EE569 Digital Image Processing

HOMEWORK#3 Report

Issued: 2/24/2021 **Due: 11:59PM, 3/12/2021**
Name: Siyu Li **ID:2455870216**
E-mail: lisiyu@usc.edu

Problem 1: Geometric Image Modification (25%)

(1): Image Warping (Project from Square to Circle)

I. Approach and Procedures

Image warping is a coordinate based geometric image modification method which transforms the original image in a certain way and gives the it a new look. In this problem, the original squared image is projected into a circle by **Elliptical Grid Mapping [1]** function. There are two ways of calculate the point in the objective image, one is forward mapping method and another is backward mapping method.

For the forward mapping method, we use transfer function to calculate the coordinate into which the pixel on the original image is projected on the objective image. However, the coordinate calculated above cannot necessarily be integer and in most cases it is fraction number which makes it impossible to create warped discrete output image directly. Therefore, we allocate the value of projected pixel on the objective image to its nearest 4 neighbors to make output of discrete image possible. The disadvantage of forward mapping is that even with the allocating value to neighbor pixel on objective image process, it is still very likely that there is missing point on objective image because in some area there will not be any pixel which makes allocating value impossible.

For backward mapping, to get the transformed pixel value in objective image, we first create a picture of the same size as the original image with zero value and project the pixels on it back into the original image and get the coordinate location. Before the projection calculation process in my coding process, imgToCoordinate.m function is applied to transform matrix index to coordinate index in order to implement the projection function correctly. In imgToCoordinate.m, we use the matrix middle row as X axis and the middle column as Y axis. The project function which project pixels in objective image which is **u** and **v** into original image where the projected pixel position is **x** and **y** is shown below **[1]**.

$$\boldsymbol{x = 0.5 * \sqrt{2 + u^2 - v^2 + 2\sqrt{2} * u} - 0.5 * \sqrt{2 + u^2 - v^2 - 2\sqrt{2} * u}}$$

$$\boldsymbol{y = 0.5 * \sqrt{2 - u^2 + v^2 + 2\sqrt{2} * v} - 0.5 * \sqrt{2 - u^2 + v^2 - 2\sqrt{2} * v}}$$

In realization coding process, EGMwrapPointReverseMapping.m is applied to calculate projected pixel position in original image. As is the similar case above, the projected pixel coordinate **x** and **y**

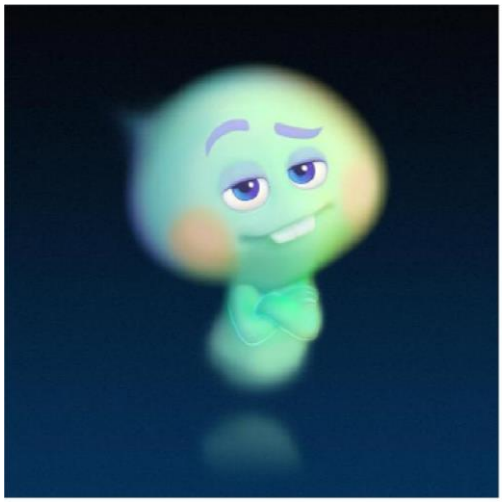
in original image is very likely to be fractional number and do not lie exactly on original image pixels. Therefore, we apply bilinear interpolation to calculate the value of projected pixel and the corresponding m function in coding is projectToCircle.m. In this m function, we only do interpolation to points which is projected from points within circle in objective image and lies within the original image. And for the value of pixel which corresponds to the boundary point of the circle in objective image, we just assign the value of nearest point in the original image to it. And for non-circle point case, we just allocate 0 value to it. The interpolation function is shown below.

$$F(x', y') = F(x + \Delta x, y + \Delta y) = (1 - \Delta x)(1 - \Delta y)F(x, y) + \Delta x(1 - \Delta y)F(x, y + 1) + (1 - \Delta x)\Delta y F(x + 1, y) + \Delta x \Delta y F(x + 1, y + 1)$$

After we calculated the interpolation and get the value of the projected pixel in original image, we get the final objective image which do not have missing point because every point in objective image has its corresponding pixels in original image.

II. Experimental Results

The original and projected images of 22.raw, Dog.raw and Forky.raw are shown in Fig 1.1 below.



(a) 22.raw Original Image

(b) 22.raw Projected into Circle

(c) Dog.raw Original Image

(d) Dog.raw Projected into Circle

(e) Forky.raw Original Image

(d) Forky.raw Projected into Circle

Fig 1.1 Original and Wrapped Image (Project from Square to Circle) of 22.raw, Dog.raw and Forky.raw

As is observed from Fig 1.1, by using backward mapping method, we successfully avoid the missing point phenomenon which is very likely to occur in forward mapping method and the result of square to circle projection turns out very well.

(2)(3): Image Unwarping and Comparison of Original Image with Unwrapped one (Project from Circle back to Square)

I. Approach and Procedures

For the image unwarping which means projecting from circle back to square, the process is basically the same as that of image warping including interpolation function which is also bilinear interpolation here. The major difference is the projection function

in EGMPointReverseMapping_Unwrap.m and the strategy that decides which point projected back from objective image in reverse mapping should be interpolated, use the nearest pixel value or just set 0 in projectToSquare.m. The projection function in image unwarping [1] is shown below.

$$\boldsymbol{u = x * \sqrt{1 - 0.5 * y^2}}$$

$$\boldsymbol{v = y * \sqrt{1 - 0.5 * x^2}}$$

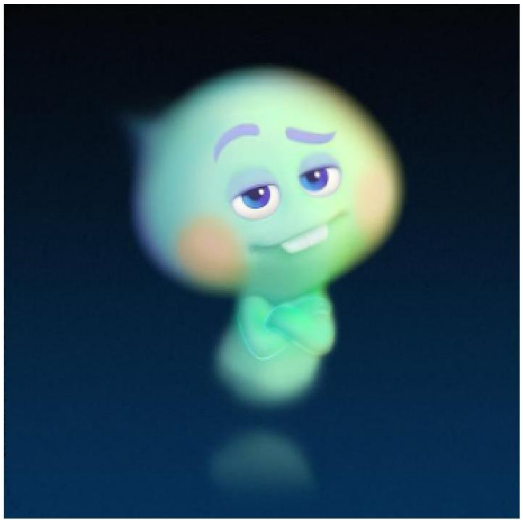
In the function, x and y are pixels coordinates in the objective image while u and v are coordinates in original image.

And as for the strategy mention above for calculate value of pixels projected from objective image back to original image, we only do interpolation to points which is projected from points within square in objective image and lies within the original image. And for the value of pixel which is originally a boundary point of the square in objective image, we just assign the value of nearest point in the original image to it. And for non-square point case, rather than just allocating 0 value to it, we allocate the value of nearest neighbor point to it instead.

Because in the process of image wrapping and unwrapping, it is certain that there will be some information loss and in the unwrapped image. If we assign 0 to the value of original image pixels which correspond to square corner in the objective image, the corner of final objective image will lack information and look black. On the contrary my method mentioned above can fixed the information lacking on the corners of objective image well.

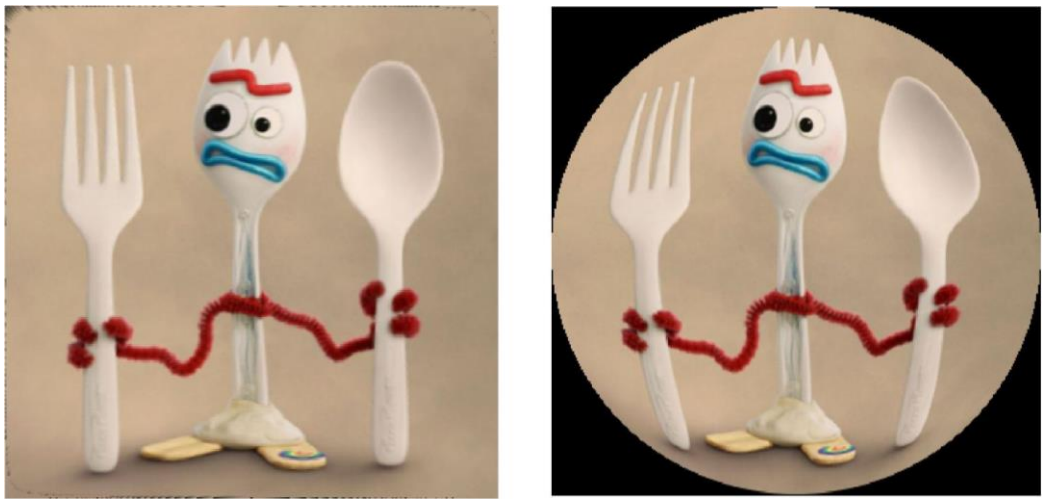
II. Experimental Results

The wrapped images and unwrapped images of 22.raw, Dog.raw and Forky.raw are shown in Fig 1.2 below.



