EE569 Digital Image Processing

HOMEWORK # 3

Aagam Shah

USC ID-8791018480

USC Email – aagamman@usc.edu

Submission Date – March 3, 2020

Problem 1: Geometric Image Modification

(a) Geometric Warping:

1. ABSTRACT & MOTIVATION

Digital Geometric Warping is an emerging branch of image processing that deals with the geometric transformations of digital images. A *geometric transformation* is an operation that redefines the spatial relationship between points in an image. Although image warping often tends to conjure up notions of highly distorted imagery, a warp may range from something as simple as a translation, scale, or rotation, to something as elaborate as a convoluted transformation. Since all warps do, in fact, apply geometric transformations to images, the terms "warp" and "geometric transformation" can be used interchangeably. However, the goal of the Geometric warping is not the geometric correction, but rather inducing geometric distortion. So, basically image warping is used to create interesting visual effects. Now let's talk about the **Spatial transformations.**

Spatial Transformation:

The basis of geometric transformations is the mapping of one coordinate system onto another. This is defined by means of a *spatial transformation* - a mapping function that establishes a spatial correspondence between all points in the input and output images. Given a spatial transformation, each point in the output assumes the value of its corresponding point in the input image. The correspondence is found by using the spatial transformation mapping function to project the output point onto the input image.

Depending on the application, spatial transformation mapping functions may take on many different forms. Simple transformations may be specified by analytic expressions including affine, projective, bilinear, and polynomial transformations.

The Geometric Image Warping find its application in various fields such as remote sensing, medical imaging, computer vision and computer graphics, etc.

In computer graphics, for example, the spatial transformation is completely specified by the parameterization of the 3-D object, its position with respect to the 2-D projection plane (i.e., the viewing screen), viewpoint, and center of interest. The objects are usually defined as planar polygons or bicubic patches. Consequently, three coordinate systems are used: 2-D texture space, 3-D object space, and 2-D screen space. The various formulations for spatial transformations, as well as methods to infer them, are discussed in Chapter 3.

2. APPROACH & PROCEDURE

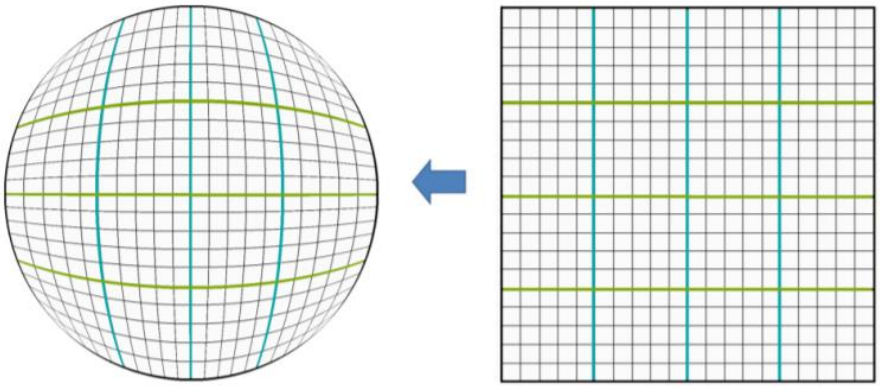
The implementation of spatial warping technique which will transform an input square image into an output disk-shaped image and again back to the square image in reverse mapping can be explained as below:

Forward Mapping:

Step 1: Initially we load the raw RGB input image on which the geometric warping is to be performed.
Step 2: Next we separate the R, G and B plane of the RGB image inside the two FOR loops, the one running for the rows and the other running for the columns.
Step 3: We basically, will use the **Elliptical Grid Mapping** approach in order to map the square to a circle and reverse map the warped image back to square.
Step 4: In order to apply the Forward mapping equations devised by Philip Nowell we need to convert the square coordinates in the range of [-1,1].
Step 5: After that we can apply the Forward mapping equations as given below:

Square to disc mapping:

$$u = x\sqrt{1 - \frac{y^2}{2}} \qquad v = y\sqrt{1 - \frac{x^2}{2}}$$



Where *x* and *y* are the coordinates of the square input image and *u* and *v* are the coordinates of the warped circular disc shaped image.

Step 6: Next we again convert the range of coordinates back to the original.

Step 7: The last step of the forward mapping is to concatenate back the R, G and B plane to get the final output as warped circular disc shaped image.

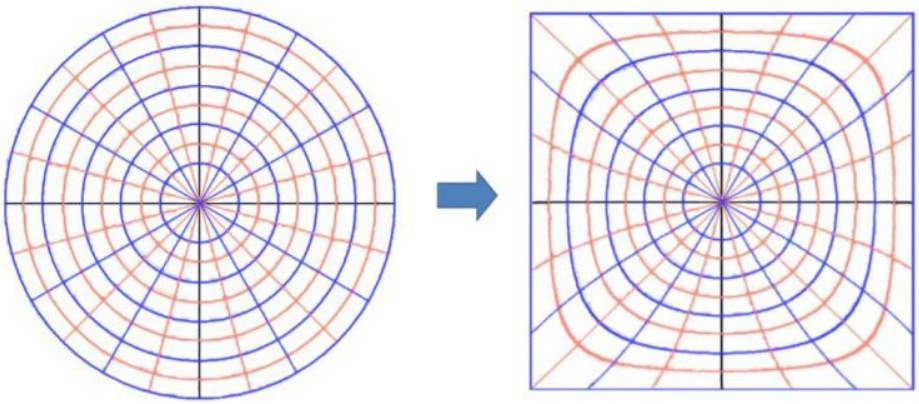
Backward Mapping:

Step 1: For the case of Backward mapping the warped circular disc shaped image obtained will act as an input image for the reverse mapping condition i.e., from circular disc shaped image back to square shaped image.

Step 2: Next we again separate the R, G and B plane of the RGB image inside the two FOR loops, the one running for the rows and the other running for the columns.

Step 3: Here again we use the **Elliptical Grid Mapping** for reverse mapping. So, for that we need to check two conditions, one is the range should be from [-1,1] and *u* and *v* which are the circular coordinates should be in the domain $\{(u,v) \mid u^2 + v^2 \leq 1\}$.

Step 4: Proceeding further we apply the equations for the reverse mapping as given below:



Disc to square mapping:

$$x = \frac{1}{2}\sqrt{2 + u^2 - v^2 + 2\sqrt{2}\,u} - \frac{1}{2}\sqrt{2 + u^2 - v^2 - 2\sqrt{2}\,u}$$

$$y = \frac{1}{2}\sqrt{2 - u^2 + v^2 + 2\sqrt{2}\,v} - \frac{1}{2}\sqrt{2 - u^2 + v^2 - 2\sqrt{2}\,v}$$

Step 5: Next we again convert the range of coordinates back to the original.
Step 7: The last step of the reverse mapping is to concatenate back the R, G and B plane to get the final output as the square shaped image, i.e., one to one mapping.

3. EXPERIMENTAL RESULTS

Fig. 1.1

Fig. 1.2

Fig. 1.3 (Reconstructed image)

Fig. 1.4

Fig. 1.5

Fig. 1.6 (Reconstructed image)

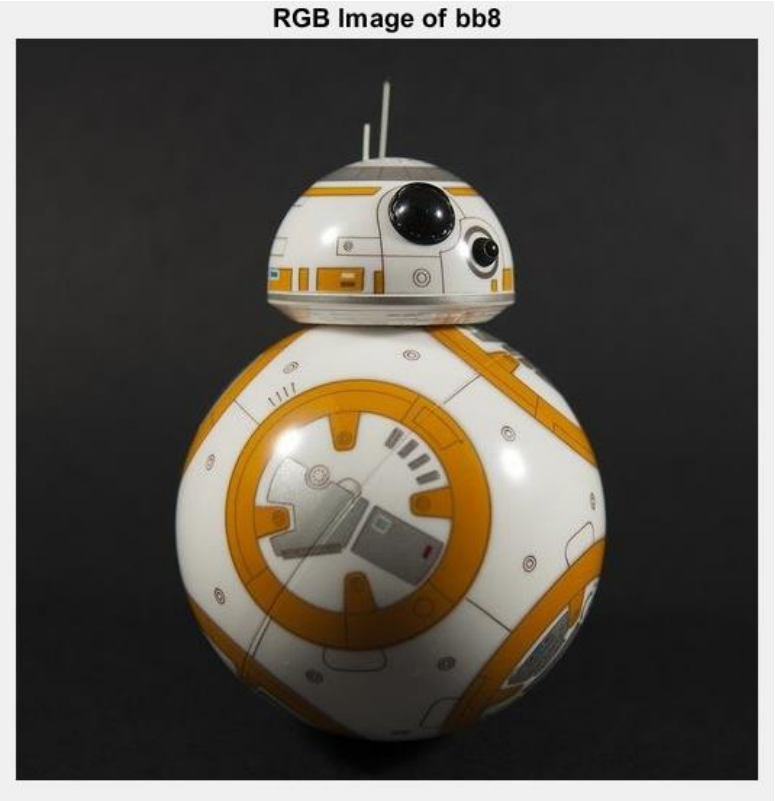


Fig. 1.7

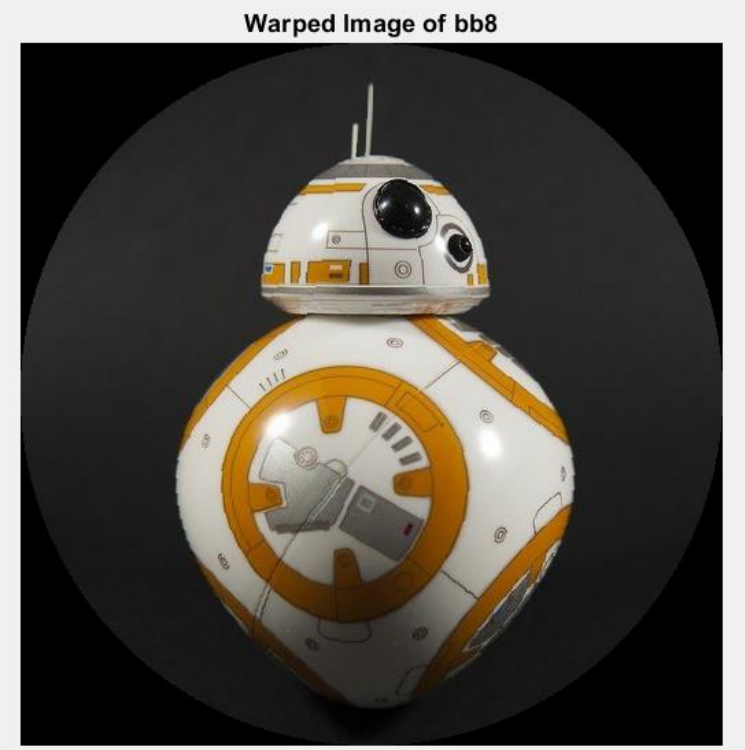


Fig. 1.8

Fig. 1.9 (Reconstructed Image)

4. DISCUSSION

Thus, the warped image satisfies the following three requirements:

- Pixels that lie on boundaries of the square still lie on the boundaries of the circle.
- The center of original images is mapped to the center of the warped images.
- The mapping is reversible, i.e., it is a one-to-one mapping.

ANSWERS:

i. The approach for the entire process or the algorithm is explained in detail in the **Approach and Procedure** section.

ii. The reverse spatial warping is also applied to each warped image to recover its original image.

iii. On comparing the recovered square image with the original square image, we find the difference is we can see the artifacts at the corners of the recovered square image has some black patches as compared to that of the original square image. Also, the recovered square image gets a bit stretched at boundaries while performing reverse mapping as compared to that of the original square image. Also, the image boundary gets a bit distorted when we reverse map the image from disc to square in case of reverse mapping the reason is, we stretch the pixels which results in image distortion. Also, the image obtained is little blurred as compared to the original or warped image.

