EE569 Digital Image Processing

HOMEWORK #2

Issued: 2/7/2017 **Due: 2/26/2017**

Problem 1: Geometric Image Manipulation (40%)

(a) Geometric Warping

I. Motivation

Geometric warping of an image is the process of modifying the geometric properties of an image. This includes transforming the size, shape, and updating the pixel value of the image. Geometric warping helps in improving the quality of an image. This process serves as a corrective measure to image distortions such as pincrush distortion and barrel distortion. Also the process is important to transforming an image from the coordinate space of the imaging sensor to another one. Geometric warping also improves the attractiveness of an image to human eyes.

II. Approach

The sample image shows that the four quadrants of the images were compressed or reduced to triangles. Hence I decided to divided the image into 4 quadrants and then applied bilinear interpolation to find the new pixel values for the triangle pixels. Algorithm 1 below further explains my approach.

Algorithm 1: Geometric Warping

```
Data : Input image F and its dimension ($N_x$ , $N_Y$).
Create 2D array G with dimension ($N_x$ , $N_Y$) for the new image
Partition F into four major quadrants $Q_1$ , $Q_2$ , $Q_3$ , and $Q_4$
Create four triangles $T_1$ , $T_2$ , $T_3$ , and $T_4$ in G by bisecting the four major quadrants (pick the triangles closer
    to the center line).

For each quadrant $Q_i$ in F
    Transform the pixels in quadrant $Q_i$ of F to pixel values in triangle $T_i$ of G.
    For each row r in $Q_i$ and $T_i$
        Apply bilinear interpolation to reduce row r in $Q_i$ to row r in $T_i$

Output : Save the new image array G to output image.
```

III. Result

Figures (1) and (2) below shows the result obtained from the geometric warping.

(a). Before Warping

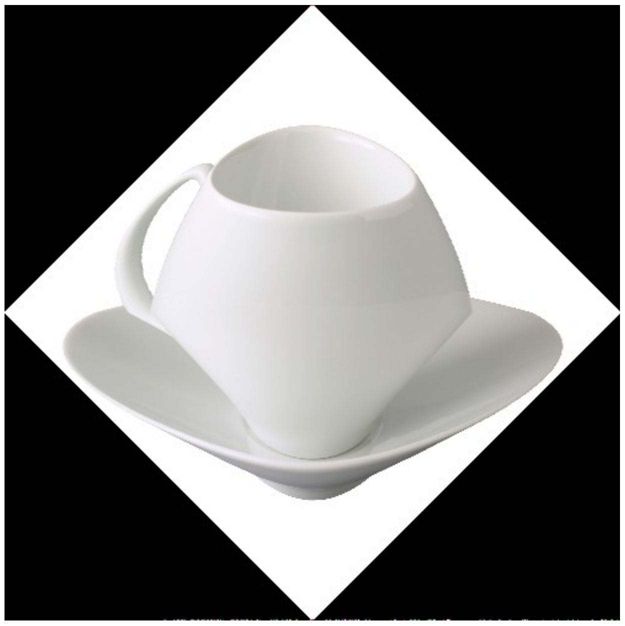

(b). After Warping

Figure (1): Geometric Warping with cup1 image.

(a). Before Warping

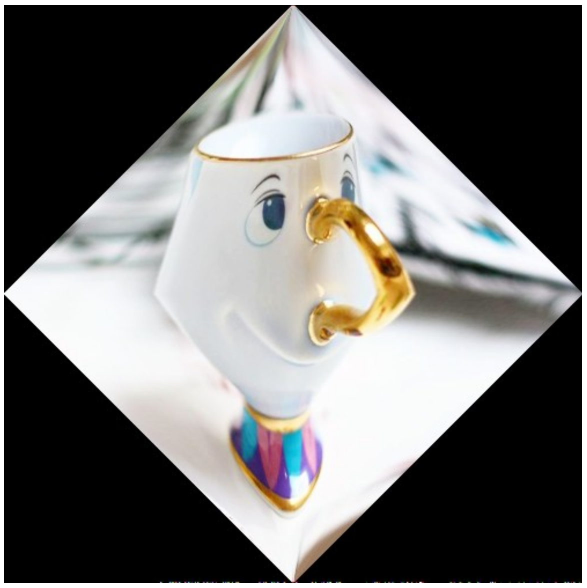

(b). After Warping

Figure (2): Geometric Warping with cup2 image.

