EE 569: Digital Image Processing
Homework #3

By
Brinal Chimanlal Bheda
USC ID - 8470655102
bbheda@usc.edu
Spring 2019

Issue date: 02/13/2019
Submission Date: 03/03/2019

Problem 1: Geometric Modification

(a) Geometric Transformation

I. Abstract and Motivation

Geometric transformations are widely used for image registration and the removal of geometric distortion. Common applications include construction of mosaics, geographical mapping, stereo and video.

Geometrical image modification consists of a set of most common operations performed in image processing. Operations such as rotation, translation and scaling of the images spatially that provides the flexibility to produce a desired result. Apart from image enhancement and restoration, it is one of the important steps in image manipulation. They are extensively used in computer graphics, image morphing and panorama stitching. In this problem we will implement all the geometrical operations and utilize them in various situations.

It is sometimes desirable to manipulate an image for considerable distortion in the shape it represents. This can be achieved by properly utilizing the concept of geometrical warping. In essence, geometrical warping is a transformation that alters the spatial configuration of an image. This is mainly used for creative purposes. In this problem we will look into the implementation and discuss a warping technique using triangles to achieve such effects.

Geometrical image modification also finds application in hole filling problems. With sufficient knowledge about the location and orientation of holes and its corresponding patches, we can implement a hole filling algorithm by heavily utilizing the geometrical image operations.

Geometric image transformation is related to the field of computer graphics. It shows the manipulation of the image in the co-ordinate domain. There are various operations that can be performed on the image like shifting, rotating, shrinking, enlarging, warping by manipulating the co-ordinates of the pixels. We need to know the relationship between image co-ordinates and Cartesian co-ordinates. We adopt the reverse address mapping where the output co-ordinates have to be integers while input co-ordinates can be fractional numbers.

a. Translation:

$Xk = Uq + tx$

$Yj = Vp + ty$

where, (tx, ty) = translational vector

b. Scaling with respect to origin

$Xk = sx*Uq$

$Yj = sy*Vp$

c. Rotation

$$\begin{bmatrix} Xk \\ Yj \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} Uq \\ Vp \end{bmatrix}$$

Compound operation: Perform first translation (tx, ty), second scaling (sx, sy) and then rotation θ

$$\begin{bmatrix} Xk \\ Yj \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} Uq + tx \\ Vp + ty \end{bmatrix}$$

But convert into matrix form i.e. affine system shown below,

$$\begin{bmatrix} Xk \\ Yj \\ 1 \end{bmatrix} = \begin{bmatrix} c1 & c2 & c0 \\ d1 & d2 & d0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Uq \\ Vp \\ 1 \end{bmatrix}$$

Translation matrix

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling matrix

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation matrix

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For example, rotate an image by θ about a pivot point (Xc, Yc) in image, we use the following equation

$$\begin{bmatrix} Xk \\ Yj \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Xc \\ 0 & 1 & Yc \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -Xc \\ 0 & 1 & -Yc \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Uq \\ Vp \\ 1 \end{bmatrix}$$

There can be various approaches for getting the different outputs by using dividing the image into triangles approach or by using polar co-ordinates i.e. r not θ.

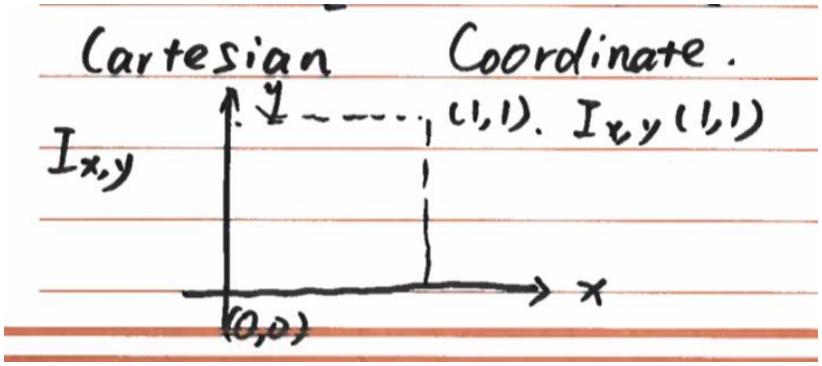
These were Linear or Affine warping. Now, we consider the Polynomial warping. This includes two types – Pincushion distortion and Barrel distortion.

(Source: Discussion Lecture dated 02/26/2019)

1) Scaling
   Sx = 2, Sy = 2
   For forward mapping, we have input image in terms of u and v and the target output image in x and y, thus dealing with 2D matrices.

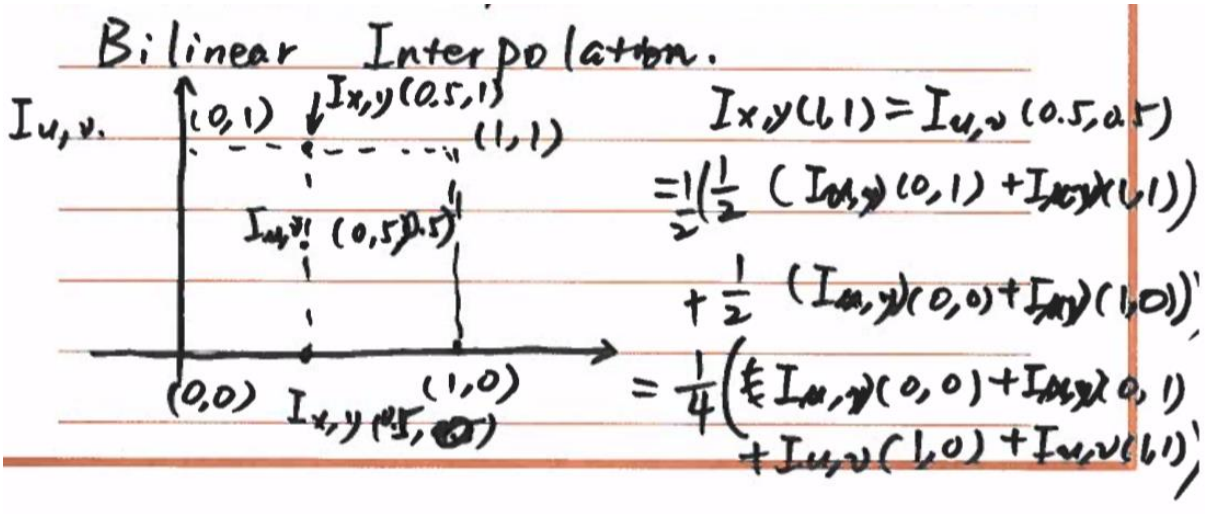
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

   We carry out this procedure in the cartesian coordinate as we doing the problem in linear algebra so cartesian coordinate is better than the image coordinate.



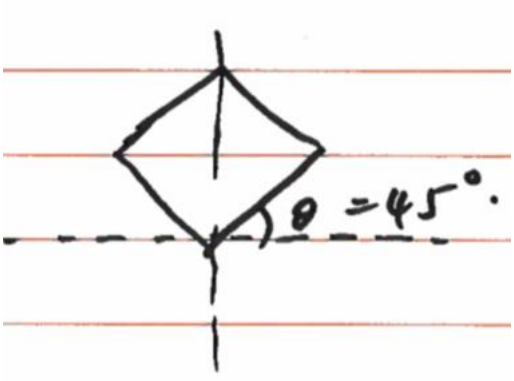
   Now for the reverse address mapping, we are not doing any geometric modification on the pixel value itself, we are calculating the pixel position values. We manipulate the pixel values when using filters, but not in geometric transformation. We perform inverse matrix calculation.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



u and v values may not be integer values and hence we use bilinear interpolation. In HDR (High Dynamic Range or high resolution) image, the low resolution image is given and we need high resolution image for display, this technique interpolates and generates imaginary pixels trying to expand it to higher resolutions.

2) Rotation
The image given a little tilted.



Forward mapping:

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

$$= \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

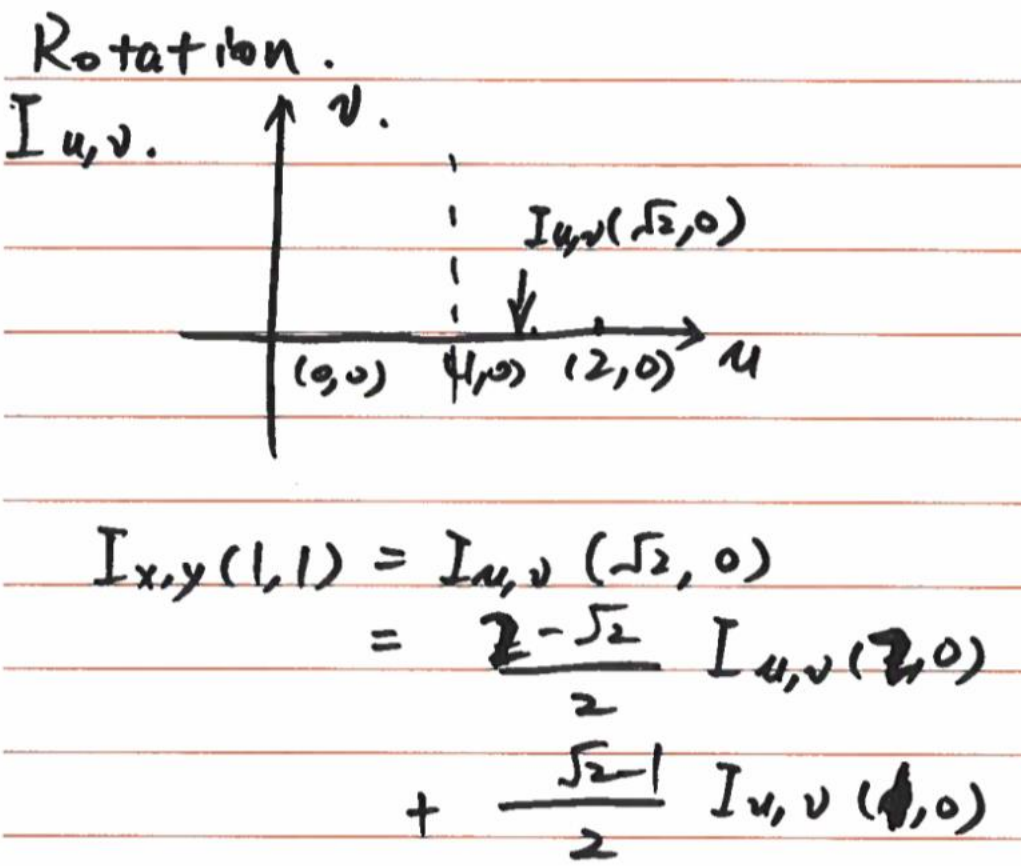
Reverse address mapping:
u,v is the image before modification, x,y is the image after modification.
The equation shows the reverse mapping from (x,y) to (u,v).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

$$= \begin{bmatrix} \sqrt{2} \\ 0 \\ 1 \end{bmatrix}$$

$$u = \sqrt{2}, \quad v = 0$$

Another example:



(Source: https://www.cis.rit.edu/class/simg782/lectures/lecture_02/lec782_05_02.pdf)

II. Approach and Procedure

Three sub-images "lighthouse1.raw" ~ "lighthouse3.raw" were cut out of an image of a scene containing a lighthouse. The three holes in the image "lighthouse.raw" are all of size 256x256. Each sub-image is from one of the hole in the image, but with different scales and orientations. We have to fill the holes in "lighthouse.raw". Hence, I wrote a program to implement a geometric transformation algorithm to create the filled image of size 512x512.

One possible way to do it is:
1) Find the coordinates of corners in each sub-image denoted by *Fi* [*x*,*y*], *i=1,2,3*.
2) Design a generic geometric transform (*qx*, *qy*) = Ξ*i* (*px*, *py*) which maps point (*px*, *py*) to point (*qx*, *qy*) in the *i*-th sub-image. Here, you need to combine one or more translation, rotation and scaling operations together. After the generic geometric

transformation, the transformed i^{th} sub-image should be a square image with its sides aligned with the horizontal and vertical axes.

3) For scaling-up, implement the interpolation function $\Theta(\cdot)$ such that $\Theta(Fi[\Xi i\ (x, y)])$ is of size 160 x 160. Drop redundant pixels when scaling-down.

4) Find the coordinates of the holes (specifically the top-left corners).

5) Fill $\{\Theta(Fi[\Xi i\ (x, y)]),\ i = 1,2,3\}$ into the holes to get the final image. We assume "lighthouse1.raw" goes to the left hole, "lighthouse2.raw" goes to top, and "lighthouse3.raw" goes to bottom-right.

Bi-linear interpolation may be needed to generate the pixel value at fractional positions.

We prefer to use the Cartesian coordinate system to modify the image geometrically. This is because operations such as scaling, rotation and translation becomes very easy when an image is viewed on the Cartesian coordinates. Hence it becomes necessary for us to compute the Cartesian coordinates from the image coordinates and vice-versa. To make the equations linear, the input and the output coordinates are augmented by appending a '1' in the end.

When an image is initialized, the Cartesian to image and image to Cartesian is set such that it can map the values between Cartesian coordinates and image coordinates. The image to Cartesian coordinate relation is given as follows:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0.5 \\ -1 & 0 & h - 0.5 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} r \\ c \\ 1 \end{bmatrix}$$

In the above equation, the input image row coordinate 'r' and column coordinate 'c' are augmented and multiplied by a matrix to get 'x' and 'y' Cartesian coordinates. Here 'h' refers to the image height. By taking the inverse of the above matrix and multiplying it with the Cartesian coordinates, we can obtain the corresponding image coordinates. These matrices are calculated every time an image object is initialized.

(Source: http://www.bmva.org/bmvc/1988/avc-88-023.pdf - A combined corner and edge algorithm)

In this problem, we will fill holes in images with its corresponding 3 patches given using geometrical transformation. This is implemented by first finding the corners of the holes and patches by Harris corner algorithm.

The Harris corner algorithm checks for changes in the x and y direction in a window and classifies the center pixel as a corner if the changes in x and y direction are significant. This is achieved by computing the x and y derivative of the image. These derivatives were calculated by convolving the image with Sobel operator. The following operations were utilized to compute the x and y gradient of the grayscale image:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad ; \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

In the above equation, G_X is the gradient along the x direction and G_Y is the gradient along the y direction of the original image A. After computing the gradients, we pass a 3x3 window through the input and compute the values of M matrix which is given by:

$$M = \sum_{x,y} w(x,y) \times \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The M matrix contains essential information about x and y gradients in the window. Gradient value of each pixel I_X and I_Y in the 3x3 window is weighted by a weight function and summed up. The weight function can be a 2D Gaussian window or a box function whose value is 1. For the sake of simplicity we have considered the value of w(x, y) = 1 for all instances. Later the M matrix is used to measure the corner response value, R. The value of R is calculated by applying the following formula:

$$R = \det(M) - k\big(trace(M)\big)^2$$

The value of R is associated with the Eigen values of M. Higher the value of R, higher the Eigen values. Here k is an empirically determined value that is between 0.04-0.06. For this problem the value of k is chosen to be 0.04.

Sometimes due to noise or unsharp corners we may get multiple corner points near the true corner. This can be eliminated by non-max suppression. To implement non-max suppression, we store all the points in a hash map which maps the corner number to its coordinates and R value. We then loop through all the corners and find the indices which are within the 5 pixel radius of the current corner. For two or more

neighboring corners, we choose the one with the highest R value and neglect the others. This way we are more robust to noise and unsharp corners.

We calculate the transformation matrix using the 4 corners obtained from the patches and 4 corners from the target images. Using reverse mapping, we have mapped the patches into the desired target images.

(Source: Discussion Lecture dated 02/19/2019)

Geometric Modification - manipulation of pixel data

How to do forward and reverse mapping to get the geometric transformation of input image

Linear transformation ie linear mapping between the input image and desired output image

1. Scaling - linear multiplication using horizontal and vertical components
2. Rotation - using sin and cos, calculating angle
3. Translation - tx and ty

Order of these operations doesn't matter as it is linear, can change order of implementation of all three

a. Use reverse address mapping because if use forward, ie multiply with input vector we do not get whole image

say in case of scaling up, there are holes in output image

hence, do reverse order - use a blank image as output image where fit in all the details pixel by pixel

b. bilinear interpolation - fit in all the details pixel by pixel

3 images for 3 holes to fill - realign them and rotate them to the correct position and translate them from one position to place in original image

a. scan the input image line by line to find the location of holes

b. find 4 co-ordinates of pixel locations for the holes - find the corners, rectangular shape → calculate width, height and angle of the image and find the central point

c. apply reverse mapping trick - multiply all the 3 matrices of processes

from output pixel value location go to corresponding input pixel location and carry the value and transfer it to output image, however

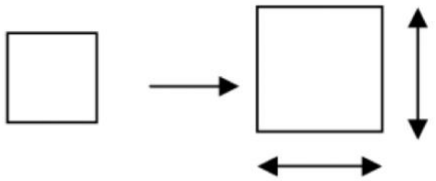
sometimes the values same around it, so take averaging at the output pixel location

Patch image is given with tilted angles and rectangular shape given - so first locate 4 corners, scan it from left to right and define some patterns to locate the corners
Top and bottom corners have different patterns as compared to right and left corners

Make use of geometric property of images
For matrix multiplication use tools like python numpy package or MATLAB
Matrix multiplication is used a lot in deep learning - TensorFlow, Keras, PyTorch

The process of scaling, rotation and transformation, reverse address mapping along with the equations is shown below:

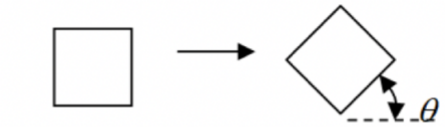
- Geometric Modification:
  - Scaling



$$x = s_x u$$
$$y = s_y v$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

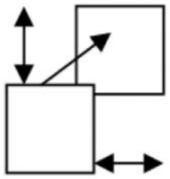
- Geometric Modification:
  - Scaling
  - Rotation



$$x = u\cos\theta - v\sin\theta$$
$$y = u\sin\theta + v\cos\theta$$

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

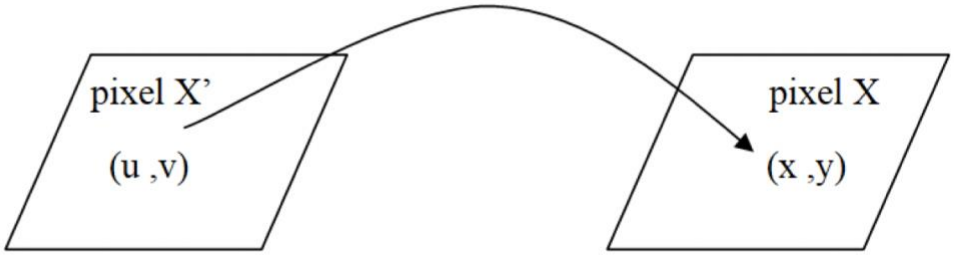
- Geometric Modification:
  - Scaling
  - Rotation
  - Translation



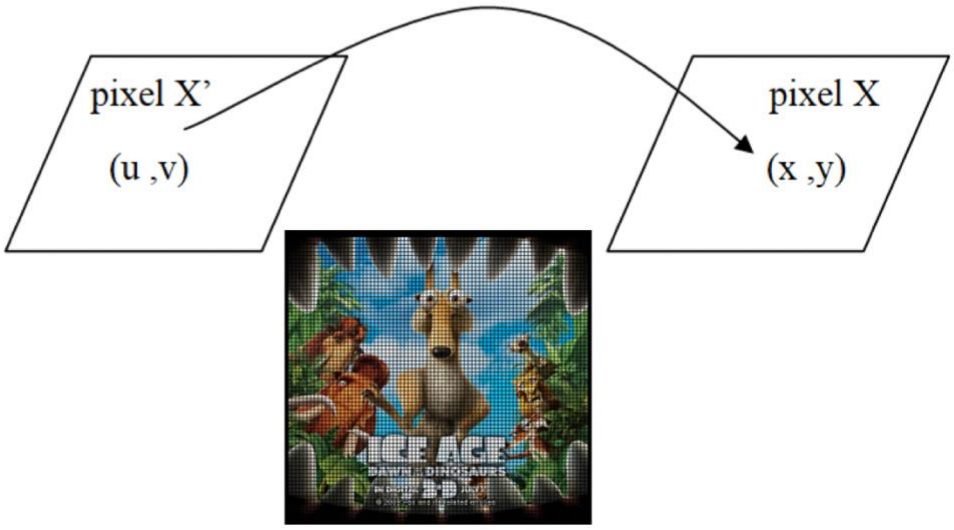
$$x = u + t_x$$
$$y = v + t_y$$

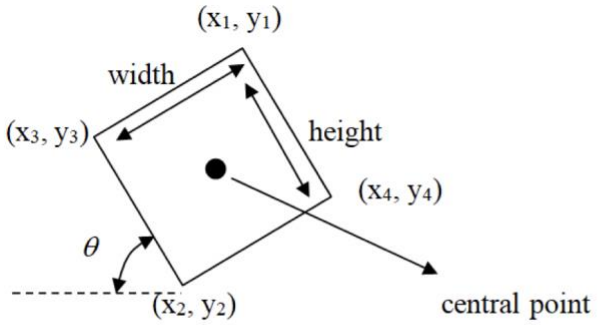
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Reverse address mapping



- Reverse address mapping



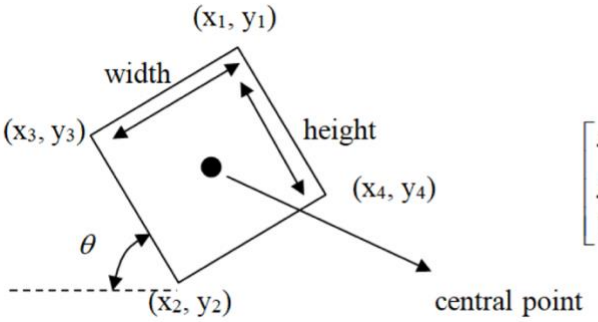


$$width = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}$$

$$height = \sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2}$$

$$\theta = \arctan(\frac{y_2 - y_3}{x_2 - x_3})$$

$$center \quad point = \left(\frac{x_3 + x_4}{2}, \frac{y_3 + y_4}{2}\right) or \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$



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

The problem of fractional indices has been solved by using bilinear interpolation. Reverse mapping results into less distortion as it is 1:1 mapping since it keeps relationship between pixels horizontally next to each other. But there is one drawback- it creates some distortion around top and bottom boundary when used in

forward mapping. After the points are mapped, we use bilinear interpolation to minimize pixel error by:

Let $\mathbf{I}$ be an $R \times C$ image.
We want to resize $\mathbf{I}$ to $R' \times C'$.
Call the new image $\mathbf{J}$.

Let $s_R = R / R'$ and $s_C = C / C'$.

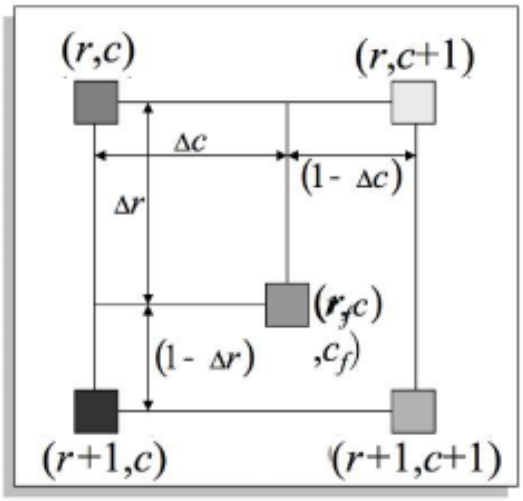
Let $r_f = r' \cdot s_R$ for $r' = 1, \ldots, R'$

and $c_f = c' \cdot s_C$ for $c' = 1, \ldots, C'$.

Let $r = \lfloor r_f \rfloor$ and $c = \lfloor c_f \rfloor$.

Let $\Delta r = r_f - r$ and $\Delta c = c_f - c$.