(b) Homographic Transformation and Image Stitching:

1. ABSTRACT & MOTIVATION

Homography is basically defined as any two images of the same planar surface in space. This is practically used in operations such as image rectification, image registration or image stitching. So, the gist of the image stitching concept in simple terms can be explained as if we want to capture a big scene but our camera provides an image only of a specific resolution and cannot capture the panoramic view, so what can be done is to capture multiple images of the entire scene and then put all the separate pieces together in one big image. Such images are known as Panoramic images. The entire process of acquiring multiple images and converting them into such panoramas is called as Image Stitching and we then finally have one big and large photographic view.

2. APPROACH & PROCEDURE

Here the basic principle is to process the consecutive pair of images. The steps followed to achieve the composition of the images or to get the image stitching can be described as follows:

Step 1: Initially we select the control points from both the adjacent images for feature detection and feature matching to detect the control points.
Step 2: Here, we use the SIFT/SURF feature detection and FLANN for feature matching. So, if we use the SIFT as the feature extractor, it returns a 128-dimensional feature vector for each key point and if we use the SURF, we get a 64-dimensional feature vector.
FLANN is basically the technique used for feature matching as a descriptor matcher.
Here we do this implementation using the Open Source code for selecting the control points.
Step 3: Next step is the matching of the key points i.e., homograph mapping by applying the homographic transformation. The homographic transformation procedure can be explained as follows:
In perspective projection i.e., the pin hole camera model, the image of points in plane from two different camera viewpoints are related by homography as

$$P2 = H*P1$$

Where H is a 3x3 homographic transformation matrix, whereas P1 and P2 denote the corresponding image points in homogeneous coordinates before and after the transformation respectively.

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

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

Step 4: In order to calculate the H matrix, we follow the steps as:

i. In order to avoid the complex calculation by taking SVD of the system, we follow the easier to solve that is, as the last component in the H vector is considered as 1, we add one more constraint equation as (h9=1) by fixing H33 = 1 so now there are only 8 parameters to be determined.
ii. Next we select the pair of 4 points in the two adjacent images in order to build eight linear equations.
iii. Then in order to get the 8 parameters of the H matrix we solve these 8 linear equations.
iv. After solving we get the H parameters and we determine the H matrix.
v. After obtaining the H matrix, we project all the points from one image to another by warping technique which is nothing, but the backward mapping followed by interpolation technique.

Step 5: Next we now warp one image onto the other adjacent image using the estimated H transformation matrix.

Step 6: Finally, we create a new big panorama image which can fit the composite warped image into it. The image composition can also be done by simply averaging the pixel values at the location where the two images will overlap.

3. EXPERIMENTAL RESULTS

Fig. 1.10

Fig. 1.11

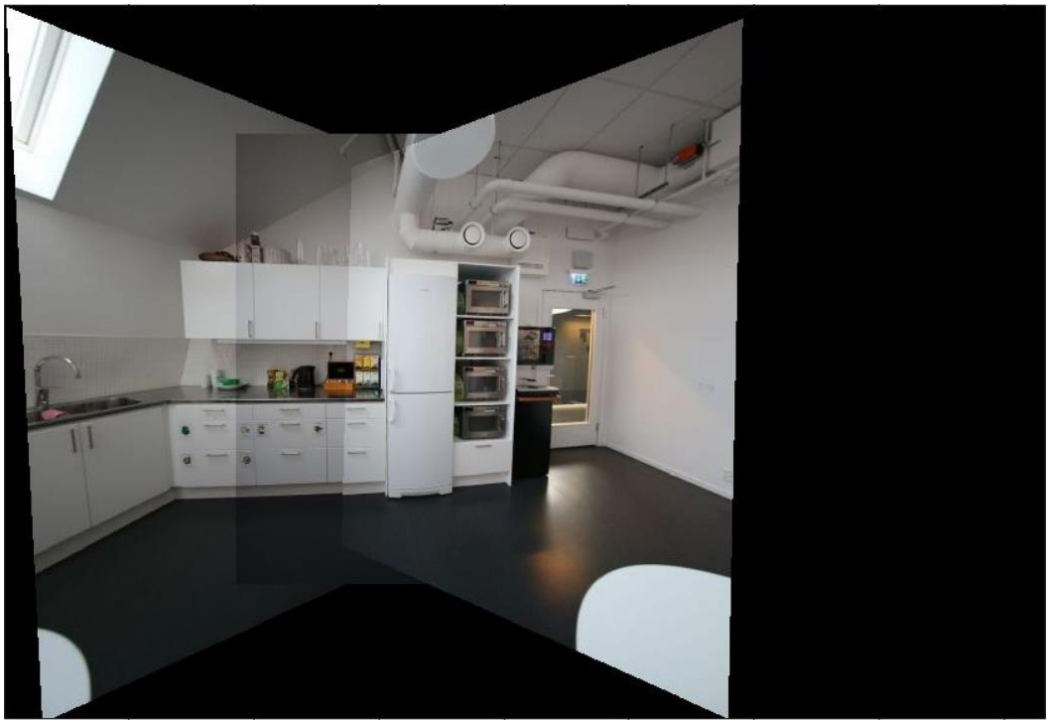


Fig. 1.12

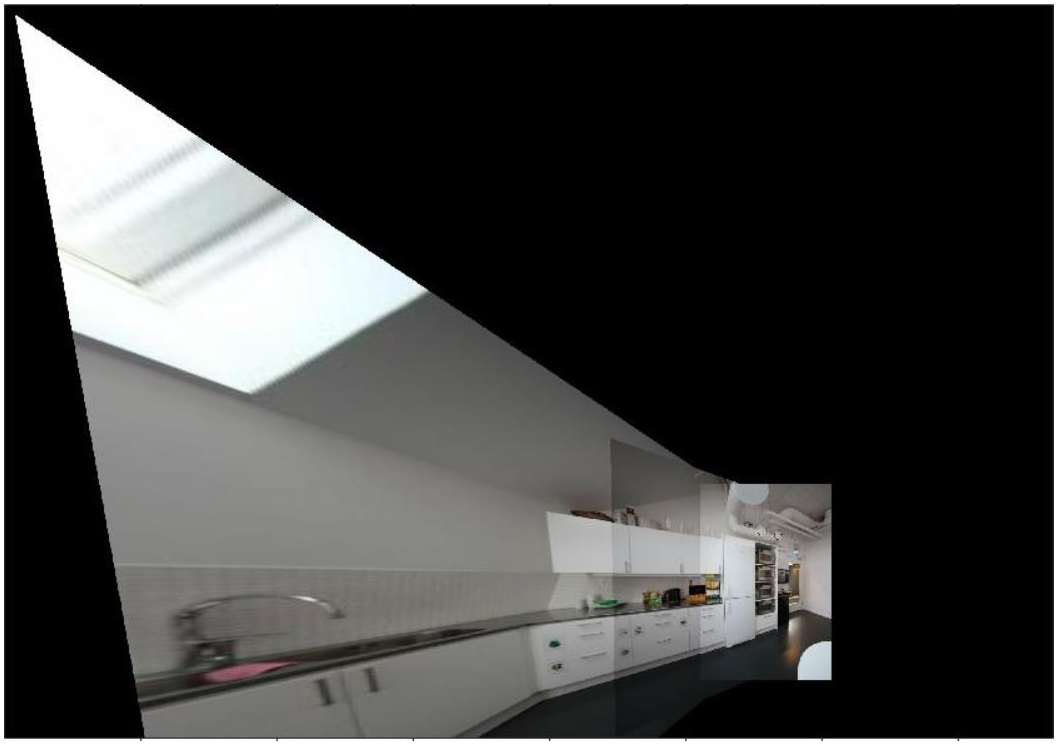


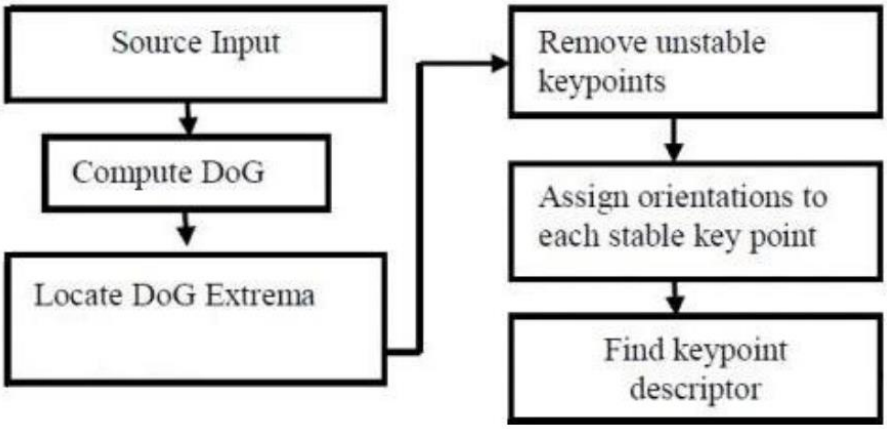
Fig. 1.13

4. DISCUSSION

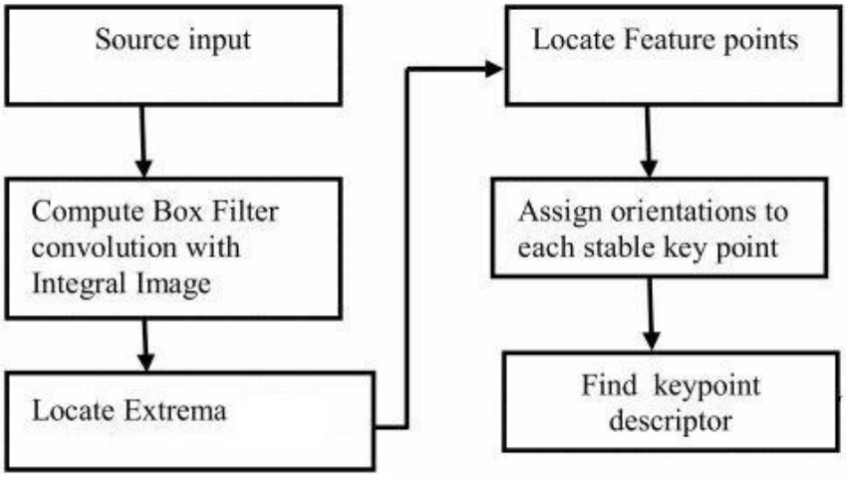
The experimental result obtained is presented above in the experimental section. Based on the implementation of homographic transformation and the stitching techniques to composite the room images answer to the questions is as follows:

i. Four control points were used for feature detection and feature matching. The corresponding control points between the left and the middle pair, and the middle and the right pair is depicted in the result section.

ii. The control points between the two images are matched by identifying their nearest neighbors. SIFT/SURF feature detection is used to match the features which can be summarized using the flowchart for both feature detection and matching algorithm using SIFT/SURF technique.



Flow Chart for SIFT feature detection



Flow Chart for SURF Feature Detection

Basically, there are majorly 4 steps involved in SIFT algorithm which are as follows:

i. Scale-space Extrema Detection – In order to detect the larger corners, we use scale-space filtering which uses Laplacian of Gaussian for images with

various sigma values. DoG is used for an approximation of LoG and if it is a local extremum it is a potential key-point.

ii. Key-point localization - Once potential key-points locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extremum is less than a threshold value, it is rejected. DoG has higher response for edges, so edges also need to be removed.

iii. Orientation Assignment - Now an orientation is assigned to each key-point to achieve invariance to image rotation. A neighborhood is taken around the key-point location depending on the scale, and the gradient magnitude and direction is calculated in that region. It creates key-points with same location and scale, but different directions. It contributes to stability of matching.

iv. Key-point Descriptor - Now key-point descriptor is created. A 16x16 neighborhood around the key-point is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histograms is created. So, a total of 128 bin values are available. It is represented as a vector to form key-point descriptor.

v. Key-point matching – Key-points between two images are matched by identifying their nearest neighbors.

