USC Viterbi
School of Engineering

DEPARTMENT
*Electrical and Computer
Engineering*

EE 569

Homework #3: Geometric Image Modification and Morphological Processing

Name: Zhiwei Deng
Date: 2/29/2020

Contents

1. Problem 1

1.1 Geometric Warping

1.1.1 Abstract and Motivation

Geometric warping is a basic operation of image processing. Change the shape of images are universal useful in industry and research area, the technique can produce good performance and results without losing too much information of the original image. Also, the geometric modification techniques are widely used in both 2D and 3D images. In this part, a warping algorithm that will change the square shape image to disk shape image will be implemented and tested on three images that shown as below in Figure 1. The performance and different techniques used will be discussed and compared then.

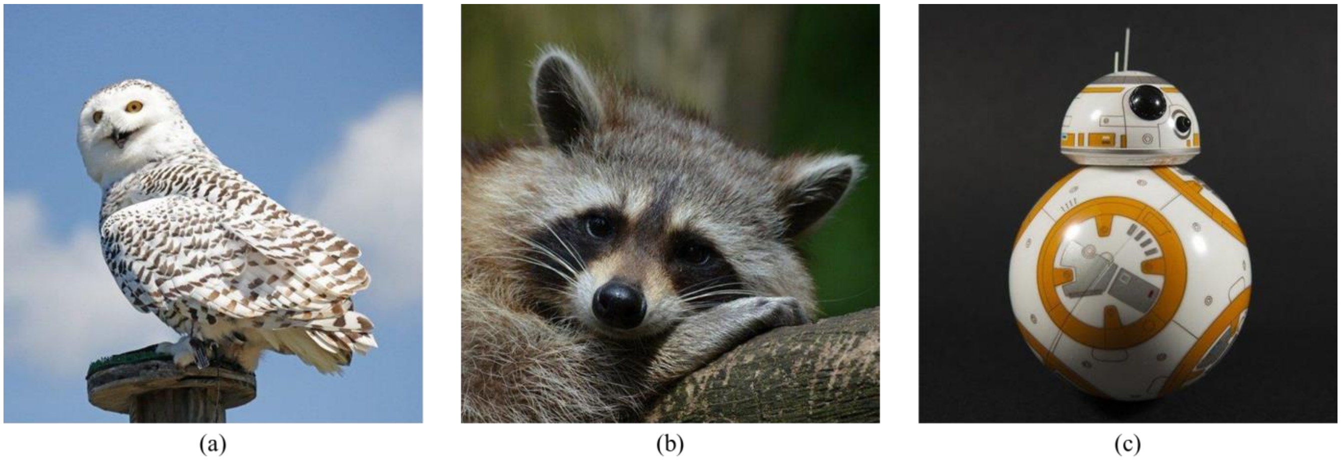


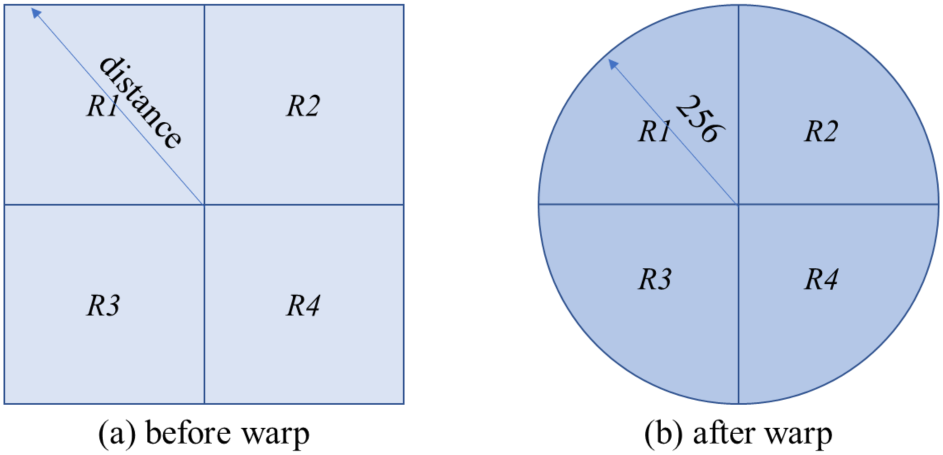
Figure 1. Test Images (a) hedwig.raw (b) raccoon.raw (c) bb8.raw

1.1.2 Approach and Procedures

Warping Directions: In this part, there are different ways to warp the images horizontally, vertically or in 45 degrees. I use the 45-degree method, which requires the implementation of polar coordinates modification. The angle keeps same, but the length is scaled in with different sizes. The formula is shown below in equation (1). The ρ is the length between each pixel with the center of the image (256, 256). Because in the disk shape image, the boundary pixels should have the ρ with 256, the scale factor of each pixel should be the *longest distance in certain angle / 256*.

$$\rho' = \rho * \frac{distance}{256} \tag{1}$$

Decomposition of the Image: The distance between each pixel and the center of the image is calculated in different ways in different image regions. So, the image should be divided into 4 different regions and processed separately. The division is shown in Figure 2(a), the center of the image is defined as (256, 256). And the warped region is shown in Figure 2(b), corresponding to (a). In this sub problem, no anchor point is needed. The distance of the certain pixel is only decided by the angle between it and the horizontal line. And for different regions, use either of sin(theta) or cos(theta).



(a) before warp (b) after warp

Figure 2. Decomposition of the Image

Forward Mapping and Inverse Mapping: There are two main methods to map an image to another. Forward mapping uses the original image pixels and tries to find the destination location in the target images. This will fully utilize the original information, while some pixels in the original image may be assigned to the same location in target image and some target image pixels might not get assigned by any pixel. This situation might cause the black dots in the target image and produce the artifacts. Inverse mapping is the other way around, it scans the target pixels and tries to find the original location in the original image, then assign it. In this way, some pixels might also come from several different locations, but we can make sure that every pixel in the target image has been assigned which means there is no artifacts like black dots. So, to avoid the artifacts like black dots, I used inverse mapping rather than forward mapping.

Bilinear Interpolation: Because the sin and cos functions are not linear, and the distances of each pixel might not be integers. After the transformation, most of the locations are not integers but decimals. Which makes it hard to identify which pixel in the original image should we take to assign it. If the interpolation is substituted by the round () function. The error can be enlarged between pixels, and the visual effect can be worse than interpolation. To avoid this situation, bilinear interpolation is applied in this problem.

Bilinear interpolation is an extension of linear interpolation, it applies linear interpolation in both X and Y directions. Its basic idea is to simulate the decimal pixel location values by weighted add the integer position pixel values. The weight of each pixel is determined by the distance between it and the decimal ones. The formula is shown as Figure 3 and equation (2). The result obtained by the round () function and bilinear interpolation is shown in below and compared in the discussion part. Generally, bilinear interpolation can produce smoother and noise free results.

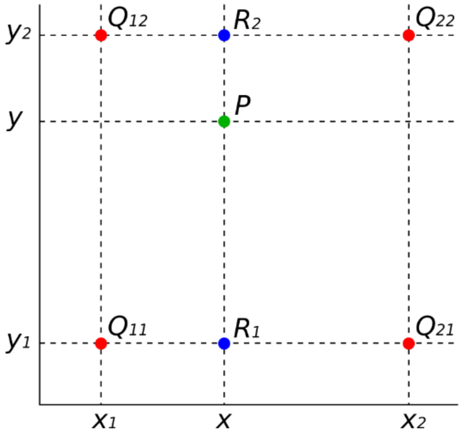


Figure 3. Bilinear Interpolation Diagram

$$\begin{aligned} f(x,y) &\approx \frac{y_2-y}{y_2-y_1} f(x,y_1) + \frac{y-y_1}{y_2-y_1} f(x,y_2) \\ &= \frac{y_2-y}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1} f(Q_{11}) + \frac{x-x_1}{x_2-x_1} f(Q_{21})\right) + \frac{y-y_1}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1} f(Q_{12}) + \frac{x-x_1}{x_2-x_1} f(Q_{22})\right) \end{aligned} \tag{2}$$

Coordinates: The diagram of the warping and reverse warping procedure is shown as below in Figure 4. This diagram is only for the 45-degree warping, i.e. polar coordinates warping. And all the pixel locations are transformed from image coordinates to 4 different Cartesian coordinates. The origin points of 4 coordinates are all defined with (256, 256).

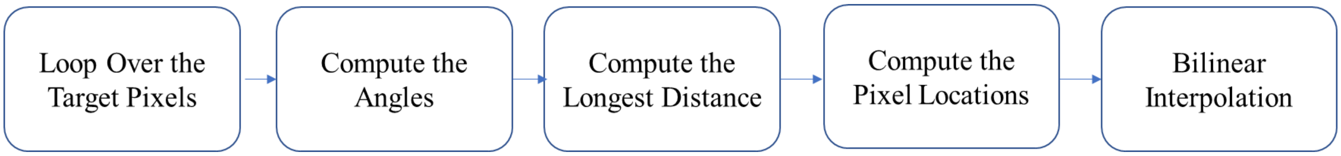
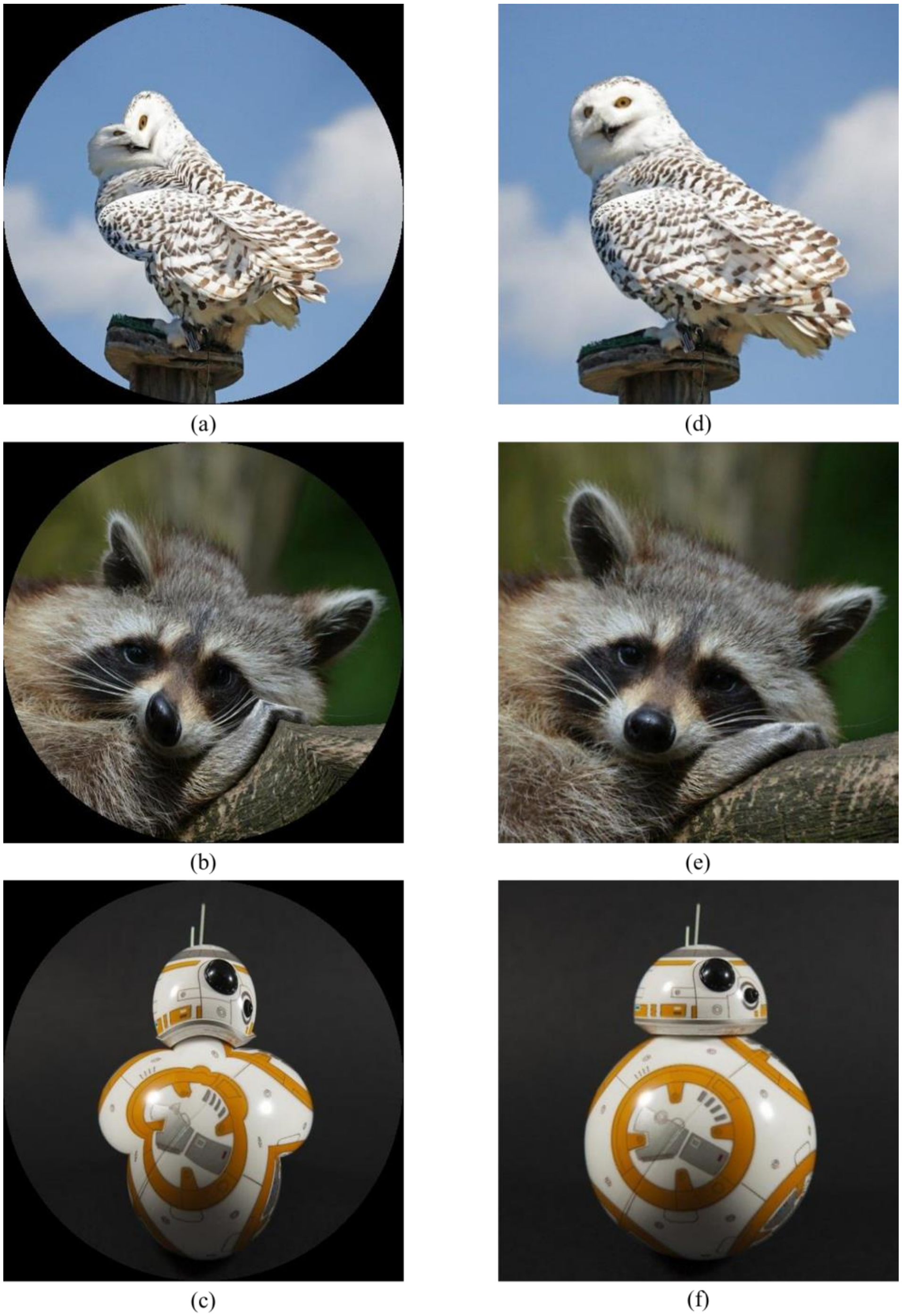

Loop Over the Target Pixels
Compute the Angles
Compute the Longest Distance
Compute the Pixel Locations
Bilinear Interpolation

Figure 4. Warping and Reverse Warping Procedure

1.1.3 Experiment Result

The warped image of test images is shown as Figure 5 (a-c). The reversed image of warped image is shown as Figure 5 (d-e).



(a) (d)

(b) (e)

(c) (f)

Figure 5. Warped and Reversed Results (Radius = 256 pixel)

1.1.4 Discussion

Method Explanation: In this part, I implemented the algorithm to warp images into diskshape. The method was described as follow. Firstly, the coordinates are transformed from image coordinates to cartesian coordinates based on the region they belongs to. There are 4 regions, left-upper, right-upper, left-down and right-down, and they share the same center of the image (256, 256) in image coordinate. After getting the coordinates, the angles of the pixels are calculated. Based on the angles, the longest distances of each angle can be obtained by the image width and height. We can get the scale factor of each angle should be the longest length divided by the radius of the disk, in this case is 256 pixels. Then, we can find the pixel in the correct position of the original image and assign it to the disk. In this process, bilinear interpolation is used. Some important parameters, center of the image is (256, 256), the radius of the disk is 256 pixel.

About the center of the image, because the row number and column number is both even with 512. Choose one of either (256, 256), (256, 257), (257, 256) or (257, 257) is reasonable. But it will also cause some problems. For example we can see that the left and upper edges of the disk image is attached to the square boundaries, and the right and lower edges has only one pixel ans shown in Figure 6. This is caused by the radius, because the left upper rigion should have smaller radius than other regions. This problem will disapper with a smaller radius of the disk. The smaller radius results are shown in Figure 7.