Then $$\begin{aligned}\mathbf{J}(r',c') = {} & \mathbf{I}(r,c)\cdot(1-\Delta r)\cdot(1-\Delta c) \\ & + \mathbf{I}(r+1,c)\cdot\Delta r\cdot(1-\Delta c) \\ & + \mathbf{I}(r,c+1)\cdot(1-\Delta r)\cdot\Delta c \\ & + \mathbf{I}(r+1,c+1)\cdot\Delta r\cdot\Delta c.\end{aligned}$$



Bilinear Interpolation Algorithm

Bilinear interpolation was utilized when needed to obtain smooth images.

(Source - https://www.codementor.io/tips/7814203938/resize-an-image-with-bilinear- interpolation-without-imresize)

Looking at the output produced below by implementing the hole filling algorithm for the lighthouse problem, we can see that the corners have been successfully detected by Harris corner algorithm. Upon closer inspection of the output, we can find sharp edges near the boundaries of the 3 holes. This is mainly because of little color variation in the lighthouse patches and the target lighthouse image. Thus I applied the median filter of a suitable window size (3x3) across the boundaries to smoothen it. The median filter procedure I used was the same as I had implemented it in the HW#1 assignment second problem. Thus, we can see and compare the output transition for the same below in the results.

In geometrical manipulation in image processing, we usually manipulate the pixel location and address in Cartesian coordinates. So, we need to translate the image coordinates to Cartesian coordinates at first. Suppose (u, v) denotes the Cartesian coordinates and (p, q) denotes the image coordinates. After translating the input image to Cartesian coordinates, based on the desired effect, we can find the

transformation matrix (address mapping) form the input image to output image in Cartesian coordinates. Then translate the output image form Cartesian coordinates to image coordinates. One important thing in geometrical manipulation is when we find the transformation matrix, we need to do reverse address mapping. If we do the forward mapping, the corresponding pixel address in output image may be fractional, so we can't decide which pixel it indicates. Using reverse address mapping, when we get the fractional address in input image, we can use bilinear interpolation to find the pixel value for the corresponding output image pixel.

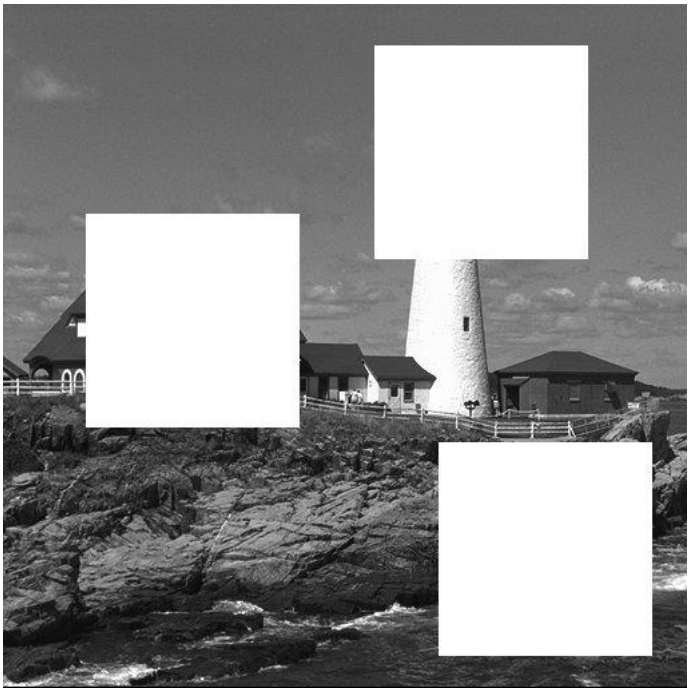
Using the equation:

$$\begin{bmatrix} Px \\ Py \\ 1 \end{bmatrix} = \begin{bmatrix} a1 & a2 & a3 \\ b1 & b2 & b3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} qx \\ qy \\ 1 \end{bmatrix}$$

Define a boundary matching score (e.g. MSE or sum of absolute value, etc.) along the sides of the modified filling images to the sides of the holes. Then rotate the images until you get the optimal matching score.

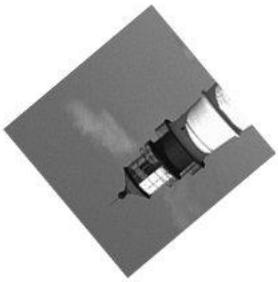
Another method which we can also use is the triangle to triangle approach to solve this problem where we divide the patches into 2 triangles and perform the scaling, rotation and transformation and reverse mapping operations. However, I have not implemented this method.

III. Experimental Results



Input image – lighthouse.raw

Input image – lighthouse1.raw



Input image – lighthouse2.raw

Input image – lighthouse3.raw

I found the positions of the 3 top left corner pixels of holes in the lighthouse image in my code in the form of (row, column) which are listed as follows:

```
Brinals-MacBook-Pro:~ brinalbheda$ \
> top left corner pixel of hole A = 31, 278 \
> top left corner pixel of hole B = 157, 62 \
> top left corner pixel of hole C = 327, 325
```

Intermediate output

Output image after improvement

IV. Discussion

Various geometrical image operations were implemented in this problem. One operation that needs significant improvement is rotation. Due to rotation, few original pixels may be lost as the presence of fractional image coordinates is inevitable. As a result of this the image gets blurry. we can see the decrease in quality due to rotation. To a certain extent, bilinear interpolation helps to preserve the quality. To further improve on rotation operation, we can implement rotation by shearing.

Shape of output image looks little distorted and we can observe distortion at the corners of the 3 lighthouse pieces added to the input image because forward mapping is not one to one mapping. Main reason being we have discrete image. For continuous images, transformation transforms every pixel into new position i.e. we can find original pixel for every pixel in transformed image. Thus, we can recover all pixels while doing reverse pixels. Because we are dealing with discrete images, we might not be able to recover all the pixels which introduces distortion. We lose information when we transform line into shorter line.

Looking at the output produced by the hole filling algorithm for the lighthouse problem, we can see that the corners have been successfully detected by Harris corner algorithm. Upon closer inspection of the output, we can find sharp edges near the boundaries of the 3 holes. This is mainly because of little color variation in the lighthouse patches and the target lighthouse image. This can be resolved by applying median filter of a suitable window size across the boundaries to smoothen it.

The output produced for house matching on the lighthouse image that is the lighthouse 1 patch doesn't look satisfactory. A lot of improvements can be done by increasing the sharpness of the lighthouse 1 house piece of image. The blurry house piece was the result of poor interpolation. A sharpening filter is applied over the house piece before merging it with the target lighthouse image.

Additionally, four coordinates were used to find 6 parameters of the matrix. This resulted in an overdetermined system. Least squares approach was used to minimize the mapping error and if we closely observe, the mapping is still one to one.

(b) Spatial Warping

I. Abstract and Motivation

Geometric transformation is a method in Image Processing, in which image pixel locations are modified and not the actual intensity levels. The basic concept used in geometrical warping is that the image is projected on another surface in order to view the image in a different perspective.

There are various applications of Geometrical Image Warping. Some of them are:

- To create special effects in an image
- To register/combine two images taken at the same scene at different times
- To morph one image into another

This concept is mainly used in photo editors, panoramas, designing logos, computer vision applications.
Warping is a process that describes the destination (x,y) using the source (u,v). A spatial warping technique is used. In this technique, points in one image are mapped

to points in another image. If the process is injective, reverse mapping is not possible, but if the process is bijective, reverse mapping of the points is possible.

We can use the spatial warping technique to create fantastic images. We can warp a square-shaped image (on the left) into another shape (on the right) as shown below.

(a) square-shaped image (b) warped image

Figure 2. Image warping

We develop a spatial warping algorithm as the one demonstrated in Figure 2 and apply it to the 512 x 512 gray scale image "hat.raw" to obtain a warped image. The height of each arc in Figure 2(b) is 128, and the tip of the arc should be located at the center line of the image. Please indicate the reference points to be used and derive the reverse address mapping function that converts Figure 2(a) to (b).

In geometrical manipulation in image processing, we usually manipulate the pixel location and address in Cartesian coordinates. So, we need to translate the image coordinates to Cartesian coordinates at first. Suppose (u, v) denotes the Cartesian coordinates and (p, q) denotes the image coordinates. After translating the input image to Cartesian coordinates, based on the desired effect, we can find the transformation matrix (address mapping) form the input image to output image in Cartesian coordinates. Then translate the output image form Cartesian coordinates to image coordinates. One important thing in geometrical manipulation is when we find the transformation matrix, we need to do reverse address mapping. If we do the forward mapping, the corresponding pixel address in output image may be fractional, so we can't decide which pixel it indicates. Using reverse address mapping, when we get the fractional address in input image, we can use bilinear interpolation to find the pixel value for the corresponding output image pixel.

(Source:http://graphics.cs.cmu.edu/courses/15-463/2011_fall/Lectures/morphing.pdf)

(ρ, θ) polar co-ordinates based image transformation. $\rho \max = 1$. Say $\theta = 0$, 45, 90… take the triangle formed and repeat the procedure throughout the input image. Change the corresponding new ρ. Also, use symmetry.

II. Approach and Procedure

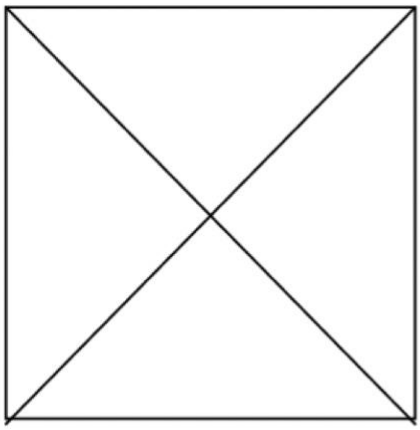
Geometrical Warping given in the question is warping of the square image to an image as shown in the figure 2. The corner points and the center of the image remain at the same location. But the mid points in the vertical and horizontal sides of the image are mapped to a different location in the form of an arc whose height is 128. This kind of Geometrical Image Modification is given by Radial Warping or by using polar coordinates.

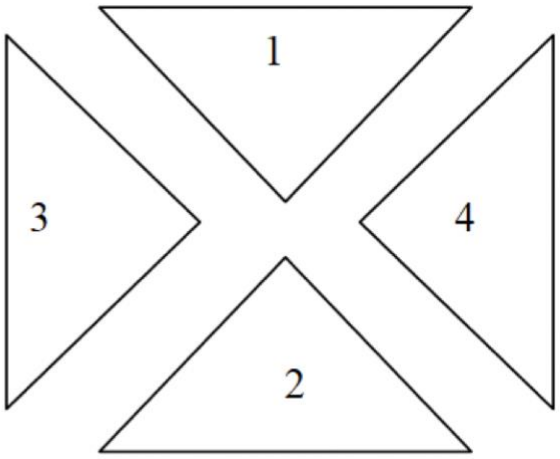
(Source: Discussion Lecture dated 02/19/2019)

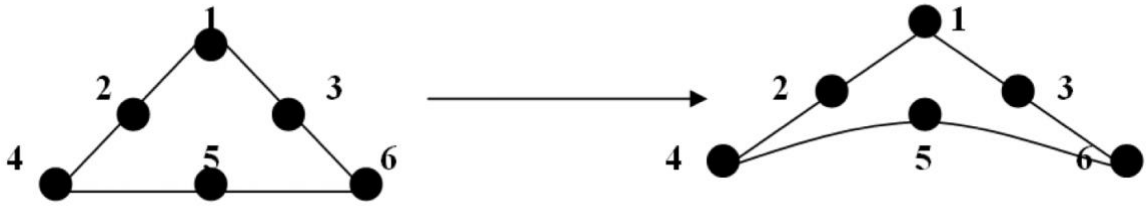
Output is non-linear operation of input
Find more number of points here as compared to 1a → these are the anchor points which are used to find the coefficients
6 number of unknowns of A and B each
So to solve find 6 points for output to input and find all the coefficients
Rest operations same as 1a

Trick is we have one output image given
Divide rectangular shape into 4 triangular regions, each triangle is mapped to a warped image
The curved edge is center pixel value of the side of the image - 128 (height by which the sides of the image is pulled to curve inside)
This is done using anchor points - each point at 3 edges and each center point of 3 sides → 6 points
Use these 6 points to calculate the coefficients of A and B

In mapping, we want to map every point to the output image with minimal distortion as much as possible.







- Spatial warping

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix}$$

- Geometric Modification:
  - Spatial warping

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} u_0 & u_1 & u_2 & u_3 & u_4 & u_5 \\ v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ y_0 & y_1 & y_2 & y_3 & y_4 & y_5 \\ x_0^2 & x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \\ x_0y_0 & x_1y_1 & x_2y_2 & x_3y_3 & x_4y_4 & x_5y_5 \\ y_0^2 & y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 \end{bmatrix}^{-1}$$

Algorithm implemented in C++:

- Read the input image “hat.raw” whose dimensions are height_size = 512, width_size = 512, BytesPerPixel = 1
- Consider the 4 triangles in the square input image and calculate the three vertices and the mid-points of each side of each of the 4 triangles, thus we get 6 values corresponding to the 4 triangles each
- Each triangle is mapped to a warped image, the curved edge is center pixel value of the side of the image subtract by 128 which forms the arc
- Use anchor points, each point at 3 edges and each center point of 3 sides
- Use the above 6 points to calculate the coefficients of A and B using the long formula shown above
- Calculate u and v for every pixel using the forward elliptical warping formula
- Retrace u and v pixel locations to 0 to 512
- Ouput the Forward warping pixel
- Using bilinear interpolation, find the pixel intensity and copy the value in output image.
- Then map all these points to the warped output image
- Repeat the procedure for the other 3 triangles for finding the coefficients of A and B
- Check for the values lie inside the image and put conditions to limit the values in the range >0 and <512
- Calculate xnew and ynew for every u,v using Reserve Address Mapping formula
- Retrace xnew and ynew to 0 to 512
- Output the Reverse warping pixel
- Map all these values obtained to the warped image

- Write the computed image data array on output.raw file using the fwrite() function

Reverse mapping results into less distortion as it is 1:1 mapping since it keeps relationship between pixels horizontally next to each other. But there is one drawback- it creates some distortion around top and bottom boundary when used in forward mapping. After the points are mapped, we use bilinear interpolation to minimize pixel error by:

Let $\mathbf{I}$ be an $R \times C$ image.
We want to resize $\mathbf{I}$ to $R' \times C'$.
Call the new image $\mathbf{J}$.

Let $s_R = R / R'$ and $s_C = C / C'$.

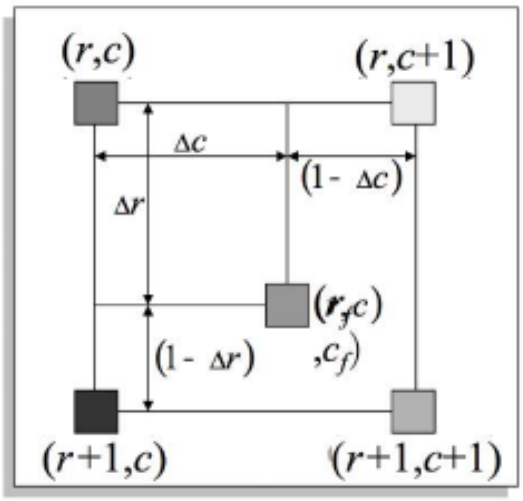
Let $r_f = r' \cdot s_R$ for $r' = 1, \ldots, R'$

and $c_f = c' \cdot s_C$ for $c' = 1, \ldots, C'$.

Let $r = \lfloor r_f \rfloor$ and $c = \lfloor c_f \rfloor$.

Let $\Delta r = r_f - r$ and $\Delta c = c_f - c$.

Then $\mathbf{J}(r', c') = \mathbf{I}(r, c) \cdot (1 - \Delta r) \cdot (1 - \Delta c)$
$+ \mathbf{I}(r+1, c) \cdot \Delta r \cdot (1 - \Delta c)$
$+ \mathbf{I}(r, c+1) \cdot (1 - \Delta r) \cdot \Delta c$
$+ \mathbf{I}(r+1, c+1) \cdot \Delta r \cdot \Delta c.$

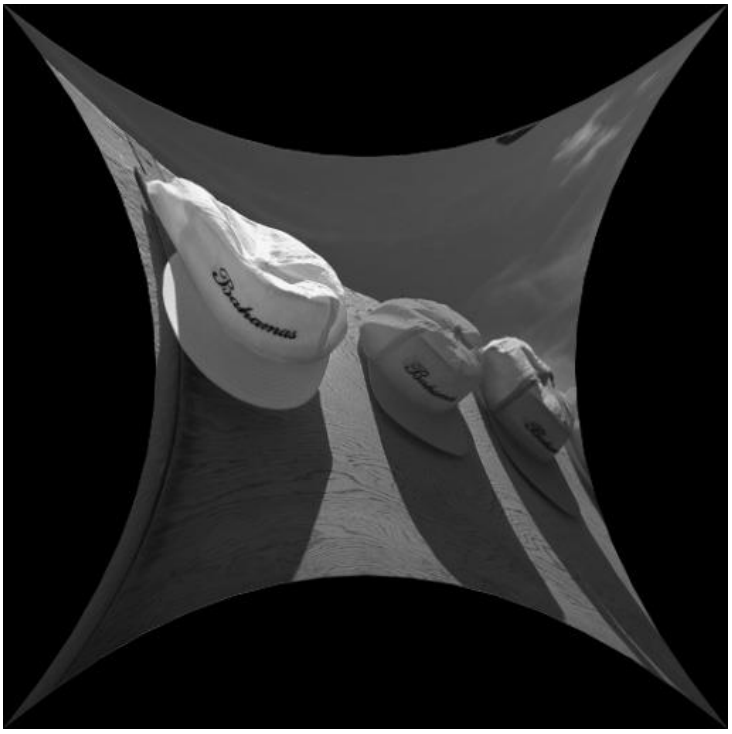


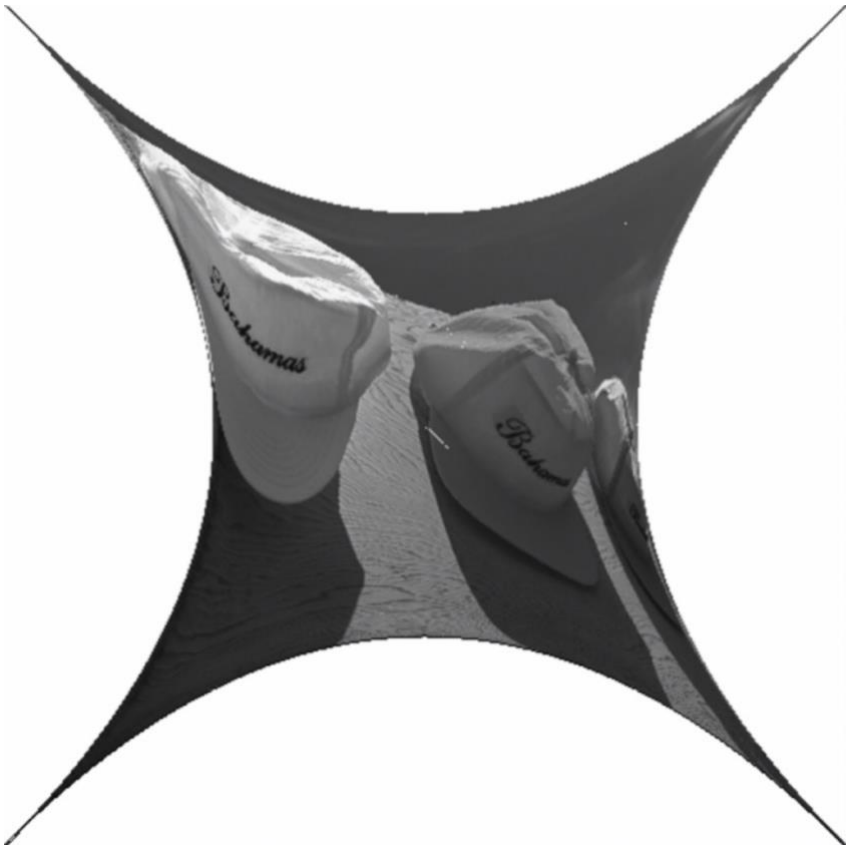
Bilinear Interpolation Algorithm

(Source - https://www.codementor.io/tips/7814203938/resize-an-image-with-bilinear- interpolation-without-imresize)

III. Experimental Results

Input image – hat.raw





Final Output image – hatSpatialWrapping.raw

Reverse mapping the output image above

IV. Discussion

In geometrical image warping, the pixel locations are changed and not the actual pixel intensity values itself. These changes in pixel locations are considered to be one-to-one mapping. Since this mapping of pixels are one-to-one mapping, they can be reversed back to original pixel locations from output pixel locations.

In the output produced by the spatial warping algorithm in figure 2, we can observe that the mapping is one to one. The shape of the output image looks distorted as desired. Upon closer inspection of the output images, we find that a lot of original pixels are lost. This is mainly because we are under sampling the original image.

Shape of output image looks little distorted and we can observe distortion at the corners because forward mapping is not one to one mapping. Main reason being we have discrete image. For continuous images, transformation transforms every pixel into new position i.e. we can find original pixel for every pixel in transformed image. Thus, we can recover all pixels while doing reverse pixels. Because we are dealing with discrete images, we might not be able to recover all the pixels which introduces distortion. We lose information when we transform line into shorter line.

The geometrical warping to convert from square image to warped image and the reverse mapping to convert from warped image to square image was performed. The outputs from the reverse mapping resembled the original image and no distortion was obtained in them. This is because rounding of the mapped co-ordinates was done. The mapping from square to warped and vice versa definitely gives an artistic look to the image.

(c) Lens Distortion Correction

I. Abstract and Motivation

(Source: https://www.sciencedirect.com/science/article/pii/S0030399209002370)

Camera lens distortion calibration is the first step in resolving any metric application with a camera. To date, lens distortion was corrected using some existing lens distortion non-metric or self-calibration methods. Using a lens distortion model

means defining a global rule to correct the entire image. This global rule does not take into account particular lens distortion effects not represented by the model. Moreover, to calibrate the model, only some features of the scene such as straight lines, circles or vanishing points are used. Since only the feature of the scene used to calibrate the model is guaranteed by the distortion rectification, it is certain that the model will not be precise. The result is an approximation of the real image distortion.

To improve the lens distortion rectification, a method without using a model is proposed. Using a set of control points distributed across the entire image, they are corrected to assure all the restrictions of the scene. With both sets of points, the points detected in the image and the undistorted ones, image local transformations are defined considering only nearby control points. Rather than calibrating a global model, local functions are characterized. The distortion correction is defined by a rectification surface composed of local surface patches each influenced by nearby control points. This method is more sensitive to local deformations and allows the image to be corrected in accordance with its distortion.

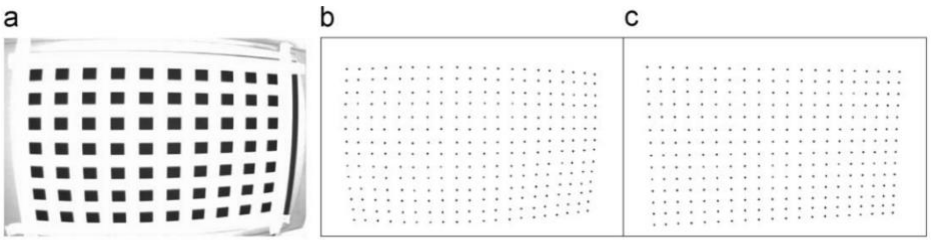


Fig. 4. (a) Image of the chessboard taken with a low-cost industrial camera VCM50. (b) Detected points in the distorted image. (c) Corrected points positions according to the cross-ratio invariability and straight line restrictions. Lens distortion is calibrated mapping the distorted point positions to their right positions.