SURF goes a little further and approximates LoG with Box Filter. One big advantage is that, convolution with box filter can be easily calculated with the help of integral images. And it can be done in parallel for different scales. Also, the SURF relies on determinant of Hessian matrix for both scale and location.

For orientation assignment, SURF uses wavelet responses in horizontal and vertical direction for a neighborhood of size 6s. Adequate gaussian weights are also applied to it. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees.

For feature description, SURF uses Wavelet responses in horizontal and vertical direction (again, use of integral images makes things easier). Another important improvement is the use of sign of Laplacian (trace of Hessian Matrix) for underlying interest point. It adds no computation cost since it is already computed during detection. In the matching stage, we only compare features if they have the same type of contrast (as shown in image below). This minimal information allows for faster matching, without reducing the descriptor's performance. In short, SURF adds a lot of features to improve the speed in every step. Analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. SURF is good

at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.

Problem 2: Morphological Processing

Morphological image processing is a type of processing in which the spatial form or structure of objects within an image are modified. Basically, the morphological image processing is a collection of non-linear operations related to either the shape or morphology of features in an image. Morphological operations generally depend on the relative ordering of the pixel values, rather than their intensity values and therefore they are specifically adaptable for the processing of binary images. A morphological operation on a binary image creates a new binary image in which the pixel has a non-zero value only if the test output results in successful at that location in an input image. The main idea behind the morphological operation is to eliminate the distortion which affects the shape and the texture of the images.

(a) Basic morphological process implementation:

1. ABSTRACT & MOTIVATION

   The dilation, erosion and skeletonizing are the three fundamental morphological operations. In dilation, an object grows uniformly in spatial extent, while in erosion an object gets shrinked uniformly. Skeletonizing results in the stick figure representation of an object. Here, we will be performing the three major morphological operations which are 'Shrinking', 'Thinning' and 'Skeletonizing'.

2. APPROACH & PROCEDURE

   Before proceeding for the understanding the concepts for Shrinking, Thinning and Skeletonizing we need to first understand the concept of Binary Image Connectivity and Binary Image Hit or Miss Transformation.

   i. **Binary Image Connectivity**:

   The binary image morphological operations are generally based on the geometrical relationship or the connectivity of the neighboring pixels that are deemed to be of the same class. On considering the following neighborhood pixel pattern:

$$\begin{matrix} X_3 & X_2 & X_1 \\ X_4 & X & X_0 \\ X_5 & X_6 & X_7 \end{matrix}$$

   Where a binary valued pixel F(i,j) = X and where X = 0 or 1 surrounded by its eight nearest neighbors X0, X1, X2, X3, X4, X5, X6 and X7.

An alternative nomenclature to label the neighbors is by the compass directions i.e., North, Northwest, West, Southwest, South, Southeast, East, Northeast. The representation can be as follows:

NW	N	NE
W	X	E
SW	S	SE

The pixel X is said to be *four* connected if X0 = X2 = X4 = X6 = 1.
The pixel X is said to be *eight* connected if X0 = X1 = X2 = X3 = X4 = X5 = X6 = X7 = 1.
The connectivity relationship between a center pixel and its eight neighbors can be quantified by the concept of a *pixel bond*, the sum of the bond weights between the center pixel and each of its neighbors. Each four-connected neighbor has a bond of two, and each of eight-connected neighbor has a bond of one.

ii. **Binary Image Hit or Miss Transformations**:

The morphological Hit-or-Miss Transform is a basic tool for the shape detection and image segmentation. The concept is quite simple. Morphological techniques probe an image with a small shape or template known as structuring element. Structuring element is nothing but a small odd-sized mask, which is scanned over all the possible locations on a binary image and it is compared with the corresponding neighborhood of pixels. A common practice is to have odd dimensions of the structuring matrix and the origin is defined as the center of the matrix. In simple terms, the structuring element is a small binary image, i.e., a small matrix of pixels, each with value of either zero or one:

- The matrix dimensions specify the size of the structuring element.
- The pattern of ones and zeros will specify the shape of the structuring element.
- An origin of the structuring element is usually one of its pixels.

If the binary-valued pattern of the mask matches with the state of the pixels under the mask (hit), and an output pixel in spatial correspondence to the center pixel of the mask is set to some desired binary state. For a pattern mismatch (miss), the output pixel is set to the opposite binary state.

Hit and Miss Morphological algorithms are often implemented in digital image processing hardware by a pixel stacker followed up by a look-up table (LUT). Each pixel of an input image is a positive integer, represented by a binary code which is either 1 or 0. Then the pixel stacker extracts the bits of the center pixel X and its eight neighbors and puts them in a neighborhood pixel stack. Finally, the binary number state of the neighborhood pixel stack becomes the numeric input address of the LUT whose entry is Y.

EE569 Digital Image Processing

Before proceeding towards the understanding of the procedure of implementation, let us first understand the basic concepts of Shrinking, Thinning and Skeletonizing. Shrinking, Thinning and Skeletonizing are basically various forms of erosion in which the erosion process is controlled in order to prevent the total erasure and to ensure the connectivity.

1) **Shrinking:** It is basically the process which erases the black pixels such that an object without the holes erodes to a single pixel at or near its center of mass, and an object with holes erodes to a connected ring lying midway between each hole and its nearest outer boundary.
2) **Thinning:** In this process it erases the black pixels such that an object without holes erodes to a minimally connected stroke located equidistant from its nearest outer boundaries, and an object with holes erodes to a minimally connected ring midway between each hole and its nearest outer boundary.
3) **Skeletonizing:** It is basically a skeleton or stick figure representation of an object that can be used to describe its structure.

Implementation:

As it is not possible to perform shrinking using a single stage 3x3 pixel hit or miss transform because 3x3 window does not provide enough information to prevent the total erasure and to ensure the connectivity. A 5x5 hit or miss transform could provide enough information to perform appropriate Shrinking, Thinning and Skeletonizing but it would result in excessive computational complexity. So, therefore we implement two stage shrinking and thinning algorithms that perform a conditional marking of pixels for erasure in a first stage, and then examine neighboring marked pixels in a second stage to determine which ones can be unconditionally erased without total erasure or loss of connectivity.

The pipeline processor version of the conditional marking scheme can be explained as follows:

In the algorithm, two concatenated 3x3 hit or miss transformations are performed to obtain indirect information about the pixel patterns within a 5x5 window. So, basically in the first stage, the states of the nine neighboring pixels are gathered together by a pixel stacker, and a following look up table generates a conditional mark M for the possible erasures. The patterns by S, T and K in the pattern table will be conditionally marked as for Shrinking, Thinning and Skeletonizing respectively for erasures.

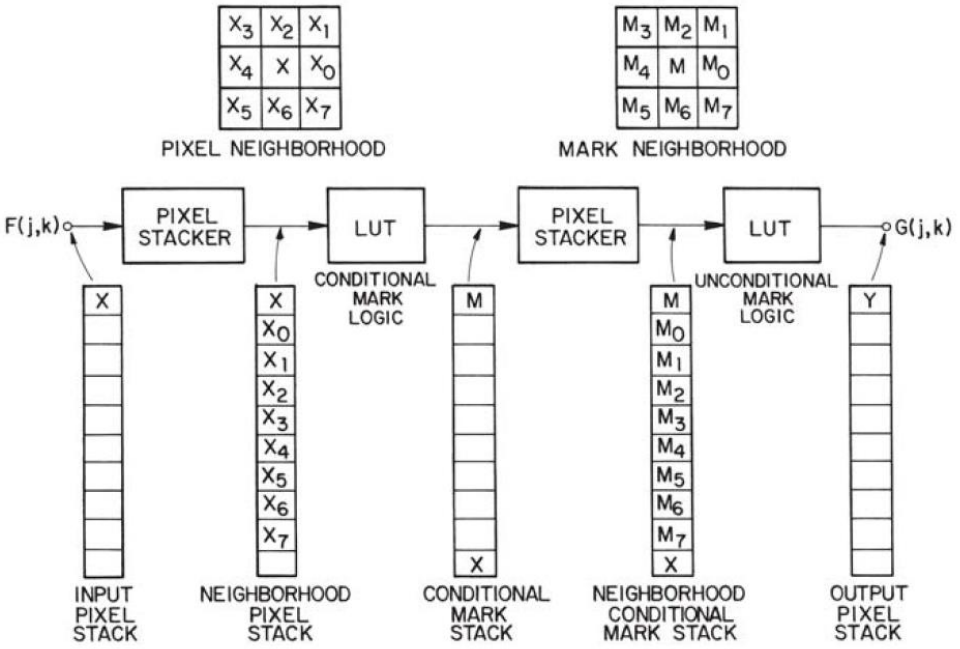
In the second stage of the algorithm, the center pixel X and the conditional marks in a 3x3 neighborhood centered about X are examined to create an output pixel.

The shrinking operation can be logically expressed as:

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

Where P(M,M0,….,M7) is an erasure inhibiting logical variable.

This two-stage process must be performed iteratively until there are no further erasures or the operation becomes idempotent.



Algorithm:

Step 1: Initially, we load the raw input image on which the morphological operations are to be performed by correctly specifying its dimensions.

Step 2: As the input image is 8-bit grayscale image, so for performing the morphological operations we first binarize the image by thresholding an input image.

Step 3: Next, we extend the boundary of an input raw image by padding it along the four edges of an input image with background pixels, which will act as a new input image.

Step 4: Next, we declare the matrix M and G as zeros matrix which will be of the same size as that of the new input image.

Step 5: Then we initialize the temporary matrix as G_old equal to G for comparing it after each iteration to check the flag condition until there are no further erasures and the operations become idempotent.

Step 6: After the above step we run the two FOR loops for rows and columns i.e., for i = 2:N+1 and for j = N+1, to check whether the center pixel of the new input image, F(i,j) is equal to 1, then if its equal to 1 then its eight neighboring pixels X0, X1, X2, X3, X4, X5, X6, X7 which are stored in an array are compared with the standard conditional mask array generated from the lookup tables of Shrinking, Thinning and Skeletonizing. If there is match, then the M(i,j) is set to 1, otherwise it is set to 0.

Step 7: Repeating this iteratively will generate M matrix which will act as an input for the second stage i.e., unconditional stage. Following the similar approach as that for the conditional logic, here also we run two FOR loops for rows and columns i.e., for i = 2:N+1 and for j = N+1 to set the G matrix. G(i,j) will be set only if the following condition is satisfied:

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

Where, P(M,M0,….,M7) are the unconditional masks for the shrinking, thinning and skeletonizing. It will be set to 1 only if there is a HIT by comapring it with the neighboring pixels such as M0, M1,….., M7 from the lookup table.

Step 8: Then we compare that whether the G matrix generated is equal to the G_old matrix, if the condition is false, then it is flagged as zero and F matrix which the new input image matrix is equated with G as (F=G) and also the G_old is stored as G and then the M matrix is flushed out and declared as again zeros matrix for next iteration. This process is repeated untill the iteration when the condition is flagged as true and then the while loop is exited and that is final morphed output for either Shrinking, Thinning or Skeletonizing.

Step 9: Finally, at this last step we display the final output morphed image as G matrix.