IV. Discussion

Geometric warping requires transforming the pixel locations of the image to a new location. This involves picking the control points and transforming the image based on the control points. For the transformation above, I tried picking 8 triangles as the control points but this did not work. The image obtained from this approach shows no compression along th edges of the image. The sample image shows that image side was compressed along with the warping process. This prompted my choice to compress the image along the rows. The approach yielded the desired result.

(b). Puzzle Matching

I. Motivation

This task involves applying basic geometrical modification methods namely translation, rotation, and scaling to fix an image piece into its location to make a complete image. The process involves putting together image fragments as a whole image. This is an essential part of designing a puzzle game.

II. Approach

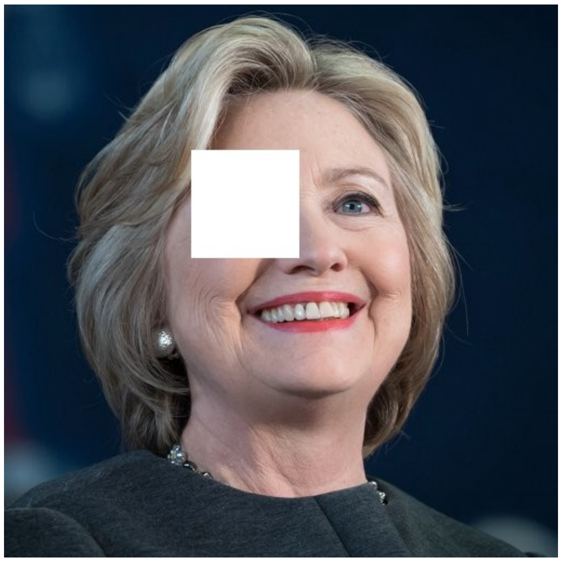
The first thing I did was to identify the configuration of the pieces and rotate the image to a flat configuration. Then I identified the location and dimension of holes in the image and I manipulated the pieces by scaling to fit the holes in the image. Then I copied the pieces into the holes. I used bilinear interpolation to find the pixel values at fractional positions. Algorithm 2 below shows the process of fixing the pieces in the holes.

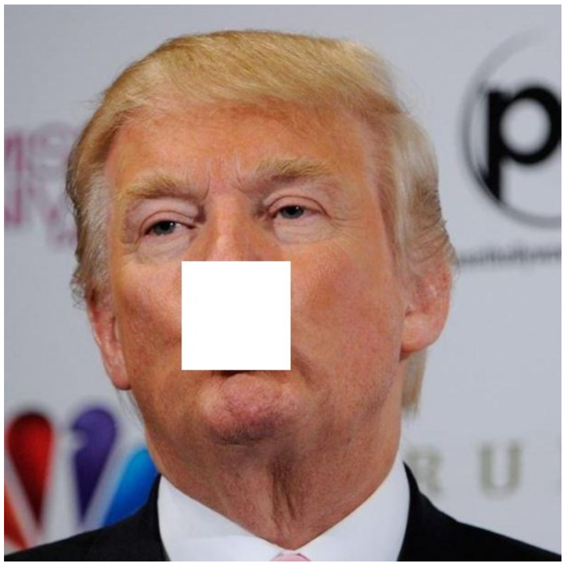
Algorithm 2: Puzzle Matching

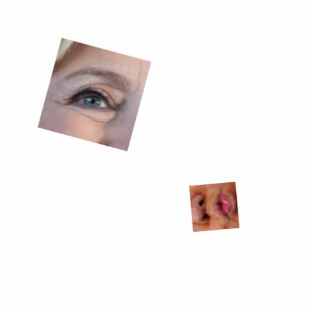
```
Data : Input image F with hole and its dimension ($N_X$ , $N_Y$).
       Input image piece G to be fixed in the hole.
  •  Determine the size configuration of piece image G.
  •  Rotate and translate the piece to assume a flat configuration.
  •  Determine the size and location of holes in F.
  •  Scale the size of piece G to the size of holes.
  •  Copy the scaled image to the hole on F.

Output : Save the new image array F to output image.
```

