't likely to allow objects to break apart, meaning that the remaining pixels can make up a simple image skeleton.

Last but not least, for the "maze" image, according to the Fig. 2.22, the Fig. 2.23, and the Fig. 2.24, the final image becomes a single white dot because shrinking can remove pixels so that objects without holes become a point. According to the Fig. 2.25, the Fig. 2.26, and the Fig. 2.27, the final image becomes a single white dot because the maze line is a curve and thinning can remove pixels so that objects without holes become a minimally connected stroke. According to the Fig. 2.28, the Fig. 2.29, and the Fig. 2.30, the final image roughly becomes a single white dot because the maze line is a curve and skeletonizing can remove pixels on the boundaries of objects but isn't likely to allow objects to break apart.

In part (b), firstly, I shrink the binarized "star" image to get the Fig. 2.33, by which I design a simple algorithm to count single white dots, thus enable us to know the number of stars is 112. Secondly, according to the TA's hint, I utilize the number of shrinking iterations as stars' size. Hence, according to the Fig. 2.34, the number of star sizes is 12 and we can easily get the frequency of these star sizes from the histogram. One thing needing to be noticed is that I might not get the same answers as others because of the threshold value I have chosen. Thirdly, I use the connected-component-labeling method to know the number of stars is 111. The number of star sizes and frequency of these star sizes can be easily known from the histogram of the Fig. 2.35. One thing needing to be noticed is that I may get the number of stars differently ranging from 111 to 113 due to the different threshold values I have chosen.

In part (c), using the procedure that I have mentioned before, as soon as I get the Fig. 2.39 after shrinking the Fig. 2.38, I can design a simple algorithm to get there are 71 white dots, which means that there are 71 holes. What's more, when I get the Fig. 2.41 after removing some

meaningless pixels on the Fig. 2.49, I can design use the connected-component-labeling method algorithm again to get there are 25 pathways present in the given PCB.

In part (d), using the procedure that I have mentioned before, I find the center of this gear as shown in the Fig. 2.46. Then I estimate the outside radius of this gear by finding the distance between the center to the top boundary in terms of the number of pixels, which is 111. Finally, letting angles and distance be two conditions of constraint, the positions of the gear teeth and the positions of the missing teeth are shown in Fig. 2.47 and Fig. 2.48 respectively.

References

[1] [Online] Available: https: // www.mathworks.com/help/vision/ref/matchfeatures.html