3. EXPERIMENTAL RESULTS

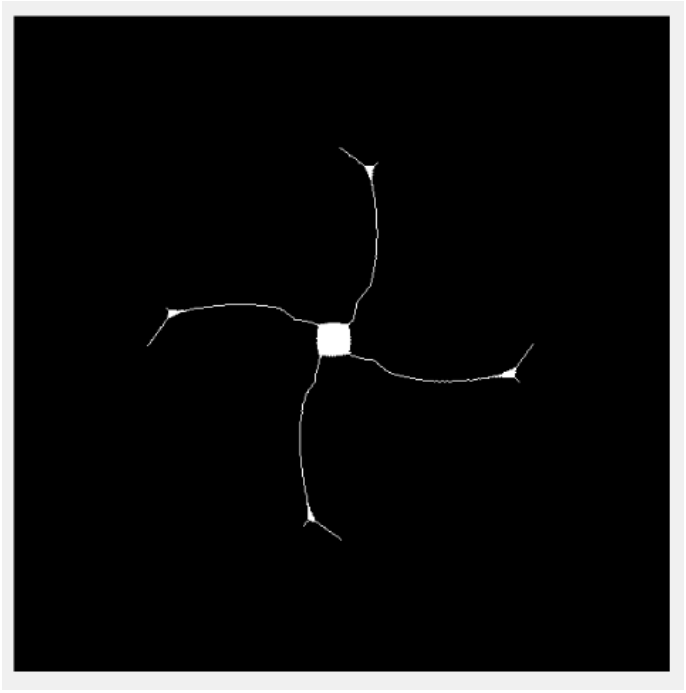


Fig.2.1 (Fan intermediate Shrink output)

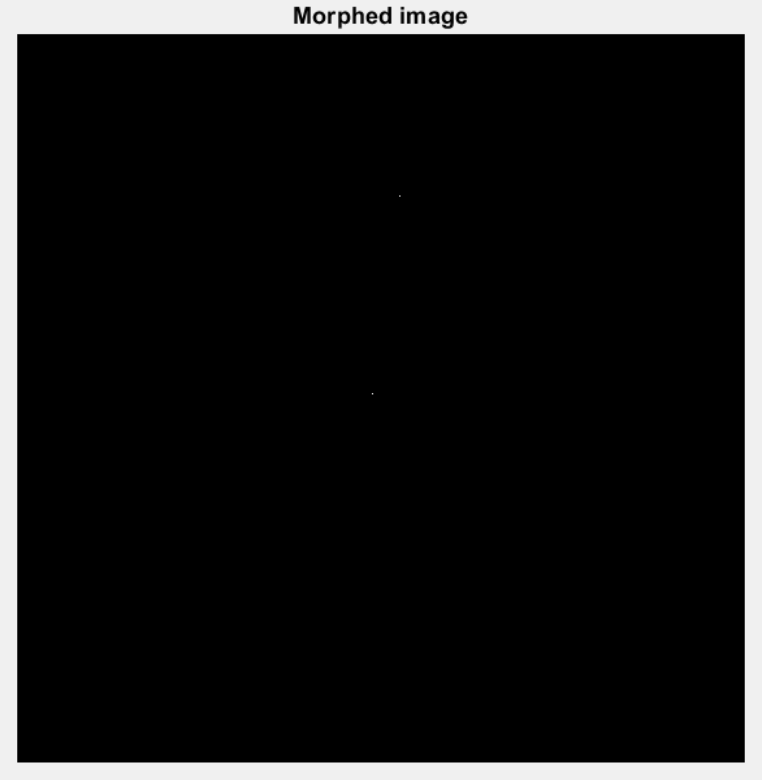


Fig.2.2 (Fan Shrinking output)

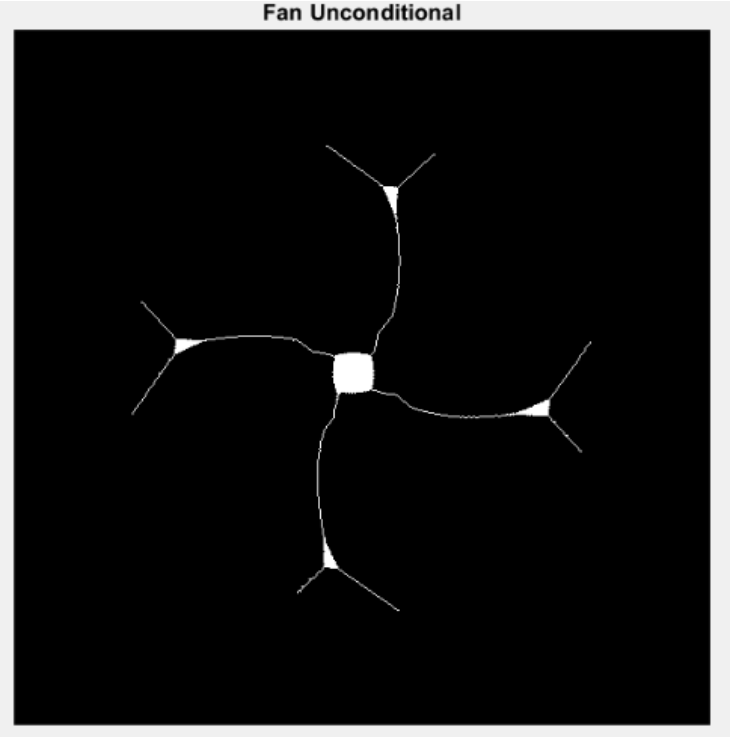


Fig.2.3 (Fan intermediate Thinning output)

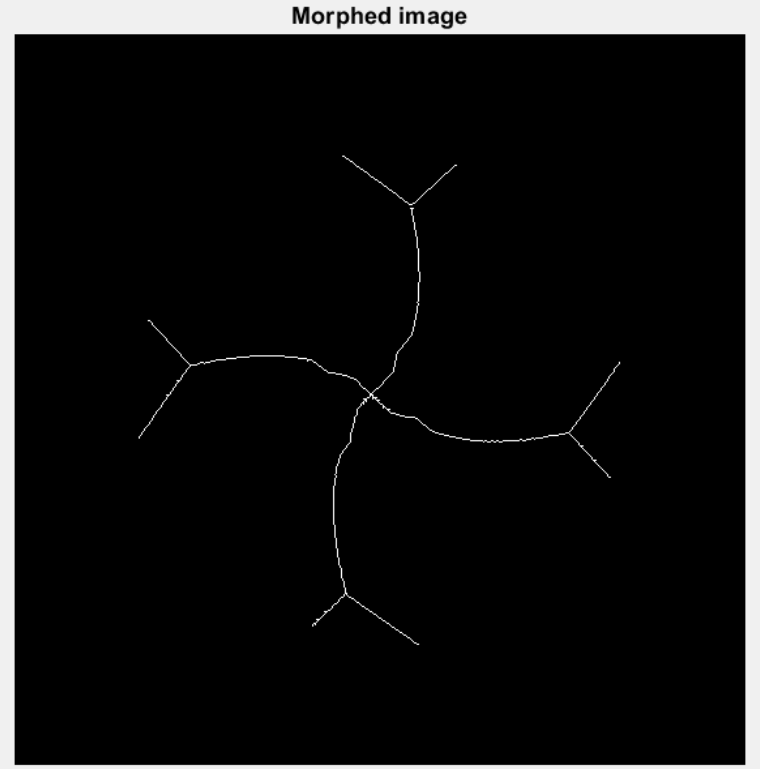


Fig.2.4 (Fan Thinning output)

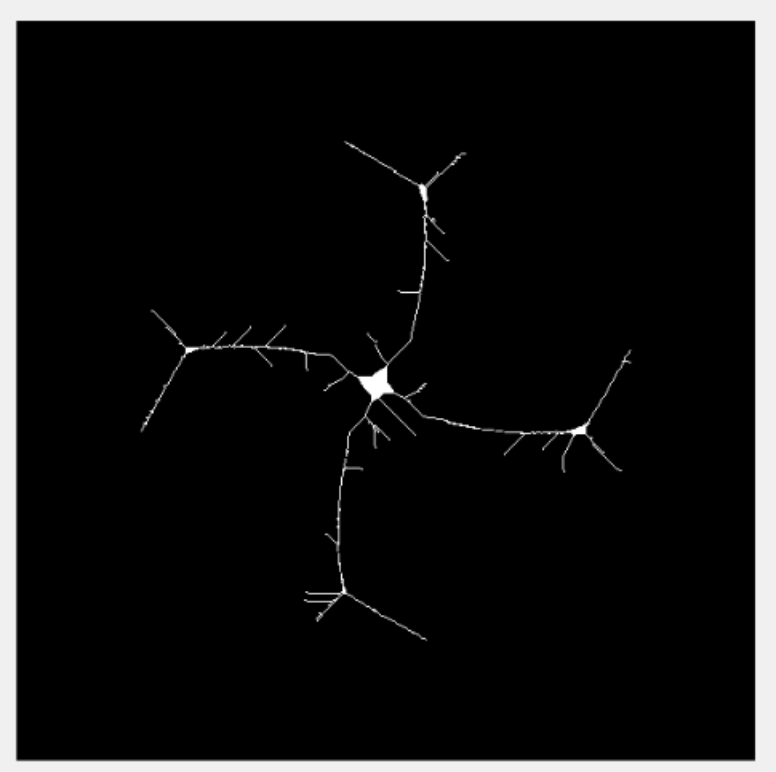


Fig.2.5 (Fan intermediate Skeletonizing output)

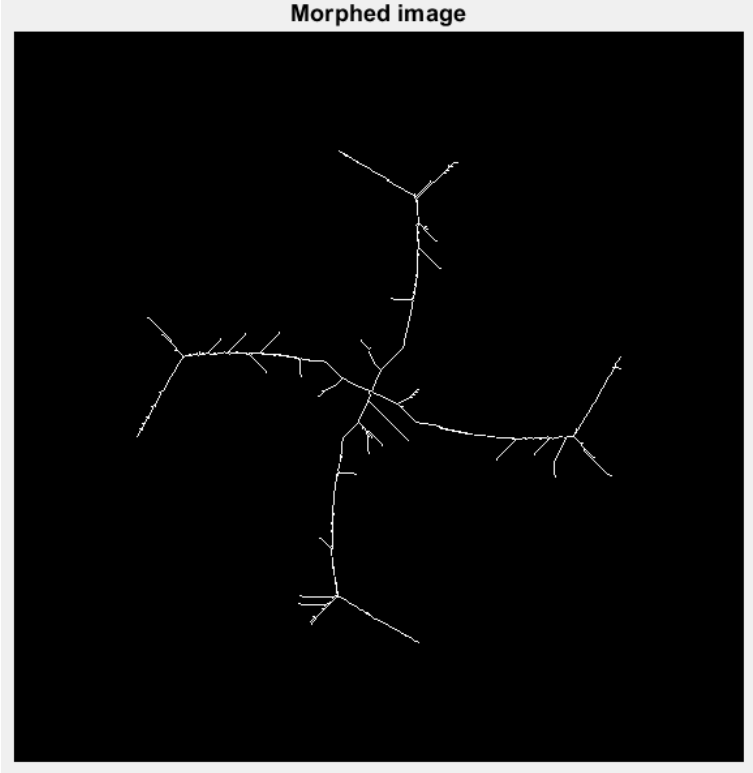


Fig.2.6 (Fan Skeletonizing output)

Fig.2.7 (Maze intermediate Shrink output)

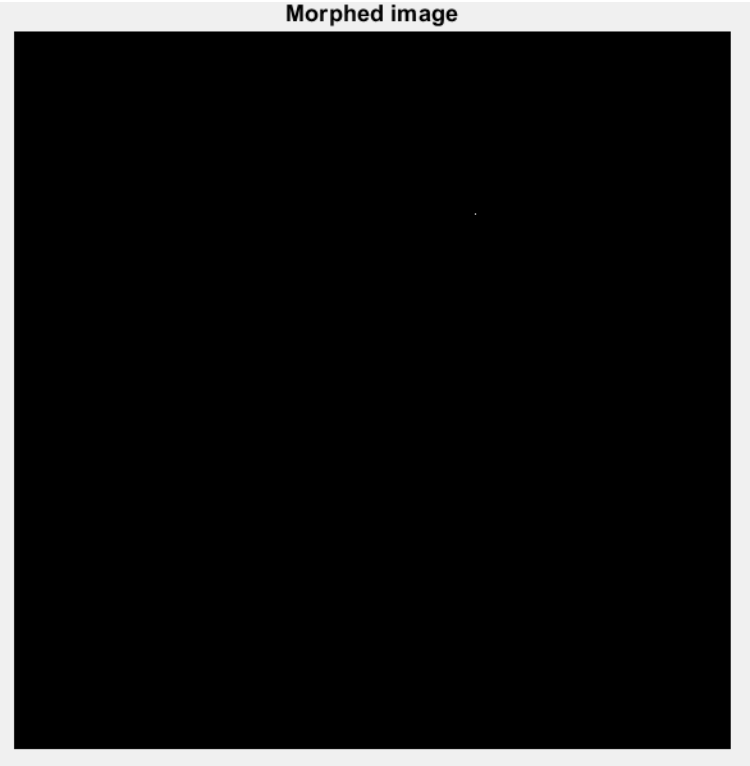


Fig.2.8 (Maze Shrinking output)