In computer graphics, we have the camera – imaging model, which transforms and projects the 3D terrain model into 2D image plane. (X, Y, Z) are the world co-ordinates and (x,y) are the projected image co-ordinates, f is the focal length of the lens.

$$x = \frac{fX}{f - Z}$$

$$y = \frac{fY}{f - Z}$$

From (X, Y, Z) → (x,y), this is many to one mapping. Suppose Z is given, we can find,

$$X = \frac{xi}{f}(f - Z)$$

$$Y = \frac{yi}{f}(f - Z)$$

In perspective transformation, we try to map a vector in the 3D space to the 2D co-ordinate space.

$$v = s \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad w = \begin{bmatrix} xi \\ yi \\ zi \\ 1 \end{bmatrix}$$

P = perspective transformation matrix

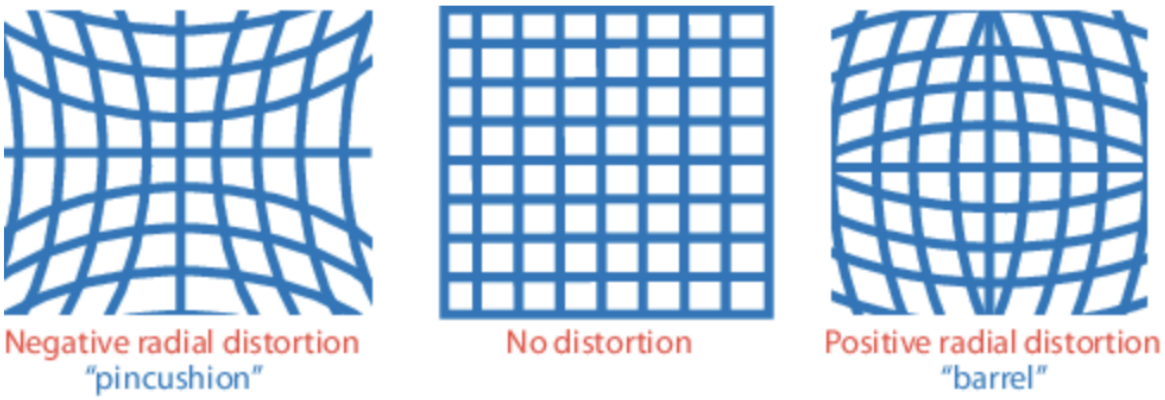
$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{f} & 1 \end{bmatrix}$$

$$w = P\,v = \begin{bmatrix} sX \\ sY \\ sZ \\ s - s\frac{Z}{f} \end{bmatrix}$$

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & Cx \\ 0 & f & Cy \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & tx \\ r21 & r22 & r23 & ty \\ r31 & r32 & r33 & tz \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

scaling factor — K: intrinsic camera matrix — [R t]: extrinsic camera matrix

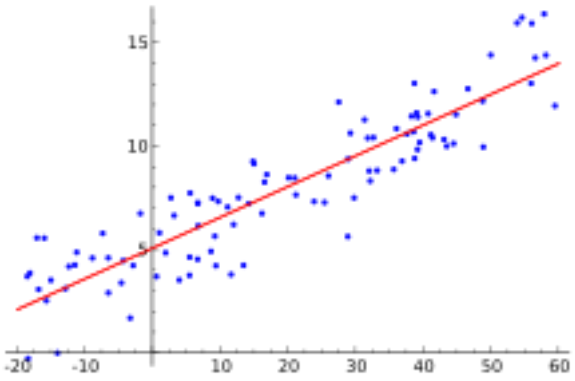
The camera parameters object stores the intrinsic, extrinsic, and lens distortion parameters of a camera.



(Source: https://en.wikipedia.org/wiki/Linear_regression)

Linear Regression:

Linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).



In linear regression, the model specification is that the dependent variable, y_i is a linear combination of the *parameters* (but need not be linear in the *independent variables*). For example, in simple linear regression for modeling n data points there is one independent variable: x_i, and two parameters, β_0 and β_1:

straight line: $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \ldots, n.$

In multiple linear regression, there are several independent variables or functions of independent variables.

Adding a term in x_i^2 to the preceding regression gives:

parabola: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, \ i = 1, \ldots, n.$

This is still linear regression; although the expression on the right hand side is quadratic in the independent variable x_i, it is linear in the parameters β_0, β_1 and β_2.

In both cases, ε_i is an error term and the subscript i indexes a particular observation.

Returning our attention to the straight line case: Given a random sample from the population, we estimate the population parameters and obtain the sample linear regression model:

$$\hat{y}_i = \widehat{\beta}_0 + \widehat{\beta}_1 x_i.$$

The residual, $e_i = y_i - \hat{y}_i$, is the difference between the value of the dependent variable predicted by the model, $\hat{y}_i$, and the true value of the dependent variable, y_i. One method of estimation is ordinary least squares. This method obtains parameter estimates that minimize the sum of squared residuals, SSR:

$$SSR = \sum_{i=1}^{n} e_i^2.$$

Minimization of this function results in a set of normal equations, a set of simultaneous linear equations in the parameters, which are solved to yield the parameter estimators, $\widehat{\beta}_0, \widehat{\beta}_1$.

In the case of simple regression, the formulas for the least squares estimates are

$$\widehat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \text{ and } \widehat{\beta}_0 = \bar{y} - \widehat{\beta}_1 \bar{x}$$

where $\bar{x}$ is the mean (average) of the x values and $\bar{y}$ is the mean of the y values.

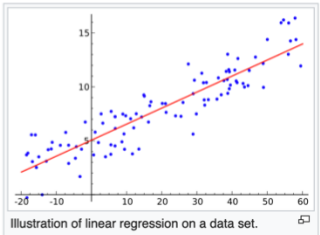


Illustration of linear regression on a data set.

Under the assumption that the population error term has a constant variance, the estimate of that variance is given by:

$$\hat{\sigma}_\varepsilon^2 = \frac{SSR}{n-2}.$$

This is called the mean square error (MSE) of the regression. The denominator is the sample size reduced by the number of model parameters estimated from the same data, $(n-p)$ for p regressors or $(n-p-1)$ if an intercept is used.[21] In this case, $p = 1$ so the denominator is $n-2$.

The standard errors of the parameter estimates are given by

$$\hat{\sigma}_{\beta_1} = \hat{\sigma}_\varepsilon \sqrt{\frac{1}{\sum (x_i - \bar{x})^2}}$$

$$\hat{\sigma}_{\beta_0} = \hat{\sigma}_\varepsilon \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum (x_i - \bar{x})^2}} = \hat{\sigma}_{\beta_1} \sqrt{\frac{\sum x_i^2}{n}}.$$

Under the further assumption that the population error term is normally distributed, the researcher can use these estimated standard errors to create confidence intervals and conduct hypothesis tests about the population parameters.

- Perform simple linear regression using the operator.
- Use correlation analysis to determine whether two quantities are related to justify fitting the data.
- Fit a linear model to the data.
- Evaluate the goodness of fit by plotting residuals and looking for patterns.
- Calculate measures of goodness of fit R^2 and adjusted R^2

II. Approach and Procedure

Solution 1: using camera co-ordinate formula and Linear Regression

(Source: Discussion Lecture dated 02/19/2019 and 02/26/2019)

Machine learning related problem
Advanced technique for finding the inverse function
Most of the real-world problem does not have a perfect forward and inverse transformation
So output is not one-to-one mapping of input
And hence there is no exact inverse function to find

So find the best approximate inverse function using machine learning based technique
Using linear regression
We provide the forward transformation function

Since lens is circular, when image captured, center part is good but the corners are tilted so there is lens distortion
Solve the lens distortion, remove it here just like the smart phone engineers
Using the forward mapping function, along with given distorted image, use forward function to calculate all the pixel positions
Calculate xd and yd - distorted positions of pixels; x and y - real world positions
Plot the graph for the above two in the 3d world and find the line passing using linear regression method which has minimum mean square error to all the points in this path

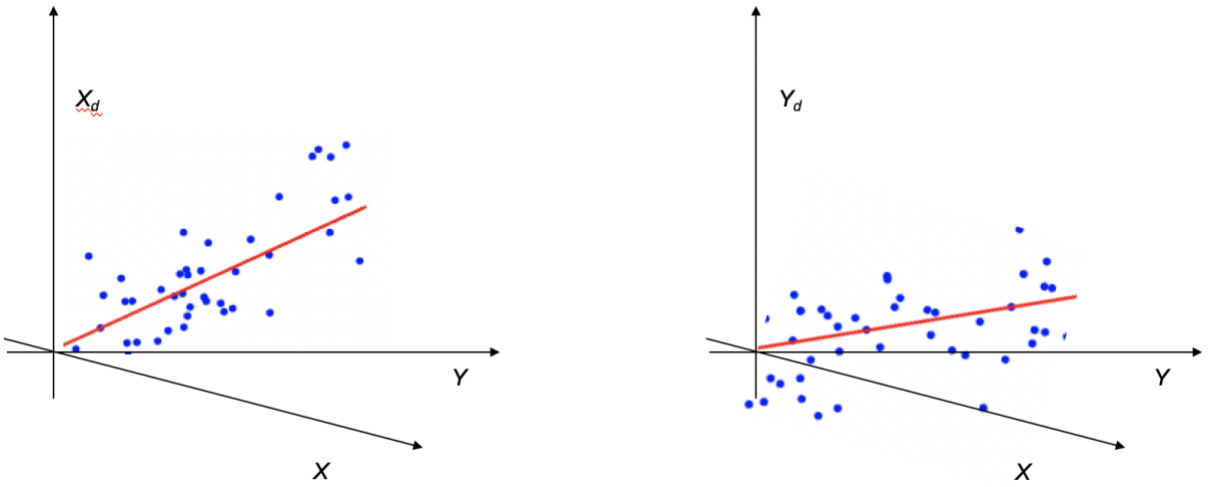
The line passes through in 3d space for the two points considered and has minimum distance ie distance is calculated using the Euclidean distance and technique is called L2 norm

L2 norm also known as MSE and try to minimize the mse (mean square error)

Linear regression package to use: Python scipy → it is fast
Partial differentiation - closed form solution but more computing power used and hence very expensive
For Mac, use stochastic gradient descent ie sgd based linear regression → optimal closed form solution ie iterative approach

Got 3 triplets - one with xd and one with yd
The two lines in 3d space are defined by the co-efficients for the inverse function
The forward function is non-linear and the inverse function to use is linear

Linear Regression
Find alpha and beta co-efficients
N is hyperparameter - determine according to you what gives optimal solution



$$X_d = \boldsymbol{\alpha}\begin{bmatrix} X \\ Y \end{bmatrix} + \boldsymbol{\varepsilon}_1, Y_d = \boldsymbol{\beta}\begin{bmatrix} X \\ Y \end{bmatrix} + \boldsymbol{\varepsilon}_2$$

$$\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_N], \boldsymbol{\beta} = [\beta_1, \dots, \beta_N]$$

Radial distortion often happens when an image is captured on a non-flat camera's focal plane. The relationship between the actual image and its distortion in the camera coordinate system is as follows:

$$x_d = x\,(1+k_1r^2+k_2r^4+k_3r^6)$$
$$y_d = y\,(1+k_1r^2+k_2r^4+k_3r^6)$$

where x, y are undistorted pixel locations, K1, K2, K3 are called radial distortion coefficients of the lens, and $r^2 = x^2 + y^2$. The x, y, xd, yd are defined in camera coordinate system, to get these values from a given digitized image, you need to convert the pixel locations from the image coordinate (u, v) to the camera coordinate (x, y) as follows:

$$x = \frac{u - u_c}{f_x}$$
$$y = \frac{v - v_c}{f_y}$$

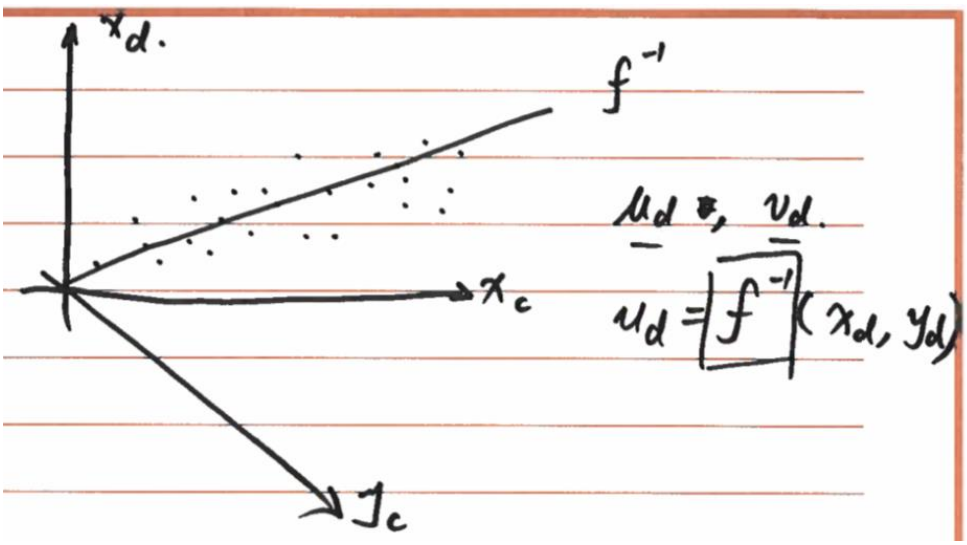
where (uc, vc) is the center of the image, fx and fy are the scaling factors. To recover the undistorted image, the (x, y), given the (xd, yd), the key lies on finding the inverse function so that:

$$x = f(x_d,y_d)$$
$$y = g(x_d,y_d)$$

However, there is no exact inverse function for xd and yd equations since the forward mapping from (x, y) to (xd, yd) is not linear. A common solution is first to project (x, y) to (xd, yd) which ends up in triplets of (x, y, xd) and (x, y, yd). We then use linear regression (refer to https://www.mathworks.com/help/matlab/data_analysis/linear-regression.html for further readings about linear regression) to find the best approximate inverse functions.

Given the classroom distorted image. Given that K1 = -0.3536, K2 = 0.1730, K3 = 0, fx = fy = 600, I implement the aforementioned method to correct the distortion. I have used third party tools (MATLAB, Python Scipy, etc.) for the linear regression part.

Given image is a distorted version of image, and forward mapping is given, do not have undistorted image, use the same trick as the above two questions. To distort again using forward mapping, to see what if the real-world image is the input image and can use the forward mapping to generate another distorted version. Given the forward mapping, we can find a bunch of triplets and put so many dots in the 3D space.



In the 3D space shown above, find and draw a line to fit through all these dots and this line is defined as the inverse mapping. Using reverse address mapping, find the distorted u and distorted v. The formula is given above, and the objective is to find the inverse function by trying to approximate it. This can be done by linear regression or the Newton method.

The function xd and yd equations have non-linear terms and especially the first and second terms cause a huge impact and thus using linear regression may not give very great results, may still have some distorted part in image.

Solution 2: using Linear approximation with Gauss-Newton method

(Source: https://ieeexplore-ieee-org.libproxy1.usc.edu/document/7361326 - Lens distortion correction method based on linear approximation)

Aiming at the influence of lens distortion to imaging measurement system, this paper puts forward an improved method for lens distortion correction based on linear approximation. Firstly, the method solves the radial distortion coefficient linearly based on the principle of cross-ratio invariance under perspective projection. Then a higher precision of the straight line is obtained according to the orthogonal distance fitting. Finally, a comprehensive distortion index function is built to get the distortion coefficient via Gauss-Newton method. Simulation results in the paper show that the average correcting error of this method is less than 0.2 pixel, in the case of 800 control points and 0.1 noise standard deviation. With the characteristic of easy to implement, strong robustness and high precision, the method proposed in this paper is suitable for high precision correction in machine vision.

The precision of correction results largely determines the accuracy of the whole system. At first, the method solves the radial distortion coefficient linearly based on the principle of cross-ratio invariance. Then we fit a straight line based on orthogonal distance by using revised points, and it has made line accuracy significantly increased. Finally, the Gauss-Newton method is used to optimize the comprehensive distortion index function including both radial and tangential distortion. The proposed method achieves high precision, low complexity and strong robustness.

According to pin-hole camera imaging theory, one point in 3D space is projected as an ideal image point (xu, yu) and the corresponding distorted point is denoted as (xd, yd). The distortion center is expressed as (u, v) and the distance from distorted point to distortion center is r. Generally, the lens distortion model can be described as:

$$x_u = x_d + x(k_1 r^2 + k_2 r^4) + (2k_3 xy + k_4(r^2 + 2x^2))$$
$$y_u = y_d + y(k_1 r^2 + k_2 r^4) + (k_3(r^2 + 2y^2) + 2k_4 xy)$$

Where x = xd−u, y = yd−v, r2 = x2+y2, k1 and k2 are the coefficients of the radial distortion. k3 and k4 are coefficients of the tangential distortion. For general application system, radial distortion coefficients k1 and k2 can fully describe the imaging distortion.

Too many nonlinear distortion parameters will cause a great amount of calculation and increase the complexity of the nonlinear optimization algorithm. However, in the industrial machine vision identification system, radial distortion only is insufficient to meet the demand of high precision applications. Incorporating the accuracy and complexity, we adopt four coefficients to correct the image distortion, both radial distortion k1, k2 and tangential distortion k3, k4.

A. Cross-Ratio Invariance

Four points A, B, C, D are lying on the same line in 3D space and the corresponding ideal image points are recorded as (xua, yua, zua), (xub, yub, zub), (xuc, yuc, zuc), (xud, yud, zud). Thus, there exits the following equation based on the cross-ratio invariability:

$$\frac{(x_{ua} - x_{uc})(x_{ub} - x_{ud})}{(x_{ub} - x_{ud})(x_{ua} - x_{ud})} = CR$$
$$\frac{(y_{ua} - y_{uc})(y_{ub} - y_{ud})}{(y_{ub} - y_{uc})(y_{ua} - y_{ud})} = CR$$

However, the lens distortion will cause the detected point to deviate from the ideal position, so we must replace the actual detected points with the ideal points in the above equations. In the situation of only taking k1 into consideration, we can still utilize equations for xu and yu to get the ideal coordinates:

$$x_u = x_d + x(1 + k_1 r^2), y_u = y_d + y(1 + k_1 r^2)$$

Where x = xd−u, y = yd−v, r2 = x2+y2. Then k1 can be obtained according to equations above with only four ideal points. In order to improve the precision of k1, we can get the points close to image edge which have severely distorted element and use multiple sets of data to participate in the calculation.

B. Orthogonal Distance Fitting

To minimize the influence of singular points to fitting precision, we correct the image primarily by k1, and then use the adjusted data for high precision fitting. In view of the inter-pixel gap that occurs during image correction, bilinear interpolation algorithm is adopted to fill up gaps. Meanwhile, for the purpose of extracting high precision control points, Harris sub-pixel corner detection algorithm is used to extract corrected points. Let us suppose that the parameter of the i-th straight line is represent as $(ai, bi, ci)^{\boldsymbol{T}}$, and the distance dij from point (xj, yj) to its corresponding line can be expressed as:

$$d_{ij} = |a_i x_j + b_i y_j + c_i| \Big/ \sqrt{a_i^2 + b_i^2}$$

Parameter of the line can be solved by minimizing the square sum of the orthogonal distance, we can get parameters of a, b and c via the following expression:

$$\begin{cases} a = Q - \sqrt{Q^2 + 4P^2}/2P, b = 1 & if \quad P > 0 \\ a = b = 1 & if \quad P = Q = 0 \\ b = Q + \sqrt{Q^2 + 4P^2}/2P, \ a = 1 & if \quad P < 0 \end{cases}$$

$$c = -\frac{1}{N}(b\sum_{j=1}^{N} y_j + a\sum_{j=1}^{N} x_j)$$

Where P and Q are expressed as follows:

$$P = \sum_{j=1}^{N} x_j y_j - \frac{1}{N}\sum_{j=1}^{N} x_j \sum_{j=1}^{N} y_j$$

$$Q = \sum_{j=1}^{N} x_j^2 - \sum_{j=1}^{N} y_j^2 - \frac{1}{N}(\sum_{j=1}^{N} x_j)^2 + \frac{1}{N}(\sum_{j=1}^{N} y_j)^2$$

Where (xj, yj) is the corrected point by k1, N is the number of points in each line.

C. Comprehensive Distortion Index Function

For analysis of the difference between the corrected line and the ideal line, a comprehensive distortion index function is built to evaluate the result of the correction.

Assume that there are M lines and N points in each line. Actual detected points are set as Dij(x_{ij}^d, y_{ij}^d), and we can acquire the corresponding ideal points Uij(xuij,yuij). M straight lines can be obtained with the orthogonal distance method, and the distance from ideal point to the fitting line is expressed as |Uij−Li|. Finally, the following complete objective function arises:

$$\min_{k_1,k_2,k_3,k_4} F = \sum_{i=1}^{M}\sum_{j=1}^{N} |\mathrm{U}_{ij} - L_i|$$

where 0<i≤M, 0<j≤N. The Gauss-Newton method is selected for nonlinear optimization. Parameter k1 that plays a main role for distortion has already obtained the appropriate initial value, and positive solutions for the remaining three parameters itself tend to zero, so the Gauss-Newton method can reach faster iteration

speed near the solution. It can also be very effective to avoid ill-conditioned coefficient matrix by improving data processing accuracy of rounding errors, thus it can achieve the perfect optimization results.

Solution 3:

(Source: https://www.sciencedirect.com/science/article/pii/S0030399209002370 - Correcting non-linear lens distortion in cameras without using a model)

Using the paper mentioned above, I used a method for lens distortion correction that uses all the geometric restrictions that are included in a standard chessboard pattern. They are straight lines, parallelism and perpendicularities. Since a chessboard is going to be used in the pin-hole calibration process, we propose using this information prior to calibrating the lens distortion. In this way, the camera calibration should be understood as a two-step procedure where first the camera lens distortion is computed and corrected and, second, the pin-hole camera model is calibrated. With the distortion correction method proposed in this paper, camera lens distortion can satisfy a number of geometric restrictions of the scene without the risk of instabilities of computing a mathematical model. It has been reported that including both the distortion center and the decentering coefficients in the non-linear optimization may lead to instabilities of the non-metric lens distortion estimation algorithm. To avoid this, some researchers used an exhaustive coarse-to-fine search for the distortion center around the image center but in the end, it has resulted in a prolonged search with no guarantees of stability. To avoid instabilities and obtain a perfect correction that includes all kinds of distortions (modelled and not modelled), the image is warped with a local transformation using the original set of points extracted from the image and the undistorted points, which accomplish all restrictions contained in the calibration template. Warping the image following no distortion model allows the handling of local deformations that do not respond to any distortion model. With this method, image warping has more degrees of freedom allowing the distortion to be corrected according to the particular deformation of each image region.

The proposed method relies on the idea that an image of a structure maintains its proportions according to a perspective projection. Therefore, there are different magnitudes within the structure that are fixed independent of the position, orientation and characteristics of the camera that takes the image. If the structure has parallel or orthogonal lines, these orthogonalities and parallelisms are kept fixed according to a perspective projection. What is proposed in this paper is a method for correcting the detected points in the image to fulfil restrictions given by the features

of the structure, and to warp the image using these two sets of points, the points detected in the image and the undistorted ones. From the point of view of the pin-hole model calibration, this method is very useful since fully structured templates such as chessboards are used to resolve them. This means that the same information can be used to calibrate the lens distortion before the camera is calibrated.

This paper proposes the bases of the image correction method. These are the geometric invariants of a structure, independent of perspective projection and how the location of the set of points in the image is corrected. It shows how the different camera lens distortion models do not satisfy the geometric restrictions of the scene independent of the calibration technique. It derives the image warping using both sets of points.

A new point of view for camera lens distortion rectification has been presented that improves the performance of existing lens distortion calibration methods. To date, lens distortion was corrected using some existing lens distortion non-metric or self-calibration methods. These methods use some features of the scene such as straight lines, circles or vanishing points to calibrate a lens distortion model. This means a global rule to correct the entire image, which guarantees only the feature of the scene used to calibrate the model, and which does not take into account particular lens distortion effects not represented by the chosen model. The result is an approximation of the real image distortion.