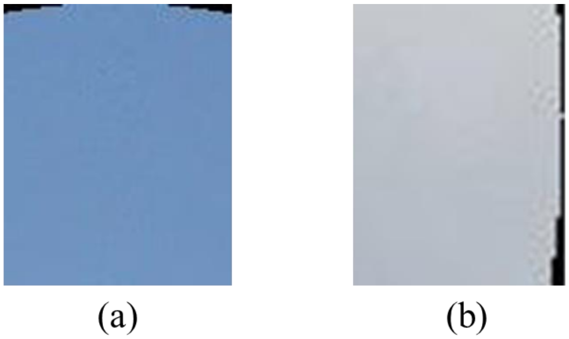


Figure 6. Upper Bound and Right Bound of Warped hedwig

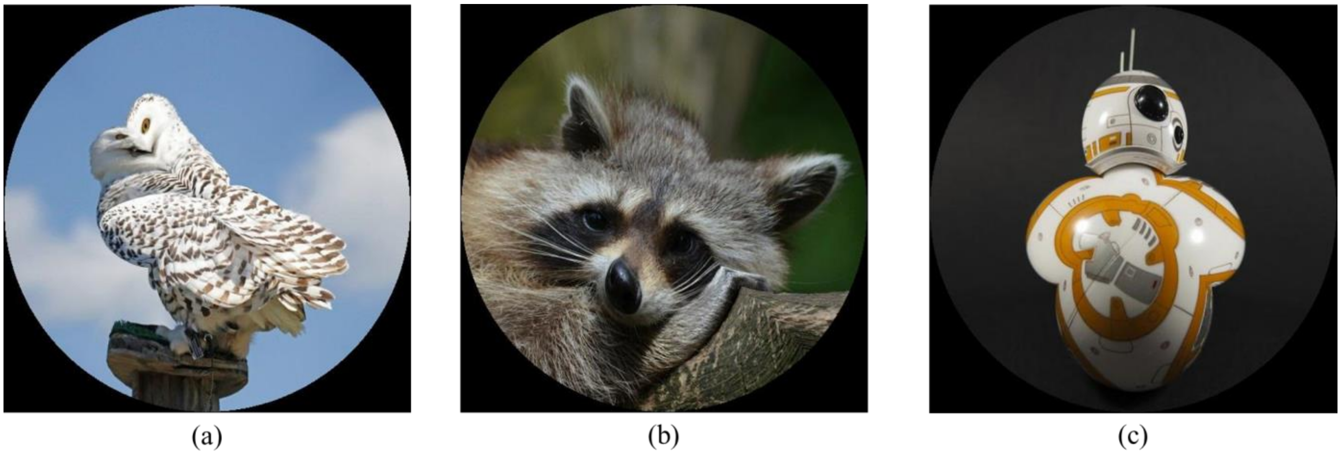


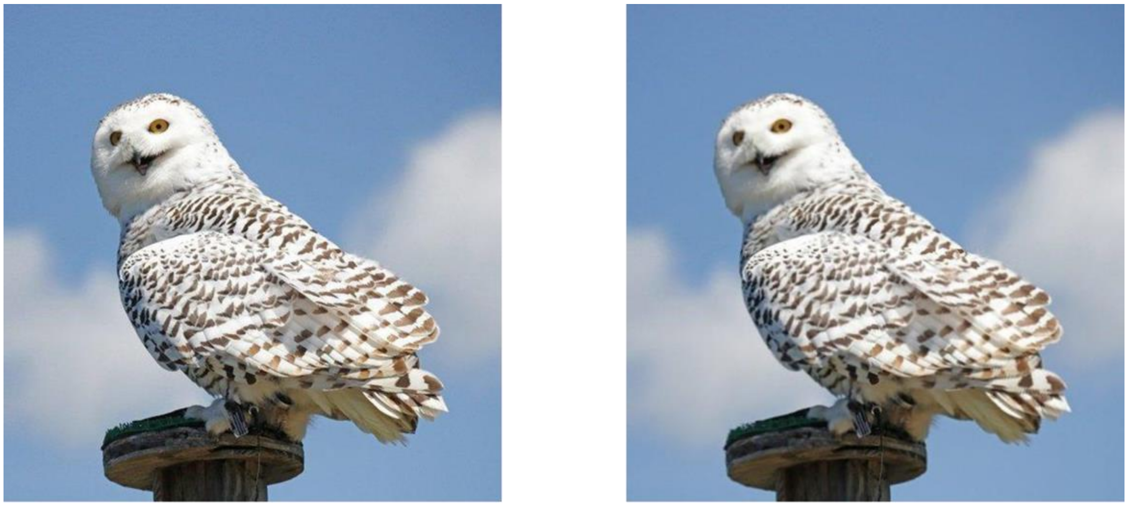
Figure 7. Warped Images with Radius = 255

Requirements: From the images, we can easily says that the boundaries and the center of the images dosenot change after warping. Also, the image is reversible and the results are shown in Figure 5 (d-f). So, all three requirements are satisfied by the code.

Filtering: Because the bilinear interpolation, there are some gray pixels on the edges of the reversed imag. On the boundary, it is easy to count the black pixels in the corners into the consideration, which would lower the overall grayscall values in each channel and produces black or gray dots on the edges. To overcome this artifact, we can apply the median filter on the reversed image due to the black dots are pepper noises. Also, we can just pick the near pixel values to assign the edges to avoid this artifact.

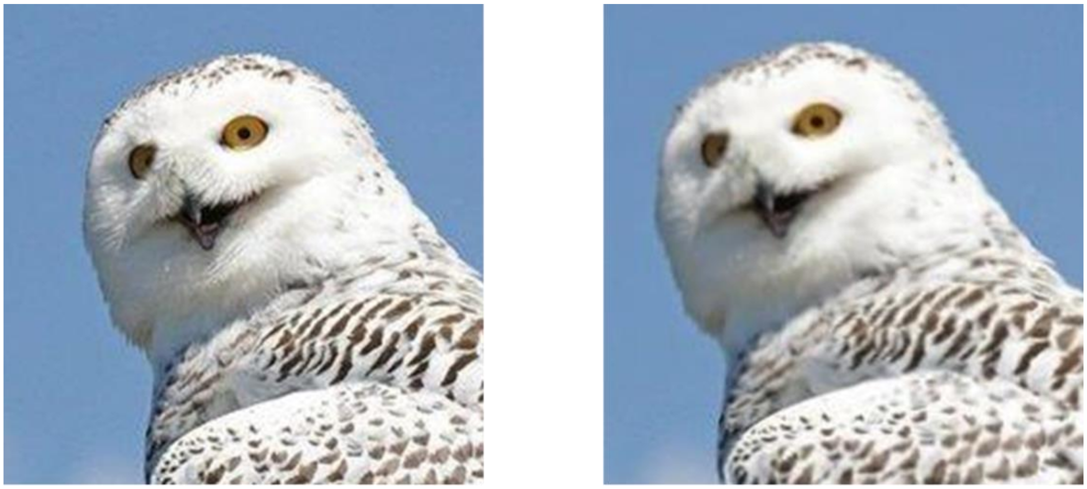
Comparison: The original image and the reversed image of hedwig are shown in Figure 8. From Figure 8, we can see that the original image is generally more clear than the reversed image. This difference is going to be more obvious if zoomed in. As shown in Figure 9, we can see that the feathers and the eyes of the hedwig in original image is much more clear and vivid than the reversed ones.

This is due to the bilinear interpolation. Bilinear interpolation takes 4 pixels into consideration at one time. The pixel values are generated by weighted average them. So, rather than just assign it to one pixel value, it acts like a low pass filter and blur the image a little bit. Moreover, I applied the bilinear inerpolation in both warping and unwarping procedures, which make the result more blurry than original one. Generally, we can think that the warping and unwarping both produces some information loss, which will cause the distortions in some level.



(a) hedwig.raw (b) reverse.raw

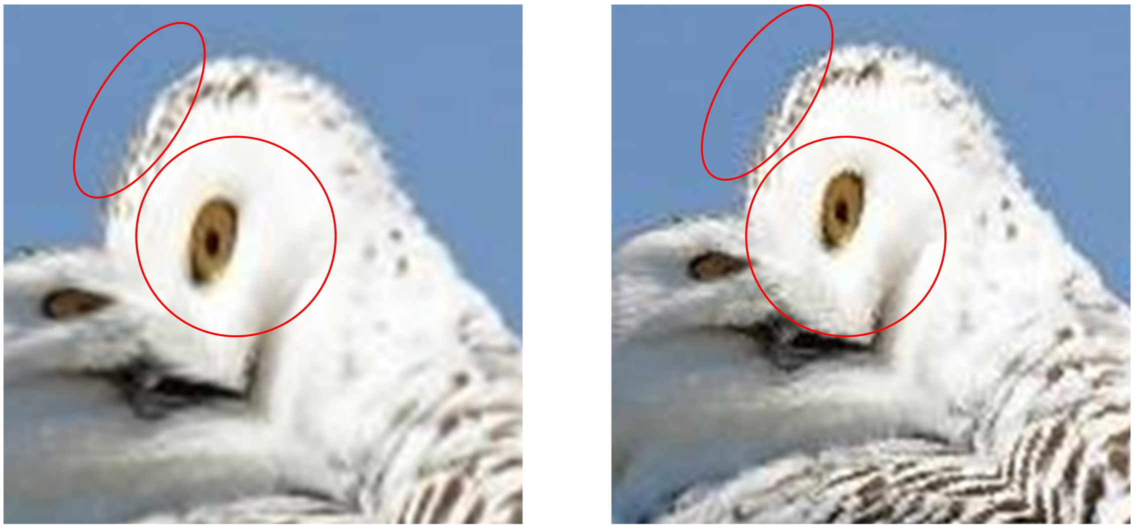
Figure 8. Original and Reversed image of hedwig.raw



(a) hedwig.raw (b) reverse.raw

Figure 9. Zoomed in Comparison

Bilinear Interpolation Comparison: The reversed result of round () funtion and bilinear inerpolation is shown in Figure 10. We can see that, the bilinear interpolation can produces smoother results and less jaggies on the eye area and edges of the head than just use the round function.



(a) Bilinear (b) round()

Figure 10. Bilinear Interpolation Comparison

1.2 Homographic Transformation and Image Stitching

1.2.1 Abstract and Motivation

Beyond the simple warping and geometric modification techniques, homographic transformation is another useful and trending skill. More and more panorama pixtures are required both in industry and daily life. Due to the limitation of the wide angle cameras, how to stitch images that taken from the same place with different angles seems to be very usful.

In this part, I run SURF to extract features from left, middle pair images and right, middle pair images. After that, I selected 4 control points for each pair to compute the transform matrixes. And then, apply the matrix to the images to produce the panorama. The test images are shown as Figure 11. Also, the selection of the control points will be discussed, the performances of different control points will be compared.

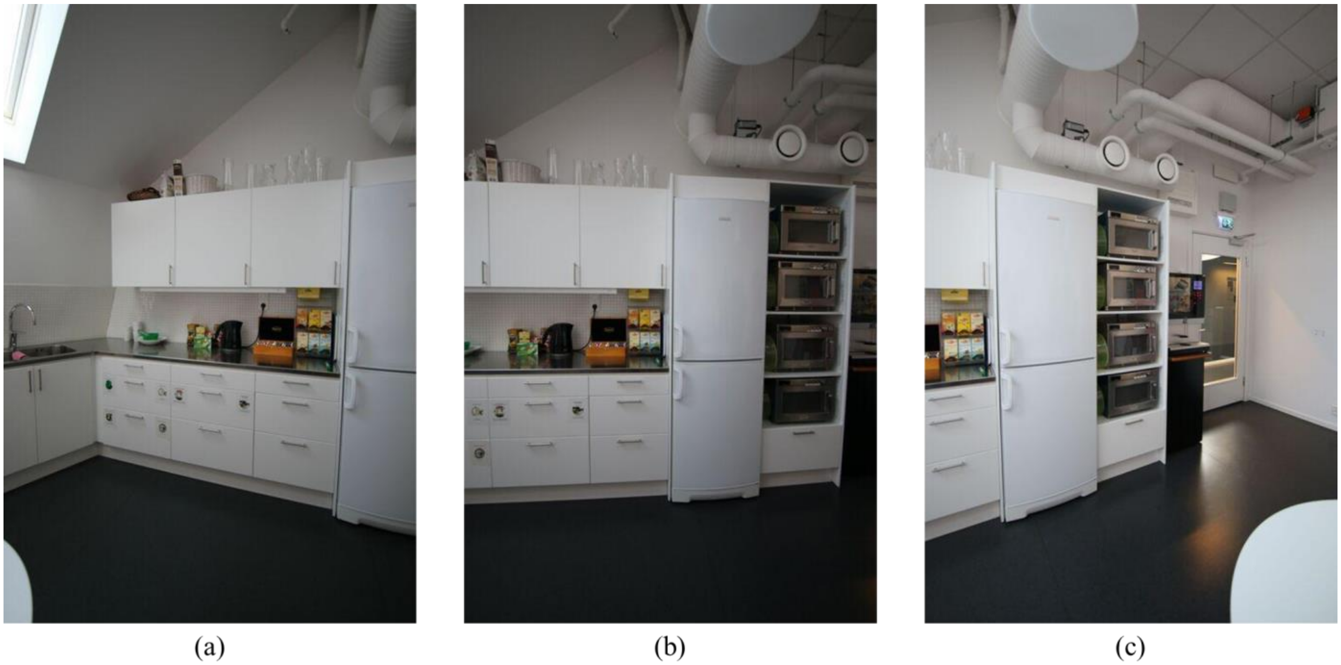

(a) (b) (c)

Figure 11. Test Images of Image Stitching

1.2.2 Approach and Procedures

SURF: There are six main steps of SURF procedure.1. Construct the Image Pyramids. In this step, we want to extract the scale invariant features, so we need to construct the image pyramid before apply the feature extraction matrix. 2. Construct the Hessian Matrix. For each scale, we use the correspongding Hessian Matrix to extract features. Before applying the matrix, a Gaussian filter is needed. 3. Generate the scale space. Rather than changing the scale of the image in octaves, SURF changes the filter scale and keep the image as the same size. 4. Non maximum suppression. After feature extraction, SURF uses NMS to locate the feature points more precisely. 5. Find the main direction of the feature points. To eansure the rotation invariant feature points, SURF analyzes the Harr wavelet feature of the neighbour pixles and gives them different weights to the feature points. 6. Construct the feature point descriptor. Use the direction and the features to creat the descriptor.

Transform Matrix: The panorama transformation follows as below as equation (3-4). Where the x1, y1 denotes the original coordinates and x2, y2 is the desired coordinates. Based on the equation (3-4), we can easily get equations (5-7).

$$\begin{bmatrix} x_2{}' \\ y_2{}' \\ w_2{}' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_2{}'/w_2{}' \\ y_2{}'/w_2{}' \end{bmatrix} \tag{4}$$

$$x_2' = H_{11} * x_1 + \ H_{12} * y_1 + H_{13} \tag{5}$$

$$y_2' = H_{21} * x_1 + \ H_{22} * y_1 + H_{23} \tag{6}$$

$$w_2' = H_{31} * x_1 + \ H_{32} * y_1 + H_{33} \tag{7}$$

Back subsitibute the equation (7) into equation (4), we can also get equation(8-9).

$$x_2' = H_{31} * x_1x_2 + \ H_{32} * y_1x_2 + x_2 \tag{8}$$

$$y_2' = H_{31} * x_1y_2 + \ H_{32} * y_1y_2 + y_2 \tag{9}$$

Back subsitibute the equation (8-9) into equation (5-6), we can also get equation(10-11).

$$x_2 = H_{11} * x_1 + H_{12} * y_1 + H_{13} - H_{31} * x_1x_2 - H_{32} * y_1x_2 \tag{10}$$

$$y_2 = H_{21} * x_1 + H_{22} * y_1 + H_{23} - H_{31} * x_1y_2 - H_{32} * y_1y_2 \tag{11}$$

Based on the equation (10-11), we can get equation (12).

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_2 & -y_1x_2 \\ 0 & 0 & 0 & x_1 & y_2 & 1 & -x_1y_2 & -y_1y_2 \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \end{bmatrix} \tag{12}$$

Based on the equation (10-11), we can get equation (12).

Equation (12) tells that the transform Matrix can be computed by knowing the target locations and original locations. Also, there are 8 unknown variables, which means we need 4 points to build 8 equations to solve (12).

The whole procedure of the image stitching is shown as a diagram below in Figure 12.

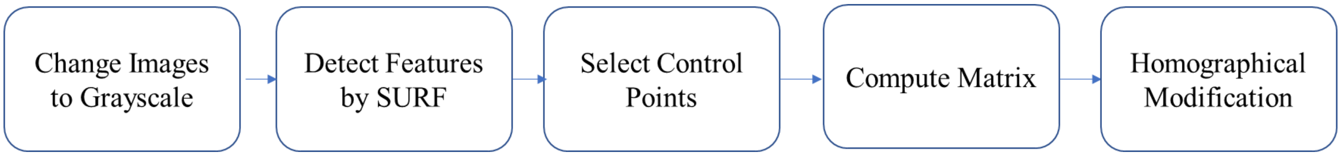


Figure 12. Diagram of Image Stitching

1.2.3 Experimental Result.

The result images that produced by pasting middle image and averaging the overlap region are shown in Figure 13 and Figure 14.