Fig.2.9 (Maze intermediate Thinning output)

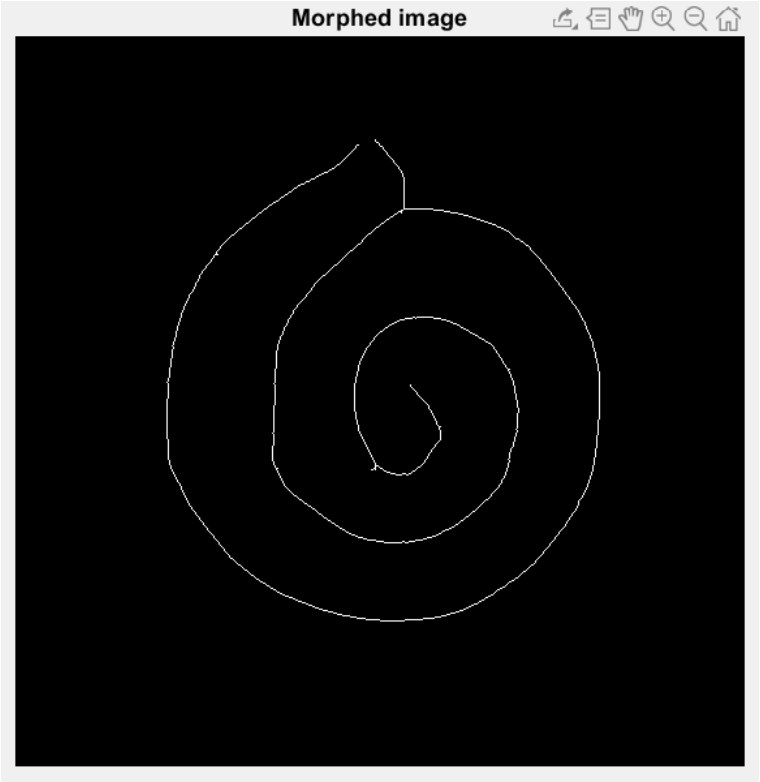


Fig.2.10 (Maze Thinning output)

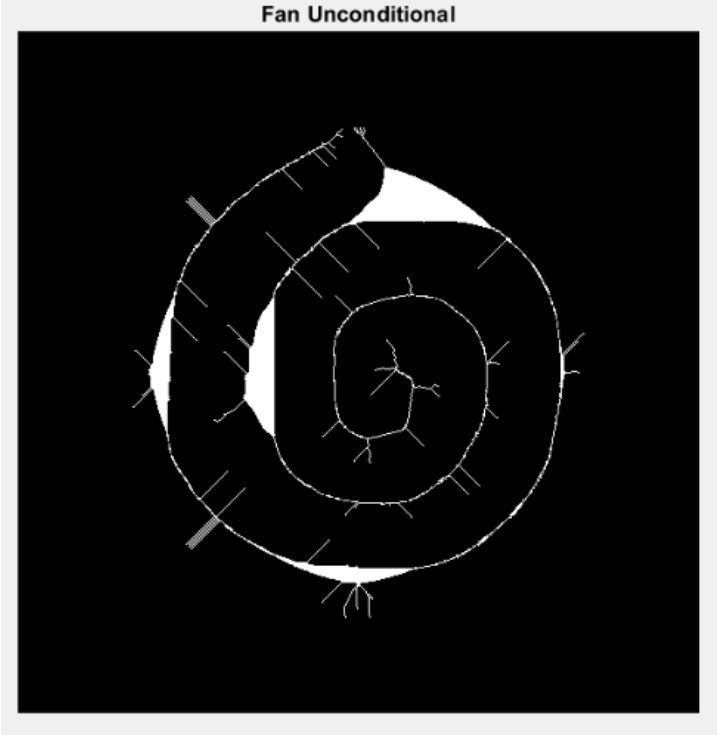


Fig.2.11 (Maze intermediate Skeletonizing output)

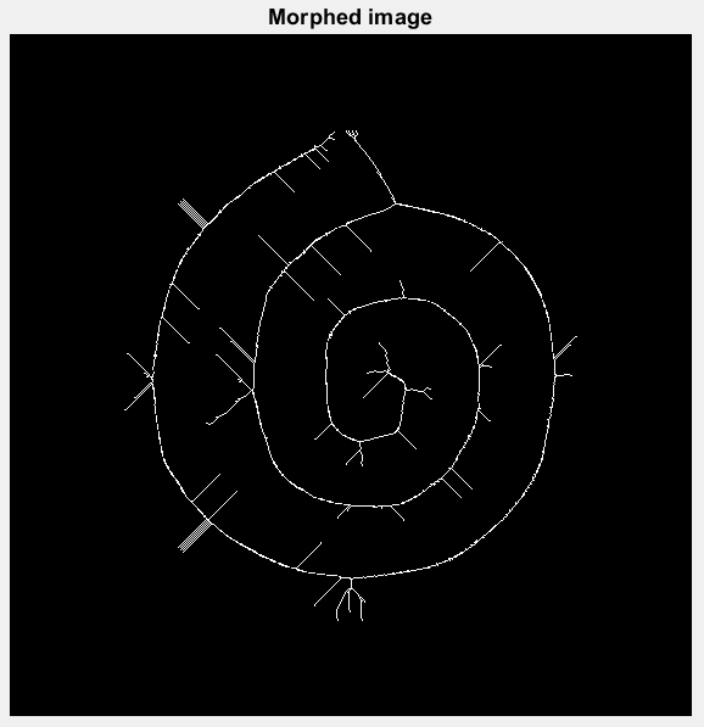


Fig.2.12 (Maze Skeletonizing output)

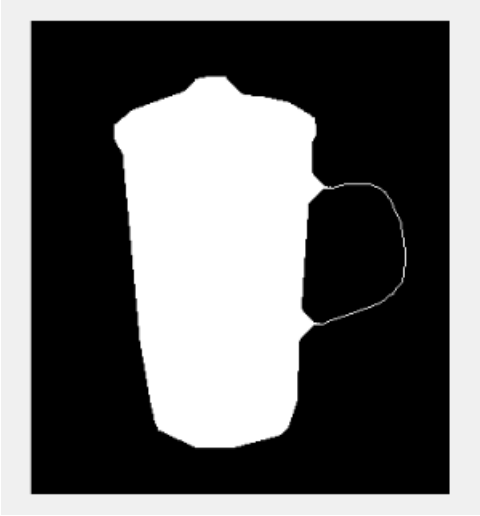


Fig.2.13 (Cup intermediate Shrink output)

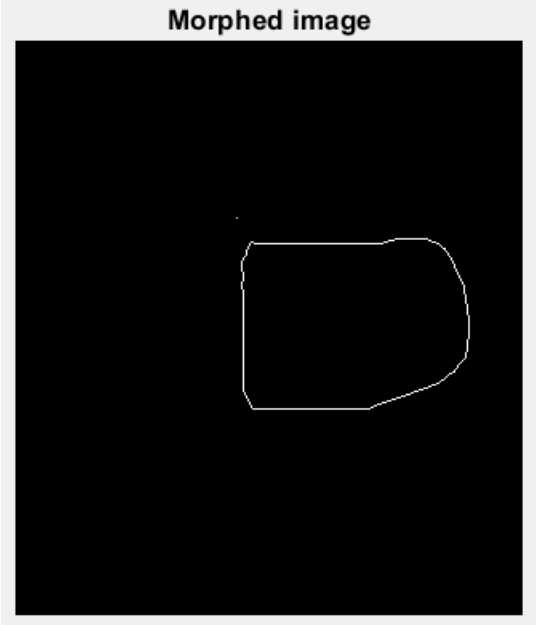


Fig.2.14 (Cup Shrinking output)

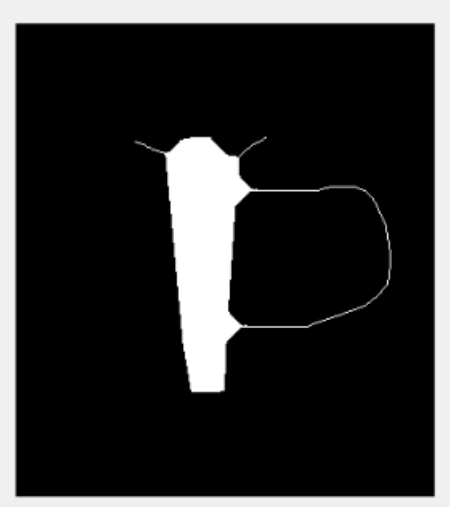


Fig.2.15 (Cup intermediate Thinning output)

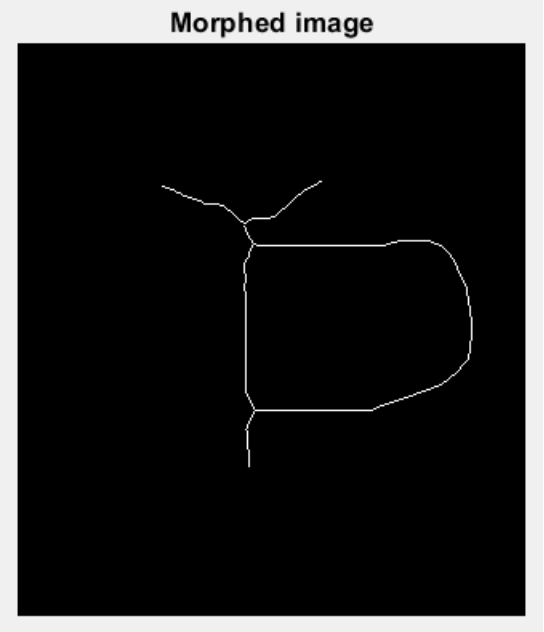


Fig.2.16 (Cup Thinning output)

Fig.2.17 (Cup intermediate Skeletonizing output)

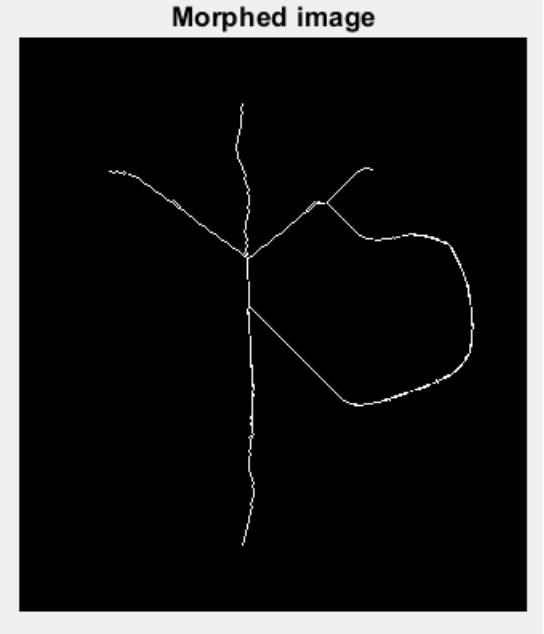


Fig.2.18 (Cup Skeletonizing output)

4. DISCUSSION

- As we discussed above that the 3x3 window size does not provide enough information to prevent the total erasure and to ensure the connectivity. A 5x5 hit or miss transform could provide enough information to perform appropriate Shrinking, Thinning and Skeletonizing but it would result in excessive computational complexity.
- From the **shrinking** results we can observe that the filled image which can be either white of black will shrink down to a single pixel near its center of mass. If the image contains any kind of hole then the black pixels within the white region will expand upon the number of iterations of the shrinking operation, while the white pixel region will try to shrink almost in the midway between the hole and the outer boundary.
- On observing the **thinning** results it can be said that the objects with holes erodes to a minimally connected ring like structure in the midway between the hole and its nearest outer boundary while for the objects which are without the holes will result in minimal connected stroke which will be located at equal distance from its nearest boundaries.
- On observing the **Skeletonization** results the objects are defined as a stick figure representation and more erasures taking place. Also the skeletonized image obtained will depend on the size of the mask as well as the set condition used for comparison.