With this new point of view, any model is used to rectify the image. This allows defining a particular correction for each part of the image and characterizing the real distortion of the image. A set of control points extracted from the original distorted image is corrected to satisfy the maximum number of restrictions of the scene. In this case, an image of a chessboard is used as the template since it has a set of points that is arranged in straight lines, parallel at a fixed distance and perpendicular to each other. So, the distorted control points are corrected to satisfy these restrictions of the scene in the corrected image. With both sets of points, the original ones and the corrected ones, local image transformations are defined considering only nearby control points. Rather than defining a global model, local functions are characterized. In this manner, the interpolated rectification surface is composed of local surface patches each influenced by nearby control points. This method is more sensitive to local deformations and allows the image to be corrected according to the distortion the image is suffering. No extra calibration template is needed since training data to calibrate the distortion can also be used afterwards for the camera calibration process.

Solution 4:

Algorithm implemented in MATLAB:

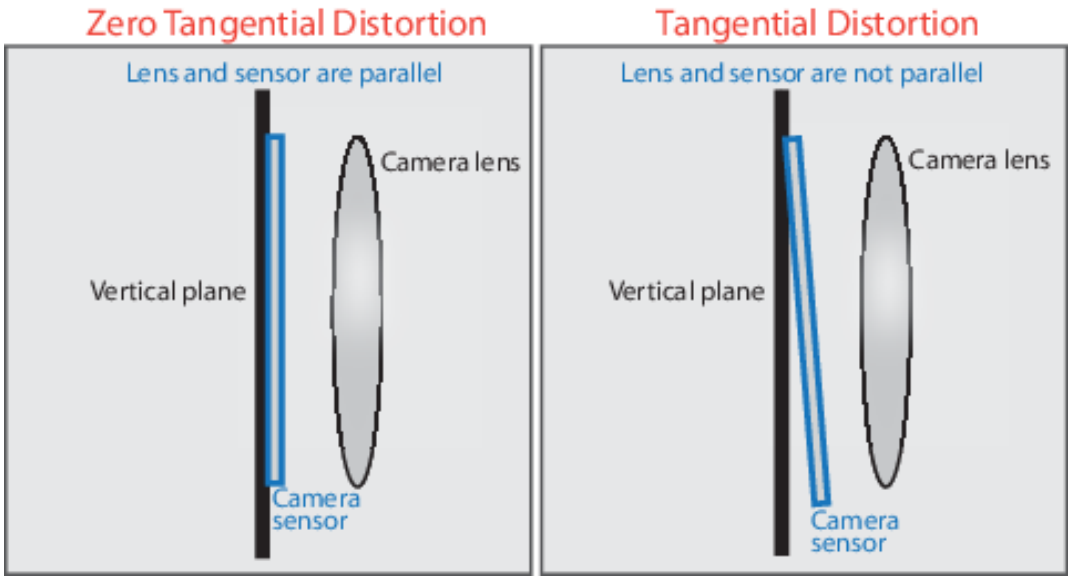
- Create a set of calibration images.
- Detect calibration pattern.
- Generate world coordinates of the corners of the squares.
- Calibrate the camera.
- Load an image and detect the checkerboard points.
- Undistort the points.
- Undistort the image.
- Translate undistorted points.
- Remove lens distortion and display results.

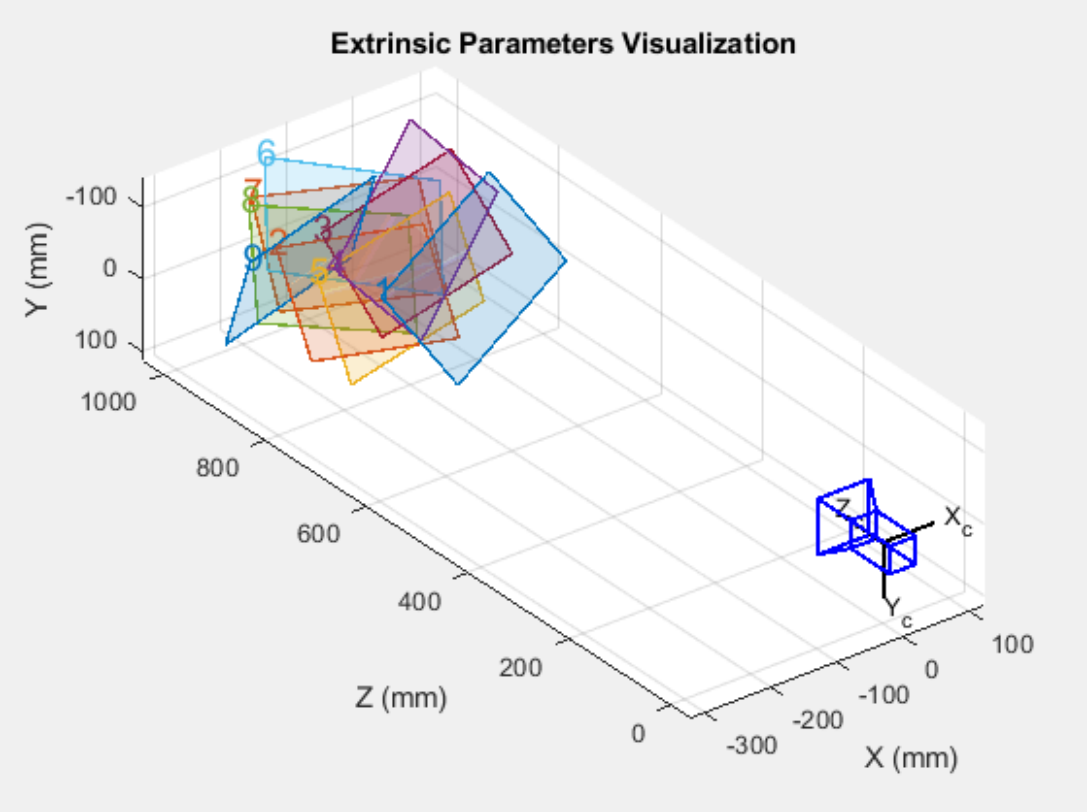
The classes are: Camera Calibration Errors, Extrinsics Estimation Errors, Intrinsics Estimation Errors, Stereo Parameters

The functions are: Detect checkerboard points, estimate camera parameters, generate checkerboard points, show extrinsics, show reprojection errors, radial distortion, tangential distortion, undistort image

(Source:
https://www.mathworks.com/help/vision/ref/cameraparameters.html#d120e103244
,
https://www.mathworks.com/help/vision/ref/estimatecameraparameters.html,
https://www.mathworks.com/help/vision/ref/undistortimage.html)

The functions in detail explanation can be seen in the links mentioned above.


Zero Tangential Distortion
Lens and sensor are parallel
Camera lens
Vertical plane
Camera sensor
Tangential Distortion
Lens and sensor are not parallel
Camera lens
Vertical plane
Camera sensor


Extrinsic Parameters Visualization
Y (mm)
-100
0
100
1000
800
600
400
200
0
Z (mm)
-300
-200
-100
0
100
X (mm)
Z
X c
Y c

Calibration Algorithm

You can use the Camera Calibrator app with cameras up to a field of view (FOV) of 95 degrees.

The calibration algorithm assumes a pinhole camera model:

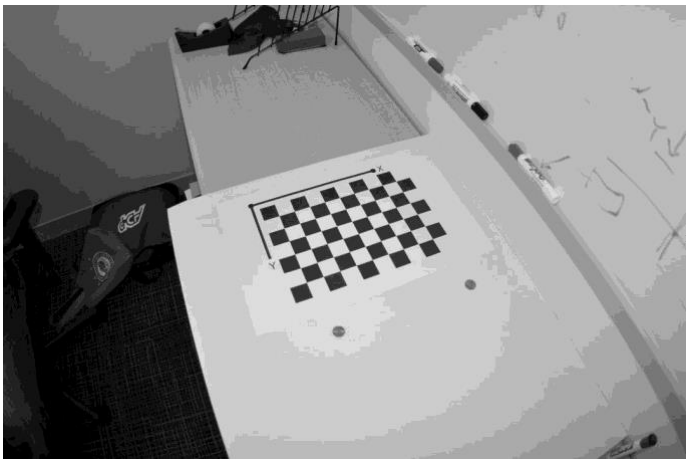
$$w\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} R \\ t \end{bmatrix} K$$

(X,Y,Z): world coordinates of a point
(x,y): coordinates of the corresponding image point
w: arbitrary scale factor
K: camera intrinsic matrix
R: matrix representing the 3-D rotation of the camera
t: translation of the camera relative to the world coordinate system

Camera calibration estimates the values of the intrinsic parameters, the extrinsic parameters, and the distortion coefficients. There are two steps involved in camera calibration:

1. Solve for the intrinsics and extrinsics in closed form, assuming that lens distortion is zero.
2. Estimate all parameters simultaneously including the distortion coefficients using nonlinear least-squares minimization (Levenberg–Marquardt algorithm). Use the closed form solution from the preceding step as the initial estimate of the intrinsics and extrinsics. Then set the initial estimate of the distortion coefficients to zero.

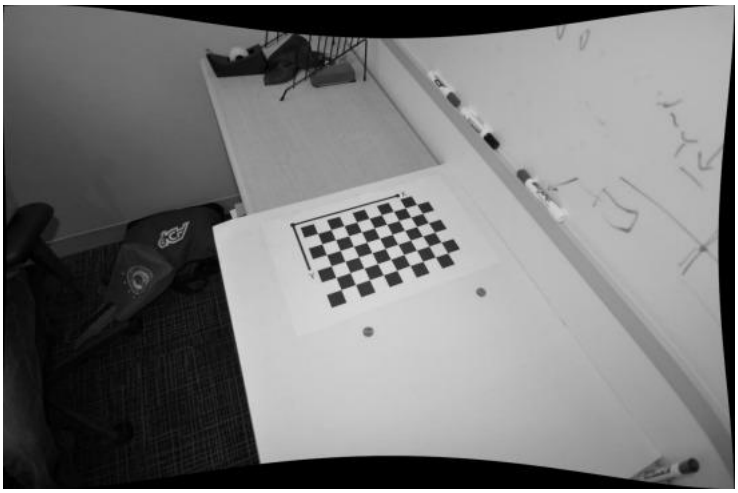
III. Experimental Results



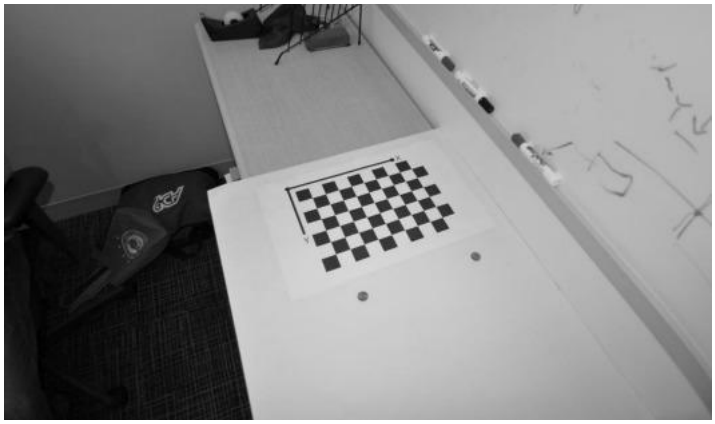
Input image – classroom.raw

Solution 1: using camera co-ordinate formula and Linear Regression

Output image matching the same size as the input image



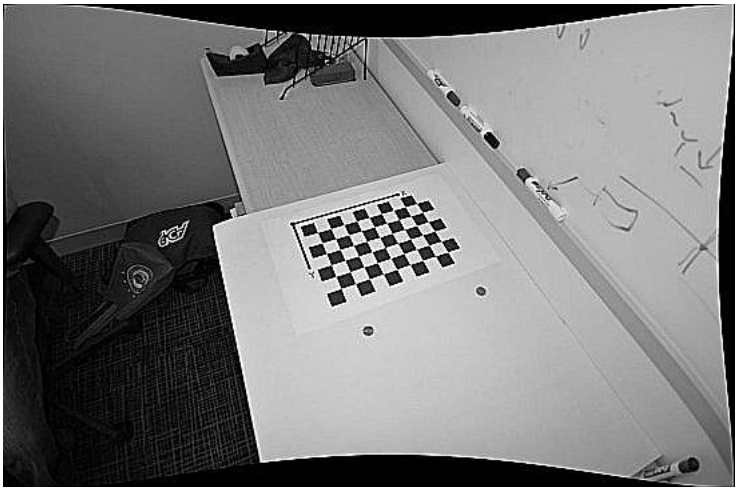
Undistorted output image containing all the pixels from input image



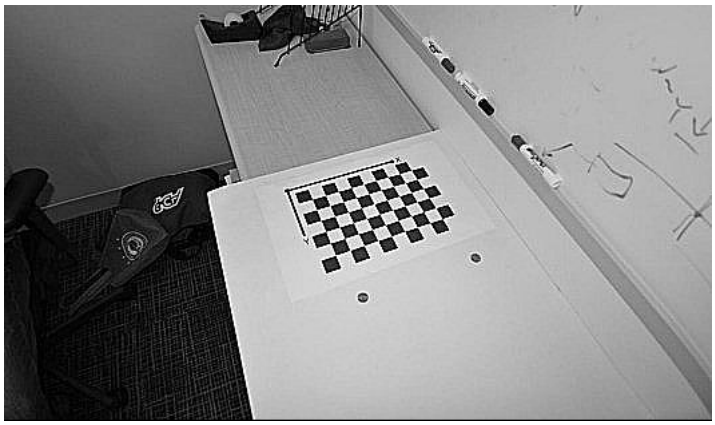
Undistorted output image containing only valid pixels from the input image

Solution 2: using Linear approximation with Gauss-Newton method

Output image matching the same size as the input image



Undistorted output image containing all the pixels from input image



Undistorted output image containing only valid pixels from the input image

IV. Discussion

We can see the various results obtained from the different methods used which are mentioned in the approach section above. We can compare the results and conclude that Newton method results that is solution 2 gives better results than the linear regression method that is solution 1 stating the following reasons. The improvement reflects in two aspects. One is the high precision fitting using adjusted points, and the other one is adding the tangential distortion and selecting the Gauss-Newton method for faster optimization. The main advantage of the proposed algorithm compared with others is its high accuracy and low complexity.

I have suggested two more solutions I found after some research above in the approach section, however, I have not implemented them due to the time shortage.

Problem 2: Morphological Processing

(a) Basic Morphological Process Implementation

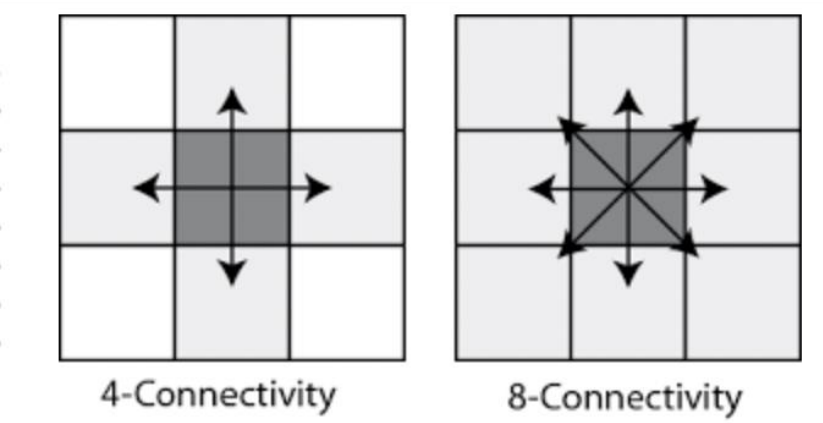
I. Abstract and Motivation

Morphology stands for study of forms/shape of plants and animals. This study is carried out by looking at the binary images or shape of say example of star or hand-written characters/digits. Morphological image processing is a method in which the image is modified depending on its shape. This technique is mainly used in object recognition to identify the objects and modify them. Morphological image processing is performed on binary images. The binary images have a lot of imperfections in them in the form of noise and textures. Morphological image processing takes into account the shape and structure of the object to remove the imperfections. It takes into account the relative ordering of pixels rather than their values.

The MNIST dataset is very famous for using in this study. It consists of hand-written datasets of digits from 0 to 9. It has 60,000 training data each number corresponding to 6000 data. The testing data consists of 10,000 numbers. There is also another MPEG-7 dataset which is a shape dataset consisting of a lot of objects with different shapes. It consists of small data samples and generates retrieval problem.

For answering the question, how to analyze the shape? We use the ImageNet dataset. Say, if we consider a dog, we usually consider the dog's face and not the body or other areas. The pixel values on the surface gives the appearance. The contour gives the shape. There are two visual pathways – appearance and shape. The CNN is good with the appearance how not so good in shape.

We develop various tools like thinning, skeletonizing, shrinking. Input is in the form of strokes – width + shape. The desired output has width as thin as possible. Shape represents the connectivity of the dots. Hence, we generate the morphological processing tools to handle this shape analysis. We consider the binary images having black as object or foreground i.e. 1 and white as the background by convention i.e. 0. By this binary image connectivity, any of it's 4 nearest neighbors is 1 i.e. 4 connected and any of it's 8 nearest neighbors is 1 i.e. 8 connected. Apply 2 rules, connectivity between 1's → 8-connectivity rule and connectivity between 0's → 4-connectivity rule. Another way to characterize connectivity is bond. For 4 connectivity, bond weight is 2 and for 8 connectivity, bond weight is 1. The bond can be calculated as the sum of all bond weights.



The morphological filters are listed below: a) hit-n-miss filter – consists of a set of odd-size masks. Scan the 3x3 filters throughout the input image at the center pixel value and if there is a hit, change the center pixel value to one/zero and if it is a miss then copy the same value. Usually in the input image, the boundary is quite big, and the object lies quite inside, so most of the times you don't need to consider the boundary effect. But if the object is stretched towards the boundaries then the boundary needs to be copied and increased outside to avoid boundary effect. If the underlying pattern of the input image matches one of the hit masks, we record the hit and change the center pixel value where 0 becomes 1 and 1 becomes 0 in the

output image. Otherwise, it is a miss and we copy the center pixel value in the output image.

For example, consider the postprocessing of edge detection. We find the gradients and apply the thresholds and calculate the edge map having 0 means no edge or 1 means edge. Say, there is an isolated edge point, we need to remove it using some morphological processing. To solve this, we generate a hit mask as follows and scan it through the whole image and when there is a hit, we convert the center pixel value to zero.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The logical expression is defined below:

$$\begin{bmatrix} X3 & X2 & X1 \\ X4 & X & X0 \\ X5 & X6 & X7 \end{bmatrix}$$

Where, X is the center and X0 ~ X7 are 8 neighbors. These are all binary thus building logical variables.
F(j,k) = X, F(j+1,k) = X0 and so on for the other 7 neighbors.
The mask relationship is shown.
If miss, G(j,k) = F(j,k) = X.
If hit, G(j,k) = (negation) F(j,k) = (negation) X.
For isolated edge or noise removal, the expression is shown below:
G(j,k) = X ∩ (X0 ∪ X1 ∪ ……. ∪ X7), where ∩ stands for logical AND and ∪ stands for logical OR.

There are two different types of filters: Additive filter and Subtractive filter.
Additive filter will convert some background points (0) to foreground points (1).
Subtractive filter will convert some foreground points (1) to background points (0).
Examples of Additive filters:
a) Interior fill

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

X0, X1, ……, X7 ← 8-bit string. The bit masks are the 16 bit strings.
For additive, check if center is 0 and change it. For subtractive, check if center is 1 and change it.
b) Diagonal fill

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

c) Bridge

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

d) Eight-neighbor dilate

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Examples of Subtractive filters:

a) Isolated edge point removal

b) Spur removal

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

c) Interior pixel removal

$$\begin{bmatrix} D & 1 & D \\ 1 & 1 & 1 \\ D & 1 & D \end{bmatrix} \rightarrow \begin{bmatrix} D & 1 & D \\ 1 & 0 & 1 \\ D & 1 & D \end{bmatrix}$$

d) H-break

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

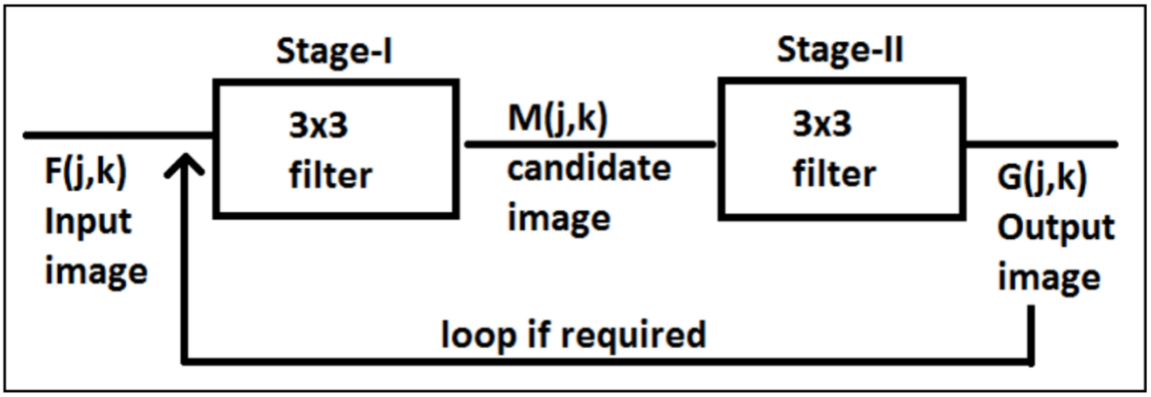
The advanced morphological filters are 5x5 or 3x3 matrices. For 5x5, 25-1 = 24, 2^24 patterns. For 3x3, 9-1 = 8, 2^8 = 256 patterns. There are some filters more complicated than 3x3 and less than 5x5. Such advanced filters are thinning gets the central line of one pixel width, shrinking gets the central pixel and skeletonizing gets the central pixel plus consideration of the boundaries/corners.

Implementation: T → thinning, S → shrinking, K → skeletonizing

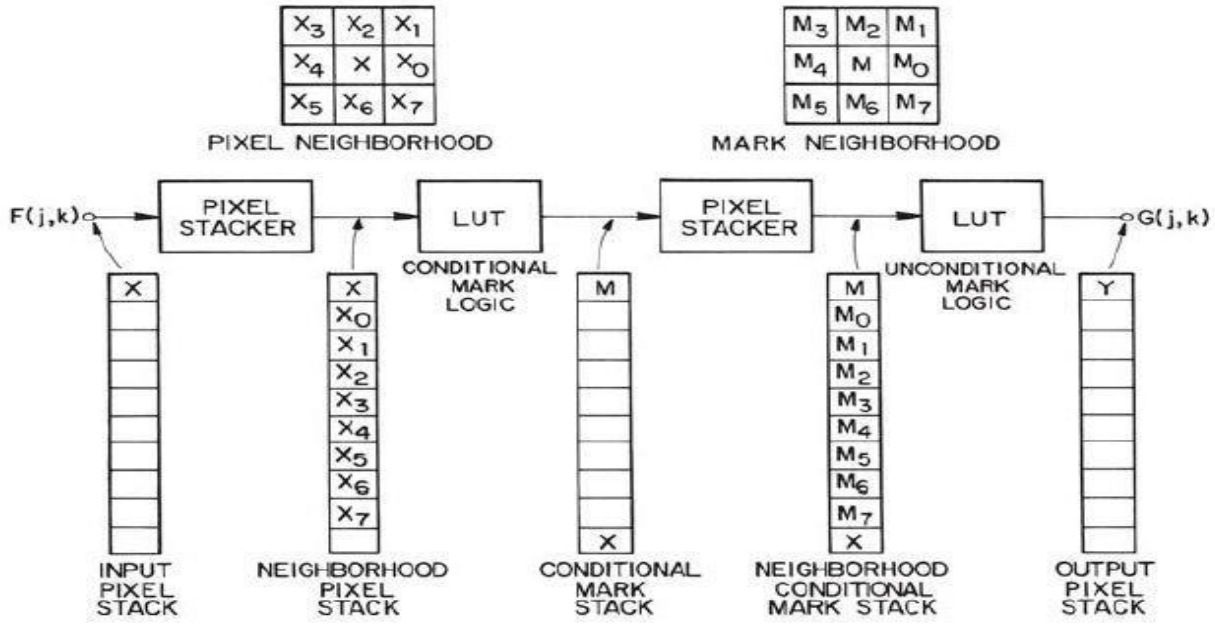
These operators are subtractive filter.