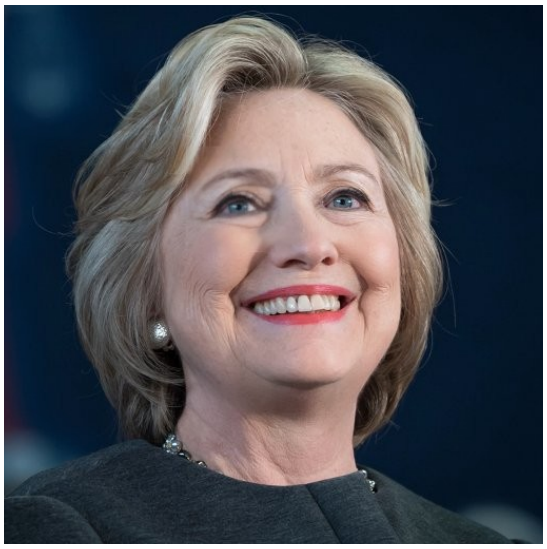
III. Results

Figure (3) below shows the image fragments and the combination after fixing the puzzle.


(a). Hillary’s image (Before Insertion)


(b). Trump’s image (Before insertion)


(c). Puzzle Pieces


(d). Hillary’s Image (After Insertion)

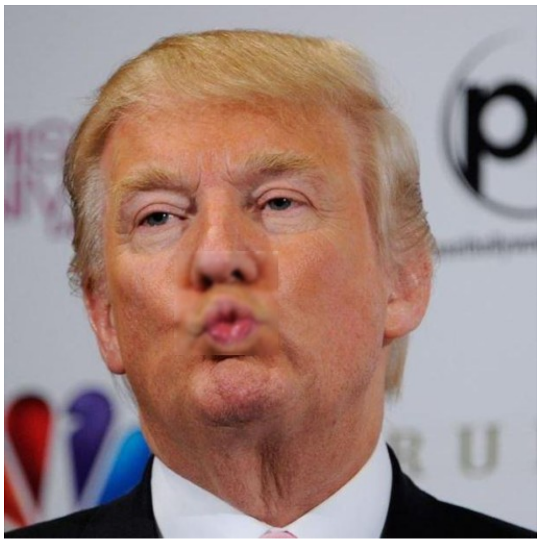

(e). Trump’s image (After insertion)

Figure (3). Puzzle Matching for Trump’s image and Hillary’s image.

I found the location of the holes in the initial image. Table 1 below shows the corners of the holes. Table 2 shows the corners of the piece.

Table 1. Location of holes in initial images.

LOCATION	**Hillary's Image**	**Trump's Image**
Left Top	[x,y] = [135, 173]	[x,y] = [236, 163]
Left Down	[x,y] = [234, 173]	[x,y] = [335, 163]
Right Top	[x,y] = [135, 272]	[x,y] = [236, 262]
Right Down	[x,y] = [234, 272]	[x,y] = [335, 262]

Table 2. Location of image piece for puzzle matching.

LOCATION	**Hillary's Piece**	**Trump's Piece**
Corner 1	[x,y] = [57, 96]	[x,y] = [290, 366]
Corner 2	[x,y] = [240, 199]	[x,y] = [371, 308]
Corner 3	[x,y] = [200, 57]	[x,y] = [296, 302]
Corner 4	[x,y] = [96, 241]	[x,y] = [359, 383]

IV. Discussion

For Trump's image, I had to scale up the piece because the size of given piece is smaller than the hole on the initial image. The process of scaling up with bilinear interpolation reduced the resolution made the piece a bit blurry as compared to the unscaled one. This created a slight noticeable difference when I inserted the piece into the hole.
For Hillary's image, the piece was scaled down . I applied bilinear interpolation to the piece and this did not make the piece blurry. Hence it appears like there is no noticeable different between the patch and the remaining part of the image after fixing the puzzle.
The major challenge I encountered was how to make smooth transition between the inserted scaled piece and neighboring pixels. I simply copied the neighboring pixels to the boundary pixels to avoid poor transition.