(a) Unwrapped Image of 22.raw **(b) Wrapped Image of 22.raw**

(c) Unwrapped Image of Dog.raw **(d) Wrapped Image of Dog.raw**

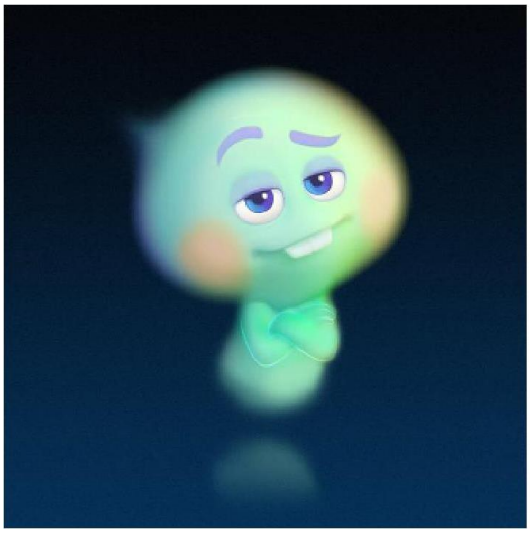


(e) Unwrapped Image of Forky.raw **(d) Wrapped Image of Forky.raw**

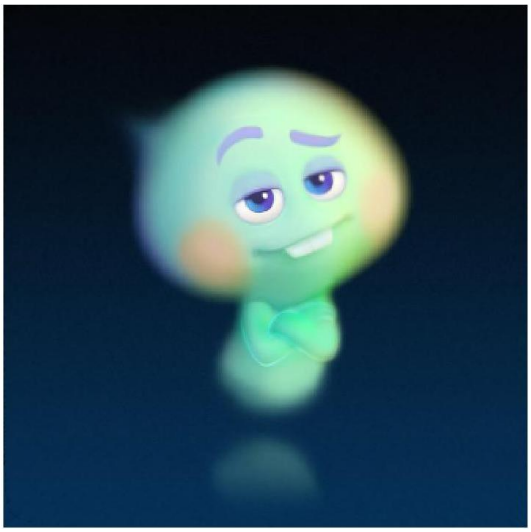
Fig 1.2 Unwrapped and Wrapped Image (Project from Circle back to Square) of 22.raw, Dog.raw and Forky.raw

As is observed from Fig 1.2, the result of unwrapped image is generally very good. The objective image corner information missing issue stated above is solved. The only defect is that there are still very little amounts of black dots on the upper left corner of unwrapped image. This issue occurs because when you calculate values for the objective image the upper left corner is traversed first so there is not any non-zeros neighbor value to be used to fix the corner information missing issue here. The solution may be doing the unwrapping process by splitting the original image to four equal sized images and then concatenating them back together.

And for **comparing the unwrapped square image with the original square image as is required in (3)**, the result images are shown in Fig 1.3 below.



(a) Original Image of 22.raw



(b) Unwrapped Image of 22.raw

(c) Original Image of Dog.raw

(d) Unwrapped Image of Dog.raw

(e) Original Image of Forky.raw

(f) Unwrapped Image of Forky.raw

Fig 1.3 Original and Unwrapped Image of 22.raw, Dog.raw and Forky.raw

First of all, from Fig 1.3, the unwrapped image is noticeably blurrier than the original image because in the back and force wrapping and unwrapping process it is certain that there is some information loss in the bilinear interpolation process. The bilinear interpolation assumes that the gray level in an image changes linearly while in fact it may not, which result in the information loss.

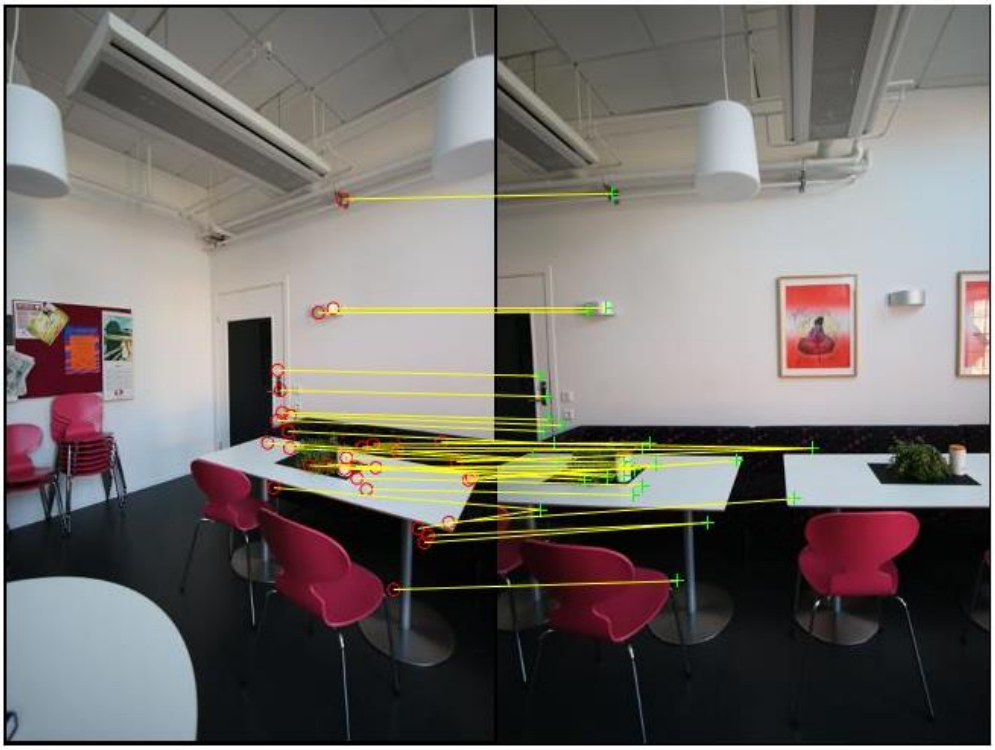
Another phenomenon worth noticing is that there is mosaic like pattern at the corner of unwrapped image especially in Fig 1.3 (d), this is because we simply use taking the nearest neighbor pixel value to fix the information loss at the objective image corner where should only be allocated 0 value as is stated before. Therefore, the pattern here is obviously different from the region calculated by interpolation for it can be regarded just as an extension of existing edge pixels.

Problem 2: Homographic Transformation and Image Stitching (25%)

I. Approach and Procedures and Experimental Results

Homographic transformation is a transformation method which can map one image to the other with corresponding points still aligned when the original and the transformed image are overlaid together. And image stitching technic can assemble imaged acquired from different angels and positions with the same object together owning to the property of homographic transformation. In this problem we will stitch 3 images share the same object but taken from different angels together.

To do the homographic transformation, the transformation matrix needs to be calculated by finding 4 matched feature point and solving equation based on the 4 points. To find the proper feature points, both automated computer calculation and manual optimal feature points selection are applied. First, we apply **detectSURFFeatures.m** in Matlab to find the most original features which are processed later by **extractFeatures.m** to be reduced to fewer but more useful features. After feature detection and reduction procedures, **matchFeatures.m** is applied to output the index of paired feature points. Then, we use the index above to find the coordinates of all paired feature points in both images needed to be stitched together. To find the useful paired feature points, we first use all of these points to plot a matched feature map by **showMatchedFeatures.m**. The results of left and middle image and right and middle image are in Fig 2.1 below.



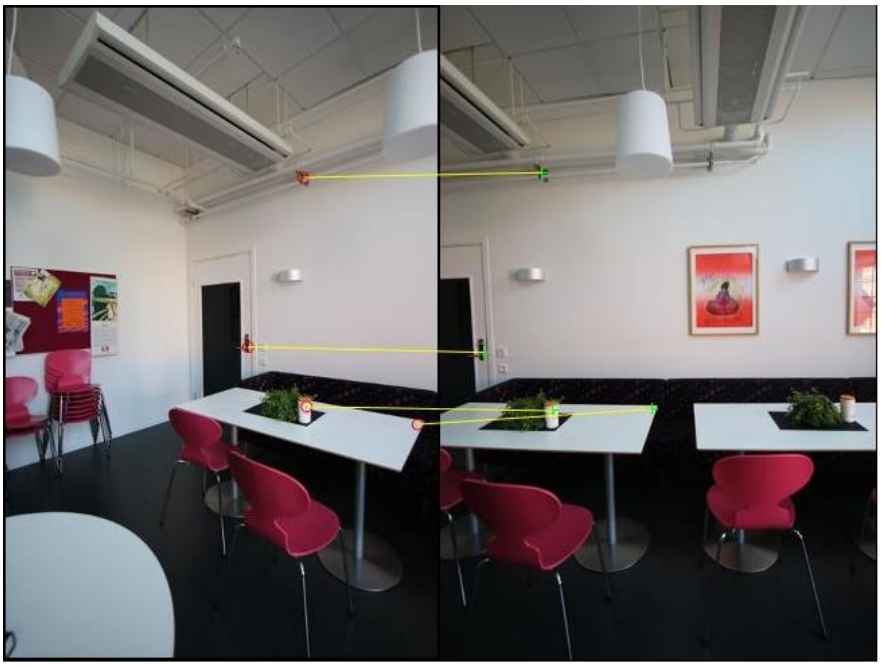
(a) All Matched Feature Points of Left and Middle Image

(b) All Matched Feature Points of Right and Middle Image

Fig 2.1 Matched Feature Point Pairs in the Image

As is observed from Fig 2.1, feature points are mainly located in the visual salient area including but not constraint to corners, objects and edges. However, some feature point

pairs are wrongly matched. Then we examine the point pairs one by one to select the best 4 matched feature point pairs and using their index to get the actual coordinate values. The **selected eight point pairs** of left and middle image and right of middle image are shown in Fig 2.2 below with different input from Fig 2.1 by the same function **showMatchedFeatures.m**.