The diagram for the processes is shown below:

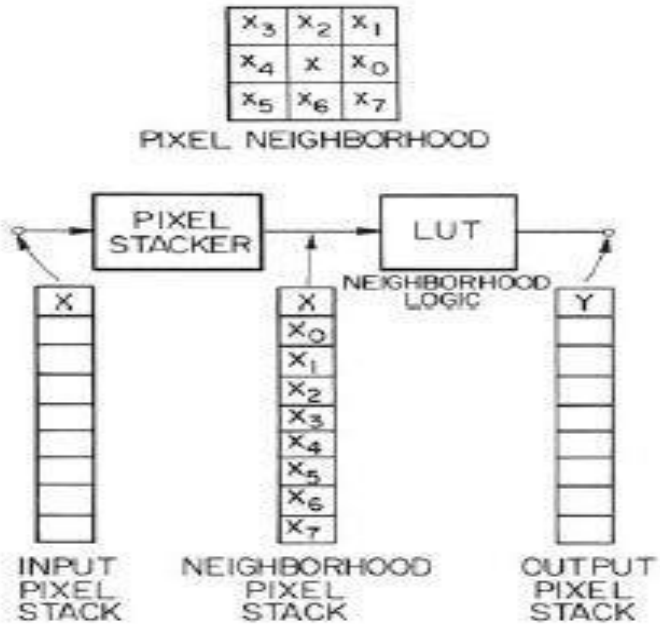
F(j,k) → (Stage 1) first 3x3 hit or miss filter → M(j,k) → (Stage 2) second 3x3 hit or miss filter → G(j,k)



Stage 1: candidates for subtraction, STK filters are given in the pattern table
To implement, first check whether the center pixel value in the filter in the input image is 1 or 0, if 0, do not change, if 1 only then use the STK filters on it. Then check the bond, whether the bond number is allowed. Third step is to check the pattern, that is check the 8 bit string pattern in the form of X X0, X1, ……, X7.
Stage 2: confirmation of subtraction operator
Here, M being the center pixel and for different patterns mentioned in the pattern table, if there is a hit, change M else not.



Look up table flowchart for binary conditional mark operations



Look up table flowchart for binary unconditional operations

A) Morphological operations are usually done considering '0' as the background and '1' as the foreground. In this problem, I will be implementing Shrinking, which is a simple morphological operation which is very "Extreme" too. I mention this Extreme because, it reduces a solid image to a single point. Using this morphological operation, I will find the output of using shrinking process on pattern 1~4.raw input images.

B) I will be implementing Thinning which is one another morphological operation. This is an operation which is not so extreme like Shrinking, but also does not give the skeleton of the image. The given input Jigsaw image is considered for Thinning operation. Similar to previous morphological operation, here also '0' is considered as the background and '1' is considered as foreground.

C) Skeletonizing is a process of reducing foreground regions in a binary image to a skeletal region which preserves the extent of connectivity and disregards most of the other foreground pixels. When compared to Shrinking and Thinning, Skeletonizing preserves most of the information and is the least extreme in extracting morphological information. In this question, I have performed Skeletonizing on the pattern 1~4.raw input image. Similar to previous morphological operation, here also '0' is considered as the background and '1' is considered as foreground. Skeletonizing is a process which converts a large number of foreground pixels in the original image into

background pixels but at the same time, maintains the connectivity of the original image.

II. Approach and Procedure

In this problem, we implement the three morphological processing operations: shrinking, thinning, and skeletonizing using the pattern tables. We apply the "shrinking", "thinning", and "skeletonizing" filters to the 4 pattern images (pattern1-4.raw), respectively and show the results below and analyze and discuss about them.

(Source: Discussion Lecture dated 02/12/2019)

1. Neighborhood definition - 4 connected and 8 connected.
2. Pixel bound - is used to simplify the implementation of 3 morphological process.
   B = 2 * no. of 4 connected neighbors + 1 * no. of 8 connected neighbors
3. Logic operations

a. NOT b. Union (A or B) c. Intersection (A and B) d. D = 0 or 1 (don't care)

Shrinking: Erase the black pixels (object) such that
-an object without holes inside erodes to a single pixel at or near it's center of mass.
-an object with holes erodes to a connected ring lying midway between each hole and its nearest outer boundary.
2 stages:
Input to loop up table (conditional) to midway value to loop up table (unconditional) to output, using two 3x3 masks.

Compare the 3x3 matrix of 8 connected neighbors with the 3x3 mask in the pattern table for each S, T and K and if it hits 1 then M for that center interest pixel is 1 else M = 0.
Compare the 3x3 matrix with the unconditional masks of the table for each S and T and table is different for K, if hit then P = 1 else P = 0.
The above was one iteration and hence get G for that pixel location using formula.

For implementation, trasfer the table matrix into a long array (1x9), also from the image mask change into (1x9) vector to do compression.
Hardcode all the masks in table.
Padding doing is optional as the background is 0 ie black.
Preprocessing to the input image - Make 0 as background as black and 255 as object as white.

Q) When to stop the loop?
Once there are no more changes in the output image anymore.
Create variable called flag.

```
flag = true
while (flag){
    S/T/K;
    if (image changed from last time)
    flag = true;
    else
    flag = false;
}
```

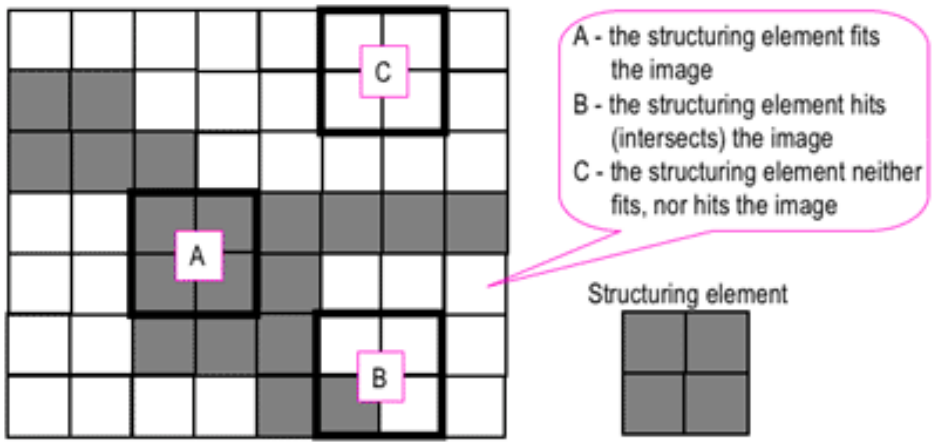
Thinning is similar to Shrinking.

Skeletonization - Repeat the steps above.
Extra step is bridging to avoid breaking of connectivity, to connect the two parts again.
Here, all "X"s refer to pixels after done with sufficient iterations of skeletonization.

We do one layer padding on the input image with the zero padding (background padding).

(Source: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)

Morphological image processing uses a structuring element which is normally an odd sized mask. The structuring element is placed over the binary image and it is designed in such a way that it performs the desired morphological operation.



The structuring element is positioned at all possible locations in the image and it is compared with the corresponding neighborhood of pixels. Some operations test whether the element "fits" within the neighborhood, while others test whether it "hits" or intersects the neighborhood. The structuring element is mainly used to perform the “hit and miss” transformation. The structuring element has both foreground and background pixels. If the structuring element exactly matches the foreground and background pixels in the input image, the underlying pixel is set to value of foreground pixel, else it is set to value of background pixel. Structuring elements play in morphological image processing the same role as convolution kernels in linear image filtering.

S → Shrinking, T → Thinning, K → Skeletonizing
The conditional mask patterns are given below:
(Source: Digital Image Processing by William K. Pratt – pg 433)

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

Type	Bond	Patterns
S	1	0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1
S	2	0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
S	3	0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1
TK	4	0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0
STK	4	0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 1
ST	5	1 1 0 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0
ST	5	0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 1
ST	6	1 1 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 0
STK	6	1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1
STK	7	1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1
STK	8	0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 0 1 1 1
STK	9	1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 1 1
STK	10	1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
K	11	1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1

The unconditional mask patterns are given below:

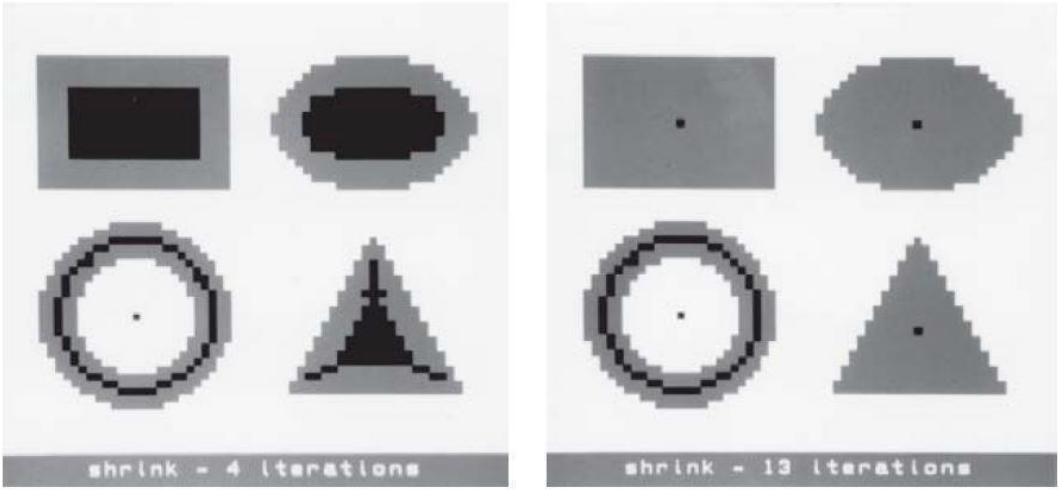
(Source: Digital Image Processing by William K. Pratt – pg 435)

Table 14.3-2 Shrink and Thin Unconditional Mark Patterns

Spur	0	0	M	M	0	0																		
	0	M	0	0	M	0																		
	0	0	0	0	0	0																		
Single	0	0	0	0	0	0																		
4-connection	0	M	0	0	M	M																		
	0	M	0	0	0	0																		
L Cluster	0	0	M	0	M	M	M	M	0	M	0	0												
	0	M	M	0	M	0	0	M	0	M	M	0												
	0	0	0	0	0	0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	0												
	M	M	0	0	M	0	0	M	0	0	M	M												
	M	0	0	M	M	0	0	M	M	0	0	M												
4-connected	0	M	M	M	M	0	0	M	0	0	0	M												
Offset	M	M	0	0	M	M	0	M	M	0	M	M												
	0	0	0	0	0	0	0	0	M	0	M	0												
Spur corner	0	A	M	M	B	0	0	0	M	M	0	0												
Cluster	0	M	B	A	M	0	A	M	0	0	M	B												
	M	0	0	0	0	M	M	B	0	0	A	M												
Corner	M	M	D																					
Cluster	M	M	D																					
	D	D	D																					
Tee Branch	D	M	0	0	M	D	0	0	D	D	0	0												
	M	M	M	M	M	M	M	M	M	M	M	M												
	D	0	0	0	0	D	0	M	D	D	M	0												
	D	M	D	0	M	0	0	M	0	D	M	D												
	M	M	0	M	M	0	0	M	M	0	M	M												
	0	M	0	D	M	D	D	M	D	0	M	0												
Vee Branch	M	D	M	M	D	C	C	B	A	A	D	M												
	D	M	D	D	M	B	D	M	D	B	M	D												
	A	B	C	M	D	A	M	D	M	C	D	M												
Diagonal	D	M	0	0	M	D	D	0	M	M	0	D												
Branch	0	M	M	M	M	0	M	M	0	0	M	M												
	M	0	D	D	0	M	0	M	D	D	M	0												
A or B or C = 1			D = 0 or 1			A or B = 1																		

A) I used the Shrinking – Conditional and Unconditional pattern masks to shrink every pattern in the input images to a single dot or point. When I apply Shrinking Conditional Masks, the output would be boundary layers that

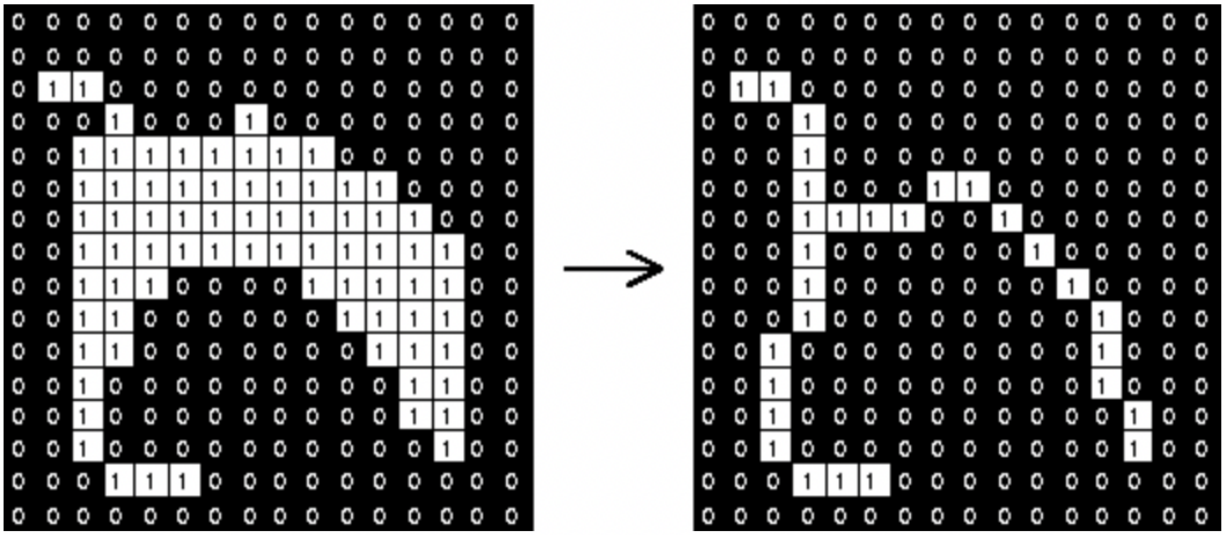
needed to be peeled off in the second stage. The second stage is where the boundaries are peeled off and a pattern with a reduced boundary is given as the output. This works in the first 3 cases. However, in the last case, the output would be boundary layers that needed to be peeled off in the second stage. Thus, in every iteration the size of the image reduces. But then since there is one more black heart object at the center of the white square object, we see that the boundary goes on reducing until the shape becomes a little deformed than a square ring towards the heart shape. However, it doesn't shrink to a single point, as there is a closed object present inside the square object, so it shrinks to a ring. If the object is without holes, the result of the shrinking algorithm gives a dot or point, whereas if the image object has a hole, the result of the shrinking algorithm is a ring.



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

B) The Thinning operation is related to hit and miss transform based on Conditional and Unconditional masks that are given in the table above. The structure elements are passed through the image pixels and a thin representation of the underlying Foreground image is visualized. An example of Thinning on a simple binary image is shown below.

(Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm)



The thinning is applied to all the pattern images and we can see that the thinning compresses the object to a thin line or dot in some cases and the outputs are shown below.
The main difference between Shrinking and Thinning are the masks. Appropriate masks are identified for Thinning that vary for every morphological operation. Also, an extra operation of Bridging is performed. This is done to connect small disconnected components in the Thinning process. Bridging is done on the last iteration to avoid the unnecessary disjoints in the output.

Bridging operation is shown and described below:
(Source: Digital Image Processing by William K. Pratt – pg 426)

Bridge. Create a black pixel if creation results in connectivity of previously unconnected neighboring black pixels.

$$G(j,k) = X \cup [P_1 \cup P_2 \cup \cdots \cup P_6] \tag{14.2-4a}$$

where

$$P_1 = \bar{X}_2 \cap \bar{X}_6 \cap [X_3 \cup X_4 \cup X_5] \cap [X_0 \cup X_1 \cup X_7] \cap \bar{P}_Q \tag{14.2-4b}$$

$$P_2 = \bar{X}_0 \cap \bar{X}_4 \cap [X_1 \cup X_2 \cup X_3] \cap [X_5 \cup X_6 \cup X_7] \cap \bar{P}_Q \tag{14.2-4c}$$

$$P_3 = \bar{X}_0 \cap \bar{X}_6 \cap X_7 \cap [X_2 \cup X_3 \cup X_4] \tag{14.2-4d}$$

$$P_4 = \bar{X}_0 \cap \bar{X}_2 \cap X_1 \cap [X_4 \cup X_5 \cup X_6] \tag{14.2-4e}$$

$$P_5 = \bar{X}_2 \cap \bar{X}_4 \cap X_3 \cap [X_0 \cup X_6 \cup X_7] \tag{14.2-4f}$$

$$P_6 = \bar{X}_4 \cap \bar{X}_6 \cap X_5 \cap [X_0 \cup X_1 \cup X_2] \tag{14.2-4g}$$

and

$$P_Q = L_1 \cup L_2 \cup L_3 \cup L_4 \tag{14.2-4h}$$

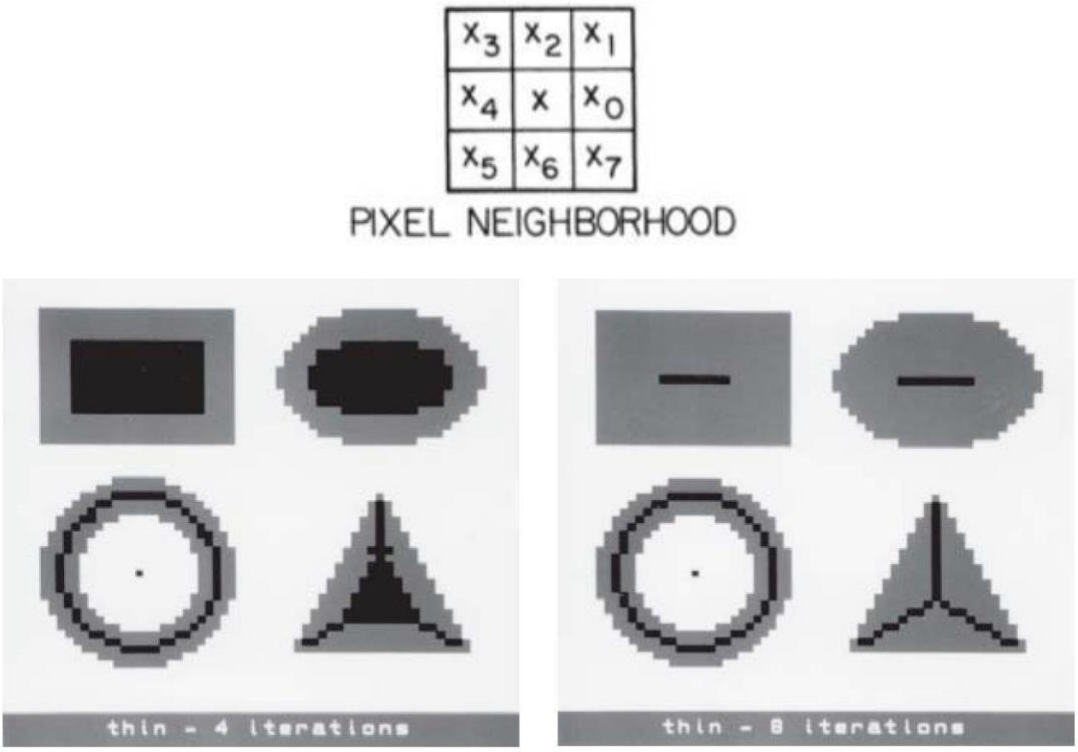
$$L_1 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap \bar{X}_7 \tag{14.2-4i}$$

$$L_2 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap \bar{X}_7 \tag{14.2-4j}$$

$$L_3 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap X_7 \tag{14.2-4k}$$

$$L_4 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap X_7 \tag{14.2-4l}$$

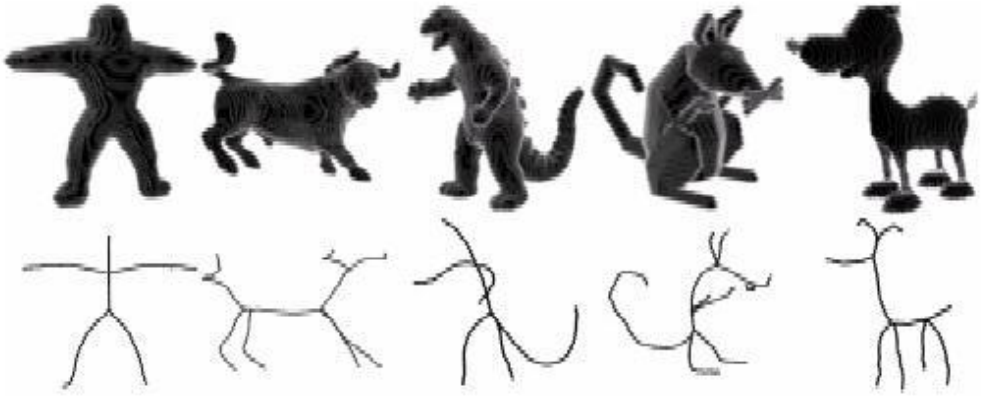
In bridging operation, a black dot is created if the creation leads to a connectivity in the previously unconnected black pixels. In bridging, the gaps in between the images are filled.



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

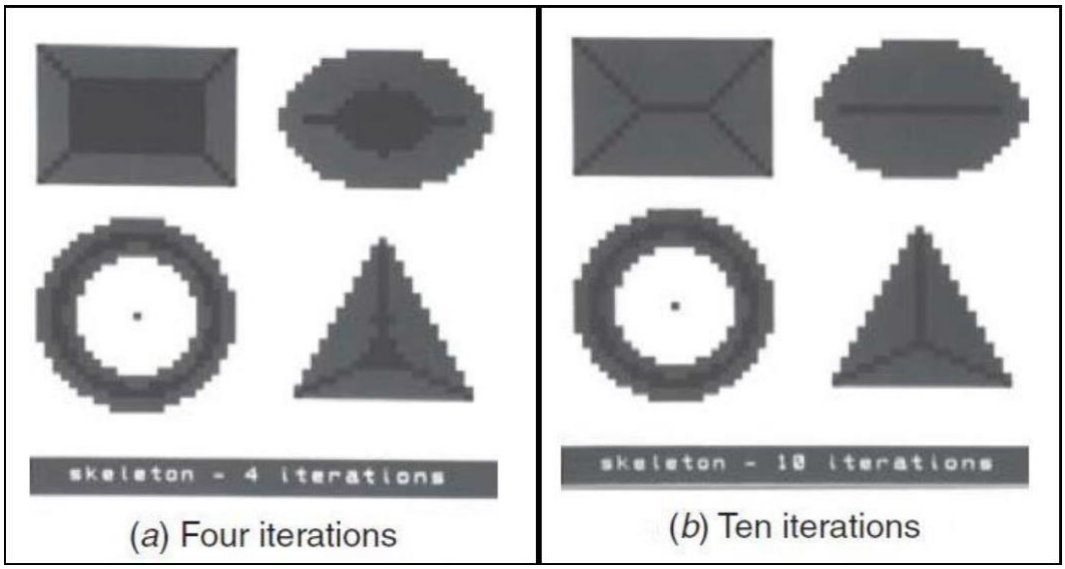
C) The Skeletonizing operation is related to hit and miss transform based on Conditional and Unconditional masks that are given in the tables above. The structure elements are passed through the image pixels and a skeleton representation of the underlying Foreground image is visualized. An example of Skeletonizing on a simple binary image is shown below.

(Source: http://coewww.rutgers.edu/www2/vizlab/skeleton)



The skeletonizing is applied to all the pattern images and we can see that the skeletonizing compresses the object giving the best connected reduced form

for the input and the outputs are shown below. An extra operation of Bridging is performed. This is done to connect small disconnected components in the skeletonizing process. Bridging is done on the last iteration to avoid the unnecessary disjoints in the output.