(b) Counting games:

1. ABSTRACT & MOTIVATION

Binary images usually contain various types of imperfections. In general, binary regions produced by simple thresholding are distorted by noise and texture. So, Morphological image processing pursues the goals of removing these imperfections by considering the form and the structure of an image.
Here in this problem of counting stars, it can be solved by two types of approaches:

a. Shrinking technique
b. Connected component analysis

2. APPROACH & PROCEDURE

For counting games of stars, we follow the same approach as described in the previous problem. We can implement the Shrinking algorithm as well as Connected Component Analysis which is other than the morphological operations to count the number of stars in an image. We also determine the different star sizes as well as the frequency of these star sizes and plot the histogram for the same. The approach and the procedure for both the different implementations can be described as follows:

I. Shrinking Algorithm:

It is basically the process which erases the black pixels such that an object without the holes erodes to a single pixel at or near its center of mass, and an object with holes erodes to a connected ring lying midway between each hole and its nearest outer boundary.

Step 1: Initially, we load the raw input image on which the morphological operations are to be performed by correctly specifying its dimensions.

Step 2: As the input image is 8-bit grayscale image, so for performing the morphological operations we first binarize the image by thresholding an input image.

Step 3: Next, we extend the boundary of an input raw image by padding it along the four edges of an input image with background pixels, which will act as a new input image.

Step 4: Next, we declare the matrix M and G as zeros matrix which will be of the same size as that of the new input image.

Step 5: Then we initialize the temporary matrix as G_old equal to G for comparing it after each iteration to check the flag condition until there are no further erasures and the operations become idempotent.

Step 6: After the above step we run the two FOR loops for rows and columns i.e., for i = 2:N+1 and for j = N+1, to check whether the center pixel of the new input image, F(i,j) is equal to 1, then if its equal to 1 then its eight neighboring pixels X0, X1, X2, X3, X4, X5, X6, X7 which are stored in an array are compared with the standard conditional mask array generated from the lookup tables of Shrinking. If there is match, then the M(i,j) is set to 1, otherwise it is set to 0.

Step 7: Repeating this iteratively will generate M matrix which will act as an input for the second stage i.e., unconditional stage. Following the similar approach as that for the conditional logic, here also we run two FOR loops for rows and columns i.e., for i = 2:N+1 and for j = N+1 to set the G matrix. G(i,j) will be set only if the following condition is satisfied:

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

Where, P(M,M0,….,M7) are the unconditional masks for the shrinking. It will be set to 1 only if there is a HIT by comapring it with the neighboring pixels such as M0, M1,….., M7 from the lookup table.

Step 8: Then we compare that whether the G matrix generated is equal to the G_old matrix, if the condition is false, then it is flagged as zero and F matrix which the new input image matrix is equated with G as (F=G) and also the G_old is stored as G and then the M matrix is flushed out and declared as again zeros matrix for next iteration. This process is repeated untill the iteration when the condition is flagged as true and then the while loop is exited and that is final morphed output for Shrinking.

Step 9: In this step we display the final output morphed image as G matrix.

Step 10: Now, we initialize the counter as zero. After doing that we run the two FOR loops, one for the rows and the other for the columns and then using IF condition i.e., if G(i,j) is equal to 1 in the shrinked image output then increment the counter which will in turn give the count of the total number of stars in an input image. As we know from the definition of Shrinking that when the shrinking operation performed on stars it will shrink all the stars to single dots thereby giving the count of the total number of stars in the image.

II. Connected Component Analysis:

Connected Component Analysis is basically an algorithmic application of the graph theory, where the subsets of connected components are uniquely labeled based on a given heuristic. Connected component analysis is used in computer vision to detect connected regions in binary digital images.

Connected Component Analysis can be precisely defined as an operator whose input image is a binary image and the output image is a symbolic image in which the label assigned to each pixel is an integer uniquely identifying the connected component to which that pixel belongs to.

EE569 Digital Image Processing

Algorithm: (8 – connectivity based)

Here, we will be implementing the **two-pass algorithm** which is relatively easy to understand and simple to implement. The two-pass algorithm basically iterates through the 2-dimensional binary data. This algorithm basically makes two passes over the entire image. The first pass is carried out to assign the temporary labels and record the equivalences and then the second pass is performed to replace the temporary label by the minimum value of the label of its equivalence class.

The First Pass:

Step 1: At first, we assign the binary values to the pixels of an image i.e., 0 to the background pixel and 1 to the foreground pixel. Then the Connected Component matrix is initialized to the size of the image matrix.
Step 2: Then we iterate through each pixel of the column followed by the iteration of rows which is nothing but known as Raster Scanning (left to right and top to bottom) on the given input image.
Step 3: A label is initialized and then incremented for every detected object in an image. A counter is also initialized in order to count the total number of different objects present in an image.
Step 4: While performing Raster Scanning, check whether the pixel is foreground or background pixel.
Step 5: If the encountered pixel is foreground pixel then check for the neighboring pixel values i.e., *north* and *west* (for 4-connectivity) and *northeast, north, northwest* and *west* (for 8-connectivity) for the given region criterion (i.e., to check for the intensity value of 1).
Step 6: If there are none of the neighbors that satisfy the criterion then uniquely assign the label to the current pixel element and then increment the counter. Else if, one of the neighbors satisfies the criterion then assign the current label to that pixel. If multiple neighbors match and all belong to the same region then assign the current label to their region. If multiple neighbors match but belong to the different region then find the neighbor with the smallest label and then assign it to the current pixel element.
Step 7: Store the equivalence between the neighboring labels.

The Second Pass:

Step 1: Again, we iterate through each pixel of the column followed by the iteration of rows which is nothing but known as Raster Scanning (left to right and top to bottom).
Step2: While performing Raster Scanning, check whether the pixel is foreground or background pixel.
Step 3: If the encountered pixel is foreground pixel then now relabel the pixel element with the lowest equivalent label.

The run time of the algorithm will depend on the size of the image and the amount of foreground pixels. The time complexity is comparable to the two-pass algorithm if the foreground pixels covera a significant part of the image. Otherwise, the time complexity is lower. However, the memory access is less structured than for the two-pass algorithm, which tends to increase the run time in practical situations.

3. EXPERIMENTAL RESULTS

Fig.2.19

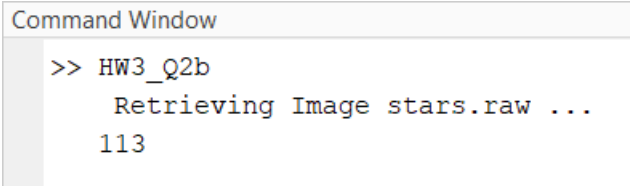


Fig.2.20

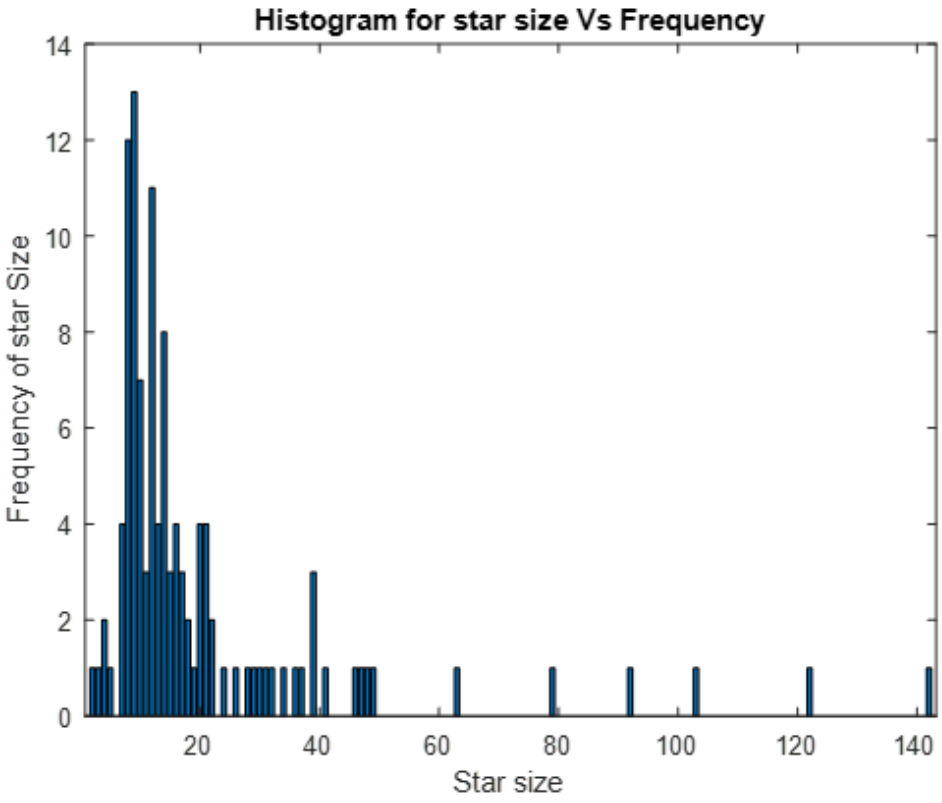

Histogram for star size Vs Frequency
Frequency of star Size
Star size
0
2
4
6
8
10
12
14
20
40
60
80
100
120
140

Fig.2.21

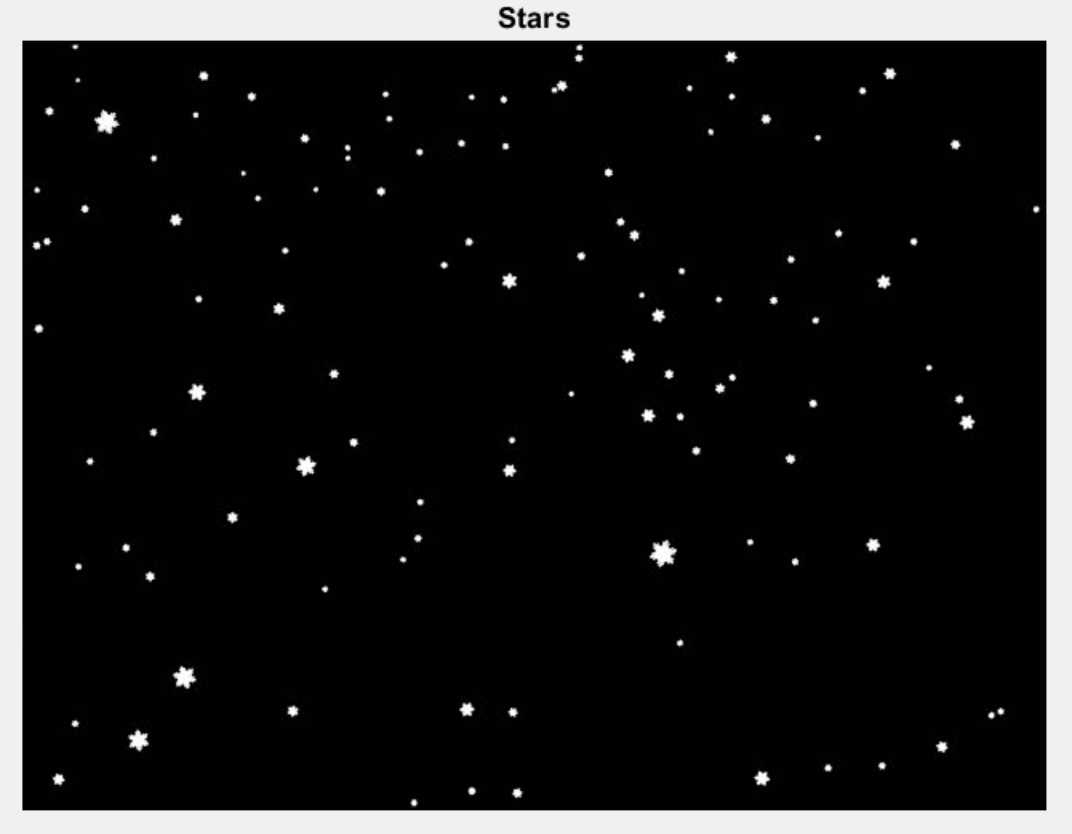


Fig.2.22

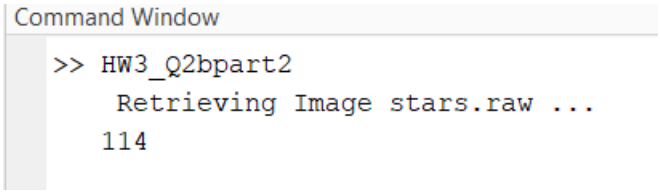


Fig.2.23

4. DISCUSSION

Applying the morphological operations for counting games of stars we arrive at the following result:

ANSWERS:

1) The total number of stars obtained in the image by the process of Shrinking is **113.**
2) The different types of stars as well as the frequency of these star sizes present in the image can be analyzed from the plot of the histogram of the star size with respect to the frequency.