(a) 4 Selected Matched Feature Points of Left and Middle Image

(b) 4 Selected Matched Feature Points of Right and Middle Image

Fig 2.2 Manually Selected Matched Feature Point Pairs

We can observe that the point pairs we selected lie exactly on the same object in two

different images and all of them are very salient objects such as light, chairs and plants on the table. With the correct matched point pairs selected, we have calculated the transformation matrix **H** by solving linear equation system with function **cal_transform_Hmatrix.m**.

After getting the transform matrix **H**, we have the transfer function of image homographic processing which belongs to image wrapping. The process of getting wrapped image is nearly the same as the case in problem1 because both of them apply backward mapping method. The only difference between the case in this problem and problem 1 is that warping in problem one applies normalization before processing so the size of original image and the objective image is exactly the same.

Meanwhile in this case, we have to make the matched points in the warped image able to overlay on the corresponding points in the middle image so we need the objective image have its supposed to be original size. Normalization will not be applied transferred point location calculation here. We first do the forward transform to know the projected pixel coordinate values in objective image.

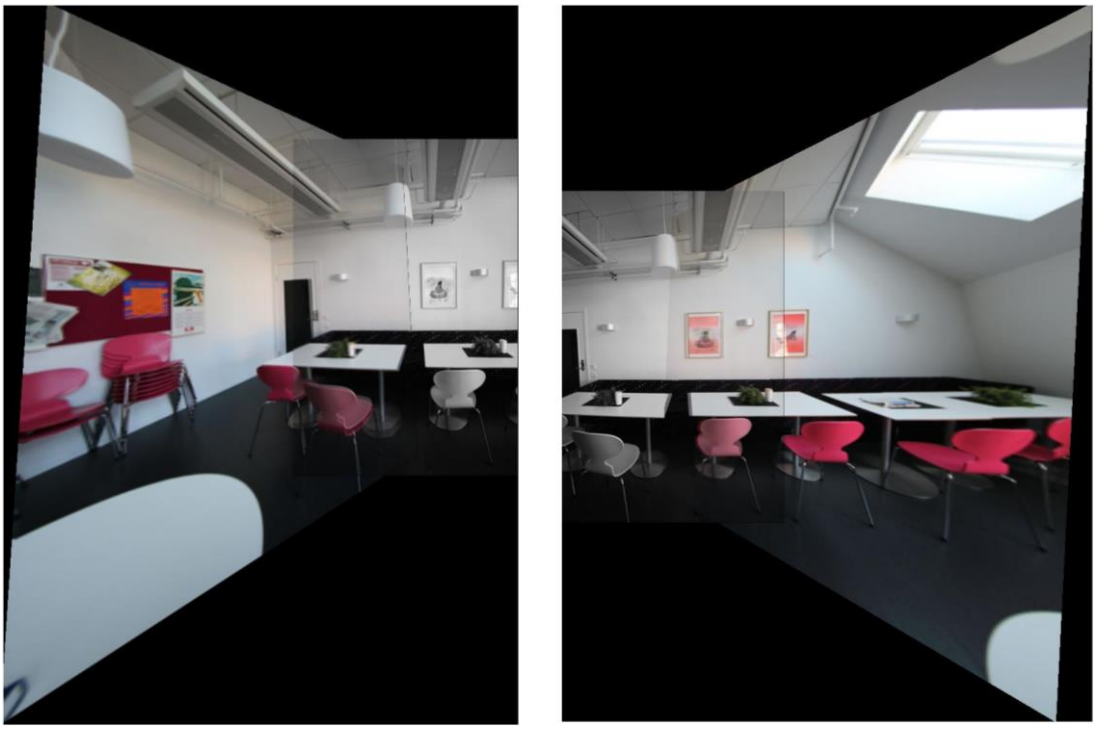
In order to have the proper size of wrapped image, we need to find the points with minimum x and y coordinate values in objective image to remove the irrelevant non-object background and also find the maximum ones to decide the boundary on the other side. The objective image size is calculated by **calOutputSideSize.m**. Then, we calculate corresponding point coordinate values in original image projected inversely from objective image pixels. **Before doing inverse transform, the offset in objective image pixel coordinate resulting from remove relevant pixels should be added back to objective image pixel coordinates in order to get the correct projected coordinate values in original image.**

With these coordinate values, we can get the pixel values in wrapped image by interpolation from the original image with region-based interpolation strategy which is similar to the one mention in problem 1. **The wrapped or homographic transformed images of right and left are shown in Fig 2.3 below.**

(a) **(b)**

Fig 2.3 Result of Homographic Transform of Image Right (a) and Image Left (b)

After homographic transformation, what we need to do is the image stitching step. The key of stitching is using the key points coordinates and wrapped image size to build a proper size of output image and to specify different regions where different pixel-level operations will be applied. Here we selected the most upper left matched feature points on both wrapped left image and wrapped right image to do the coordinate alignment in stitching wrapped left and middle and wrapped right and middle together. The result is in Fig 2.4 as follows.



(a) **(b)**

Fig 2.4 Wrapped and Stitched Left and Middle Image (a) and Wrapped and Stitched Right and Middle Image (b)

As is observed from Fig 2.4, **the pattern in the overlapped arear is different from other regions because in the overlapped area I simply add the corresponding pixel values together and apply normalization.** In the wrapped image, bilinear interpolation which assume the change of pixel value is linear while in fact it may not is applied to calculate values. Therefore, there must be some information loss and distortion in the wrapped information and when it is overlaid with the middle image with the same object the pattern difference makes the overlapped region look slightly abnormal. **Besides, coordinate misalignment also brings some tiny unexpected black lines and dots on the boundary of middle image position in the stitched image. For the final stitched image,** the procedure is nearly the same except that we have to use more coordinate alignment points**. The result is in Fig 2.5 below.**

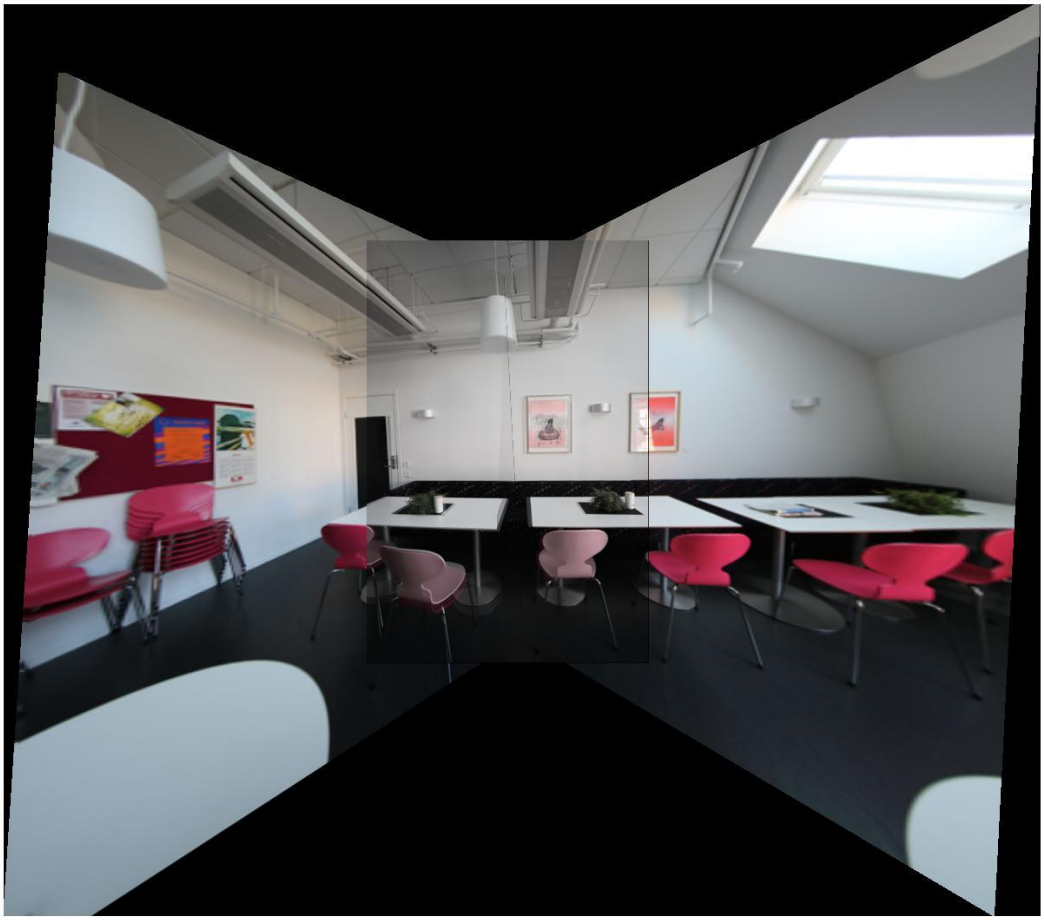


Fig 2.5 Final Stitched Result of Left Middle and Right Images

In the final result in Fig 2.5, the problems mentioned above in Fig 2.4 still exhist but overall, the performance meets the requirement and the common objects in the overlapped parts are overlaid exact together. To sum up, the homographic transform matrix is the core of transform because it controls how the images are transformed and how matched feature points are overlaid together. So, selecting proper matched feature points are extremely important for it decides how the homographic transform matrix is generated. Moreover, the process of deciding the output image size and aligning different pictures to make them fit well in one image is also very important. Because if there is any error in this step there will be lots of undesired dots, lines and even gap between images. Any of the defects above can impair the quality of output image greatly.