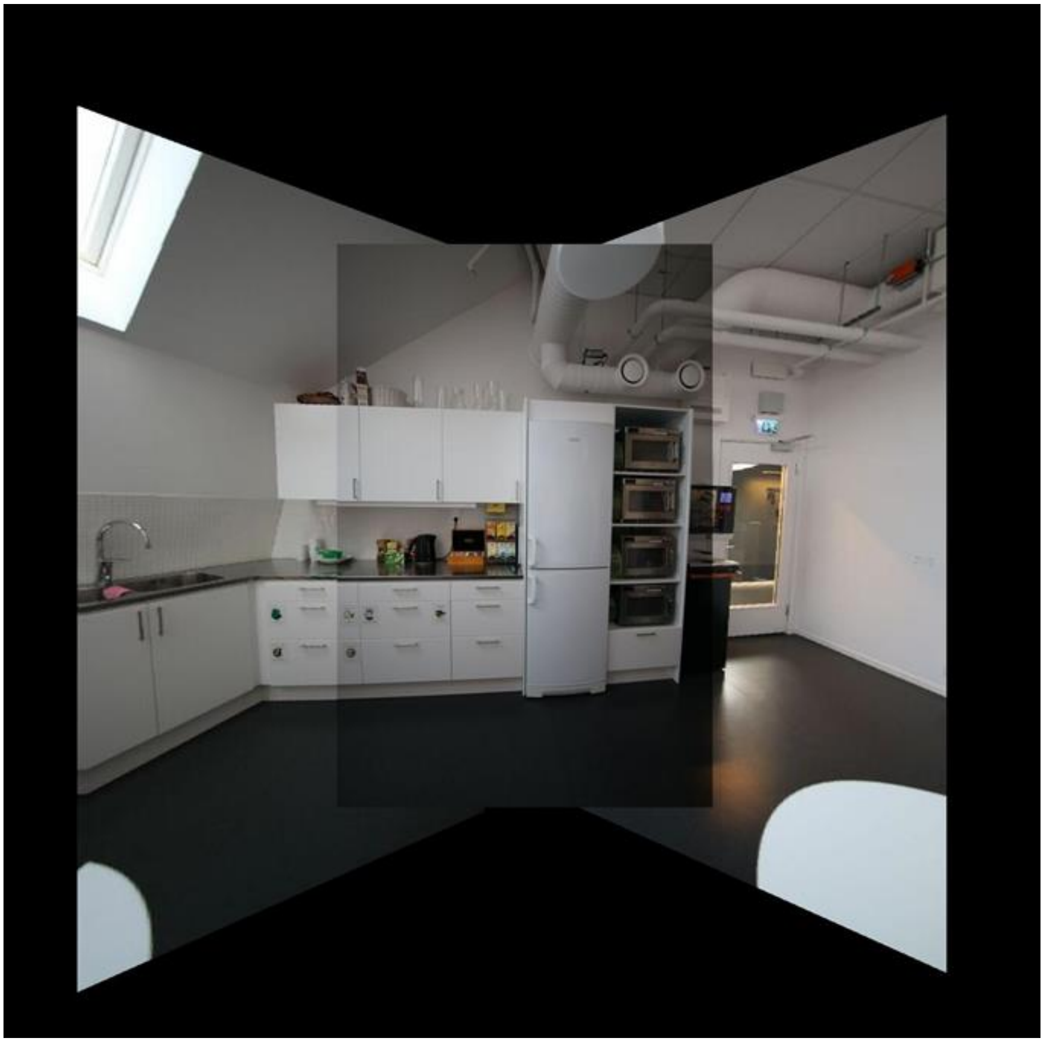


Figure 13. Result Image by Pasting Middle Image

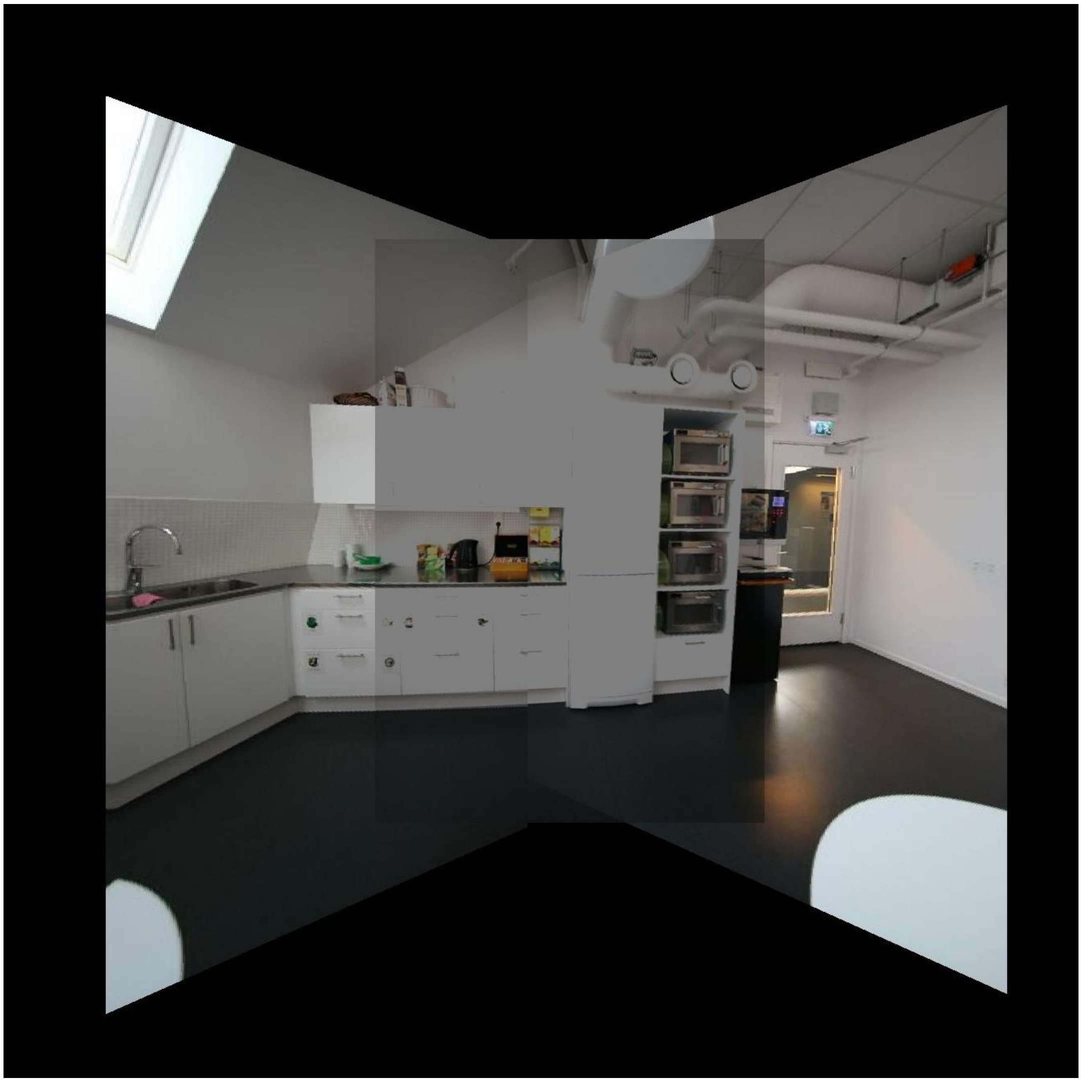
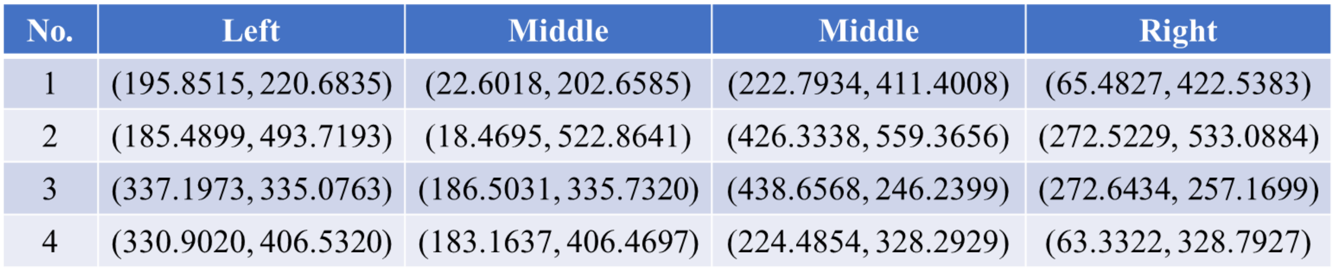


Figure 14. Result Image by Averaging Middle Image

The four control point pairs between left image and middle image and between middle image and right image are shown as Table 1.

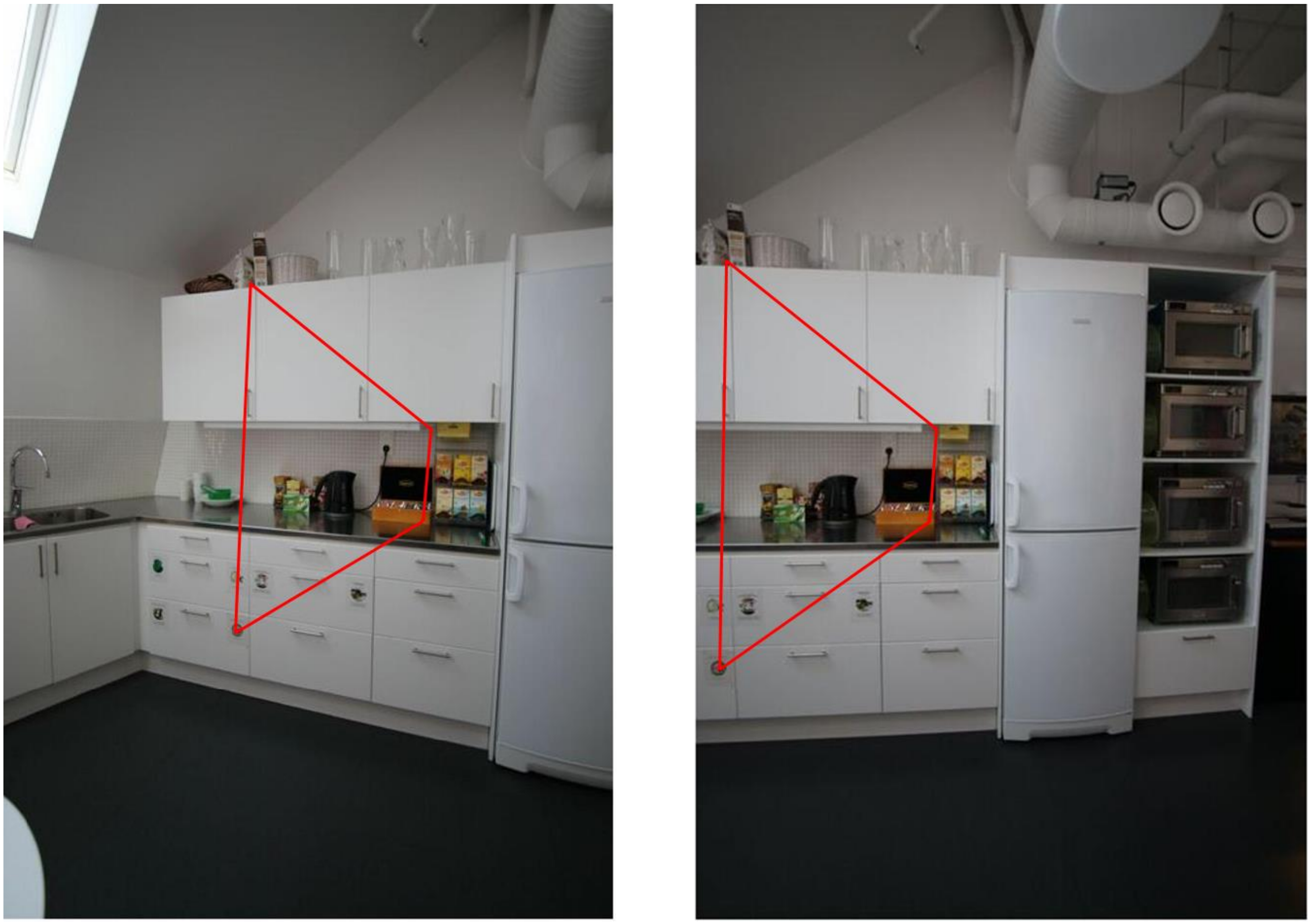
Table 1. Control Point Pairs in Panorama



No.	Left	Middle	Middle	Right
1	(195.8515, 220.6835)	(22.6018, 202.6585)	(222.7934, 411.4008)	(65.4827, 422.5383)
2	(185.4899, 493.7193)	(18.4695, 522.8641)	(426.3338, 559.3656)	(272.5229, 533.0884)
3	(337.1973, 335.0763)	(186.5031, 335.7320)	(438.6568, 246.2399)	(272.6434, 257.1699)
4	(330.9020, 406.5320)	(183.1637, 406.4697)	(224.4854, 328.2929)	(63.3322, 328.7927)

1.2.4 Discussion

The control points between left and middle image is shown in Figure 15. The red dots are connected by the red lines. So the control points and the regions can be more clearly presented. The control points formes a trapezoid-like region, which located at the center of the image.

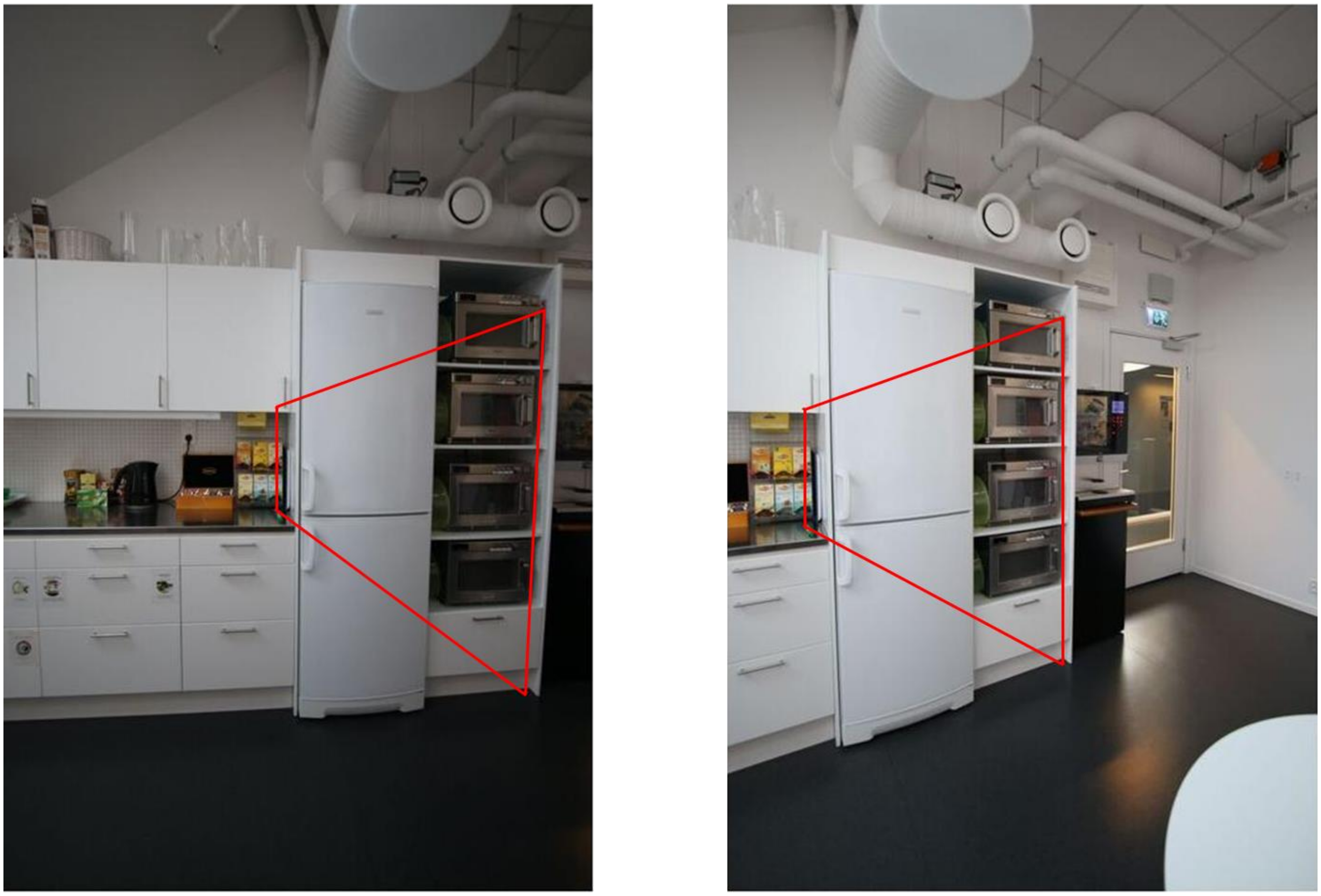


(a) Left Image Control Points

(b) Middle Image Control Points

Figure 15. Control Points Between Left and Middle Image

The control points between middle and right image is shown in Figure 16. The red dots are connected by the red lines. So the control points and the regions can be more clearly presented.



(a) Middle Image Control Points

(b) Right Image Control Points

Figure 16. Control Points Between Middle and Right Image

Number of Control Points: In this part, I used 4 pair of control points to calculate the transform matrix between each image pair. This number of conrol point is the minimum number to solve the equation (12) to get the transform matrix. From Figure 13 and Figure 14, we can see that the three images matches really well based on the control points. It means the minimum number of points can be sufficient to get the correct matrix if the points are selected properly. For example, I selected 4 points which can form a trapezoid-like region. This trapezoid locates near the center of the image and big enough to contain most of the feature points.