3) Other than the morphological operations, the other method which can be implemented to do the above both parts is **Connected Component Analysis** which is very well explained in detail in the Approach and Procedure section.

(c) PCB Analysis:

1. ABSTRACT & MOTIVATION

The morphological image processing is a collection of non-linear operations related to either the shape or morphology of features in an image. Morphological operations generally depend on the relative ordering of the pixel values, rather than their intensity values and therefore they are specifically adaptable for the processing of binary images. Basically, the mathematically the morphology is used to extract the image components that are useful in the representation and description of the region shapes. It also helps in image segmentation. In PCB analysis in order to find out the number of holes in the PCB and the total number of pathways present in the PCB we use the two approaches here too respectively which are as follows:

a. Shrinking Technique
b. Connected Component analysis

One of the applications of the morphological image processing is it can be used for pre as well as post image processing techniques.

2. APPROACH & PROCEDURE

For counting the total number of holes as well as the connected pathways we follow the similar approach as implemented for the previous question. We will implement the Shrinking algorithm as well as Connected Component Analysis which is other than the morphological operations to count the total number of holes and the connected pathways respectively. The approach and the procedure for both the different implementations can be described as follows:

1) Shrinking Algorithm:

It is basically the process which erases the black pixels such that an object without the holes erodes to a single pixel at or near its center of mass, and an object with holes erodes to a connected ring lying midway between each hole and its nearest outer boundary.

Step 1: Initially, we load the raw input image on which the morphological operations are to be performed by correctly specifying its dimensions.

Step 2: As the input image is 24-bit RGB image, so for performing the morphological operations we first convert the RGB into the grayscale image and then we binarize it by thresholding an input image.

Step 3: Next, we extend the boundary of an input raw image by padding it along the four edges of an input image with background pixels, which will act as a new input image.

Step 4: Next, we declare the matrix M and G as zeros matrix which will be of the same size as that of the new input image.

Step 5: Then we initialize the temporary matrix as G_old equal to G for comparing it after each iteration to check the flag condition until there are no further erasures and the operations become idempotent.

Step 6: After the above step we run the two FOR loops for rows and columns i.e., for i = 2:N+1 and for j = N+1, to check whether the center pixel of the new input image, F(i,j) is equal to 1, then if its equal to 1 then its eight neighboring pixels X0, X1, X2, X3, X4, X5, X6, X7 which are stored in an array are compared with the standard conditional mask array generated from the lookup tables of Shrinking. If there is match, then the M(i,j) is set to 1, otherwise it is set to 0.

Step 7: Repeating this iteratively will generate M matrix which will act as an input for the second stage i.e., unconditional stage. Following the similar approach as that for the conditional logic, here also we run two FOR loops for rows and columns i.e., for i = 2:N+1 and for j = N+1 to set the G matrix. G(i,j) will be set only if the following condition is satisfied:

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

Where, P(M,M0,….,M7) are the unconditional masks for the shrinking. It will be set to 1 only if there is a HIT by comapring it with the neighboring pixels such as M0, M1,….., M7 from the lookup table.

Step 8: Then we compare that whether the G matrix generated is equal to the G_old matrix, if the condition is false, then it is flagged as zero and F matrix which the new input image matrix is equated with G as (F=G) and also the G_old is stored as G and then the M matrix is flushed out and declared as again zeros matrix for next iteration. This process is repeated untill the iteration when the condition is flagged as true and then the while loop is exited and that is final morphed output for Shrinking.

Step 9: In this step we display the final output morphed image as G matrix.

Step 10: Now, we initialize the counter as zero. After doing that we run the two FOR loops, one for the rows and the other for the columns and then using IF condition i.e., if G(i,j) is equal to 1 and the remaining 8 neighbors are 0 in the shrinked image output then increment the counter which will in turn give the count of the total number of holes in an input image of PCB. As we know from the definition of Shrinking that when the shrinking operation

performed on holes of PCB it will shrink all the holes to single dots thereby giving the count of the total number of holes in the image of PCB.

2) **Connected Component Analysis:**

Connected Component Analysis is basically an algorithmic application of the graph theory, where the subsets of connected components are uniquely labeled based on a given heuristic. Connected component analysis is used in computer vision to detect connected regions in binary digital images.
Connected Component Analysis can be precisely defined as an operator whose input image is a binary image and the output image is a symbolic image in which the label assigned to each pixel is an integer uniquely identifying the connected component to which that pixel belongs to.

Algorithm: (8 – connectivity based)

Here, we will be implementing the **two-pass algorithm** which is relatively easy to understand and simple to implement. The two-pass algorithm basically iterates through the 2-dimensional binary data. This algorithm basically makes two passes over the entire image. The first pass is carried out to assign the temporary labels and record the equivalences and then the second pass is performed to replace the temporary label by the minimum value of the label of its equivalence class.

The First Pass:

Step 1: At first, we assign the binary values to the pixels of an image i.e., 0 to the background pixel and 1 to the foreground pixel. Then the Connected Component matrix is initialized to the size of the image matrix.
Step 2: Then we iterate through each pixel of the column followed by the iteration of rows which is nothing but known as Raster Scanning (left to right and top to bottom) on the given input image.
Step 3: A label is initialized and then incremented for every detected object in an image. A counter is also initialized in order to count the total number of different objects present in an image.
Step 4: While performing Raster Scanning, check whether the pixel is foreground or background pixel.
Step 5: If the encountered pixel is foreground pixel then check for the neighboring pixel values i.e., *north* and *west* (for 4-connectivity) and *northeast, north, northwest* and *west* (for 8-connectivity) for the given region criterion (i.e., to check for the intensity value of 1).
Step 6: If there are none of the neighbors that satisfy the criterion then uniquely assign the label to the current pixel element and then increment the counter. Else if, one of the

neighbors satisfies the criterion then assign the current label to that pixel. If multiple neighbors match and all belong to the same region then assign the current label to their region. If multiple neighbors match but belong to the different region then find the neighbor with the smallest label and then assign it to the current pixel element.
Step 7: Store the equivalence between the neighboring labels.

The Second Pass:

Step 1: Again, we iterate through each pixel of the column followed by the iteration of rows which is nothing but known as Raster Scanning (left to right and top to bottom).
Step2: While performing Raster Scanning, check whether the pixel is foreground or background pixel.
Step 3: If the encountered pixel is foreground pixel then now relabel the pixel element with the lowest equivalent label.

The run time of the algorithm will depend on the size of the image and the amount of foreground pixels. The time complexity is comparable to the two-pass algorithm if the foreground pixels cover a significant part of the image. Otherwise, the time complexity is lower. However, the memory access is less structured than for the two-pass algorithm, which tends to increase the run time in practical situations.