Problem 3: Morphological processing (50%)

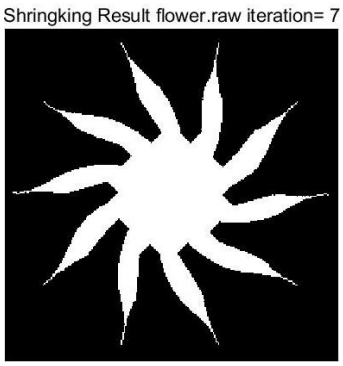
(a) Basic morphological process implementation (15%)

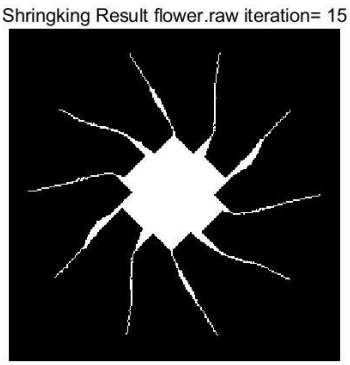
I. Approach and Procedures

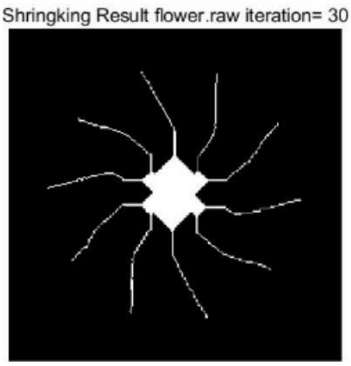
Shrinking, thinning and skeletonizing are the most common morphological processing method. In this problem, I realize the three operation by two stage filter. First, bond of each pixel with its neighbor is calculated because in the later two stage filter design different mask and operation will be applied when the bond value of a pixel is different. Here **calculateBond.m** is applied to calculate the bond. In this function, if a pixel is not background, for each 1 in strong 4 neighbor region the bond increases by 2 and for each 1 in the other 4 neighbor region in 8 neighbor of this pixel the bond increases by 1. And in the stage 1 conditional filter whose corresponding function is **stage1Filter_S.m**, **stage1Filter_T**.m and **stage1Filter_K**.m, the 8 neighbor pixel values are compared with the patterns used based on the condition of bond value of the center pixel. If a mask is absolutely the same as corresponding 8 neighbor value, we can that a hit and the logic value of this comparison is 1. Then we take the 'OR' (take the union) operation among logic values of all comparisons based on different masks and then the filtered value of central pixel is decided. The stage2 unconditional filter is very similar except some nuance in logical operations and masks used. Shrinking and thinning share the same function **stage2Filter_ST.m** and skeletonizing uses **stage2Filter_K.m**. Finally based on the output filtered matrix in unconditional pattern step, a certain pixel will be deleted if its corresponding value is 1and will still be kept if that is 0. And a loop is designed to do the iterations of morphological operation, we set the parameter **countmax** which means the max number of iterations 1000 in shrinking and 200 for the rest to make all the iterations continue until they completely converge.

II. Experimental Results

The result of shrinking, thinning and skeletonizing and their intermediate steps of three images flower.raw, jar.raw and spring.raw is shown below in Fig 3.a.1, Fig 3.a.2 and Fig 3.a.3.







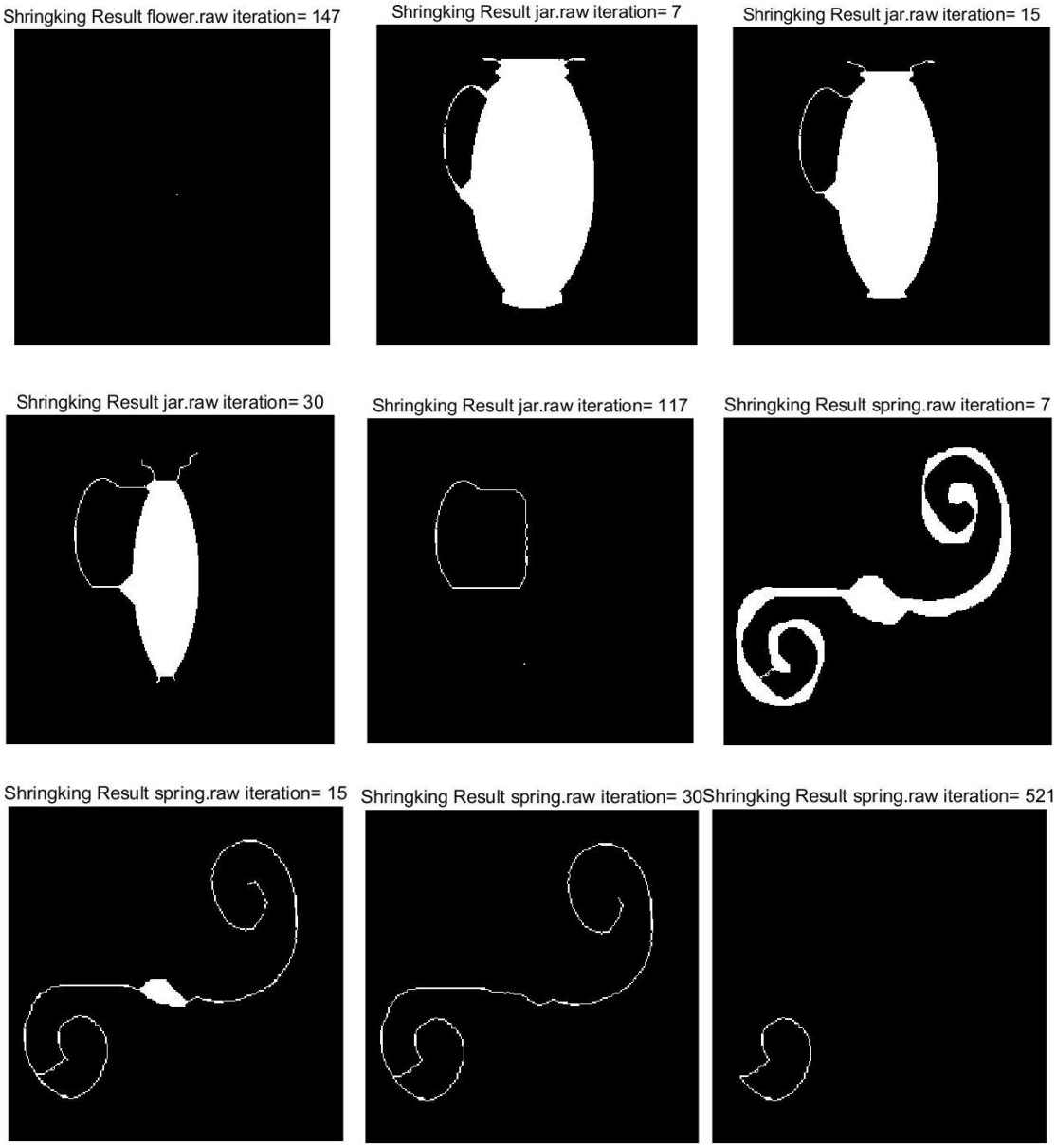
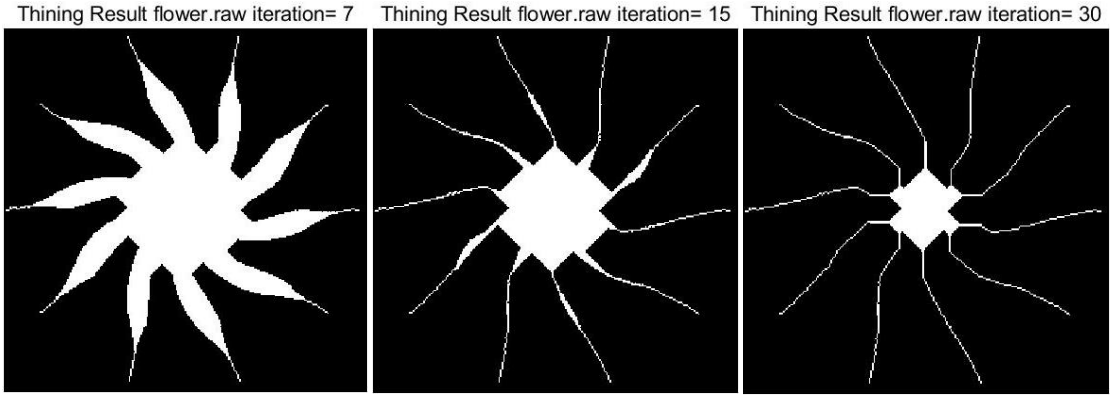


Fig 3.a.1 Shrinking Result and Intermediate Step of Image flower.raw, jar.raw and spring.raw (The image with the largest number of iterations is the converged one)