Averaging: From Figure 13 and Figure 14, we can see that directly paste the middle image into the canvas has a better perfomance. But it is not practical if we donot know the order of the images. The averaged middle image has some blurry pixels and unclear edges. Thi s is caused by the error of the left and right image position, simply average the pixel values enlarge this error on the overlapped region. Also, the three images have different brightness, the averaging caused some pixels and region to be too bright like the pipe on the roof and the surface of the refrigrator. Although, average causes some artifacts and problems, it is still the most scientific way to composite the panorama regardless of the order of the images.

Selecting the Control Points: The selection of the control points can significantly affect the stitching performance. The comparison of different selection of the control points are shown as Figure 17. And the location of the control points are shown in Figure 17, too. We can tell from Figure 17 that the performance is much worse on the right image transformation if the control points are changed. The 4 control points forms a triangle-like region, which means three of them are nearly on the same straight line. Based on this, we can consider it in linear algebra that this case can cause an unstable system. The error of the position can be easily enlarged by this error-sensitive system. Also, the control region is too small, it will not be precise to estimate the whole image pixels based on the matrix that generated by this region.

Procedure to find the proper control points, is not easy. Firstly, I use SURF to detect features and the matched points between left and middle,

middle and right image pairs. I got 67 mathed points for each image pair. For these possible control points, I changed the corresponding pixels and their neighbours to red in the original image to view them directly. From the red dots images, I tried to select four points that forms a rectangle, but the matched points are mostly located on the boxes next to the fridge. It is hard to find a large enough rectangle which can form a robust system equation and large enough to get the correct calculation. So, I decided to go with the trapezoid like region which is large enough in the image.

Of course, I tried many other points, and find a best performed one as shown in Figure 13 and 14. There are different metrics to select control points in different cases. But in this problem, I find some useful metrics which can help to create better performances.

1. The control points should form as large region as they can. In this case, the transformation of the whole image can be fully represented rather than only projected based on a small persperctive.
2. The control points should not be on the same line. The equation (12) can be an unstable system to solve if the 4 points are not entirely linearly independent.
3. A rectangle or quadrilateral region is prefered. To achieve the bullet two, a rectangle region might be the best case for 4 linearly independent points.
4. More control points can get more precise result. It is easy to think that more points will increase the precision of equation (12).

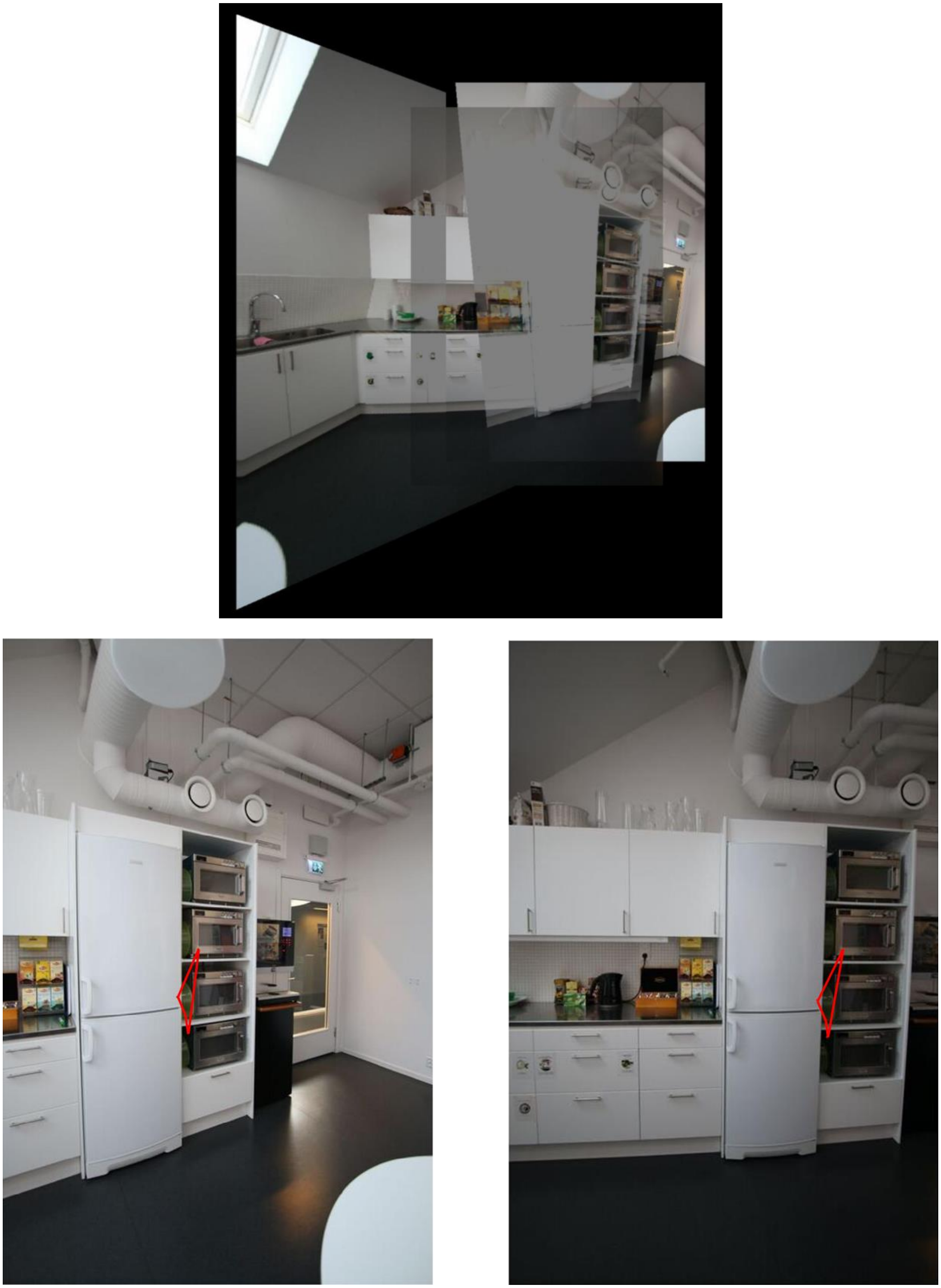


Figure 17. Other Choice of Control Points and Result

2. Problem 2

2.1 Basic Morphological Process Implementation

2.1.1 Abstract and Motivation

To view the image or object structures in digital images, morphological process is used in bineary images. There are two kinds of morphological operations—dilation and erosion. Dialation inserts white pixels to enlarge the objects and extend the boundaries of the objects. Erosion dose the opposite, removes the white pixels to reduce the obeject boundaries. In the industry, erosion operations and morphological process can be used in defect detection and object counting,

In this part, I implemented three functions of erosion, shrinking, thining and skeletonizing based on the given pattern table for each operatsion. The algorithms are applied on three grayscale images shown in Figure 18. The shortcomings and problems of the pattern tables and the algortithm it self will be discussed.

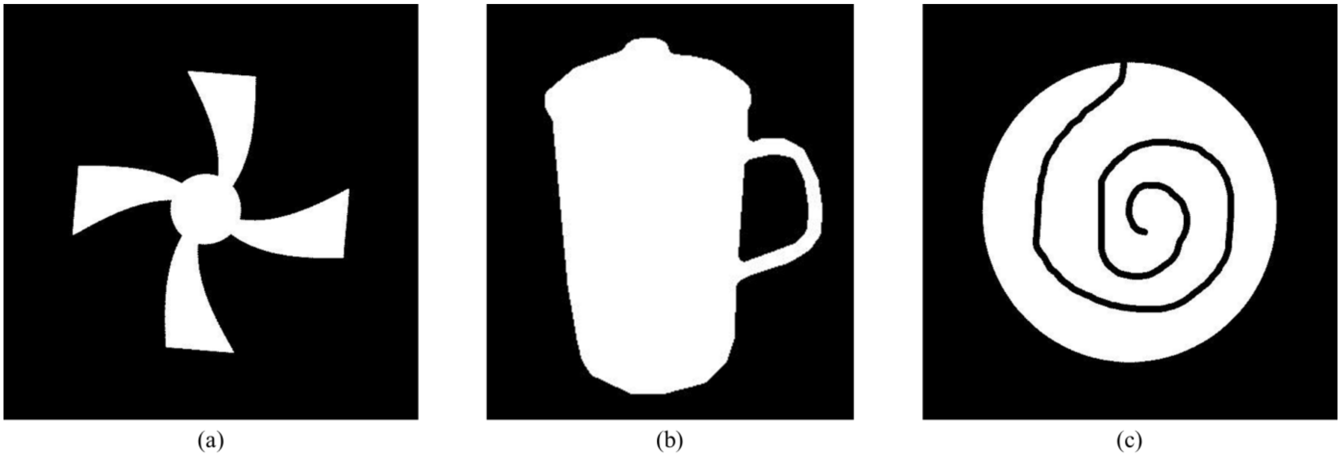


Figure 18. Test Images of Morphological Operations

2.1.2 Approach and Procedures

Hit or Miss： Hit-or-miss transform is an operation that used to detect the given patterns in binary image. It is one of the most basic operations in image morphological processing. The operator constructs the structure of the center pixel and compare it to the given pattern table. If the foreground pixel values and background values exactly matches some pattern in the table. The pixel would be set to the foreground color, otherwise, it will be set as the background color. This scheme are widely used in morphological processing while comparing the current pixel pattern to the conditional and uncondtional pattern tables.

Encode: There are many ways to encode the eight-neighbour of a certain pixel. We can use 2-D arrays or bit strings or character strings to encode the pattern tables. To save the storage room and speed up the algorithm, I use a structured sum of the eight-neighbour pixels to identify every pattern. Each pixel of the eight-neighbour of current pixel has been assigned a value from 1, 2, 4, 8, 16, 32, 64 or 128. So, the neighbour of certain pixel of a binary image can be expressed uniquely by a number between 0~255. Each number represents a unique structure, which prevents the error detection and overlap. The structure of the encodes are shown in Figure 19. This scheme can only use one byte(0~255) to store each pattern, and it also precvents the multiple logic check for the pattern comparison.

In general, this scheme can speed up the algorithm and save the storage usage in programming.

128	*64*	*32*
16	*1*	*8*
4	*2*	*1*

(a) Encode Scheme

1	*0*	*0*
1	*1*	*0*
0	*1*	*1*

(b) Encode Example
E = 1+2+16+128

Figure 19. Encoding Scheme

Procedures: These three operations' procedure are very similar. Firstly, we scan the image in raster order and find the eight-neighbor pixels of the current pixel. Calculate the sum of these pixels by the encoding method and get the pattern. Compare this pattern with the corresponding conditional pattern table of each operation. After comparing with conditional table, we can get the Mask Image. Then compare the mask image with the unconditional pattern tables and inverse the mask value if the pattern matches. Recursively apply this scheme to the image until the image converges, which means no pixel value changes. The diagram of the procedure is shown as Figure 20.

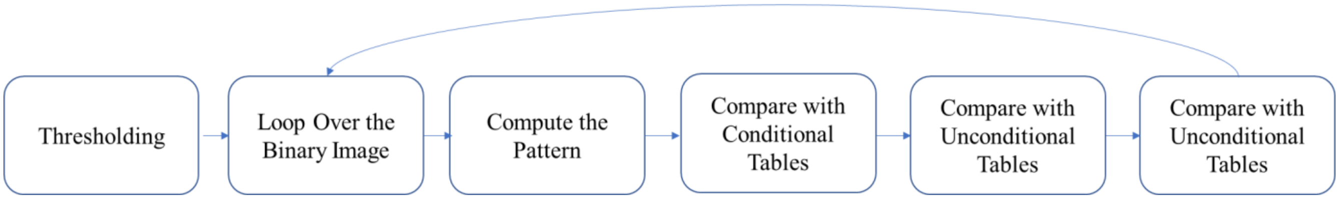

Thresholding
Loop Over the Binary Image
Compute the Pattern
Compare with Conditional Tables
Compare with Unconditional Tables
Compare with Unconditional Tables

Figure 20. Procedure of Basic Morphological Processing

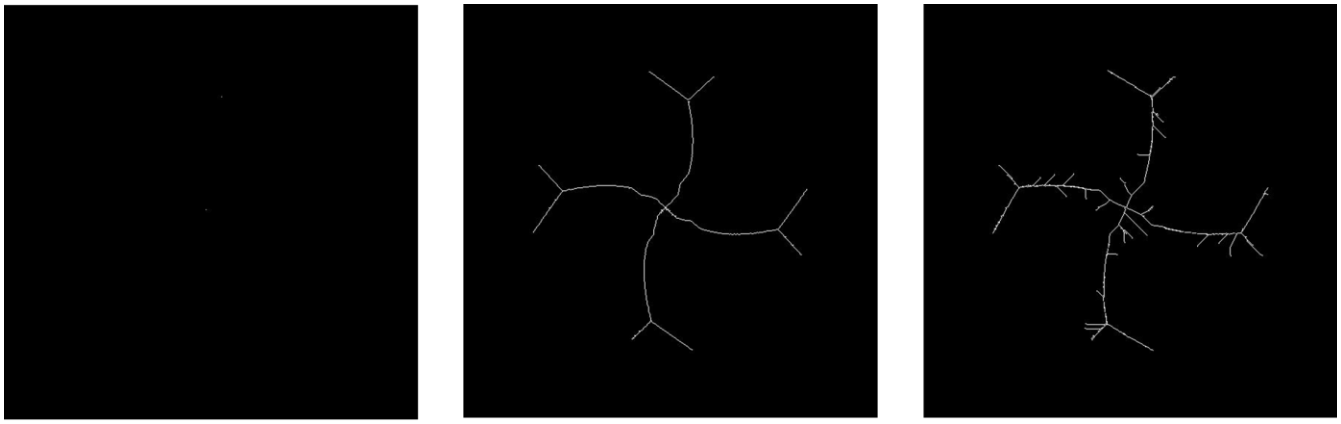
Padding: In previous homework problems, I implemented the reflection padding, which was used in the image demosacing and image denoising. In this part, to avoid generate the inexisted pattern at the wrong locations. The zero padding should be applied rather than reflection padding. It is easy to see that there is no white pixels on the boundaries of the test images. So, it is also reasonable to do no padding and start the loop from the second row and the second row.

Thresholding: Beause the given test images are grayscale rather than binary. It is necessary to do the thresholding before applying the morphological operator. In the experiment, I found that the threshold can effect the result significantly, a proper theshold can prevent an object breaks into two and creat more precise result.

2.1.3 Experimental Result

Final Results:

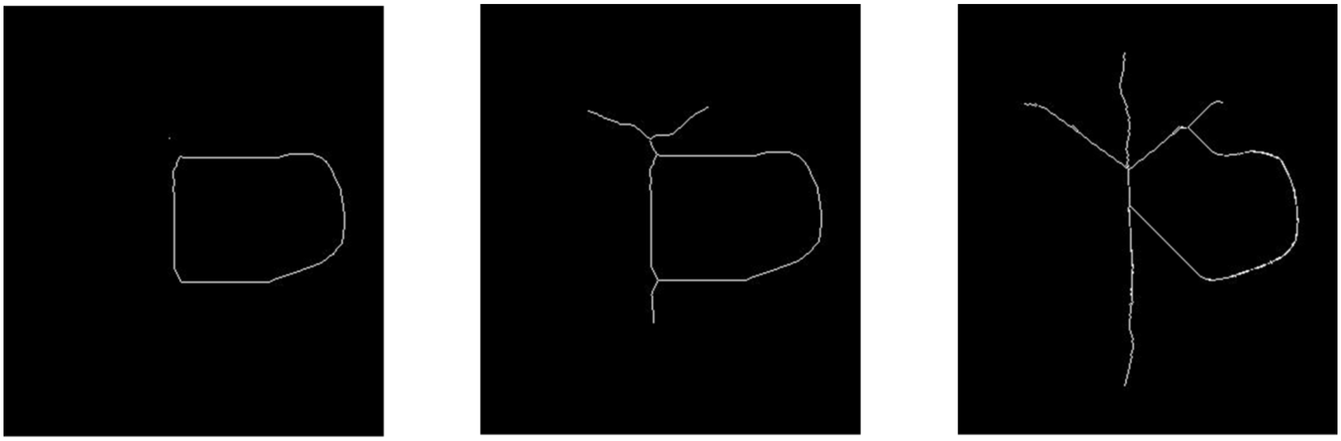
The result images of fan.raw are shown in Figure 21.