(*a*) Four iterations (*b*) Ten iterations

The Conditional Mask Patterns for Shrinking, Thinning and Skeletonizing are given in one single table and hence is the same in all the sub questions. The Unconditional Mask Patterns are same for Shrinking and Thinning. Whereas it is different for Skeletonizing. Since Shrinking and Thinning are very extreme in extracting morphological information, they have similar Unconditional Mask Patterns. But Skeletonizing is not so extreme whereas it preserves most of the information in a binary image to give the skeleton of the image structure, it has different Unconditional Mask Patterns.

Algorithm implemented in C++:

Shrinking

- Read the input image "pattern1~4.raw" whose dimensions are height_size = 375, width_size = 375, BytesPerPixel = 1
- Convert the grayscale image into binarized image by using fixed thresholding such that
  If input pixel value <=127, set output value to 0; else 255
- Apply the shrinking algorithm on the binarized input image

- The 3x3 mask is converted into a 9-array elements which is converted into a string for making the calculation easy
- Compute the Stage 1 steps:
    - Initialize the intermediate matrix with all zeros
    - Using two for loops, transverse through the whole image using the 3x3 mask
    - Find every single pixel if it is 0 or 1
    - If it is 1, calculate the bond value using the formula mentioned above and compare the 3x3 neighbor of that pixel with all the bond's corresponding conditional masks
    - If it is 0, copy the same value and don't change it
    - If it is a hit, put 1 in the intermediate matrix
    - If it is a miss, put 0 in the intermediate matrix
- Compute the Stage 2 steps:
    - We have the input matrix and the intermediate matrix
    - Using two for loops, transverse through the whole image using the 3x3 mask
    - Find every single pixel if it is 0 or 1
    - If it is 1, compare the 3x3 neighbor of that pixel with all the conditional masks
    - If it is 0, copy the same value to the output and don't change it
    - If it is a hit, copy input to output
    - If it is a miss, put 0 in the output pixel value
- Perform stage 1 and stage 2 for many iterations until get the desired output
- Check every element of the 3x3 neighbor with every element of the mask, such that if one element doesn't match, break and move on to the next one
- Iterate through all the masks similarly and if one mask completely matches, break and get the hit or miss output
- The desired output is obtained when there is no more change to the image for another iteration and it saturates
- Write the computed image data array on output.raw file using the fwrite() function

Thinning

- Read the input image "pattern1~4.raw" whose dimensions are height_size = 375, width_size = 375, BytesPerPixel = 1
- Convert the grayscale image into binarized image by using fixed thresholding such that

If input pixel value <=127, set output value to 0; else 255

- Apply the shrinking algorithm on the binarized input image
- The 3x3 mask is converted into a 9-array elements which is converted into a string for making the calculation easy
- Compute the Stage 1 steps:
    - Initialize the intermediate matrix with all zeros
    - Using two for loops, transverse through the whole image using the 3x3 mask
    - Find every single pixel if it is 0 or 1
    - If it is 1, calculate the bond value using the formula mentioned above and compare the 3x3 neighbor of that pixel with all the bond's corresponding conditional masks
    - If it is 0, copy the same value and don't change it
    - If it is a hit, put 1 in the intermediate matrix
    - If it is a miss, put 0 in the intermediate matrix
- Compute the Stage 2 steps:
    - We have the input matrix and the intermediate matrix
    - Using two for loops, transverse through the whole image using the 3x3 mask
    - Find every single pixel if it is 0 or 1
    - If it is 1, compare the 3x3 neighbor of that pixel with all the conditional masks
    - If it is 0, copy the same value to the output and don't change it
    - If it is a hit, copy input to output
    - If it is a miss, put 0 in the output pixel value
- Perform stage 1 and stage 2 for many iterations until get the desired output
- Check every element of the 3x3 neighbor with every element of the mask, such that if one element doesn't match, break and move on to the next one
- Iterate through all the masks similarly and if one mask completely matches, break and get the hit or miss output
- The desired output is obtained when there is no more change to the image for another iteration and it saturates
- Perform bridging to join the disjoint lines in output
- Using two for loops, transverse through the whole final iteration image using the 3x3 mask
- Get X, X0, X1, X2, X3, X4, X5, X6, X7
- X is the center pixel and all others are surrounding neighbor pixels
- Calculate L1, L2, L3, L4, P1, P2, P3, P4, P5 and P6 using the above mentioned Logical Operations

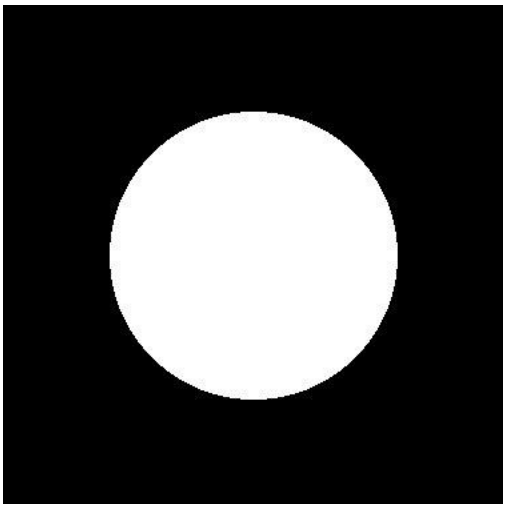
- Calculate the pixel intensity value using the above variables
- Get the final bridged image
- Write the computed image data array on output.raw file using the fwrite() function

Skeletonizing

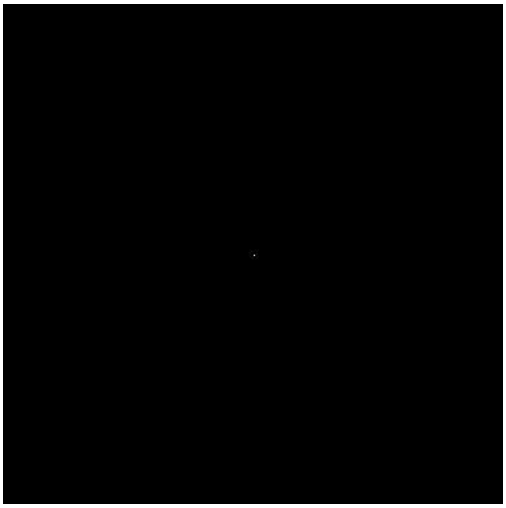
- Read the input image "pattern1~4.raw" whose dimensions are height_size = 375, width_size = 375, BytesPerPixel = 1
- Convert the grayscale image into binarized image by using fixed thresholding such that
  If input pixel value <=127, set output value to 0; else 255
- Apply the shrinking algorithm on the binarized input image
- The 3x3 mask is converted into a 9-array elements which is converted into a string for making the calculation easy
- Compute the Stage 1 steps:
  - Initialize the intermediate matrix with all zeros
  - Using two for loops, transverse through the whole image using the 3x3 mask
  - Find every single pixel if it is 0 or 1
  - If it is 1, calculate the bond value using the formula mentioned above and compare the 3x3 neighbor of that pixel with all the bond's corresponding conditional masks
  - If it is 0, copy the same value and don't change it
  - If it is a hit, put 1 in the intermediate matrix
  - If it is a miss, put 0 in the intermediate matrix
- Compute the Stage 2 steps:
  - We have the input matrix and the intermediate matrix
  - Using two for loops, transverse through the whole image using the 3x3 mask
  - Find every single pixel if it is 0 or 1
  - If it is 1, compare the 3x3 neighbor of that pixel with all the conditional masks
  - If it is 0, copy the same value to the output and don't change it
  - If it is a hit, copy input to output
  - If it is a miss, put 0 in the output pixel value
- Perform stage 1 and stage 2 for many iterations until get the desired output
- Check every element of the 3x3 neighbor with every element of the mask, such that if one element doesn't match, break and move on to the next one

- Iterate through all the masks similarly and if one mask completely matches, break and get the hit or miss output
- The desired output is obtained when there is no more change to the image for another iteration and it saturates
- Perform bridging to join the disjoint lines in output
- Using two for loops, transverse through the whole final iteration image using the 3x3 mask
- Get X, X0, X1, X2, X3, X4, X5, X6, X7
- X is the center pixel and all others are surrounding neighbor pixels
- Calculate L1, L2, L3, L4, P1, P2, P3, P4, P5 and P6 using the above mentioned Logical Operations
- Calculate the pixel intensity value using the above variables
- Get the final bridged image
- Write the computed image data array on output.raw file using the fwrite() function

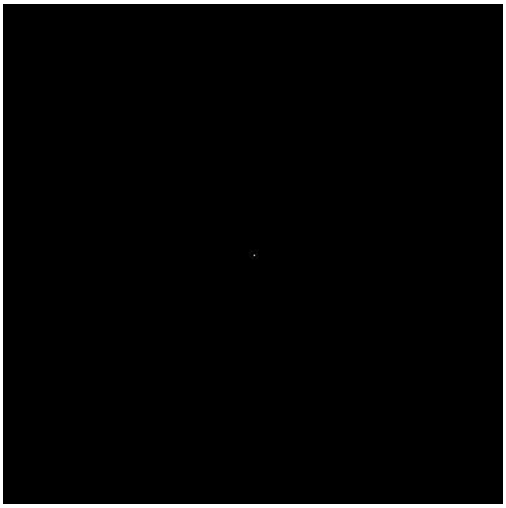
III. Experimental Results



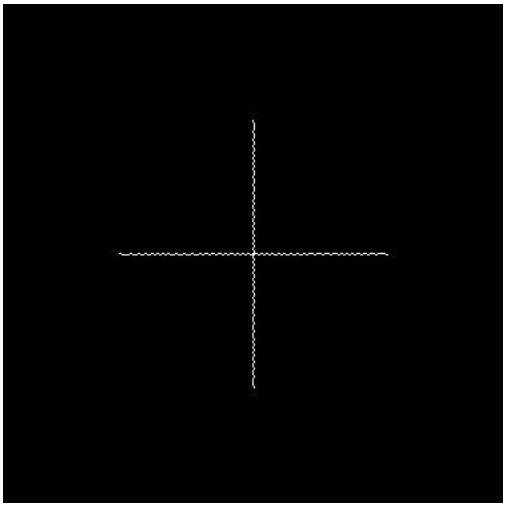
Input image – pattern1.raw



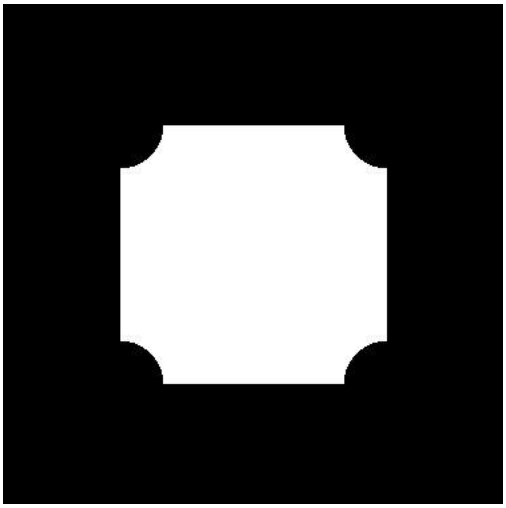
Output of pattern1 when applied shrinking



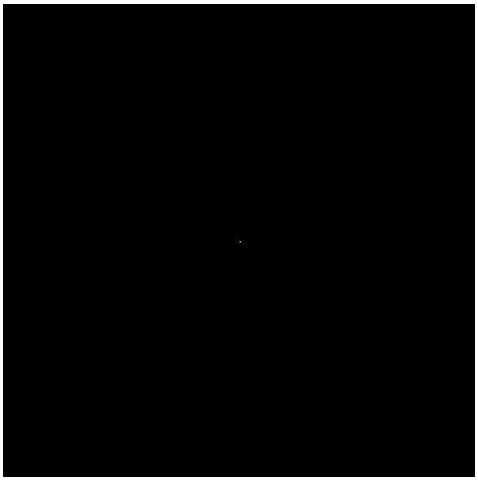
Output of pattern1 when applied thinning



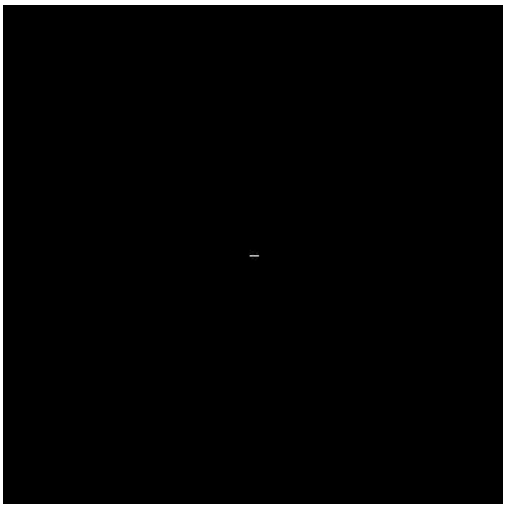
Output of pattern1 when applied skeletonizing



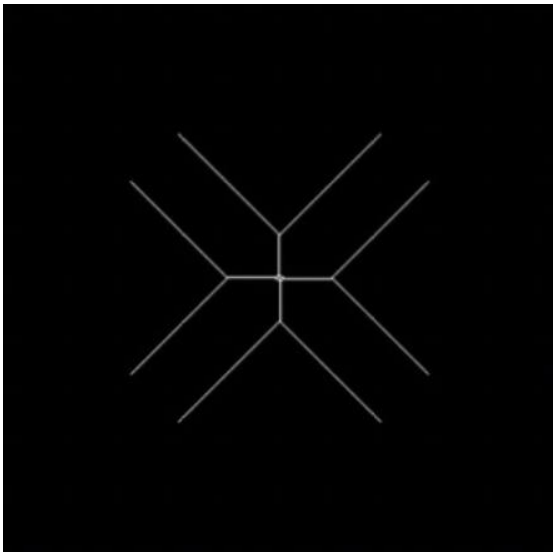
Input image – pattern2.raw



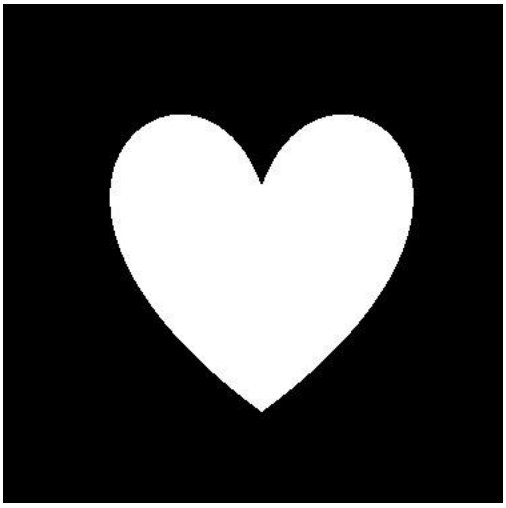
Output of pattern2 when applied shrinking



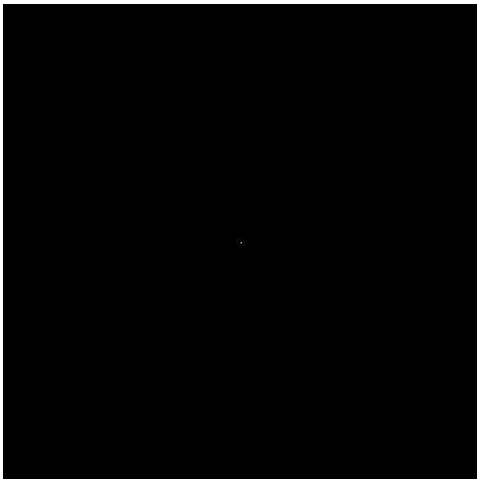
Output of pattern2 when applied thinning



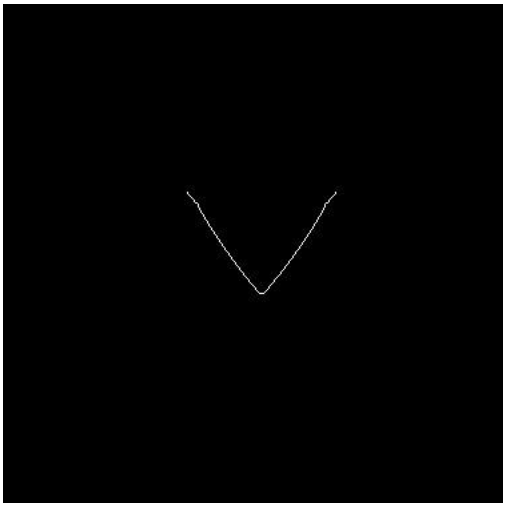
Output of pattern2 when applied skeletonizing



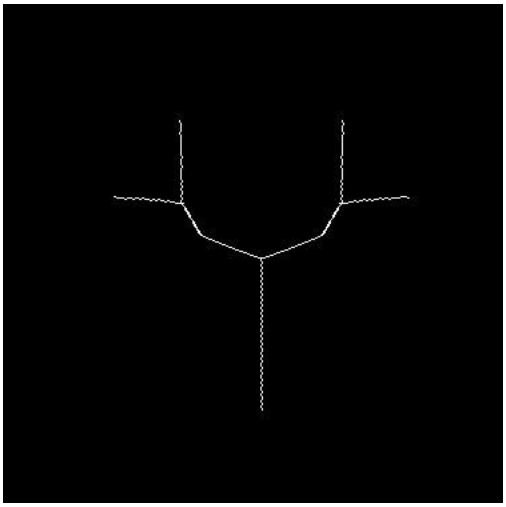
Input image – pattern3.raw



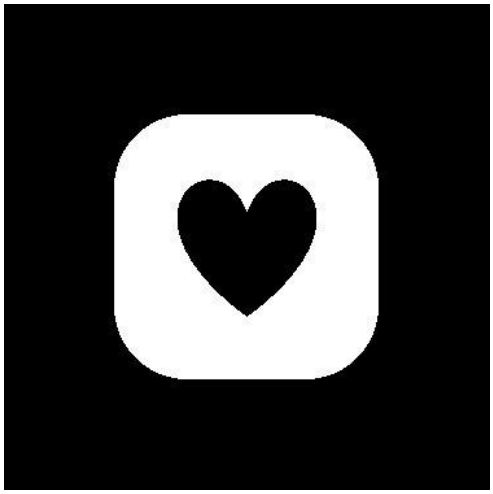
Output of pattern3 when applied shrinking



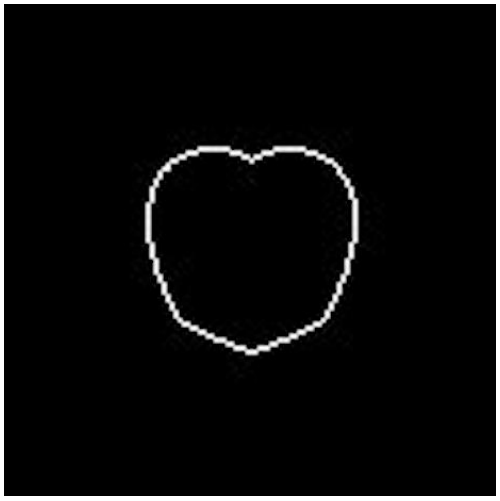
Output of pattern3 when applied thinning



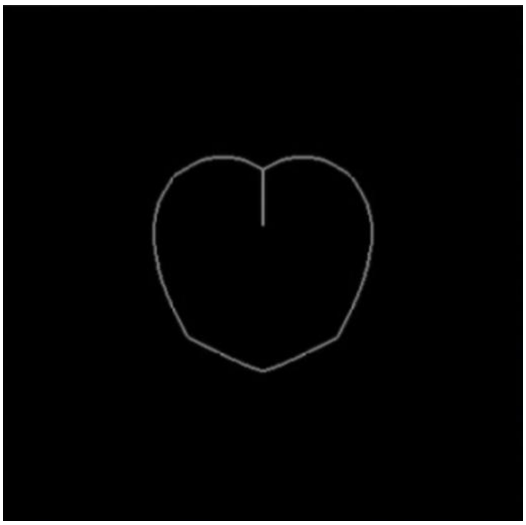
Output of pattern3 when applied skeletonizing



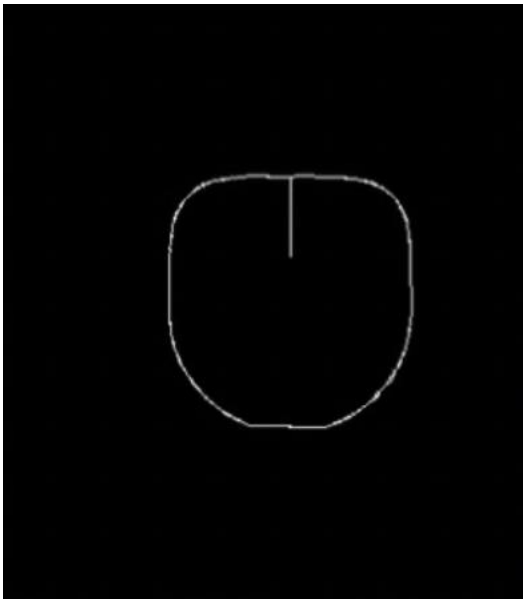
Input image – pattern4.raw



Output of pattern4 when applied shrinking



Output of pattern4 when applied thinning



Output of pattern4 when applied skeletonizing

IV. Discussion

Shrinking

Shrinking morphological operation was performed on the pattern1~4.raw input images using the above mentioned procedure. With every iteration, the size of the patterns goes on reducing. The shrinking algorithm is implemented using a lot of conditional and unconditional masks. The good part of the algorithm is that after the image has been reduced to a single dot or point or line (ring) in some cases, even if the number of iterations goes on increasing, the output remains constant, it saturates and further does not change.

Shrinking is not able to produce the whole structure of the object. It shrinks the object to a single point and hence a lot of the information is lost. It can be used to detect the presence of the object however, cannot determine how the object looks. But when there is another back object inside the white object it shrinks to form a ring and cannot be shrinked further. This helps to detect the presence of another object inside the white object. It can be used to find the total count of the objects present in the image. It is used in applications like getting the count of the number of objects in the image.

For pattern 1, it is a circular object and after applying stage 1 and 2, it shrinks to a single dot at the center. For pattern 2, it is a square object with inward curved circular corners, after applying stage 1 and 2, it shrinks to a single point at the center. For pattern 3, input image contains a heart shaped object, after applying stage 1 and 2, it shrinks to a single point at the center. For pattern 4, it is a square shaped object with curved corners and contains a black heart shaped object inside it, after applying stage 1 and 2, it shrinks to form a ring shape. This ring shape is not exactly square shaped, but it resembles a little similarity with the object inside that is the heart shape.

Thinning

Thinning morphological operation was performed on the pattern1~4.raw input images using the above mentioned procedure. With every iteration, the size of the patterns goes on reducing. The thinning algorithm is implemented using a lot of conditional and unconditional masks. The good part of the algorithm is that after the image has been reduced to a single point in some cases or line or ring, even if the number of iterations goes on increasing, the output remains constant, it saturates and further does not change. As there was a gap in the output obtained after the thinning operation, I implemented the bridging operation. Due to this, the breaks present in the thinning output gets eliminated and the output composes of one stroke.

Thinning process helps us to understand the structure of the object of interest better as compared to the shrinking process. It is used in applications like automation detection of handwriting. For example, this might give a better understanding of the written letter or number.

For pattern 1, it is a circular object and after applying stage 1 and 2 and using bridging, it thins to a single dot at the center. This happens so because we have thinning equally from all the sides as the pattern given is circular. The output obtained here is similar to the shrinking output. For pattern 2, it is a square object with inward curved circular corners, after applying stage 1 and 2 and bridging, it thins to a single horizontal line at the center. For pattern 3, input image contains a heart shaped object, after applying stage 1 and 2 and bridging, it thins to a V shaped line at the center. For pattern 4, it is a square shaped object with curved corners and contains a black heart shaped object inside it, after applying stage 1 and 2 and bridging, it thins to form a ring shape. This ring shape is not exactly square but it resembles a little similarity with the object inside that is the heart shape. The output obtained here is similar to the shrinking output.