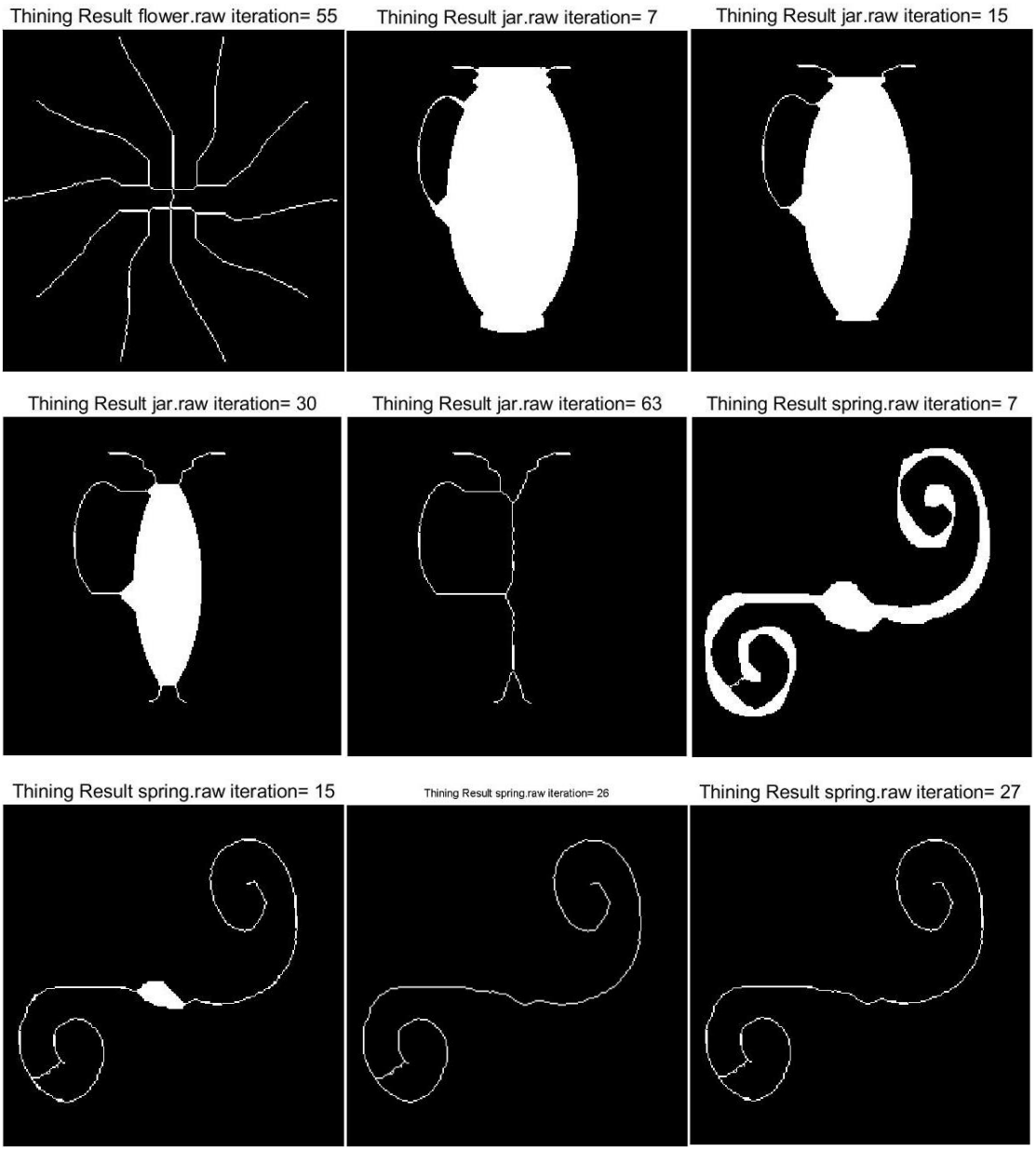
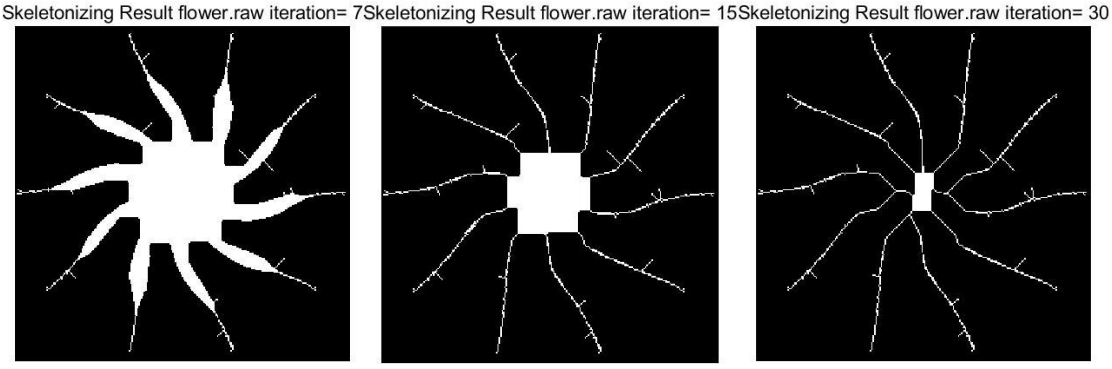


Fig 3.a.2 Thinning Result and Intermediate Step of Image flower.raw, jar.raw and spring.raw (The image with the largest number of iterations is the converged one)



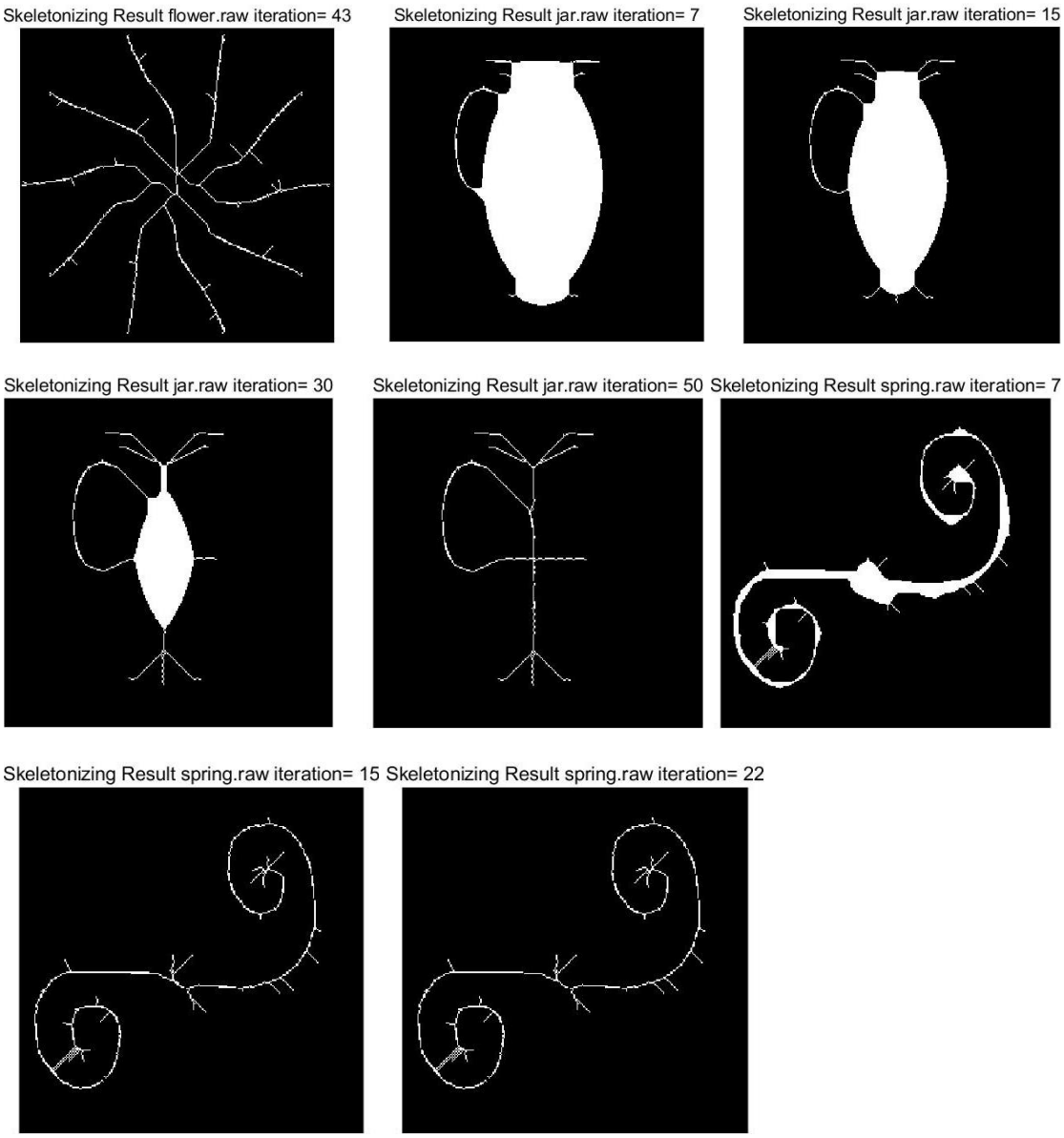


Fig 3.a.3 Skeletonizing Result and Intermediate Step of Image flower.raw, jar.raw and spring.raw (The image with the largest number of iterations is the converged one)

Based on the observation of Fig 3.a.1, Fig 3.a.2 and Fig 3.a.3, we can conclude that the difference between shrinking and thinning results from the conditional patterns. Shrinking process mark the pixels of bond value 1, 2 and 3 so it makes a line reduced to a single point regardless the way pixels in a line are assembled together. We noticed that even for very complicated-shaped spring.raw, if a huge number of iterations of 521 are made, it will converge to a circle finally. And we also noticed that image with single object without a loop will finally converged to a single point as the converged shrinking result in Fig 3.a.1 flower.raw.

At the meantime, thinning process does not mark pixels of bond value 1, 2, and 3. Therefore, it makes points at the end of some lines remain. Different from shrinking,

thinning will not only show the connection region on the converged picture, it will still tell the information about the length between the longer and shorter contours. From the result of flower.raw of shrinking and thinning, we can observe from flower.raw result in Fig 3.a.1 and Fig 3.a.2 that the result is totally different. Shrinking result is a point which can only tell how many unconnected objects there are while thinning result can tell the length of each petal of the flower.