(a) Fan Shrinking (b) Fan Thinning (c) Fan Skeletonizing

Figure 21. Results of fan.raw

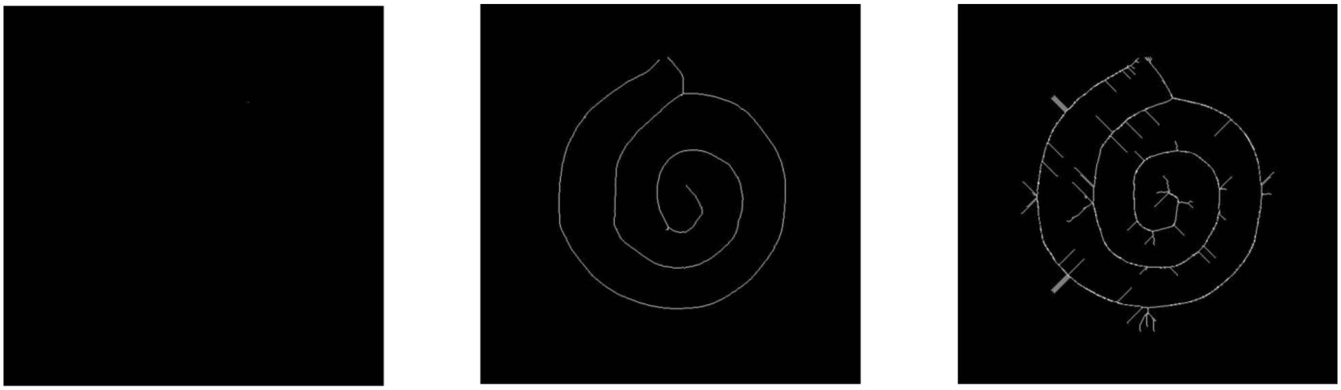
The result images of cup.raw are shown in Figure 22.



(a) Cup Shrinking (b) Cup Thinning (c) Cup Skeletonizing

Figure 22. Results of cup.raw

The result images of maze.raw are shown in Figure 23.



(a) Maze Shrinking (b) Maze Thinning (c) Maze Skeletonizing

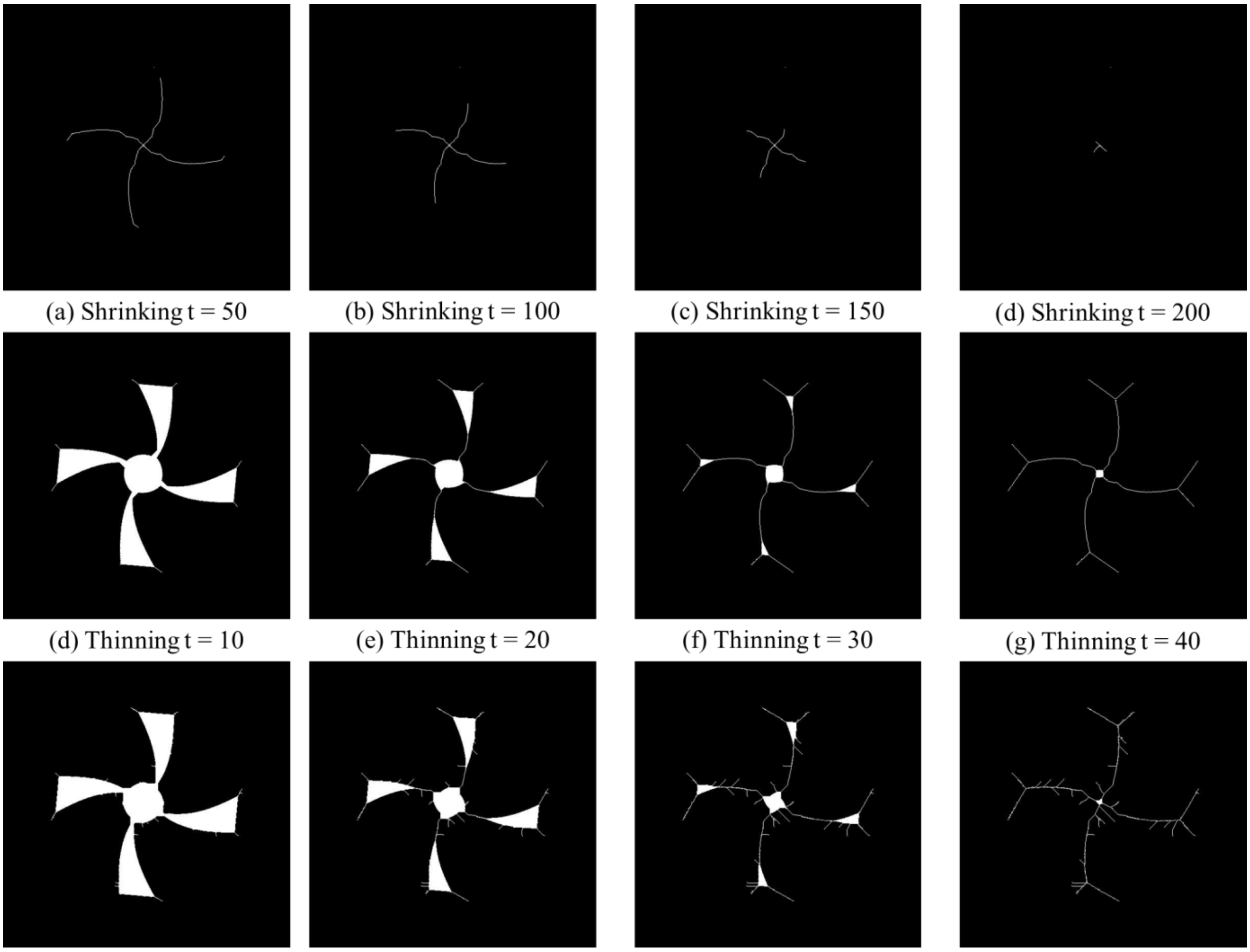
Figure 23. Results of maze.raw

Intermediate Results:

The intermediate processing result image can show the whole procedure how the erosion can be down. For each image, there are different iteration times it uses. So, the intermediate processing image is shown as the order of the iteration times.

For each operation, based on their iteration thimes, I divided the process into 4 phases and noted the iteration times.

The intermediate processing result images of fan.raw and cup.raw are shown in Figure 24 and Figure 25.



(a) Shrinking t = 50

(b) Shrinking t = 100

(c) Shrinking t = 150

(d) Shrinking t = 200

(d) Thinning t = 10

(e) Thinning t = 20

(f) Thinning t = 30

(g) Thinning t = 40

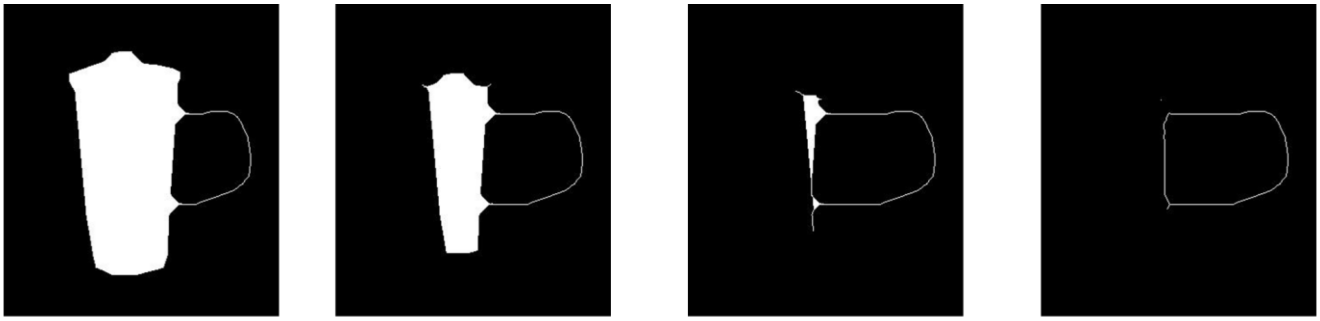
(h) Skeletonizing t = 7

(h) Skeletonizing t = 15

(h) Skeletonizing t = 23

(h) Skeletonizing t = 31

Figure 24. Intermediate Processing Results of fan.raw



(a) Shrinking t = 25

(b) Shrinking t = 50

(c) Shrinking t = 75

(d) Shrinking t = 100

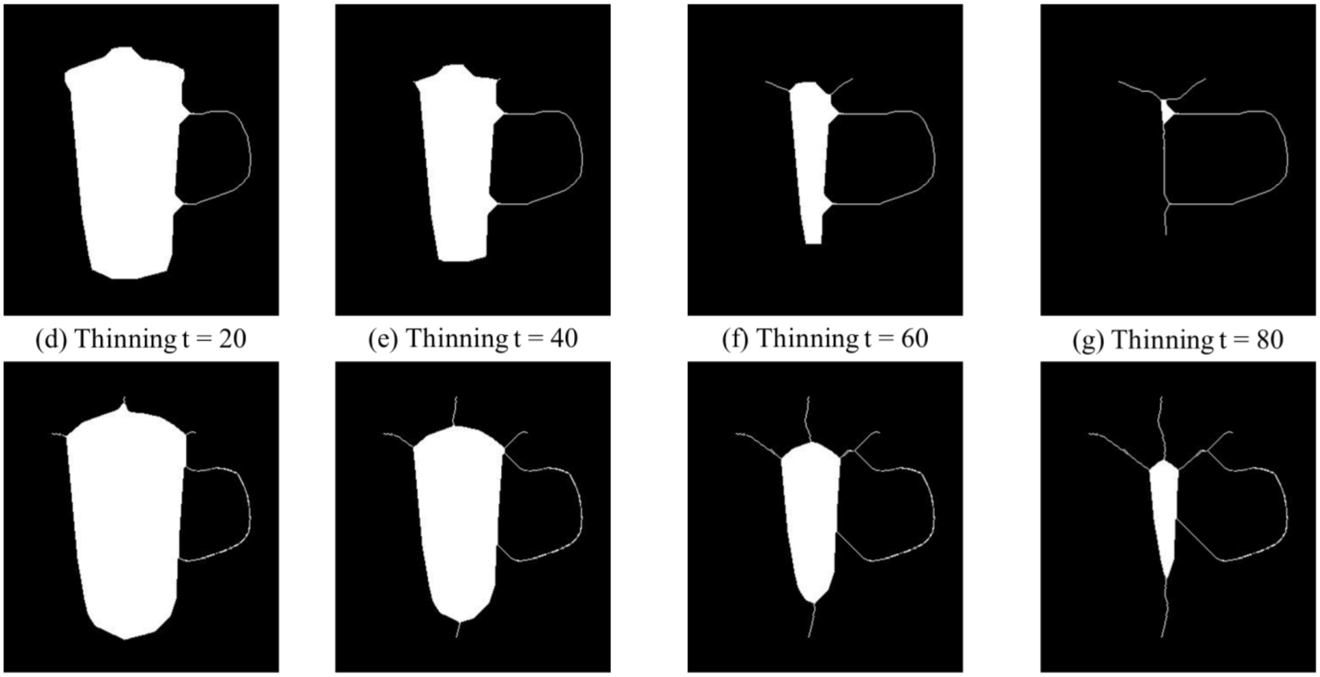
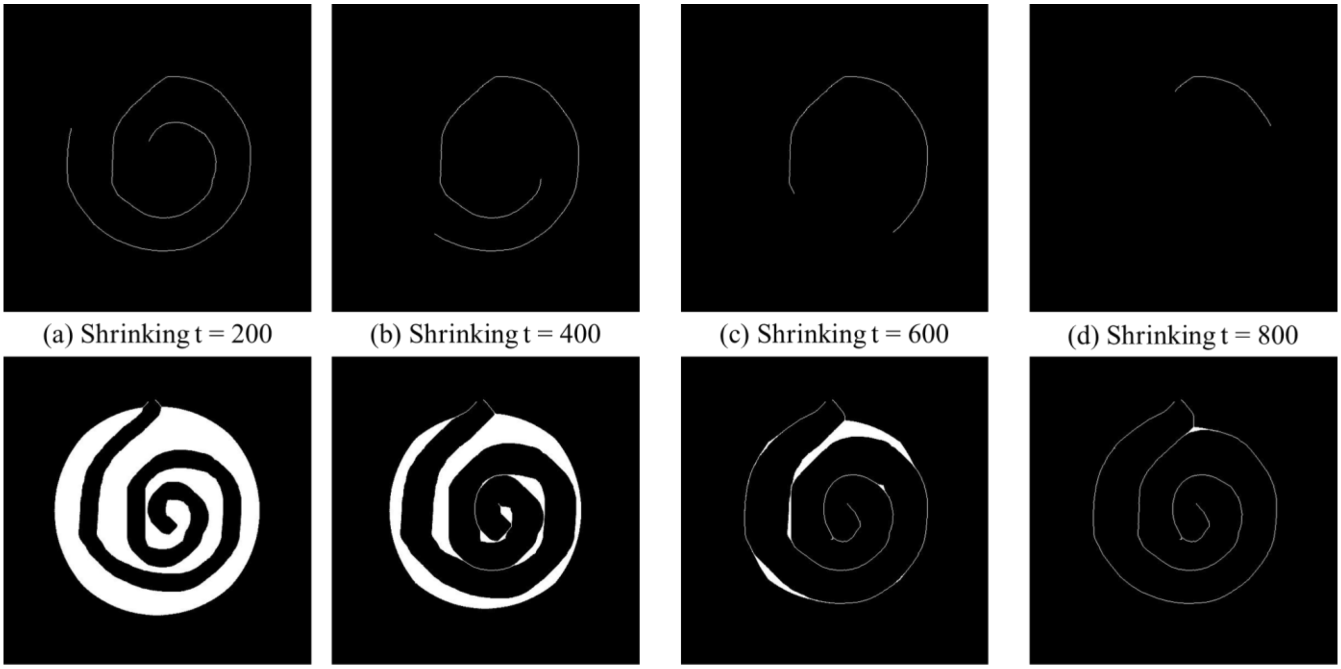
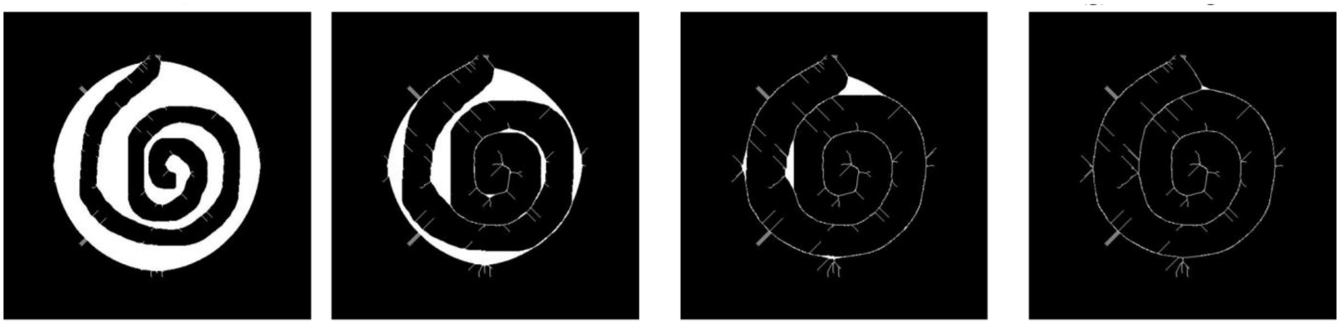

(d) Thinning t = 20
(e) Thinning t = 40
(f) Thinning t = 60
(g) Thinning t = 80
(h) Skeletonizing t = 15
(h) Skeletonizing t = 30
(h) Skeletonizing t = 45
(h) Skeletonizing t = 60

Figure 25. Intermediate Processing Results of cup.raw

The intermediate processing result images of maze.raw are shown in Figure 26.


(a) Shrinking t = 200
(b) Shrinking t = 400
(c) Shrinking t = 600
(d) Shrinking t = 800
(d) Thinning t = 12
(e) Thinning t = 24
(f) Thinning t = 36
(g) Thinning t = 48



(h) Skeletonizing t = 10 (h) Skeletonizing t = 20 (h) Skeletonizing t = 30 (h) Skeletonizing t = 40

Figure 26. Intermediate Processing Results of maze.raw

The iteration times before converging of each images with diffferent methods are shown as Table 2.