Skeletonizing

Skeletonizing morphological operation was performed on the pattern1~4.raw input images using the above mentioned procedure. With every iteration, the size of the patterns goes on reducing. The skeletonizing algorithm is implemented using a lot of conditional and unconditional masks. The good part of the algorithm is that after the image has been reduced to a skeleton or lines with branches, even if the number of iterations goes on increasing, the output remains constant, it saturates and further does not change. As there was a gap in the output obtained after the skeletonizing operation, I implemented the bridging operation. Due to this, the spaces present in the skeletonizing output gets eliminated and the output looks better. For skeletonizing, output is similar to thinning but has more edges at sharp corners. It shows stick like structure.

As we see from the results above, skeletonizing gives better outputs, more information about the structure or skeleton of the object in the image as compared to the shrinking and thinning process. However, there is a disadvantage that skeletonizing is more sensitive to noise since it preserves more information corresponding to almost all the edges as compared to the other morphological processes above. Skeletonizing is used in applications like a simple compact information of the skeleton of the image is required.

For pattern 1, it is a circular object and after applying stage 1 and 2 and bridging, it outputs a skeleton in the form of horizontal and vertical lines intersecting each other at a center point forming similar to a plus sign covering the area of the object. For pattern 2, it is a square object with inward curved circular corners, after applying stage 1 and 2 and bridging, it outputs a structure of horizontal and vertical lines intersecting each other at a center point forming similar to a plus sign being smaller in size as corners as eroded. For pattern 3, input image contains a heart shaped object, after applying stage 1 and 2 and bridging, it outputs a skeleton in the form of Y shape with two more branches. This gives a very close structure to the heart as compared to both shrinking and thinning which were unable to give the structure for the object. Also, by using bridging the disjoint edges were joint and we got connected output. For pattern 4, it is a square shaped object with curved corners and contains a black heart shaped object inside it, after applying stage 1 and 2 and bridging, it outputs a skeleton in the form of horizontal and vertical lines intersecting each other at a center point forming similar to a plus sign, however, it does not correctly represent the object. This structure obtained by skeletonizing is not accurate as compared to the outputs got by the shrinking and thinning process. They resemble the object more closely than the skeletonized processed output.

(b) Defect Detection and Correction

I. Abstract and Motivation

Morphological image processing is a collection of techniques that are used for analyzing geometrical structures in an image. Morphological operations such as dilation, erosion, opening and closing use a small template known as structuring element to transverse into an image. These operations can be combined and applied to basic image processing algorithms for boundary extraction, hole filling, extraction of connected components, convex hull and so on.

Morphological operations are used extensively in shape classification. A method for representing shape is required to extract the shape information of the objects in an image. Various morphological functions like erosion, dilation, shrinking, thinning, skeletonizing and hit or miss are used to provide solutions to hole filling and boundary smoothening problems. In this problem we will make an attempt to implement these solutions for the hole filling. Hole filling is generally a pre-processing step for getting the shape information of the object in the image.

One of the applications of morphological processing technique is defection detection. Suppose a deer.raw image is designed for product decoration, and it will be enlarged to fit multiple product sizes later. For consistent product appeal, image defect detection to insure no black dots in the main deer body is necessary. Thus, we apply morphological process to help determine whether the deer.raw image is defectless. If it is not defectless, we determine the defect regions and correct them to white dots.

Binary images usually contain a lot of imperfections and pixels with redundant information. To remove these, we need to account for the form and structure of the image. This thing basically means the ordering of the pixel value i.e 0 or 1. By studying this feature, we can apply a combination of non-linear operations to the binary image to get successful morphological processing.

(Source: https://docs.opencv.org/3.3.0/d9/d61/tutorial_py_morphological_ops.html)

II. Approach and Procedure

(Source: Discussion Lecture dated 02/12/2019)

Think in a reverse way of problem 2a, connected edges, how many number of objects, number of connected backgrounds due to connectivity - 4 connected not 8. Use connected components wiki (edge connected) - try something different.

The input image is first read whose dimensions are heigh_size = 375 and width_size = 375 and BytesPerPixel = 1. Since the object is present near the corners and the edges of the image, we do image padding, in this case zero padding is done as the background of the input image is black (0), to avoid the boundary effect.

Holes in the object produce incorrect results when morphological operations are performed because these holes tend to produce incorrect hit or miss results. But this problem is solved by implementing the following simple hole filling algorithm, which is done by using the formula below:

$$X_k = (X_{k-1} \oplus B) \cap A^C$$

The output X_k is recursively found until $X_k = X_{k-1}$ where k = 1,2,3 and so on. The previous output is dilated with a structuring element B. The dilation is then intersected with A compliment that would limit the result to inside the target region. For this, the structuring element B is as follows:

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

We set the pixel value at 0,0 to start with, in this dilation process. The main idea is to dilate the background and find the missing holes. After the background is dilated, the following logical operation is performed that results in the hole locations shown as below:

$$H = \overline{A \cup X_k}$$

The hole locations H is calculated by taking the complement of the union of input and dilated images. The hole locations are then filled by taking the union of H with the input.

The black holes in the deer image can be seen when we display the input image. The number of black dots present inside the deer object can be counted by using the shrinking process. To calculate the number of black dots present in the image, I use a 5x5 filter as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

This filter is used for counting such that if hit, count is 1 and if miss, count is unchanged. This is not a necessary step, as we need to remove all the black holes or dots irrespective of the count.

I applied the hole filling method to the black dots to convert them into white dots. Before the actual method used, I also did some preprocessing wherein I tried eliminating few black dots using the subtractive filter. For this method, first find the black pixels surrounded by the white pixels and then change it's value from 0 to 1. The pattern filter used is shown below:

$$\begin{bmatrix} D & 1 & D \\ 1 & 0 & 1 \\ D & 1 & D \end{bmatrix}$$

where D is don't care term, either 0 or 1. If there is a hit, change the value from 0 to 1.

Now I apply the Closing method, which is reverse of opening, it is dilation followed by erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object. It also helps to smoothen the boundaries. Small holes inside blobs can be filled by performing dilation followed by erosion. This operation is called morphological closing. Thus, this is an important operator from the field of mathematical morphology. It is derived from the fundamental operations of dilation and erosion.
The mathematical expression of dilation and erosion is shown below:

$$Dilation \quad A \oplus B = \{z \mid (\hat{B})z \cap A \ominus \neq \emptyset\}$$

$$Erosion \quad A \ominus B = \{z \mid (B)z \cap A^C \neq \emptyset\}$$

I used the 3x3 structuring element matrix:

$$H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The closing operation is the dilation followed by the erosion using the above structuring element. F is the input and G is the output. Thus,

$$G = (F \oplus H) \ominus H$$

For $\oplus$ and $\ominus$, set the center of H as the original position. $\oplus$ means moving the images to 8 directions, then take the union. $\ominus$ means moving the image to 8 directions, then take the joint.

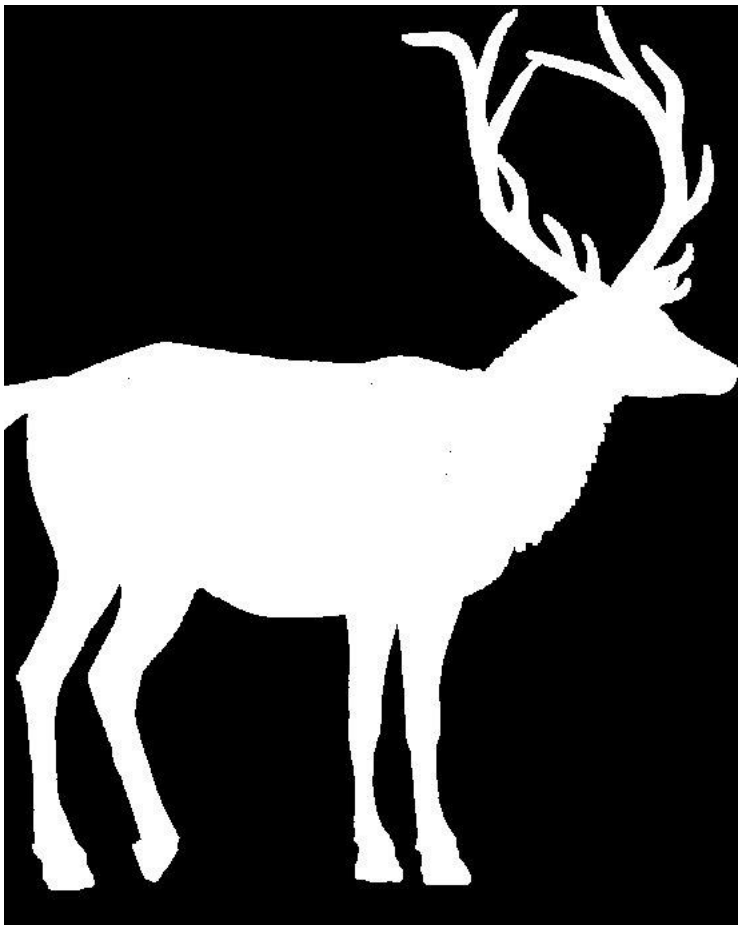
Therefore, we apply the subtractive filter to fill the single pixel hole or black dot and use other types of structure elements to do the closing to get the improved black hole-filling output image.

One additional thing that I tried was to add smoothing boundary filter which is used to smooth the boundary of the object. The closing operation with 2 filter of dilation and erosion method were used in this process.

$$\text{Dilate filter} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\text{Erode filter} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

III. Experimental Results



Input image – deer.raw

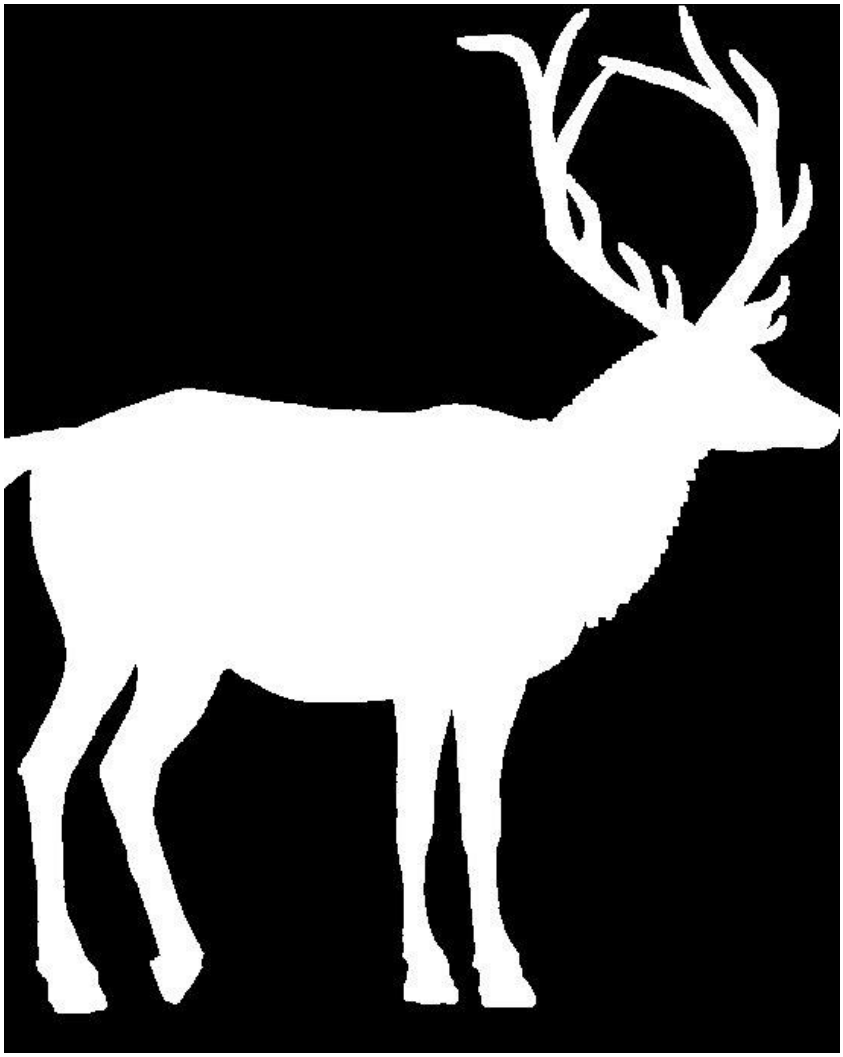
The answer for detection and finding the location of the black dots defect pixel is shown below,

I have found the values of the black dots pixel locations corresponding to the rows and columns of the black dots in the deer image:

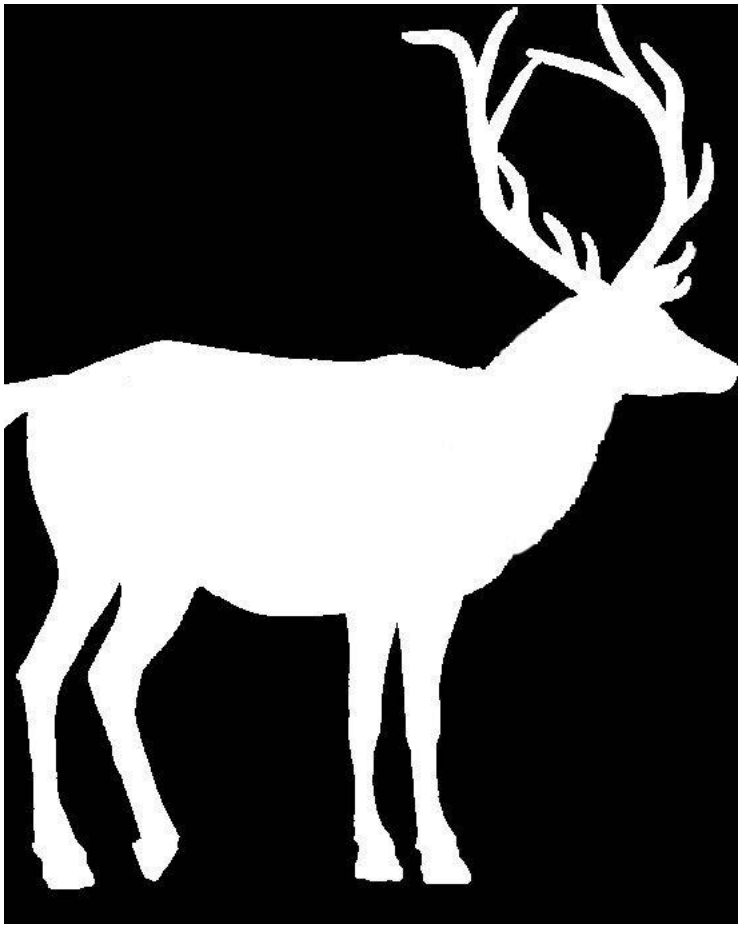
The first answer is the row of the first black dot and the second answer is the column of the first black dot. The third answer corresponds to the row of the second black dot and the forth answer corresponds to the column of the second black dot and hence it is a pattern of rows and columns of the black dots.

```
Brinals-MacBook-Pro:source codes brinalbheda$ g++ -o main 2b_defect_detection_correction.cpp
Brinals-MacBook-Pro:source codes brinalbheda$ ./main deer.raw out.raw 1
Black dot location rows: 207
Black dot location columns: 497
Black dot location rows: 280
Black dot location columns: 92
Black dot location rows: 284
Black dot location columns: 274
Black dot location rows: 335
Black dot location columns: 333
Black dot location rows: 352
Black dot location columns: 330
Number of black dots or defects in the deer body: 5
```

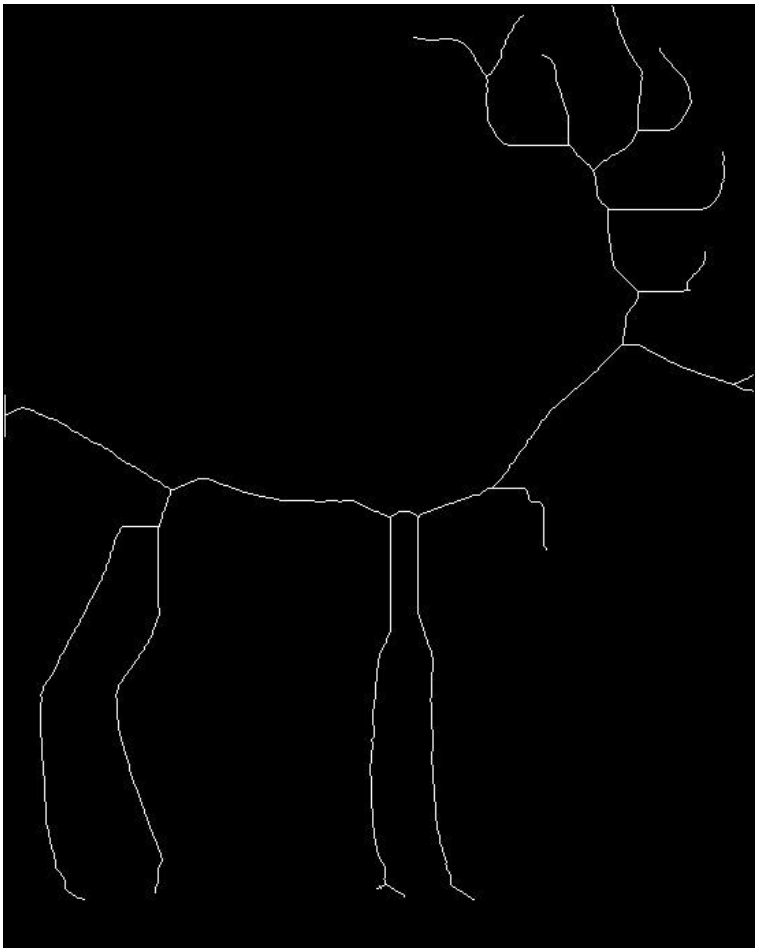
To find these defects locations, I used a 3x3 filter of [1 1 1 , 1 0 1 , 1 1 1] and transverse across each pixel value of the input image and where it caused hit, I displayed the corresponding row and column and increased the count by 1 for every hit. I then converted this pixel location to white by assigning 255 to the pixel value.



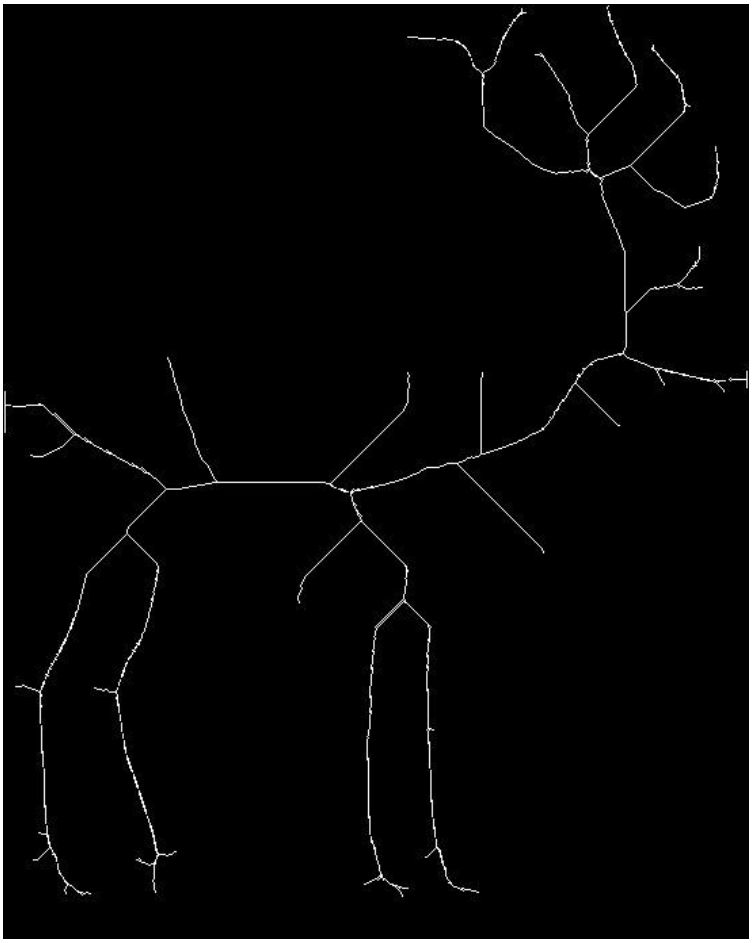
Output image – deer.raw with the black dots removed



Output image – smoothened deer.raw



Output of deer when applied thinning



Output of deer when applied skeletonizing

IV. Discussion

1. Using the morphological process of shrinking, we see that the deer doesn't shrink to a single dot, this proves there are black holes present in the deer body. Hence, the deer is not defectless.
2. Since the deer is not defectless, the defect regions are the ones where there are black dots or holes present on the white object in the foreground, in our case, there are black dots on the deer body. We need to correct them to white dots so that suppose if the deer image is used for product decoration, where the image is enlarged to fit the multiple product sizes later, it should have a clean and clear design of the object for consistent product appeal and not show the black holes enlarged on the object. Say when a poster is needed to make and the size dimensions of the poster are 10 times the current size of the image we possess, the black dots which are present on the deer object in the image will become more prominent and visible to the human eyes as the scaling of the dots will also go 10 times to enlarge the object in the poster. And hence it is

very necessary to remove these defects so that they do not increase the error or affect the visuality or correctness of an image in the future for other applications.

Black holes filling is an important method to be performed as pre-processing in the morphological image processing techniques as it leads to incorrect results for shrinking, thinning or skeletonizing. The image after removing the holes plays an important role in thinning and skeletonizing. The pixels of the object get transformed to the shape of the object as we observe in the output above. The holes may create extra boundaries and hence we do not see the correct shape or form in the thinning or skeletonizing outputs. Thus, by removing these black dots helps us to extract the main shape of the object in the image successfully.

(c) Object Analysis

I. Abstract and Motivation

Morphological image processing is a collection of techniques that are used for analyzing geometrical structures in an image. Morphological operations such as dilation, erosion, opening and closing use a small template known as structuring element to tranverse into an image. These operations can be combined and applied to basic image processing algorithms for boundary extraction, hole filling, extraction of connected components, convex hull and so on.

(Source: https://www.researchgate.net/publication/262732484_Classification_of_Basmati_Rice_Grain_Variety_using_Image_Processing_and_Principal_Component_Analysis)

Rice grain inspection is a procedure to define rice quality in the marketplace. Figure rice.raw is an example image for rice grain type examination. We need to convert the incoming RGB image to grayscale image first, then convert it into a binarized image by applying appropriate threshold and then apply morphological processes. Some pre-processing or post-process may be needed to deal with unwanted dots and holes. Then count the total number of rice grains. Compare the size of rice grains. Rank the grain's size from small to large in terms of type.

In this problem, I have implemented the rice grain inspection to define rice quality using morphological operations explained in detail in section 2(a). Different thresholding algorithms were tested to obtain a perfect binary image. The final thresholding algorithm that gives us the perfect binarization is discussed in detail. Different thresholding algorithms were used as we can see in the input image the different shades of the rice. Most of them are on the whiter range however, one type of the rice is particularly dark and hence the thresholding technique should be such that even the dark rice bunch should be visible in the binarized image. Shrinking operation was used to reduce the rice grains to single dots which helped us with counting the actual number of rice grains in the input image.

II. Approach and Procedure

Convert the input image to the binarized image of 0 as background and 1 as foreground to perform the morphological operations. Also, do some preprocessing and cleaning. We determine a thresholding method to use.

The input image is first read whose dimensions are heigh_size and width_size and BytesPerPixel = 3. Then the image is processed from RGB color to gray-scale image using the luminosity formula given as:

$$Y(i,j) = 0.299 * R(i,j) + 0.587 * G(i,j) + 0.114 * B(i,j)$$

And then converted it to binary image by using some threshold between 0~255 to convert into black and white. The threshold generally taken into consideration is usually 127. However, since this is a special case, we consider implementing different thresholding algorithms and using the one giving the best clear output, especially including the dark colored rice grains. Since we have some darker colors of rice, we may need to use adaptive thresholding. After thresholding, the image will consist of only two intensities i.e. 0 and 1. The 0 intensity pixels are considered as background pixels whereas the 1 intensity pixels are considered as foreground pixels or object pixel values. Then use shrinking to find the number of rice grains. For next question, look at the average size of each part and then rank the size from smallest one and count the whole object area.