(c). Homographic Transformation and Image Overlay.

I. Motivation

Homographic Transformation is the process of projecting an image at a certain angle onto a base. We can then apply overlaying methods to synthesize the projected image onto the base layer. The combination of these two methods is essential merging two different images into a single piece. It also adds aesthetic effects to the image and improves it quality of the image to human eyes.

II. Approach

The homographic transformation uses the homographic transformation matrix to convert a pixel P_1 from the original coordinate to P_2 on the projected surface. The points P_1 and P_2 are the control points for this transformation and the homographic transformation matrix H transforms the image as shown in equation (1) below.

$$P_2 = H\ P_1 \tag{1}$$

I picked the control points and I found the values of the homographic transformation matrix to be as follows. I used MATLAB to solve for the values of homographic transformation H. The values are shown below.

$H_{11}=1.9767$	$H_{12}=0.7422$	$H_{13}=363.00$
$H_{21}=0.7410$	$H_{22}=0.0501$	$H_{23}=571.00$
$H_{31}=0.0026$	$H_{32}=0.0000$	$H_{33}=1.0000$

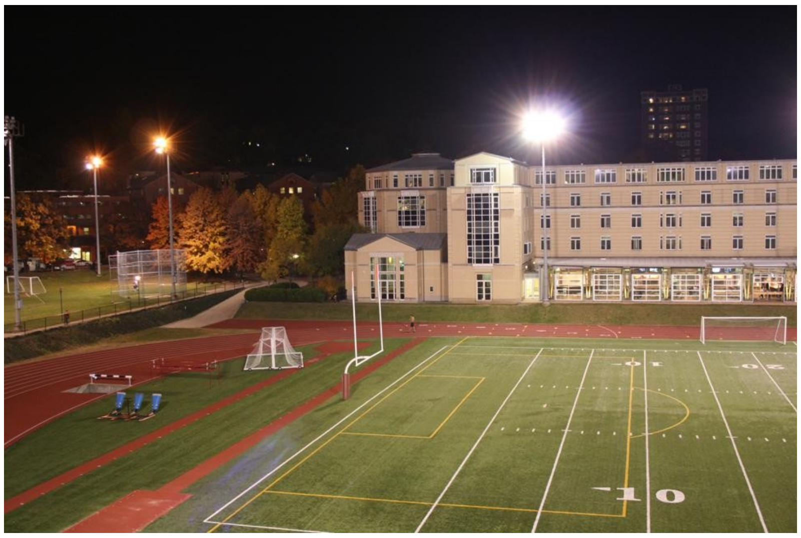
After projecting the top image, I transferred the image from the top layer to the base layer. Algorithm 3 below shows the approach for this task.

<u>Algorithm 3: Homographic Transformation with Overlay method</u>

```
Data : Top image (Trojan) F with hole and its dimension ($N_X$ , $N_Y$).
       Base image G and its dimension ($M_X$ , $M_Y$).
  •  Initialize image array L  with dimension ($M_X$ , $M_Y$) for the projection
  •  Pick the control points for the transform of the top image
  •  Compute the holographic transformation matrix H.
  •  Set every pixel in L to 255
For every pixel (i, j) in image F
  •  Find the homographic transformation of top image
     [a,b] = H  *  [i,j]

For every pixel (a,b) in L
     if L(a,b) != 255
        Set G(a,b) = L(a,b)
     end

Output : Save the new image array F to output image.
```

III. Result

Figure (4) below shows the result of the homographic transformation and overlay method that prints the “Trojans” text image on the field image.