For skeletonizing, it has the least conditional patterns so it exhibits the tendency of keeping most of the information on sharp angles in a contiguous region. This operation will also make the region with loop converge to a circle and eliminate some boundary pixels of the region, but it will leave the most information compared to the other two above because details with sharp angels are maintained here. We can attest to these properties by comparing any of the objects among three figures above.

(b) Solution to the maze (15%)

I. Approach and Procedures

From the previous discussion and analyzation, we concluded that shrinking exhibit the property of shrink every single object to a single point or a circle. For the maze we have to deal with in this problem, we can observe form Fig 3.b.1 that the white part which is the path in the maze and the boundary can form circles. And if there is a solution path, it is certain that it can connect the boundary and form two closed loops. Therefore, the final shrinking result will be loops indicates that we can find the solution path by eliminating all the non-solution lines during shrinking process. Here shrinking with infinite or very high maximum iteration number is applied to find the path because of the high complexity of maze shape. The solution path will be a very thin white line connecting the entrance and the exit which are parts of white boundary path.

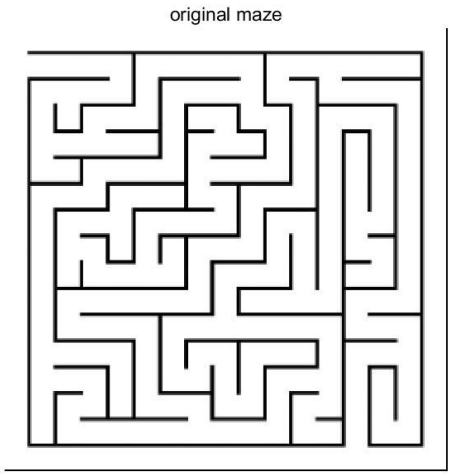


Fig 3.b.1 The Original Maze

II. Experimental Results

In the result exhibition part, the result and the solution maps are overlaid with the same weight in order to show a better visual result in Fig 3.b.2. We can see that after 858 iterations the result converges and we can find the solution path.

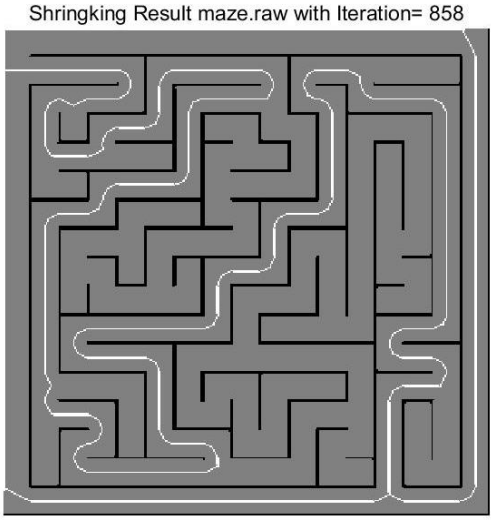


Fig 3.b.2 Maze Solution Converged with 858 Iterations

(c) Defect detection and count (20%)

(1) Count the total number of disconnected defects in the image by using morphological method.

I. Approach and Procedures

By the property of shrinking that it will reduce unconnected single object to a single point or circle, we can reduce all defects to a single point and then count the number of single points which is exactly the number of defects to solve the problem. In specific, we first do binary thresholding to eliminate the noise and get the exact boundary of defects. Here the number of defects will vary with different thresholds because low threshold will more likely to suppress the weak link between original defect and partition it into two defects in binary image. So, the lower the threshold is, the more the defect numbers will be. My threshold is 128 in the problem. And to make it possible to do shrinking operation on defects, binary image is inversed to make the value of defect point 1. Then we just do shrinking operation on the image until it converges. At last, we use countIsolatePoint.m to count the single point to reflect the total number of defects. The function countIsolatePoint.m is realized by increase the total number by one when all the values of 8 neighbor pixels of a point valued 1 are zero.

II. Experimental Results

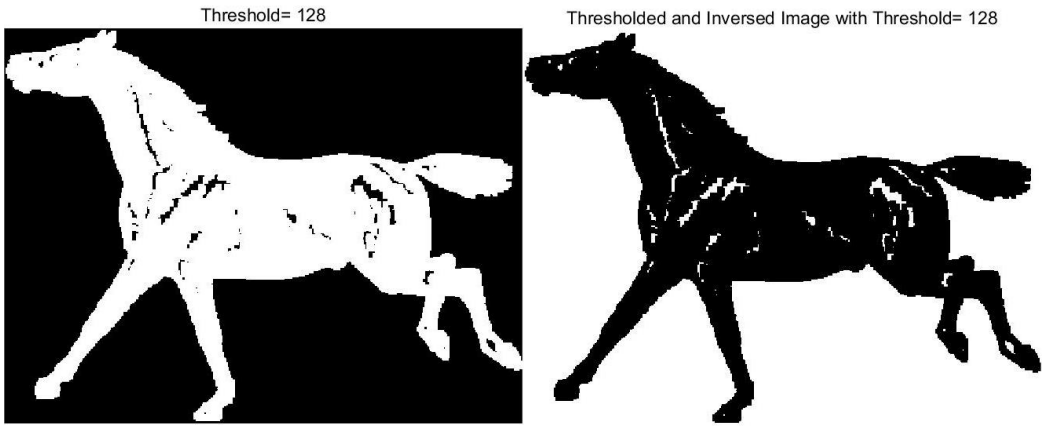


Fig 3.c.1 Threshold Binary Horse Image and Inversed Threshold Binary Horse Image (Threshold=128)

The threshold binary image and the inversed threshold binary image are in Fig 3.c.1 above. We can see the defects with value 1 are ready to be processed with shrinking.

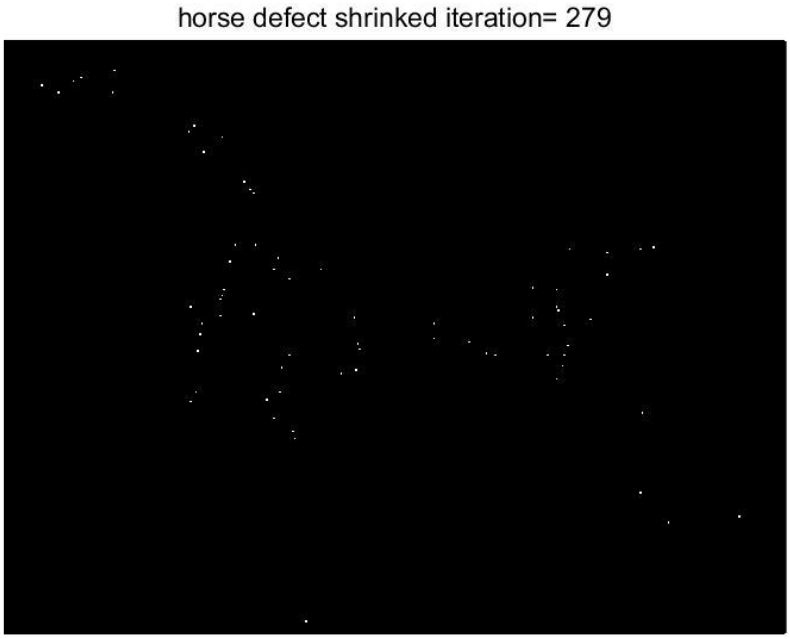


Fig 3.c.2 Converged Result of Shrinked Defect with 279 Iterations

We can see that in the shrinkage result, every single point represents a defect in the original image. By counting them using countIsolatePoint.m, we finally conclude that **there are 70 defects in total in the case that threshold equals 128.**

(2) Number of Different Defects Size and Their Frequency

I. Approach and Procedures

With the restriction that we can only apply morphological method to this problem, we cannot know how many pixels exactly a defect has. So, we assume that two defects are of the same size if they take equal number of iterations to converge to a single point. The problem of getting the numbers of pixels in all defects have been transformed into count the number of defects which converge to single point at certain number of iterations. To realize this function, we will record the newly generated single alone point when each shrinking iteration is performed and send the counted number to the location indexed by number of current iterations in a vector. Then we use this vector to plot histogram.