We use different algorithms to convert the gray scale image to the binarized image. However, most of the binarization techniques fail because some rice object in the image was on the brighter or darker side than the background and hence could not be displayed properly. The standard thresholding algorithm used would also perform poorly as it would not be able to separate or differentiate between the foreground

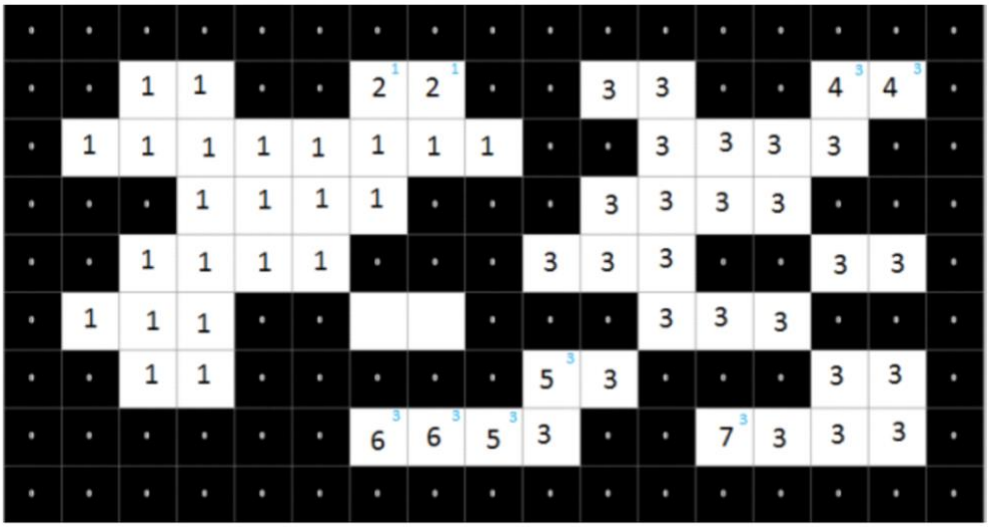
and the background adaptively. A brief analysis of the various algorithms used for thresholding to get a clean output is mentioned in the Discussion below.

For implementing the process of getting a high quality of binarization, firstly split all the three-color channels from RGB to R, G, B respectively. Then apply a median filter of size 3x3 to each channel. This will make the boundaries smooth and reduce the impulse noise in the image. Then apply a mode based binarization technique. This method consists of two steps: a) Count the mode pixel value of the image. In most cases, the mode value accurately represents the background pixel value. b) Set the pixel values as 0 or 1 based on certain condition. Iterate through all the pixels and set the value to 0 if it lies within a certain radius around the mode value, using the knowledge of the background pixel value. If the pixel value lies away from the mode, then it is set to 1. The mode value represents the background pixel value of each channel when I calculated and compared the histogram of each channel.

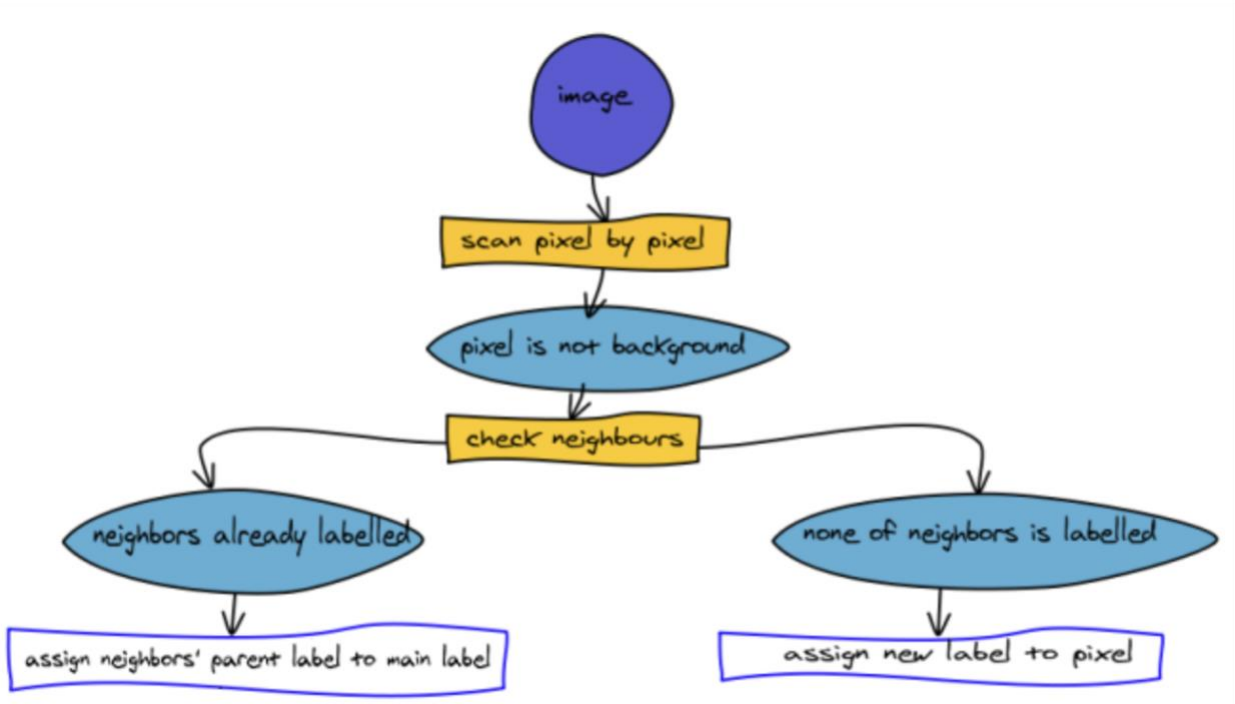
The radius is selected through the trial and error method. The best value of radius was found to be 15. Then create all channels with binary values using the luminosity equation. Apply the median filter with size 5x5 to smoothen the grain surface and to remove the small sized holes in the binary image. The values obtained after the synthesis is not binary, so we perform mode based binarization with a radius of 15 to the obtained synthesized image. This process leads to the output of a smooth and cleaned binarized image. Using this method, we find the count 55 rice grains by shrinking the image.

We apply thinning to the binary image obtained above. Apply thinning using the conditional and unconditional pattern tables as mentioned in the procedure for thinning in problem 2(a).

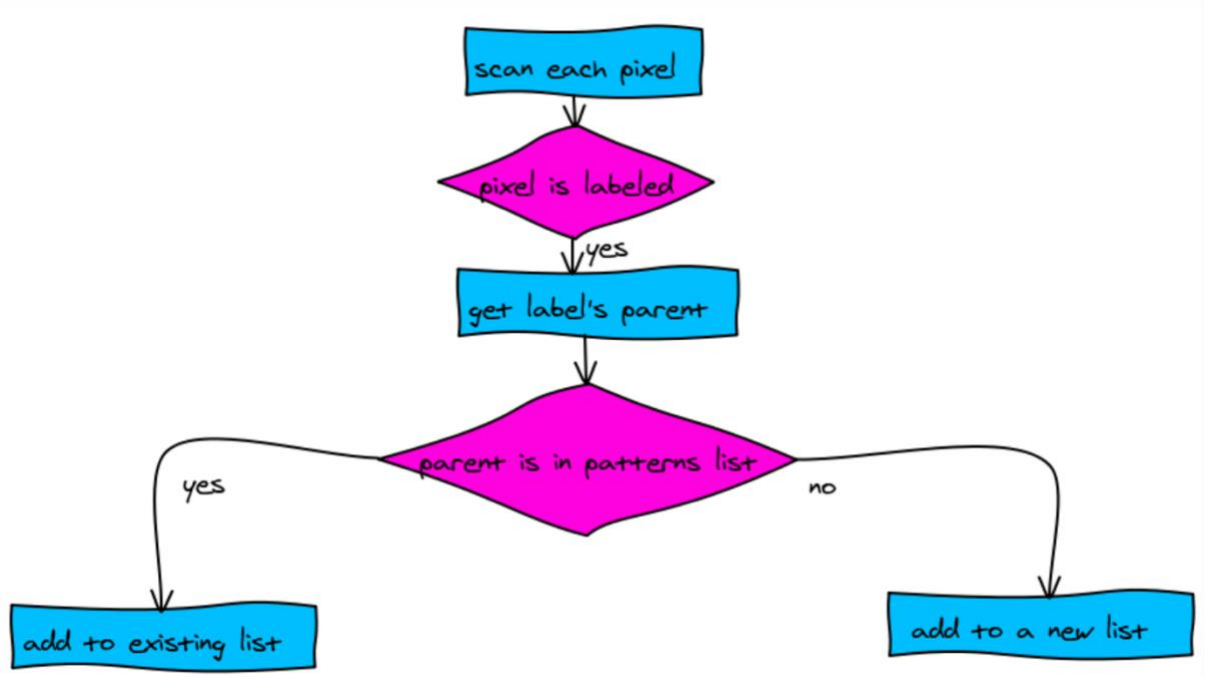
Connected Component Analysis:
Connected Component Analysis scans the entire image and group the pixels into components according to pixel connectivity.
(Source:https://www.codeproject.com/Articles/336915/Connected-Component-Labeling- Algorithm)



Sample output of Connected Component Labelling



Flowchart for the first stage



Flowchart for the second stage

(Source: https://en.wikipedia.org/wiki/Connected-component_labeling)

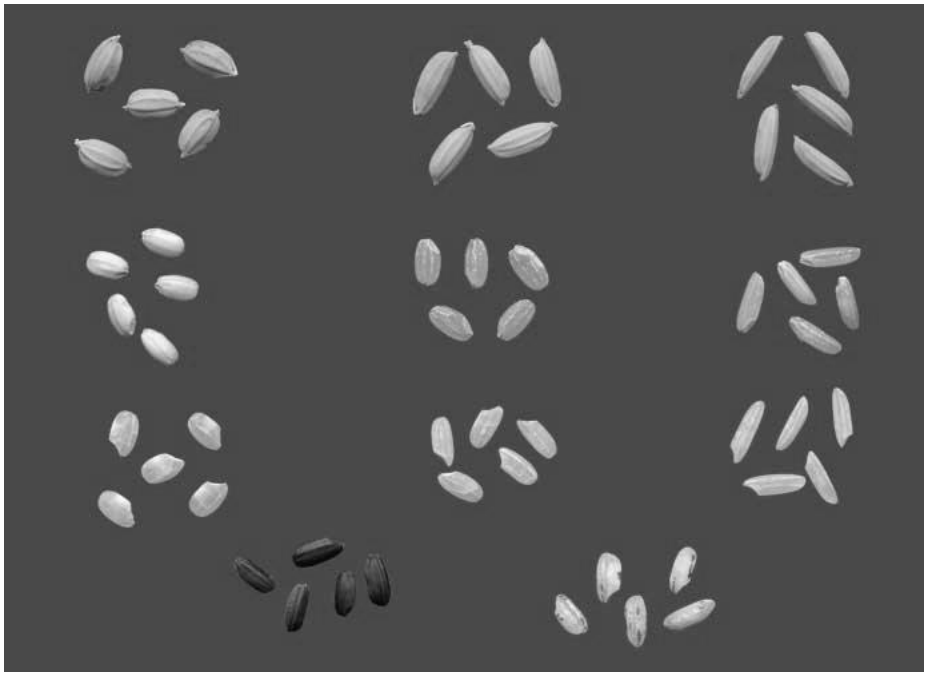
In Connected Component Analysis, all pixels that are connected to each other are labelled with a single pixel value whereas pixels which are not connected have different values. The entire image is canned from top to bottom and left to right to identify pixels having connectivity i.e. same intensity values. Connected Component Analysis is mainly used in Automated Image analysis applications.
While labelling the elements in any scan for the connected component analysis, four neighbors of the center pixel are considered. Those are the one above the center element, left one and left diagonal and right diagonal terms. The labels are then assigned as follows:

a) If all 4 neighbors of the center pixel are 0, assign a new label to the center element.
b) If a single element in the 4 neighborhood has a non-zero value, assign that value to the center pixel.
c) If more than 2 elements of the four neighborhoods have non-zero values, assign the minimum value to the center pixel.

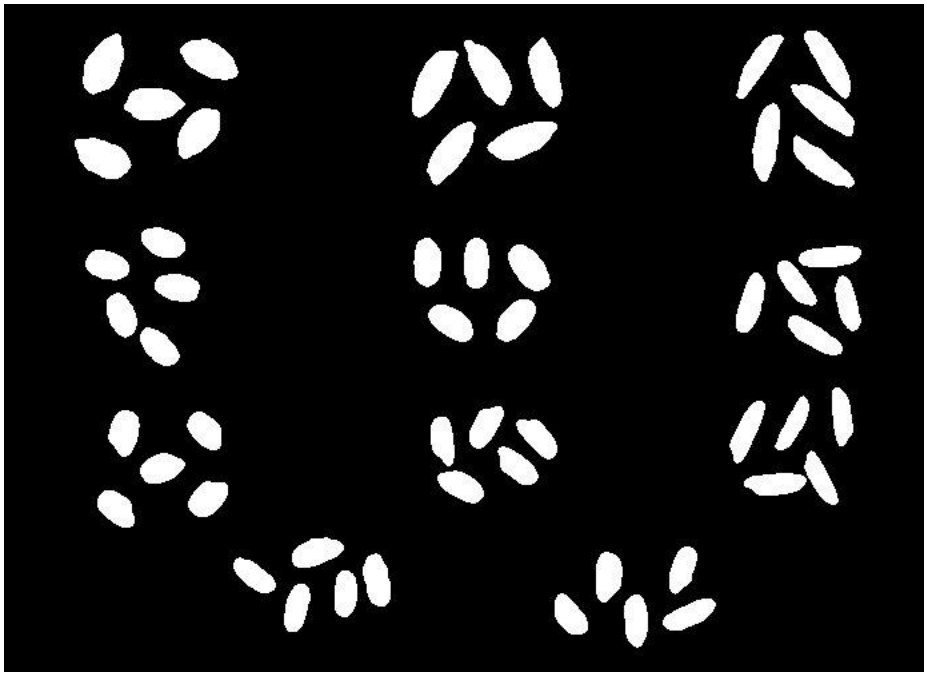
After first scan, the equivalent label pairs are placed in a look up table and in the second scan each label is replaced by the equivalent look up table value. This is how, the components in the image are separated using separate labels in the image.

III. Experimental Results

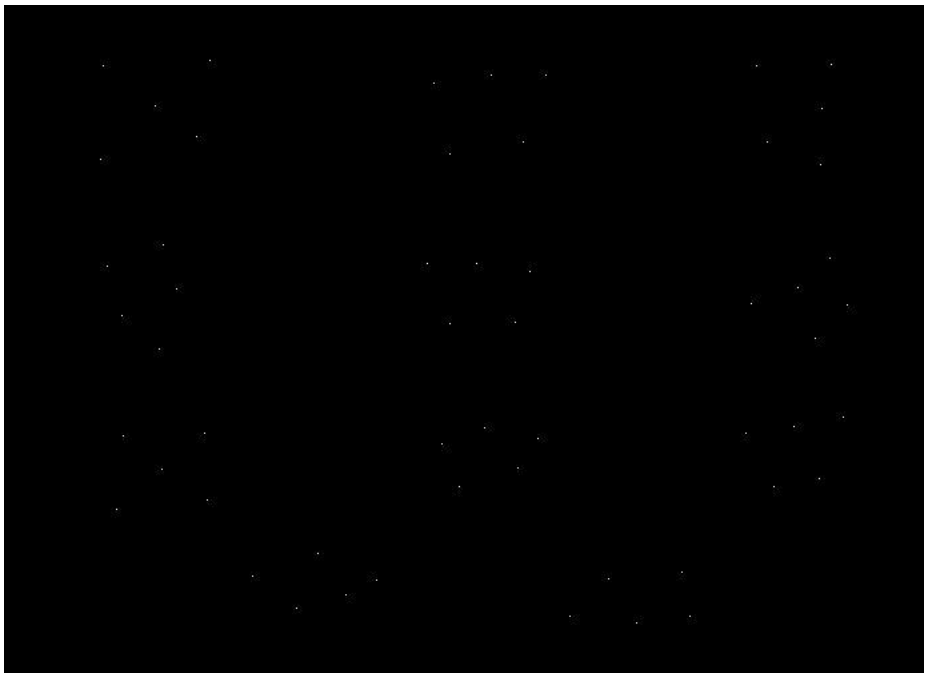
Input image – rice.raw



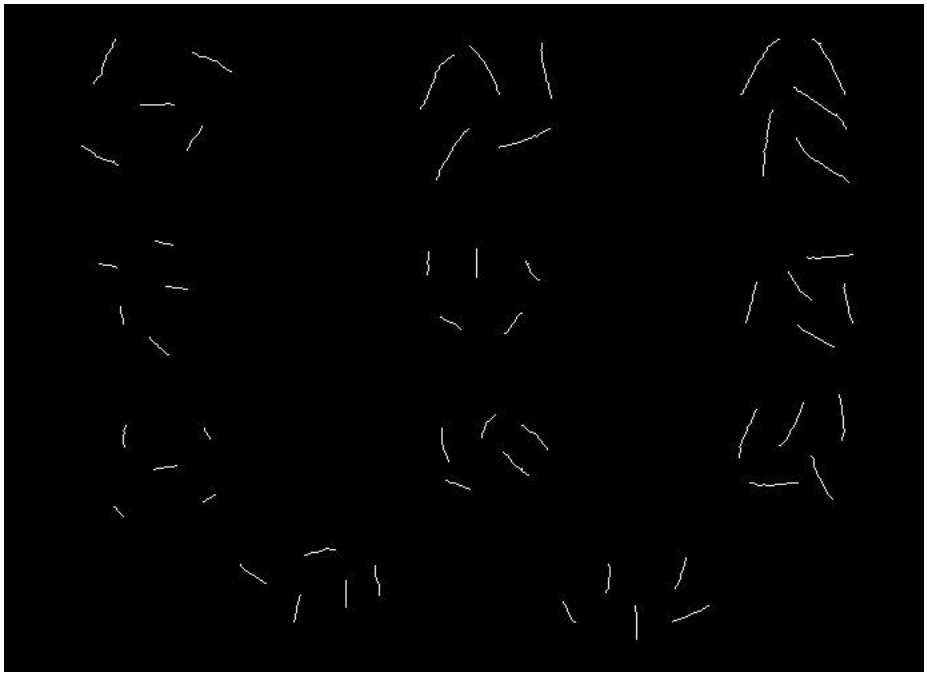
Gray scale input image – rice.raw



Binarized rice image with 1 as object and 0 as background



Output of rice when applied shrinking



Output of rice when applied thinning

```
[Brinals-MacBook-Pro:HW2 brinalbheda$ ./main HW\ 2\ images/Rice.raw rice_out 3 69]
0 500 5
The  number of rice grains are: 55
Brinals-MacBook-Pro:HW2 brinalbheda$
```

Output showing total number of rice grains

IV. Discussion

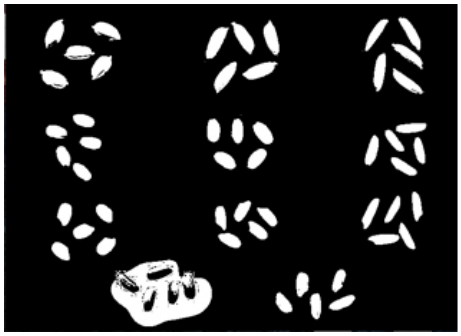
1. The total number of rice grains is 55.
2. The size of the rice grains is compared by using thinning and using connected component labelling, we rank the grain's size from small to large in terms of type that is each rice grains stack by comparing the average size. Considering the rice stack denotation as the row and column number. Starting from the smallest one that is rank 1 – 3rd row and 1st column, rank 2 – 2nd row and 1st column, rank 3 – 2nd row and 2nd column, rank 4 – 4th row and 1st column, rank 5 – 3rd row and 2nd column, rank 6 – 4th row and 2nd column, rank 7 – 1st row and 1st column, rank 8 – 2nd row and 3rd column, rank 9 – 3rd row and 3rd column, rank 10 – 1st row and 2nd column, rank 11 which corresponds to largest area – 1st row and 3rd column.

Firstly, I'm discussing about the different thresholding algorithms used and the reason why they failed. The list of algorithms tried and failed along with their images

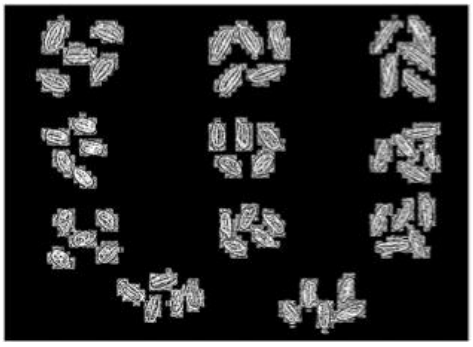
showing the kind of thresholding the algorithms implemented on the rice image are shown below:

Fixed Threshold

Adaptive Mean



Adaptive Gaussian



Niblack

The main reason the algorithms above failed and are flawed for the rice image was because they could mask the background from the foreground. Due to this reason, we see the bottom left portion of the rice grains missing in the first figure. Other methods consider windows to determine local threshold rather than the global threshold. Thus, the objects darker than the foreground were binarized incorrectly as we see in the second and third figures. The output for Niblack's thresholding method isn't good because of the background noise. Thus, we use the method mentioned in the procedure section which helps us implement the thresholding in a successful manner giving a good binarized output.

The method used to count the number of rice grains is shrinking and described above in the procedure. We can count the total number of dots or points in the image as each rice grain shrinks to a single dot after applying the shrinking morphological operation. Thus, the number of dots in the shrinked output image represents the number of rice grains in the original image. We can use erosion and dilation as well for calculating the number of rice grains such that after using shrinking the object does not reduce to more than a single point. The shrinking process helps to count the total number of rice grains however it does not assign a unique number to each rice grain.

For counting the number of rice grains in the image, there are multiple approaches to find the total number. One of them is connected component analysis which is explain in the 2(b) part in detail. This method compares its neighbors and assigns label with particular neighboring pixels. But the algorithm was very tedious. Hence, I used a better and efficient way to find the number of rice grains by shrinking the image using different threshold values by observing pixel intensities.

The comparison of rice grains can be made by first applying connected component labelling. Each grain in the binary image would correspond to a digit between 1 and 56. We would then proceed to count the number of such digits occurring in the image and store it in a hash map with the grain number as key and label count as its value. This would give us the grain size in terms of number of pixels. Larger grains would have larger values and smaller grains would have lesser number of pixels. We would then proceed with calculating the average of each type of rice grain and sort it. This would give us the relative size comparison of each grain type. The grain type would be stored in line as 1 to 11 and then depending on the average size of the stack of similar rice, the stack of rice would be ranked from 1 to 11 corresponding from small to large.

The rice grain can be categorized based on their physical parameters such as length, area and color. Morphological feature can be extracted and used for training a simple classifier. I learned reading the paper published mentioned in the source above that the following features can be useful to consider while classifying rice grains:

1. **Area**: This calculation can be made by counting the number of pixels in the connected component labelled image.
2. **Major Axis length**: This could be found by applying thinning and connected component labelling. The number of pixels under each label gives us the value for major axis length.
3. **Minor Axis length**: This could be found by computing the minimum bounding boxes for the binarized image. The bounding boxes would enclose the grain and the breath of such bounding box would give us the value for minor axis length.
4. **Eccentricity**: This gives us a measure of how circular a rice grain is. This can be calculated by taking the ration of distance between the foci and its major axis length.
5. **Perimeter**: This can be computed by applying canny edge filter. These features provide a rich insight about the grain types.

Instead of the whole procedure mentioned above doing in detail coding for every step, another easier way is to calculate the total number of rice grains and the rank of each type of rice depending on the size by using the morphological functions mentioned below:

MATLAB: https://www.mathworks.com/help/images/morphological-filtering.html

OpenCV: https://docs.opencv.org/3.4.1/d9/d61/tutorial_py_morphological_ops.html

However, I have not implemented my results using the built-in functions, I have implemented the codes by myself as mentioned in the approach above. Thank you.