Table 2. Iteration times of Different Methods and Test Images

Iteration Times	fan.raw	cup.raw	maze.raw
Shrinking	219	106	902
Thinning	49	90	54
Skeletonizing	35	80	44

2.1.4 Discussion

Three Methods Comparison: The three operators peroforms differently in image erosion. Form the images above, we can see that the shrinking operator shrinks image into a dot if the image is ideal and no hole in it. If there is a black hole in the object, a ring might be generated by shrinking operator since the erosion can been both done from the outside and the inside. For thinnig operator, it should shrink the image until the object has only one pixel in its length or height. In these three test images, it gives a overall sructure of the object. While we still can tell from the maze image that the width of the object has been erased to 1 pixel only. The skeletonizing gives the more detailed structure of the object—the skeleton of the object.

Shortcomings and Improvments: We can see from the thinning images and skeleton images that there are some jaggies in both of them. But for thinned images, there are fewer jaggies in the object outline. The jaggies in the skeletonized image is more obvious than the shrinking and thinning ones. And for skeletonized image, the result might not be 1-pixel wide, this might caused by the lack of the pattern tables. To improve this unexpected problem, we can use a more robust pattern table or check the eight neighbour region to erase some useless pixels.

For fan.raw, because there is no hole in the object, so the shrinking result should be one pixe, while the real result is two. And the thinning result gives the outlines of the fan. The skeletonizing result generates its skeleton and some branches. For cup.raw, there is a hole in the right of the cup, so the shrinking result is a ring, with an unexpected dot above it. The thinning and skeletonizing gives the structure of the cup. For maze.raw, the shrinking result is a white dot, and the skeletonized result has unexpected branches, too.

Unexpected Results: The result images have some unexpected artifacts which caused by the incorrect pattern tables. First of all, the shrinking image of fan.raw generates two white pixels rather than one. And the skeletonizing of fan.raw and maze.raw generates some unexpected branches. These problems might caused by the incorrect pattern tables. There are no mask in the unconditional table to preserve the patterns like Figure 27 shows. Some connection of pixels are erased because of this problem. And the division and the branched are generated then.

0	*0*	*0*
0	*1*	*1*
0	*1*	*0*

(a)

0	*1*	*1*
0	*1*	*0*
0	*0*	*1*

(b)

Figure 27. Example of Missing Patterns

2.2 Counting Games

2.2.1 Abstract and Motivation

The morphological operations are really useful in the real life in the object analysis. Counting objects based on the digital images is another application of the morphological techniques.

In this part, I used two different methods to count the stars in a binary image, where stars are represented by the white regions. The number of the stars is going to be counted, a histogram of stars' sizes are going to be draw and the comparison of the methods are going to be discussed. The test image is shown as below in Figure 28.

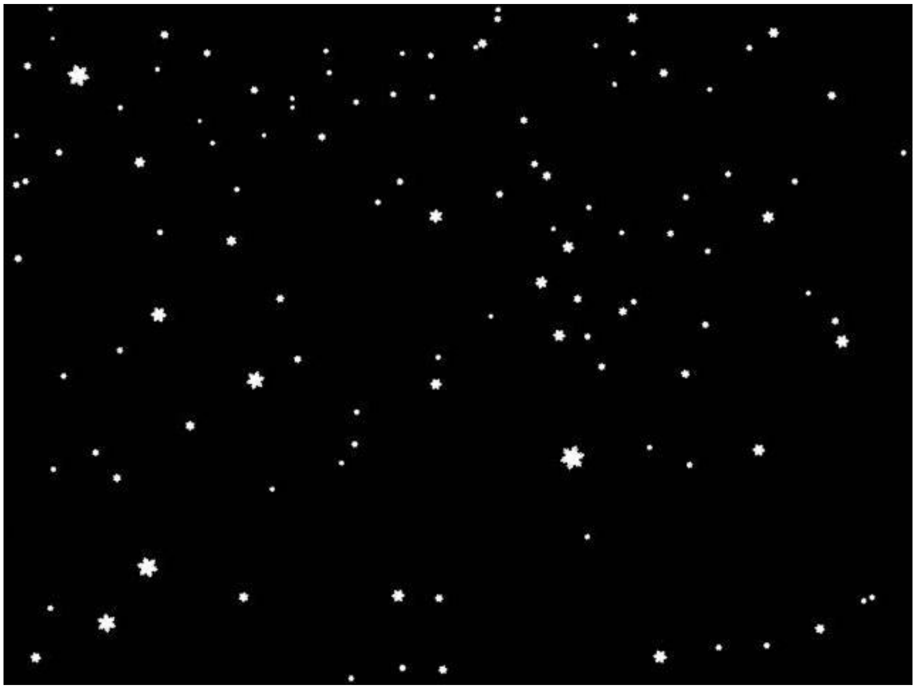


Figure 28. Stars.raw

2.2.2 Approach and Procedures

Morphological Method: As the image shown above, we can see that the stars are all closed objects, which means there are no holes or rings in them. So, the shrinking method could be useful to count the star number. Because the stars will shrink to a dot, and count the white pixels in a binary image is an easy job to do. About the satistics of the star size, we can use the original image to find every white pixel belongs to which star. And get the number of pixels of each star, then plot the histogram. The diagram of the morphological method is shown as below in Figure 29.

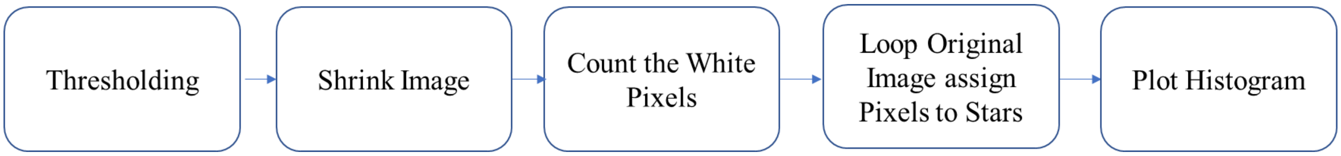


Figure 29. Morphological Method Procedure

Connected Component Labeling: The connected labeling is anther method to count the number of distinct objects in an image. It uses the 8-connectivity of a pixel to label the white pixel. And has a special scheme to check whether two labels are equivalent to simplify the labeled image. This method's procedure is shown as below in Figure 30.

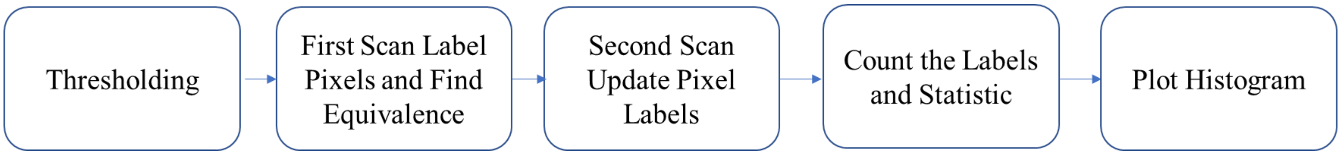
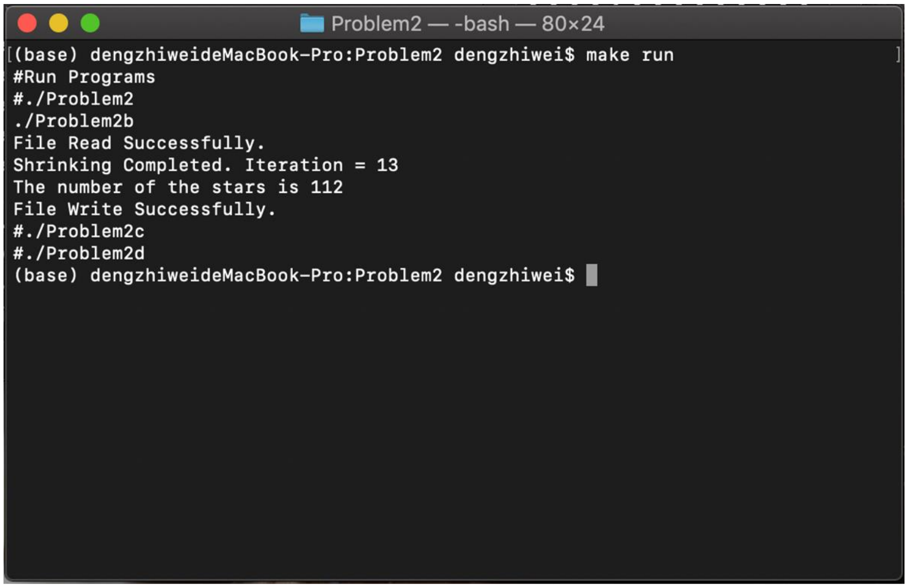

Thresholding
First Scan Label Pixels and Find Equivalence
Second Scan Update Pixel Labels
Count the Labels and Statistic
Plot Histogram

Figure 30. Connected Component Labeling Procedure

Histograms: The size of a star can be measured in two ways, pixel numbers of the star or the longest L2 distance between two pixels that belong to the same star. In this part, I use the former one since the pixel numbers are all integers, which are more convinient to plot. Also, the morphological process changes the information of the image, which means that the L2 distance might be not precise enough to measure a star's size.

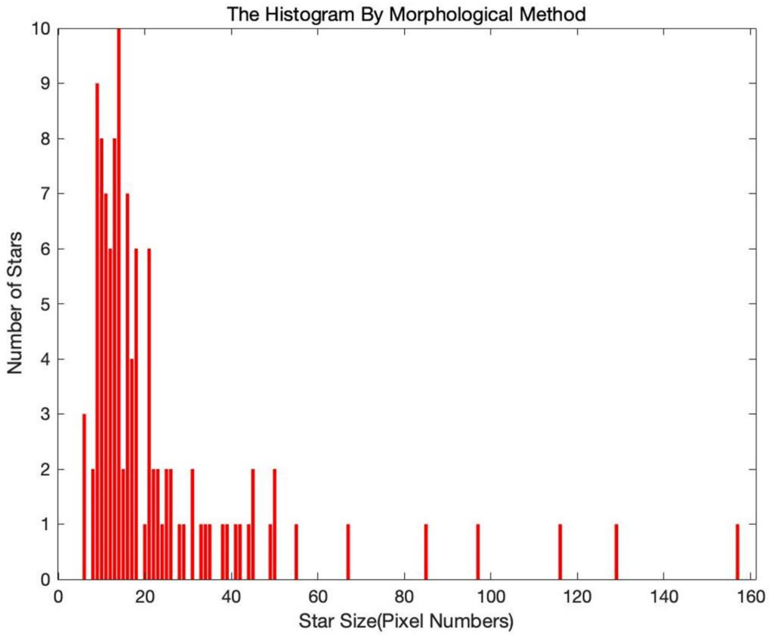
2.2.3 Experimental Result

The result generated by Morphological method is shown as below in Figure 31.



```
[(base) dengzhiweideMacBook-Pro:Problem2 dengzhiwei$ make run
#Run Programs
#./Problem2
./Problem2b
File Read Successfully.
Shrinking Completed. Iteration = 13
The number of the stars is 112
File Write Successfully.
#./Problem2c
#./Problem2d
(base) dengzhiweideMacBook-Pro:Problem2 dengzhiwei$ 
```

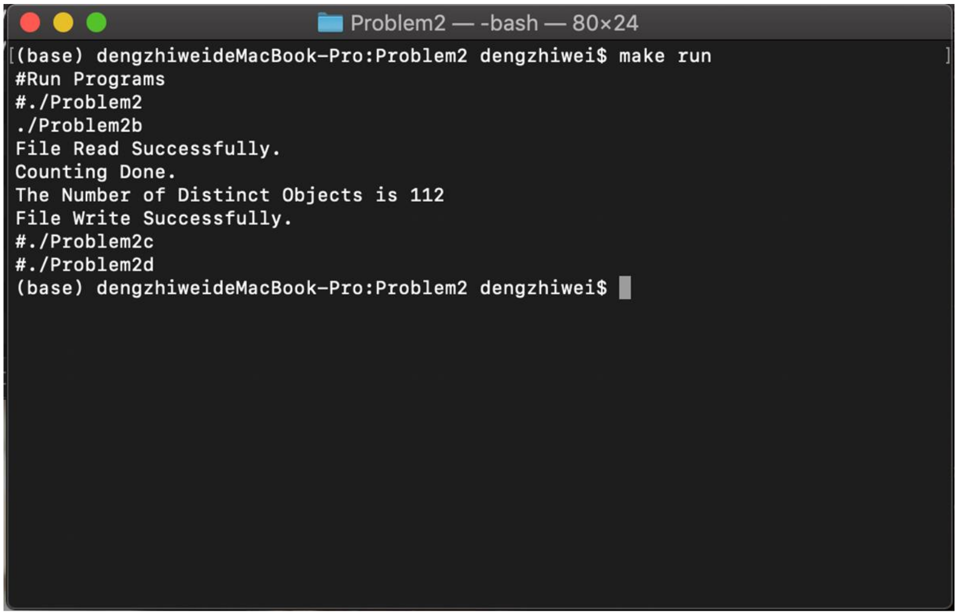
(a) Counting Result



(b) Histogram

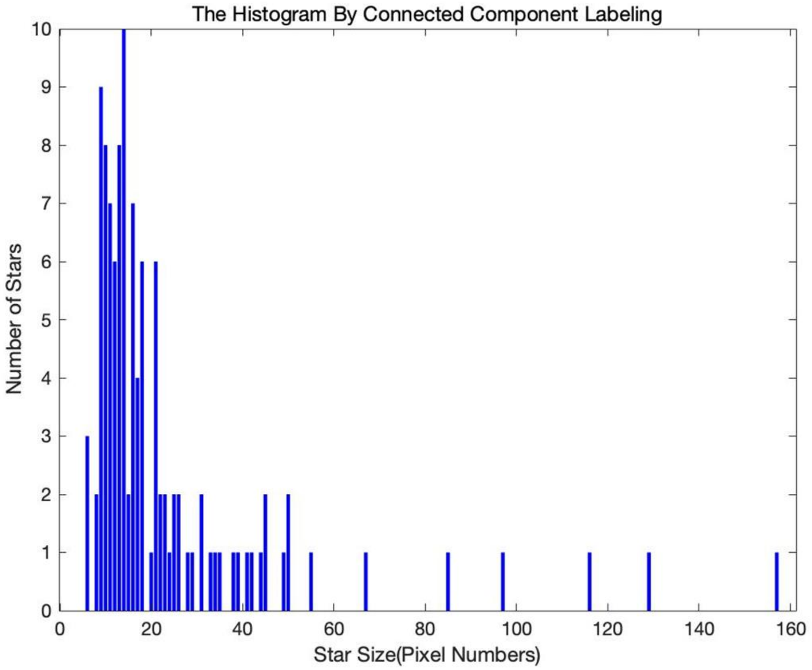
Figure 31. Morphological Counting Result

The result generated by Connected Component Labeling is shown as below in Figure 32.



```
[(base) dengzhiweideMacBook-Pro:Problem2 dengzhiwei$ make run
#Run Programs
#./Problem2
./Problem2b
File Read Successfully.
Counting Done.
The Number of Distinct Objects is 112
File Write Successfully.
#./Problem2c
#./Problem2d
(base) dengzhiweideMacBook-Pro:Problem2 dengzhiwei$
```

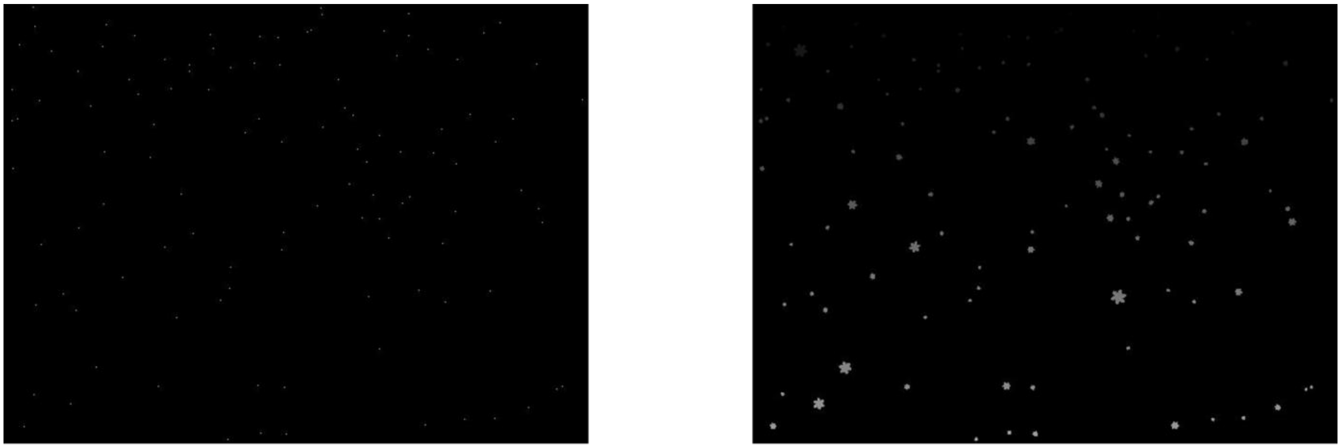
(a) Counting Result



(b) Histogram

Figure 32. Connected Component Labeling Result

The shrinked image of Stars.raw and the labeled image with different grayscale are shown in Figure 33.