II. Experimental Results

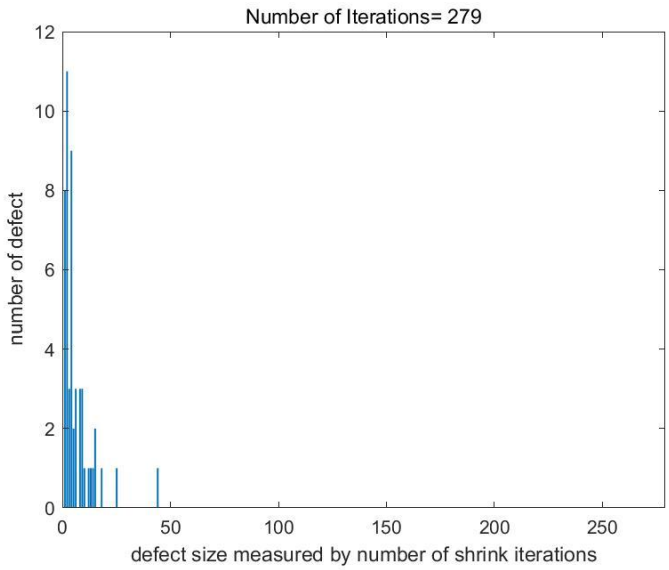


Fig 3.c.3 Complete Histogram of Number of Defects and Defect Size Measured by Number of Shrink Iterations (Maximum Iteration =279)

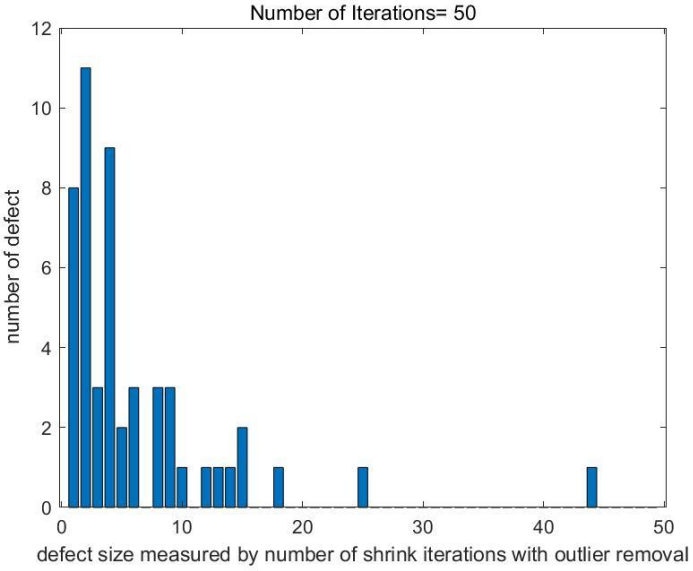


Fig 3.c.4 Rescaled Histogram of Number of Defects and Defect Size Measured by Number of Shrink Iterations with Outlier Removal (Maximum Iteration =50)

From the result in Fig 3.c.3, we acquired the Complete Histogram of Number of Defects and Defect Size Measured by Number of Shrink Iterations. The number of different defect sizes and the frequency of these defect sizes can also be observed from Fig 3.c.4. It is observed that most of the defects converge to a single point within 25 iterations so the majority of defects are small sized. To avoid the improper and inconvenient scale of Complete Histogram, we use smaller number of iterations 50 to acquire the proper-sized partial histogram in Fig 3.c.4 to observe distribution. And we can see that there is quite a number of very small size defect which only take one to three iteration to become a single point.

(3)(4) Using Connected-component labeling (CCL) Method to Get Clear Horse Image and do problem (1)(2) again

I. Approach and Procedures

The constraint that we cannot measure the exact number of pixels in a defect and record the position of these pixels in the previous problem prevent us to do pixel-level operation to get the clear horse image and generate the histogram based on defect size by number of pixels rather than size by number of iterations.

To solve this problem, Connected-component labeling (CCL) **[2]** is introduced and implemented. CCL is implemented by traversing the image and give the same label to pixels in the same connected region. With different labels assigned to pixels in different defect, we can get all the exact locations of pixels in a certain defect and the number of pixels in a certain defect very easily.

The CCL method is a two-step process. The first step is to assigned label to the non-background pixels. For a non-background pixel whose value is 1, we first detect its neighbor pixels composed of upper left, upper, upper right and left pixel and if all the value is 0, new label value is assigned to this center pixel. New label value is continuous and increases by one each time. While there are non-zero values in its neighbor stated above, we use the least non-zero value in the neighbor to perform as the label for central pixel.

The second stage is needed because in the first stage due to the reason of special patterns existing in non-background region, pixels in the same defect can get different labels. Fig 3.c.5 shows how this situation happens. The left image is the original binary image where 0 represents zero-padded boundary or background while 1 represents object. Then we can observe from the right image that even if the pixels with label 1 and 2 have the different labels they locate in the same region. This this where the problem lies in stage one result.

0	0	0	0	0
0	1	0	1	0
0	1	1	1	1
0	1	1	1	1

0	0	0	0	0
0	1	0	2	0
0	1	1	1	1
0	1	1	1	1

Fig 3.c.5 Original Binary Image (Left) and Result of CCL First Stage (Right)

The second stage is applied by allocating each central pixel with the least non-zero value in its 8 neighbor pixels and doing iterations until all the labels no longer change. It should be noticed that the first stage should be processed only once while the second stage should be processed until it converges. We can observe that after the entire CCL process, the pixels located in one connected region will have the same label in Fig 3.c.6.

0	0	0	0	0
0	1	0	1	0
0	1	1	1	1
0	1	1	1	1

Fig 3.c.6 Result of Complete CCL Process

II. Experimental Results

The CCL method is realized with my function CCL_LabelMap.m. To get the image of clear horse in **(3)**, we set all the pixels with values greater than 1 on the label map 1 and overlay it on the threshold binary image. We will get a binarized clear horse image as is shown in Fig 3.c.7.

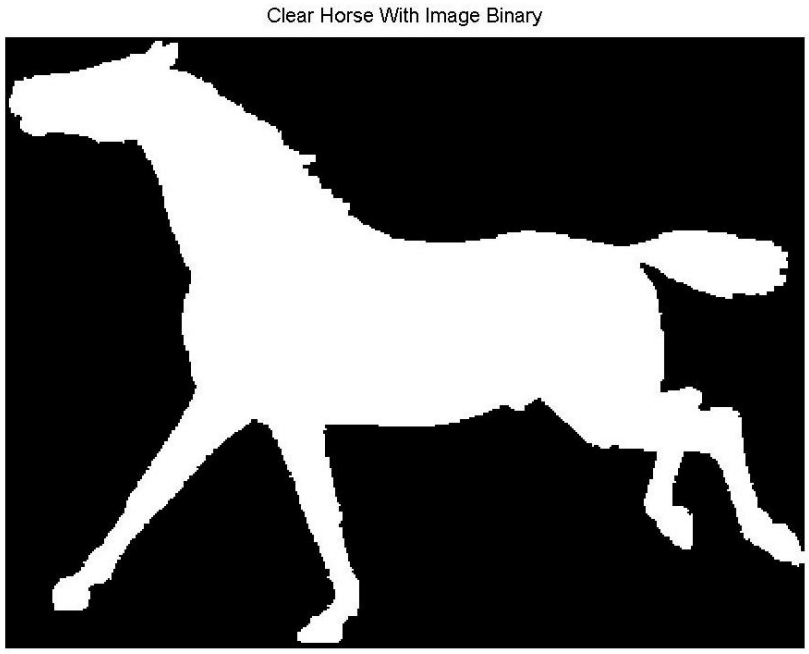
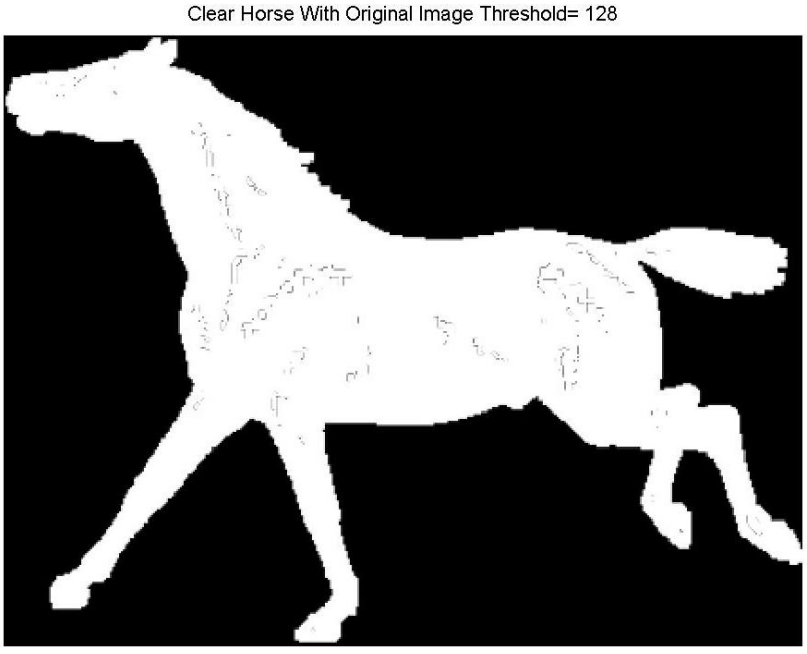


Fig 3.c.7 Result of Binarized Clear Horse

But there is still some room for improvement. Because in binarized image the edge information which is reflected by the contour is damaged, which still makes it different from the original horse.raw image.

Now we set all the pixels with values greater than 1 on the label map 255 and overlay it on the original image rather than the threshold binary image to get the clear horse image based on original horse image. Fig 3.c.8 shows the result of clear horse acquired by original horse image with threshold 128 and 250.