3. EXPERIMENTAL RESULTS

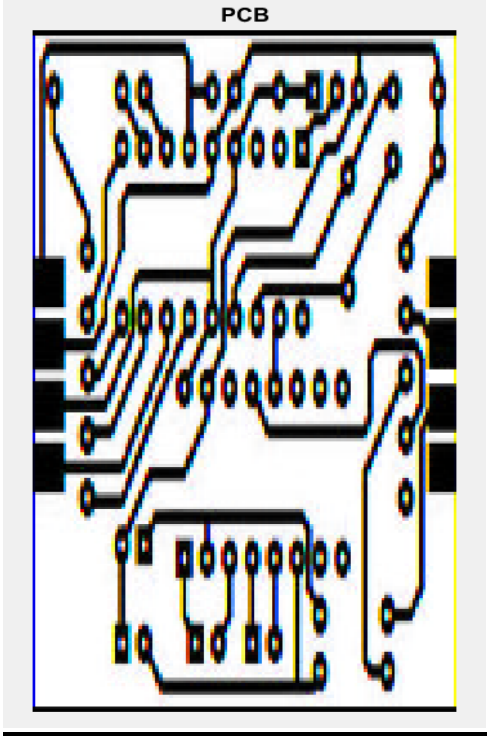


Fig. 2.24

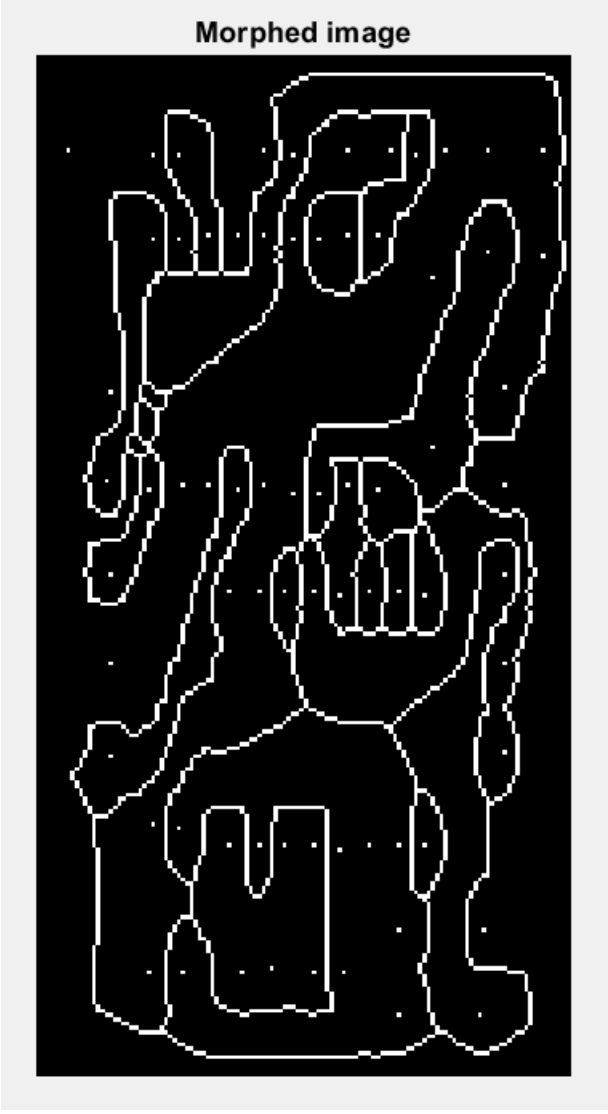


Fig. 2.25

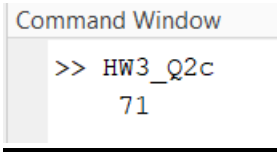


Fig. 2.26

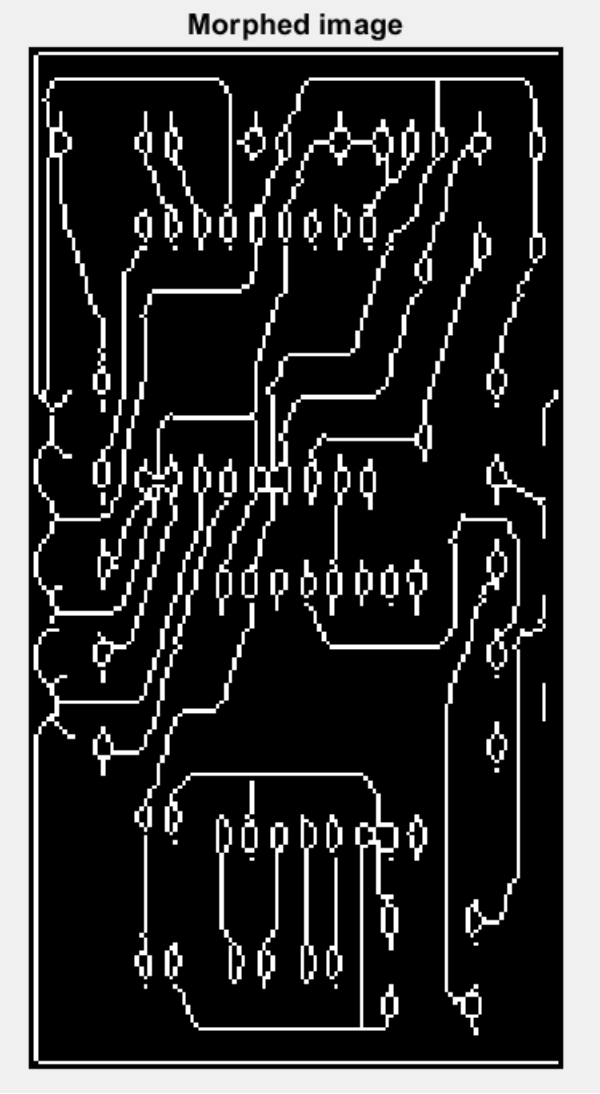


Fig. 2.27

4. DISCUSSION

An image processing algorithm that uses morphological and logical operations to find the features of the PCB (Printed Circuit Board) is implemented and based on that the answers to the questions is discussed below:

i. There are in total **71** holes present in the given PCB which was obtained by applying the **shrinking technique** which is discussed in detail in the Approach and Procedure section.

ii. The total number of pathways present in the given PCB is **25** excluding the solder pads which is obtained by the **Connected Component Analysis** technique which is also described in detail in the Approach and Procedure section.

(d) Defect Detection:

1. ABSTRACT & MOTIVATION

Morphological image processing is a type of processing in which the spatial form or structure of objects within an image are modified. Binary images usually contain various types of imperfections. In general, binary regions produced by simple thresholding are distorted by noise and texture. So, the one of the primary goals of morphological operations is to remove the distortions which affects the shape and texture of images. So, defect detection is one of the applications of the morphological image processing techniques.

2. APPROACH & PROCEDURE

As the given Gear-tooth image has two missing teeth at different positions and upon visualization technically there are 12 teeth uniformly distributed around the cylinder of a normal gear for the Gear-tooth image. So, if we divide the 360° by 12, we see that each tooth of the Gear should be present at 30°. So, sweeping in anticlockwise direction we can make out that the angles at which the teeth of the Gear are missing are at 210° and 360° i.e., at the lower left and the right positions of the wheel. So, here I have designed morphological algorithm for detecting the missing tooth of a gear.
Basically, here first I am reconstructing the given input image which will have all the 12 teeth and then I will subtract the reconstructed image from the given input image (with missing 2 teeth) in order to find the location of the 2 missing teeth in the defected input image of Gear-tooth.
The algorithm for the implementation can be described as follows:

Step 1: Initially, we load the raw input image on which the morphological operations are to be performed by correctly specifying its dimensions.

Step 2: As the input image is 8-bit grayscale image, so for performing the morphological operations we first binarize the image by thresholding an input image.

Step 3: Next, we extend the boundary of an input raw image by padding it along the four edges of an input image with background pixels, which will act as a new input image.

Step 4: In this step I horizontally divide the input image into half keeping just only the upper half of the image and store it in one variable (F_half) and then rotate (F_half) by 180° in the anticlockwise direction in order to preserve the orientation and store it as another variable (J).

Step 5: Next vertically concatenate the (F_half) and (J) in the form of new image as (C) variable.

Step 6: In this step I vertically divide the image (C) into half keeping just only the left half of the image and store it in one variable (C_half) and then flip (C_half) in the circular fashion i.e., from left to right and store it as another variable (K).

Step 7: Next again flip (K) in circular fashion i.e., from bottom to top and store it as another variable (L).

Step 8: Then fuse the image obtained in (K) and (L) and store it in the variable as (M).

Step 9: Next flip (M) in circular fashion i.e., from right to left and store it as another variable (N).

Step 10: Here, in this step horizontally concatenate (N) and (M) and store it in variable as (O) thereby getting the reconstructed image of the teeth of the Gear. As the image generated in the (O) variable is the fused image, which is RGB, so it has to be converted into grayscale image.

Step 11: Next as the morphological operations can be performed only on the binary images so we will now binarize the grayscale image using thresholding technique.

Step 12: Now, we perform the dilation operation and fill in the inner four black holes of the Gear as they are of no importance in finding the missing teeth of the Gear. This dilation operation is performed on the reconstructed as well as the original defected image.

Step 13: Lastly, we subtract the defected image from the reconstructed image in order to find the position of the missing teeth of the Gear.

3. EXPERIMENTAL RESULTS

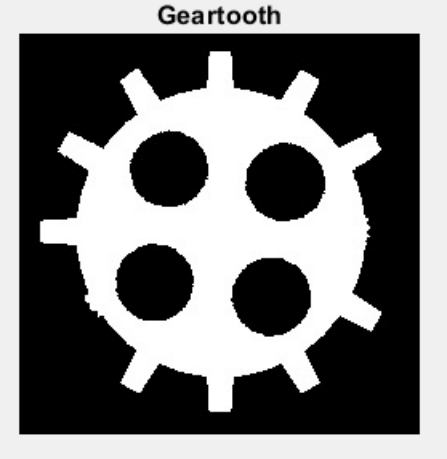


Fig. 2.28

Fig. 2.29

Fig. 2.30

Fig. 2.31

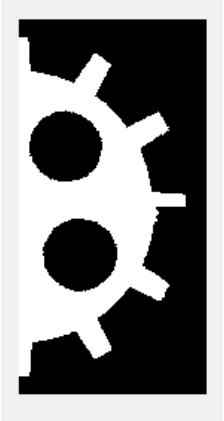


Fig. 2.32

Fig. 2.33

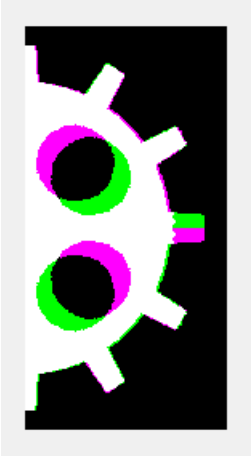


Fig. 2.34

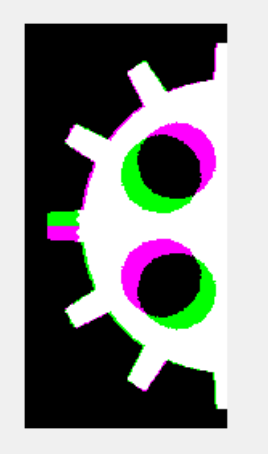


Fig. 2.35

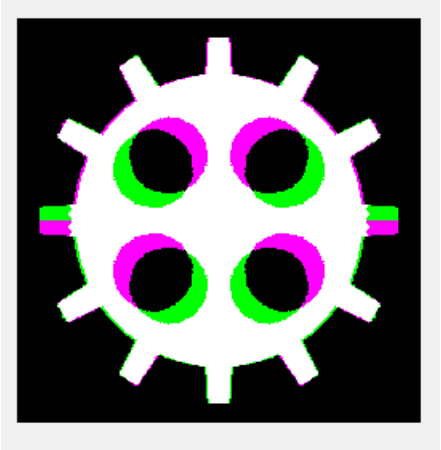


Fig. 2.36

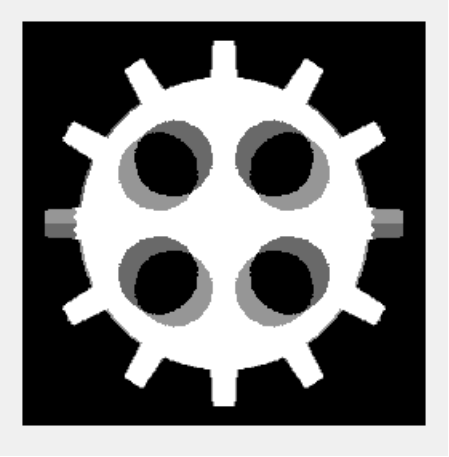


Fig. 2.37

Fig. 2.38

Fig. 2.39

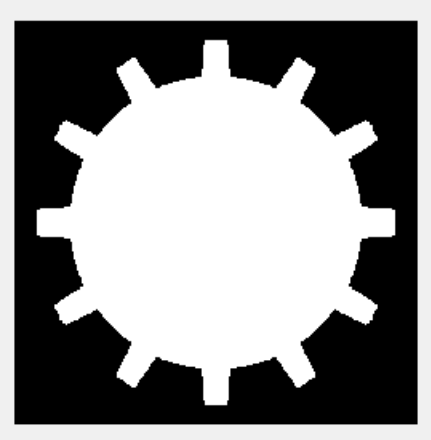


Fig. 2.40

Fig. 2.41

4. DISCUSSION

In this question I have followed a different approach to design a morphological processing algorithm for automatic missing teeth detection other than the one stated in the Hint section of the question. The entire flow of the algorithm is discussed in detail in the approach and the procedure section as well as the results of each stage is attached in the result section.

References:

1. http://www.cse.iitd.ernet.in/~pkalra/csl783/Morphological.pdf
2. https://homepages.inf.ed.ac.uk/rbf/HIPR2/hitmiss.htm
3. William K. Pratt: Digital Image Processing, 4th Edition, John Wiley & Sons Inc., 2007. (ISBN 9780471767770).
4. DigitalImageProcessing-Gonzalez_org.pdf
5. https://medium.com/analytics-vidhya/image-stitching-with-opencv-and-python-1ebd9e0a6d78
6. http://mathproofs.blogspot.com/2005/07/mapping-square-to-circle.html
7. https://arxiv.org/ftp/arxiv/papers/1509/1509.06344.pdf
8. http://squircular.blogspot.com/2015/09/mapping-circle-to-square.html
9. https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm
10. https://en.wikipedia.org/wiki/Image_stitching
11. https://www.pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/
12. https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/
13. https://medium.com/analytics-vidhya/image-stitching-with-opencv-and-python-1ebd9e0a6d78
14. https://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html
15. https://courses.engr.illinois.edu/cs498dwh/fa2010/lectures/Lecture%2017%20-%20Photo%20Stitching.pdf
16. https://towardsdatascience.com/image-panorama-stitching-with-opencv-2402bde6b46c
17. https://math.stackexchange.com/questions/494238/how-to-compute-homography-matrix-h-from-corresponding-points-2d-2d-planar-homog
18. https://en.wikipedia.org/wiki/Homography_(computer_vision)
19. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
20. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

21. https://www.researchgate.net/figure/Flow-Chart-for-SURF-Feature-Detection_fig2_318565053
22. https://www.worldcat.org/wcpa/servlet/DCARead?standardNo=0818689447&standardNoType=1&excerpt=true
23. https://en.wikipedia.org/wiki/Connected-component_labeling
24. https://www.xarg.org/2017/07/how-to-map-a-square-to-a-circle/