(a) Stars shrink

(b) Connected Component Labeling

Figure 33. Shrinked Image and Labeled Image

2.2.4 Discussion

Number of Stars: 112. This number is counted by both morphological method and the connected component labeling method.

Histograms of Star Sizes: As shown in Figure 31 and Figure 32.

Explanation of Methods: 1. For morphological method, firstly, I thresholded the original image to get the binary image. Then I used shrinking transform to change the stars regions to individual white pixels. Then, to count the number of the stars, I just checked the number of the individual white pixels as the number of the stars. Also, I recorded every white pixel’s location as the center of each star. As to the histogram, I used the unshrinked binary image, for every white pixel, I calculated the distances between the pixel and every center of the stars and chose the minimum value index as the star it belongs to. The minimum distance means that the pixel is nearest to the certain star, so the star's size was added by 1. After all procedure, I output it as a .txt file as its histogram. Then used Matlab to plot it.

2. For connected component labeling, there are two scans of the image to get the final label of each pixel. In first scan, it checks the previous neighbours of current white pixel, if the 4 pixels are all 0, then mark it as a new label. If the 4 pixels shares one same non zero value, then the pixel was assigned by this label. If the 4 pixels have different non zero values, pick the smallest one as its label and put these values into the equivalent array. The second scan is to merge the equivalent labels. It makes sure that all equivalent labels are assigned same, which means these pixels are connected. I used an array to store the parent label index for each label, after the second scan, I just subtracted the number of non zero elements in parent label arrays from the total number of the label to get the number of distinct labels, which is the number of stars.

Thresholding: The methods that described above is binary image based. So, the image should be converted to binary map first and then apply the corresponding method on it. The threshold can effect the result significantly, you can easily get 111 or 113 with incorrect threshold. One way to tune the threshold of this sub problem is to run the algorithm in different thresholds and find the most stable intervals.

More Methods: Besides the morphological process and the connected component labeling. There are other algrithms that can be applied to this sub-problem. Like depth first search, it uses self-called function and visit map to simplify the complexity of the algorithm. Its time complexity is O(m*n).

Comparison: The result of both methods are same with 112 stars and the same histogram. While the computation complexity of this two methods varies. The morphological algorithm has time complexity with O(k*m*n), where the k is the iteration times. While the connected component labeling is O(2*m*n) due to the two scan. So, the speed of later method is much faster.

Verification: The results of different methods are vverified by each other since there data and counts are exactly same. While the number of the stars are verified by myself manually, too. This result are also verified by the opensource connected labeling algorithm and so on.

2.3 PCB Analysis

2.3.1 Abstract and Motivation

The morphological techniques are also widely used in object analysis field. Using image processing techiniques to find objects and categorize them is another application of morphological transform.

In this part, I implement an algorithm to detect the number of holes and path ways of a given PCB image. The number of the holes and pathways are counted, the algorithm is going to be explained in detail and the performance of the algorithm is going to be discussed. The test image is shown as below in Figure 34.

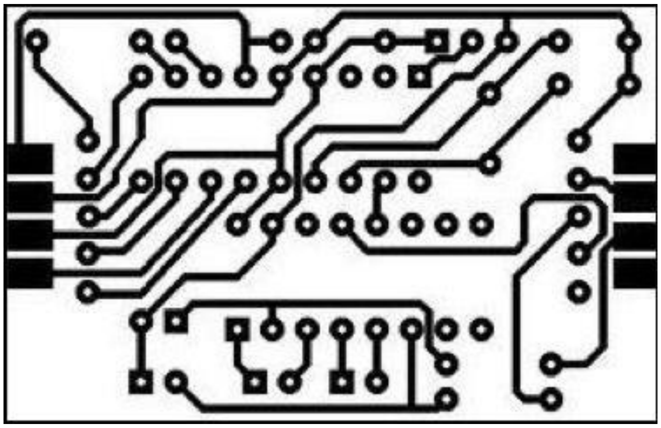


Figure 34. PCB.raw

2.3.2 Approach and Procedures

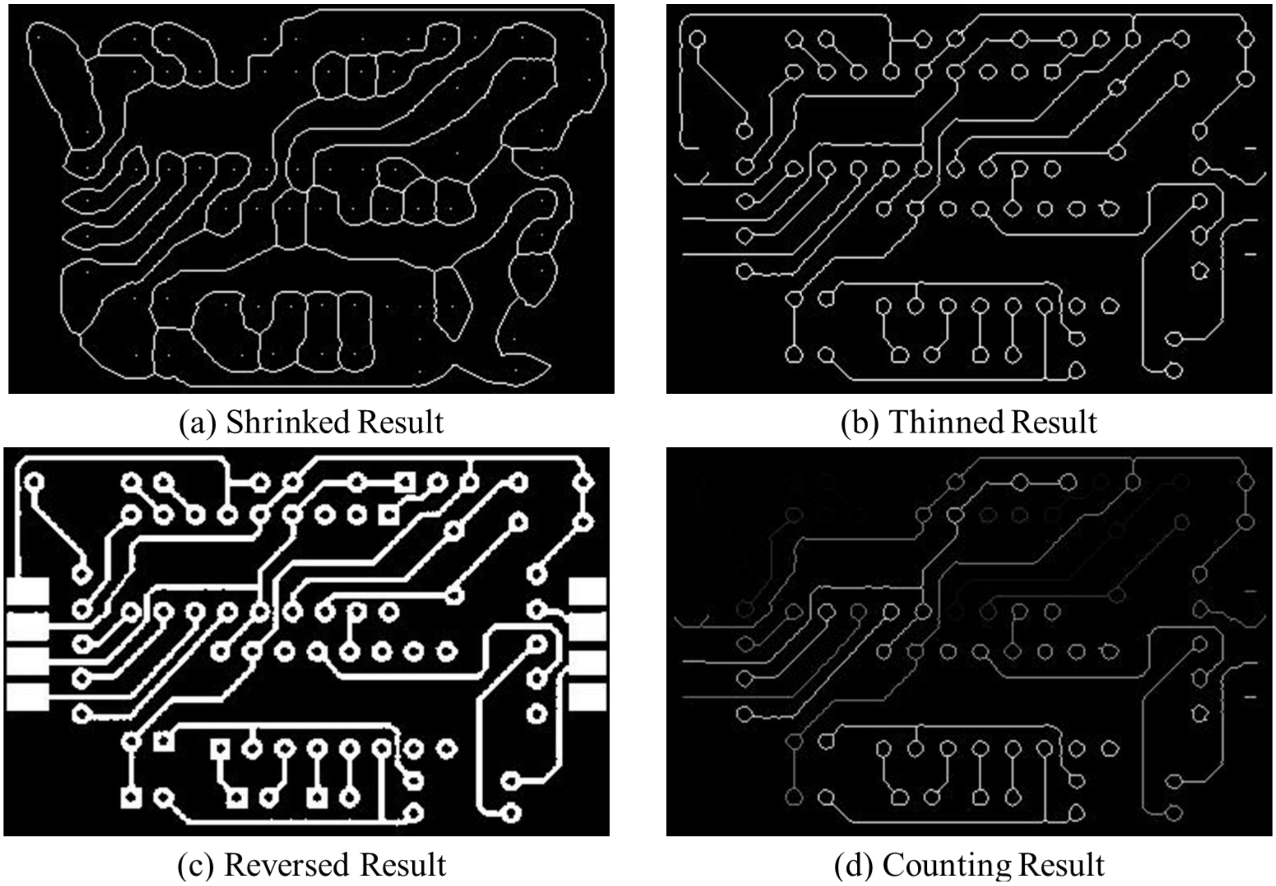
Pre-processing: In this sub problem, we are required to figure out the number of the holes and the pathways. For both of them, binarization is required. A threshold should be set to convert the image from grayscale to binary image. The threshold can significantly effect the result of the counting, if the threshold is too big, some holes' white pixels might get filtered and missed. If the threshold is too low, many gray pixels might be kept as foreground and some pixels might be misjudged as holes as well. There is a robust way to tune the threshold is that apply the algorithm in different thresholds and choose the ones that generate most stable number of holes. Because the holes in the original image is already white, so there is no need to reverse image pixel values. But for the pathways detection, we need to apply the reverse function to change the white pixels to black and black pixels to white before applying the mophological filter to the image.

Also, because the image boundary is 2 pixel width of black pixels, it will connect other pathways through the edges, which will cause the error in the counting. So, a boundary cropping process is a another necessary pre-process before applying the shrinking or thinning filter.

Methods: For holes detection, we can see from Figure 34 that the holes in the PCB image is isolated from the black circles. The white circles without any black holes means that it will become one white pixel after shrinking process. Like the morphological method described above, we can apply the shrinking operator to the image and then count the individual white pixels as the hole number. For pathways detection, because the pathways are black, we need to reverse the image and do the edge cropping befor applying any operator. After the pre-processing, we can use the thinning operator to make all the distinct objects one pixel wide. Then use the connected component labling to find distinct objects. To filter away the individual holes to prevent misjudgments, I set a threshold to discard the labels that has too few pixels.

2.3.3 Experimental Result

The results of PCB.raw are shown as below in Figure 35.



(a) Shrinked Result

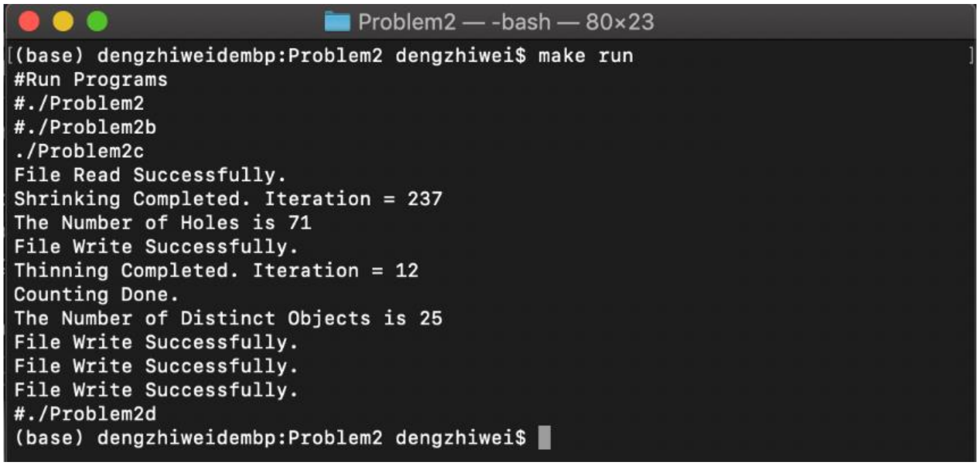
(b) Thinned Result

(c) Reversed Result

(d) Counting Result

Figure 35. Results of the PCB.raw

The counting results are shown as Figure 36.



```
[(base) dengzhiweidembp:Problem2 dengzhiwei$ make run
#Run Programs
#./Problem2
#./Problem2b
./Problem2c
File Read Successfully.
Shrinking Completed. Iteration = 237
The Number of Holes is 71
File Write Successfully.
Thinning Completed. Iteration = 12
Counting Done.
The Number of Distinct Objects is 25
File Write Successfully.
File Write Successfully.
File Write Successfully.
#./Problem2d
(base) dengzhiweidembp:Problem2 dengzhiwei$
```

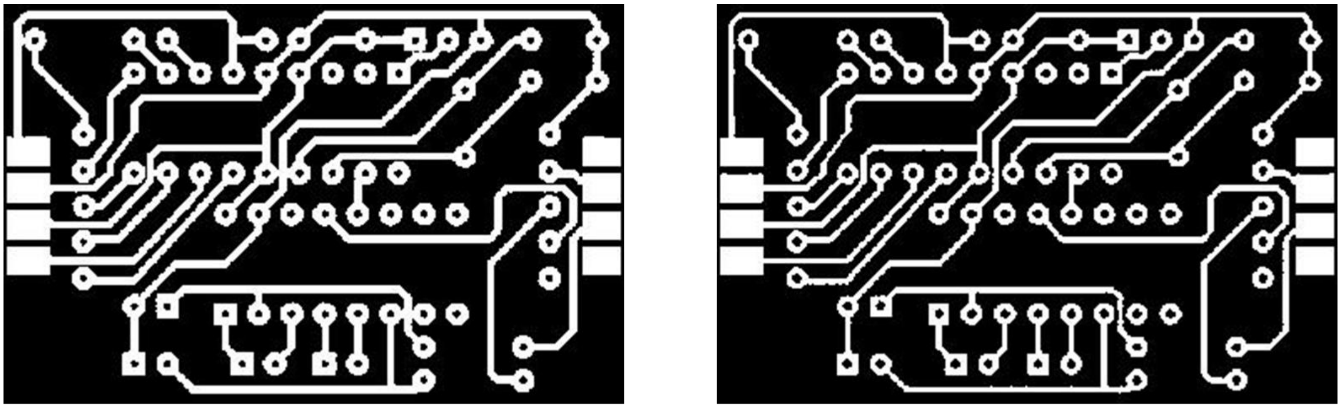
Figure 36. Count Results

2.3.4 Discussion

Number of Holes and Pathways: 71 and 25.

Assumptions: The pathways that contains more than 2 holes are counted as one pathway. The single hole and the single pad are not counted as a pathway. A single pad is not recognized as a hole, either. The black line of the board is not counted as connected pathways and is erased by the code.

Thresholding: In this sub-problem, some holes are very close to the pathways or to each other. To separate them clearly, we need to do the thresholding much more carefully. As the same idea as above, I iterate for each threshold to find the best fit threshold which can create the most clear objects like Figure 35 (c). Also, we can see from Figure 37 that the threshold can affect the result significantly by change the margin of pathways and holes. The holes get connected with the pacths when the threshold is too large.



(a) Threshold = 128

(b) Threshold = 50

Figure 37. Differences of Different Thresholds

Explanation of Methods: As discussed above, for counting the holes in the board, I applied the shrinking method to shrink each hole in to a individual dot as I did in the last sub-problem. Also, there are more than just individual white pixels in the shrinked image. So, we need to do the filter for the holes counting. For each white pixel, I checked the 3 by 3 windows of it and made sure that the neighbours are all 0 for a pixel if it is counted as a hole. For pathways counting, I assumed that the unconnected objects like individual holes and pads are not pathways. After the thinning job is done, the connected pathways became to lines and circles, which clearly has more pixels than the individual holes (a single circle) and pads (a dot). So, I implemented a filter into the Connected Component Labeling algorithm, to filter away the objects that have too few pixels to exclude the holes and pads.

Verification: In this problem, while there are not only white individual pxiels or pathways in the morphological transformed results. So, the connected component is avoidable for the counting job. While the total number of the distinct objects are 35, which is verified by the opensource connected component labeling algorithm. Also, the number of the holes and pathways are verified by myself manually.

2.4 Defect Detection

2.4.1 Abstract and Motivation

Another object analysis that uses morphological tranforms is defect detection. This technique is widely used to find the flaws in the manufactured components and mechanical parts in the industry.

In this part, I implement the algorithm which uses morphological transforms to detect the properties of the gear and the defects of the gear object. The propertis are the center and the radius of the gear. Also, the missing toothes are located in the image. The performance will be discussed and the steps of the solution will be explained and shown in details. The test image is shown below in Figure 38.