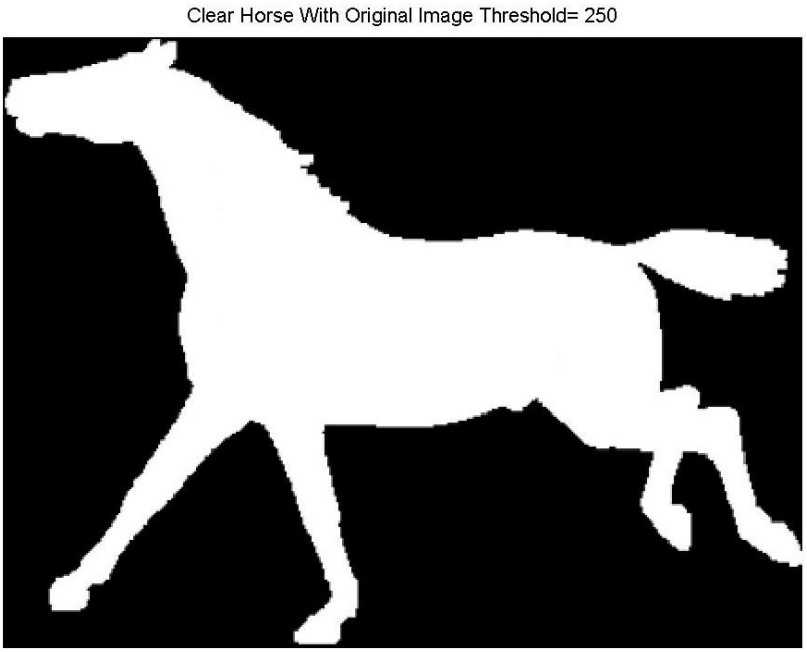


Fig 3.c.8 the Result of Clear horse acquired by Original Horse Image with Threshold 128 and 250

From Fig 3.c.8 we can observe that the result with 128 threshold have some residual edge information of defects. The reason is illustrated as follows. For example, when the threshold is 128, a point valued 129 will not be regarded as defect but finally it may locate among corrected defect points whose values are all 255. Therefore, the 129 point will appear noticeably darker compared to neighbor pixel valued 255. To solve this issue, threshold should be set anywhere smaller than but close to 255. I applied 255 finally and **the clear horse which is the lower image in Fig 3.c.8 (My final result) performs very well.**

To know the total defect size, we need to know the number of labels which still appear on the processed CCL label map. The problem we encounter here is that in the stage 2 label reduction process a lot of pixels replace the original assigned label with the least non-zero label in it 8 neighbor pixels so some labels do not exist in the label map any more. First, we count the number of pixels of each label to form a vector and plot label and number of pixels of each label histogram. As is shown in Fig 3.c.9, we have two histograms, the left one contains label 0 and label 1 which are black background label and white background label. The black background label region corresponds to the area of horse without defects while the white background label region corresponds to the area which is not in the horse. **Consequently, the pixels number of the two labels is very huge and they are not relevant to the defect which we want to discuss on so we delete them in Fig 3.c.9 left to observe the histogram better**

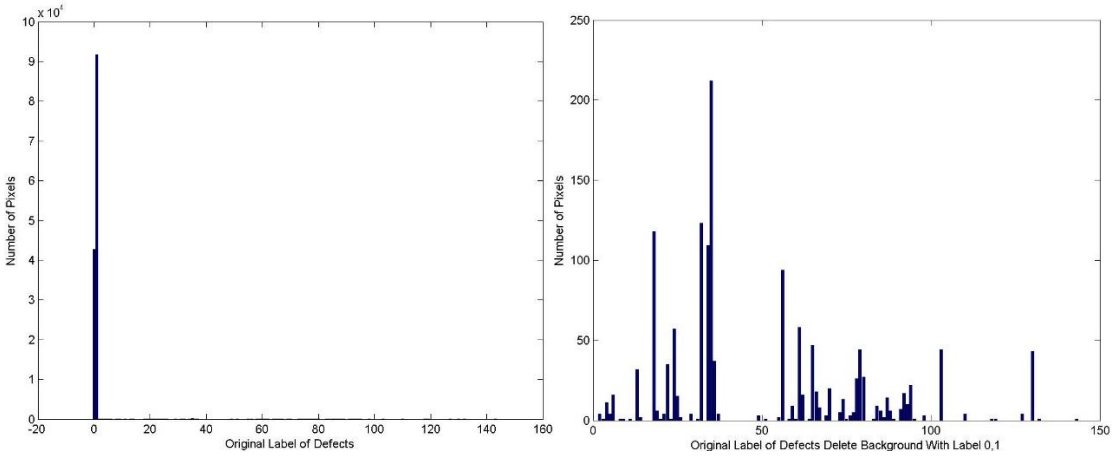


Fig 3.c.9 Label and Number of Pixels with This Label Histogram

However, Fig 3.c.9 only shows the **intermediate histogram** because we want to show there are lots of labels with zero pixels having their label values in the final label map after CCL. We can observe that there is near 150 labels in the histogram while only have of them have corresponding pixels. Then, in Fig 3.c.10, all the labels with 0 pixels having it in final label map are deleted and realization process can be seen in hw3problem3c3and4.m in row 124 to 148. **Here after label reduction, we can finally observe from Fig 3.c.10 that the total number of defects is 70.**

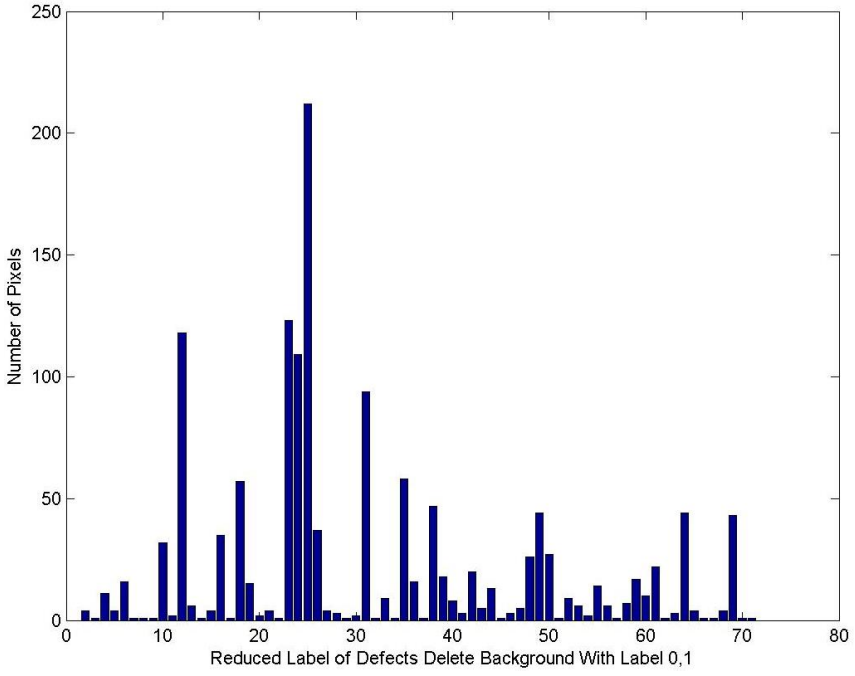


Fig 3.c.10 Reduced Label and Number of Pixels with This Label Histogram

The last step is to transform the histogram to defect size and number of defects with this size histogram. We generate the vector for new histogram by using the largest number of pixels in a label to initialize it. And the value in vector for histogram in Fig 3.c.10 is applied as index to the new vector. When traversing each element in the old

vector, the element in new vector indexed by value in old vector will increase by one. After traversing all elements in old vector, we have made **the vector for histogram of defect size and number of defects with this size which is shown in Fig 3.c.11. (The result for (2) by CCL method)**

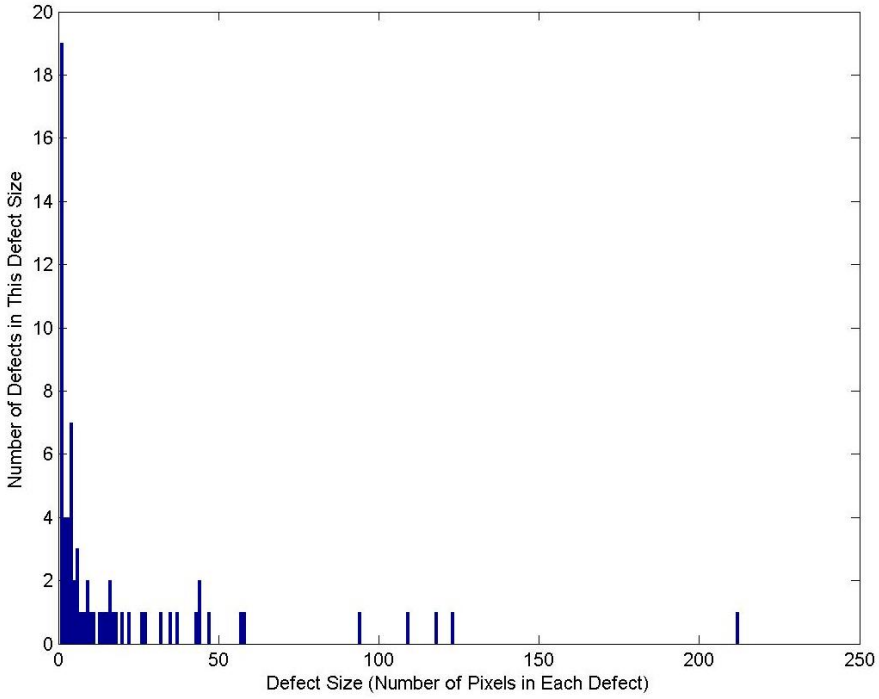


Fig 3.c.11 Histogram of Defect Size and Number of Defects with This Size

It can be concluded from Fig 3.c.11 that most of the defect sizes are very small and within the total pixel number of 25. Some of them are even noise point sized only one pixel. The several defect with defect size larger than 50 are the most visual salient defect on the original image horse.raw.

References:

[1] Fong C. Analytical methods for squaring the disc. arXiv preprint arXiv:1509.06344. 2015 Sep 21.

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