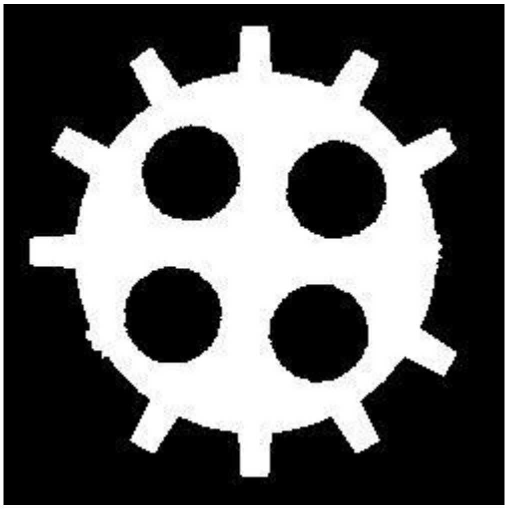


Figure 38. GearTooth.raw

2.4.2 Approach and Procedures

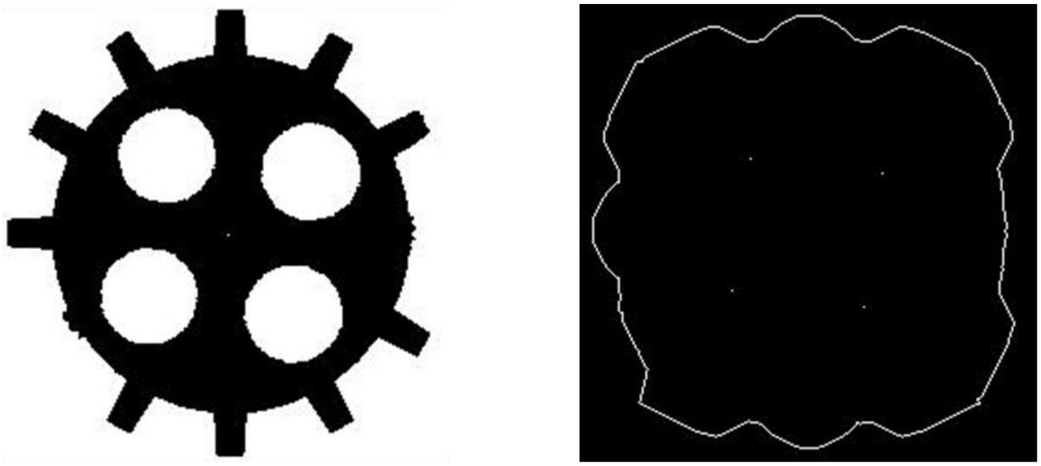
The procedure of dectecting the missing toothes of the gear is as below. There are 4 steps according the hints. It includes the gear object analysis and defect detection, states as below.

1. Find the center of the gear object. Because there are four black holes in the gear, which will be shrinked to 4 white dots if we reverse the image first as stated above in 2.3. Then averaging the 4 points' locations, we can get the center of the whole gear.
2. After get the center of the gear, we can calculate the distance between the top white pixel and the center location to figure out the outside radius of this gear.
3. Find the positions of the geartooths of the gear, using the center and the outside radius to locate each tooths of the gear and display them. By observing the image, we can say that the gear tooths only appears in certain degrees of the center.
4. Check the certain angles of the image to see whether are there missing tooths or not. If there are missing teeths, change the value to the foreground value and display it.

Automation: The method of finding missing tooth is based on the linear vector transformation. We can see the gear tooth is located in the 0~360 degrees to the center of the gear divided into 12 pieces. Of course, we can just check this 12 directions in the image to see whether there is a teeth or not. But this ad hoc procedure is nor suitble for other cases if the image has been rotated, this method will not work in these cases. So, for the generalization of the code and the automation of the algorithm, I applied a vector rotation to detect the missing tooth. This method will be detailed explain in the discussion part.

2.4.3 Experimental Result

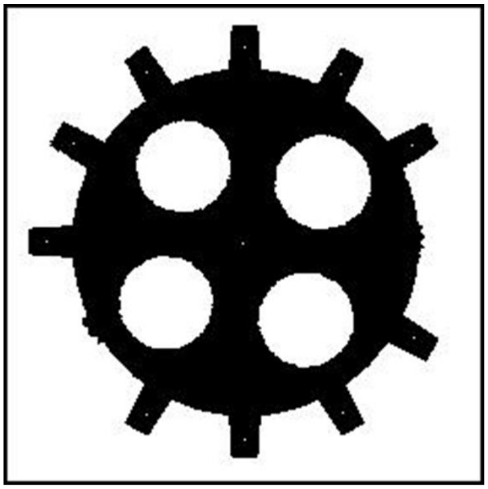
The results of the gear cneter is shown as Figure 39.



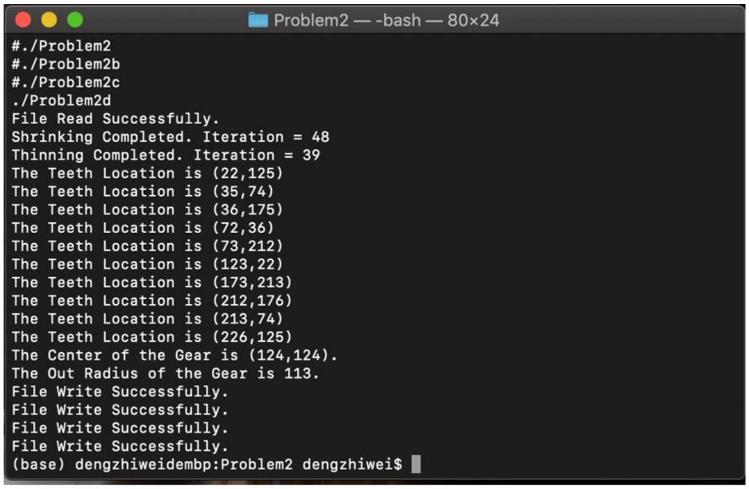
(a) Center (124,124) (b) Reverse Shrinked Image

Figure 39. Center of the Gear (white pixel) and the Reversed Shrink Result

The results of the gear tooth locations are shown as Figure 40.



(a) Detection of Tooth



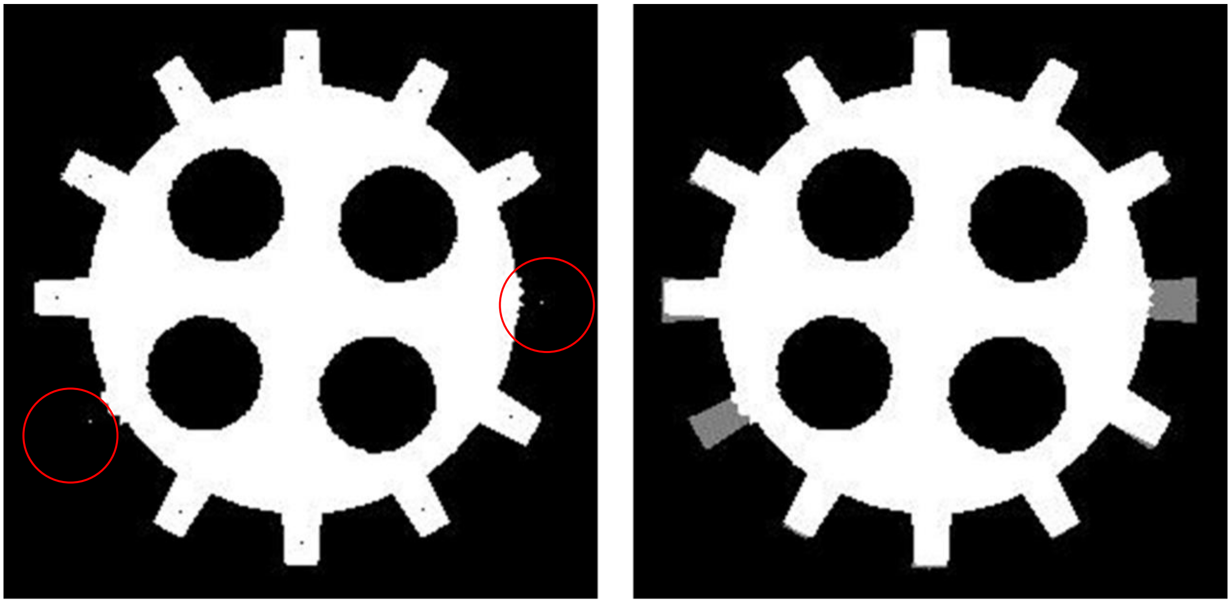
```
#./Problem2
#./Problem2b
#./Problem2c
./Problem2d
File Read Successfully.
Shrinking Completed. Iteration = 48
Thinning Completed. Iteration = 39
The Teeth Location is (22,125)
The Teeth Location is (35,74)
The Teeth Location is (36,175)
The Teeth Location is (72,36)
The Teeth Location is (73,212)
The Teeth Location is (123,22)
The Teeth Location is (173,213)
The Teeth Location is (212,176)
The Teeth Location is (213,74)
The Teeth Location is (226,125)
The Center of the Gear is (124,124).
The Out Radius of the Gear is 113.
File Write Successfully.
File Write Successfully.
File Write Successfully.
File Write Successfully.
(base) dengzhiweidembp:Problem2 dengzhiwei$ 
```

(b) Tooth Locations

Figure 40. Gear Tooth Locations (White Pixels in the image)

The results of the missing gear tooth locations are shown as Figure 41. And the defectless image was obtained by the use of the radius and the center.

Note: The background of Figure 39 and 40 are set to be white, because in 39 and 40, the backgrounds are the gear itself. So, to show the points and positions more clearly with white dots. The gears are set to be black and the background are set to be white.

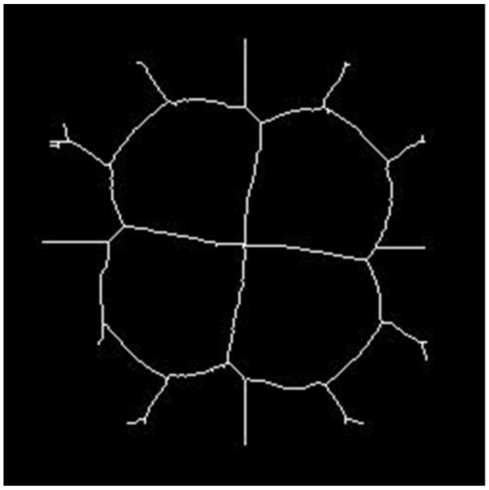


(a) The Missing Tooth Position

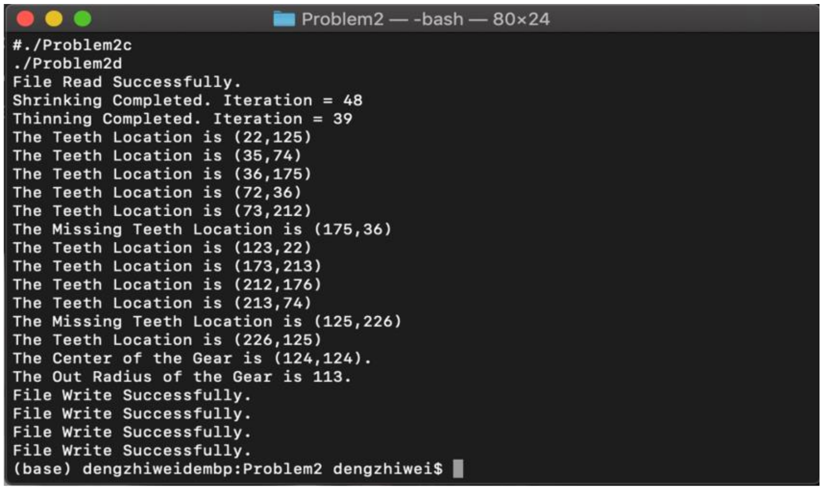
(b) The Defect less Gear

Figure 41. Missing Gear Tooth Locations (White Pixels in the left image)

The positions of the missing teeth and the geartooth image is shown as below in Figure 42.



(a)Thinned Geartooth.raw



```
#./Problem2c
./Problem2d
File Read Successfully.
Shrinking Completed. Iteration = 48
Thinning Completed. Iteration = 39
The Teeth Location is (22,125)
The Teeth Location is (35,74)
The Teeth Location is (36,175)
The Teeth Location is (72,36)
The Teeth Location is (73,212)
The Missing Teeth Location is (175,36)
The Teeth Location is (123,22)
The Teeth Location is (173,213)
The Teeth Location is (212,176)
The Teeth Location is (213,74)
The Missing Teeth Location is (125,226)
The Teeth Location is (226,125)
The Center of the Gear is (124,124).
The Out Radius of the Gear is 113.
File Write Successfully.
File Write Successfully.
File Write Successfully.
File Write Successfully.
(base) dengzhiweidembp:Problem2 dengzhiwei$
```

(b) The Positions of Missing Tooth

Figure 42. Thinned Image and Missing Gear Tooth Positions

2.4.4 Discussion

Thresholding: Thresh holding is also an important pre-process. In this sub-problem, I used the same scheme to select the proper threshold to binarize the grayscale image.

Explanation of Methods: 1. For the center and radius calculation, I used the shrinking method to the reversed images. So that the four circles of the geartooth is going to be shrinked to one pixel dot. Then I used these four individual white pxiel locations and average them to figure out the center of the gear is (124, 124). The average method is just plain averaging. Then I used the first white pixel of the image to calculate the radius of the gear, the radius equals the distance between it and the center point.

2. For finding the existing gear tooth, I applied the thinning method on the original image. After thinning, the existing geartooth will generate longer lines than the defected ones as shown in Figure 42. So ,we can set another threshold to detect the long lines and discard the short ones by a fixed radius circle, i.e claculating the white pixels' distance with the center. Because the thinning operator gives the center line of each tooth, the position that marked in Figure 40 is almost the center of each tooth.

3. By detecting the existing tooth, we also knows that the angles between neighbour tooth is 30 degrees. So, I used a linear vector rotation to estimate each tooth's neighbour's position. The transform is shown as Figure 43. And the calculation formula is shown as equation (13-14). Based on the center points and existing tooth locations, I estimated the positions of enxt

tooth of each existing tooth in a anti-clock wise way. Then check the corresponding position is white or not to decide the tooth is missing or not.

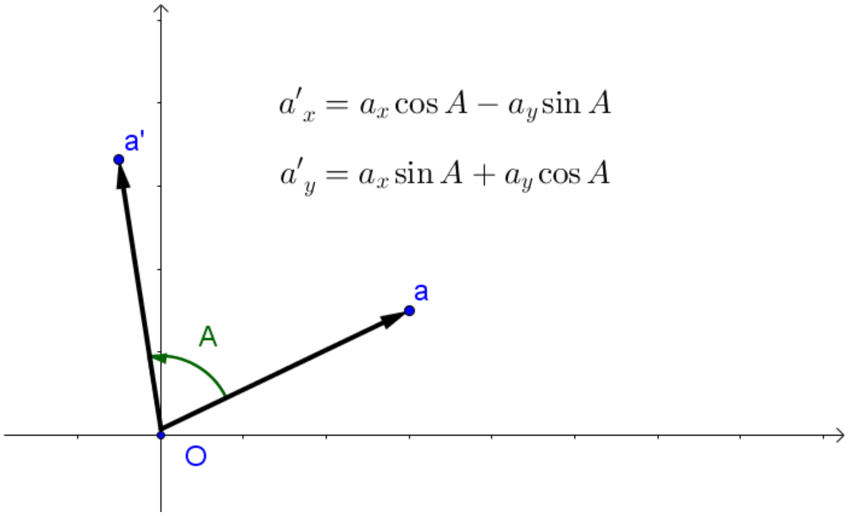

$a'_x = a_x \cos A - a_y \sin A$
$a'_y = a_x \sin A + a_y \cos A$
a'
a
A
O

Figure 43. Diagram of Vector Rotation.

$$x = (x1 - x2)cos\theta - (y1 - y2)sin\theta + x2 \qquad (13)$$

$$y = (y1 - y2)cos\theta - (x1 - x2)sin\theta + y2 \qquad (14)$$

This method is based on the existing tooth locations and the center of the gear. It is more robust than any ad hoc algorithms for this problem since this one can detect the missing tooth regardless of the orientation of the gear. This improves the automation of the code.

Defectless Image: I designed another algorithm to fill the image with the missing tooths. This is based on the hardcoded degrees check and the radius of the gear. I check the disctances between each pixel and the center, then check its direction relative to the center. If the distance is correct and the direction is in some ranges, the pixel will be set to gray. Although there are some artifacts in the image, the visual quality is surprising to view. I will modify this algorithm to make it more robust in the future.

References

[1] MATLAB: https://www.mathworks.com/help/vision/examples/feature-based-panoramic-image-stitching.html

[2] Connected Component Wikipedia: https://en.wikipedia.org/wiki/Connected-component_labeling.