(a). Field Image

(b). Trojans text image

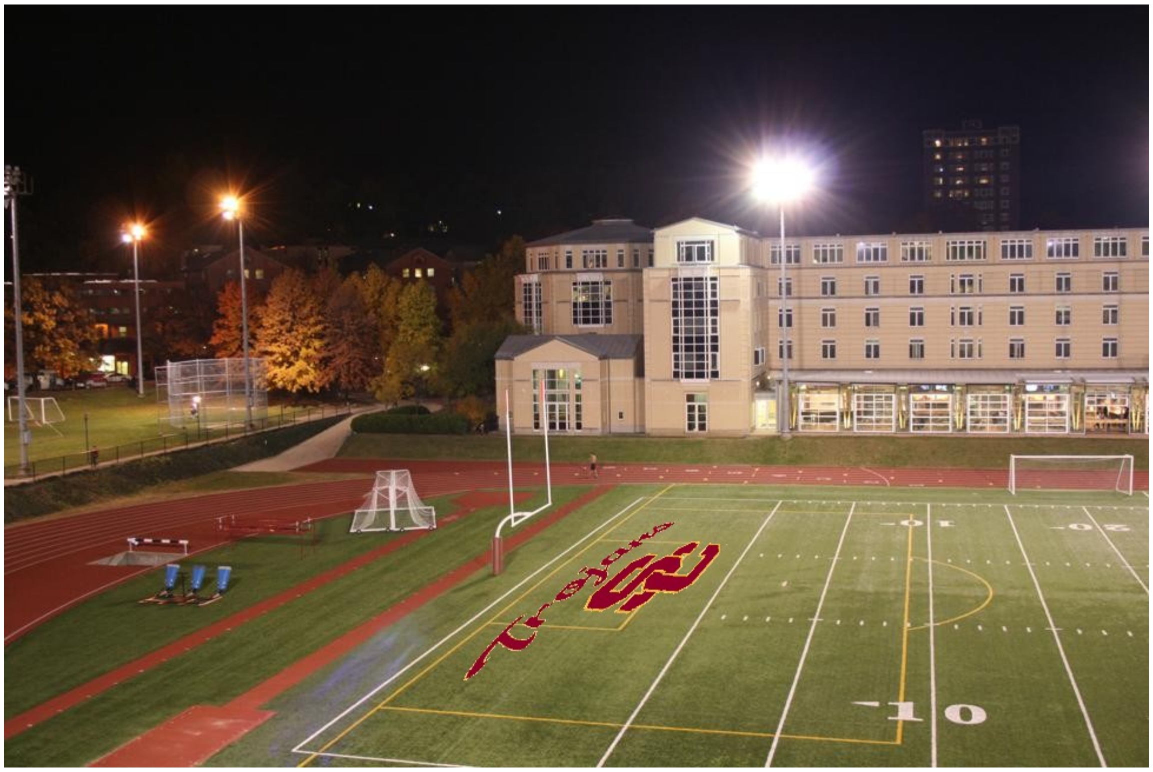

(c). Field image with the projection of Trojans text image.

Figure (4): Homographic Projection with Overlay method.

IV. Discussion

Figure (4) above shows the result of homographic projections with overlay methods. The image shows that outcome after projecting the Trojans text image on a field image. The transformation projected the image on to a flat surface. When I transferred the projected image to the base layer. I noticed some extra white pixels along the edges of the project Trojans text image. I applied a threshold pixel value to filter off these pixels and improve the quality of the image.

Problem 2: Digital Halftoning (30 %)

(a). Dithering

I. Motivation

Dithering is the process of representing unavailable colors with few available colors. Digital Halftoning uses only two possible pixel values to represent a digital image. Despite having just two possible pixel values, the values are arranged such the human eyes can perceive more than two colors. This approach is very important to black and white printing. This is a case where we only have two possible colors for printing an image. Despite the small number of colors, image dithering helps to create more shades of color. This process is very common with display hardware including video adapters, liquid crystal displays (LCDs), and inkjet printers.

II. Approach

This process involves applying matrix filters to digital images. Bayer matrices are commonly used for image dithering. The matrices I considered are listed below.

$$I_2=\begin{bmatrix}0 & 2\\ 3 & 1\end{bmatrix}$$

$$I_4=\begin{bmatrix}0 & 8 & 2 & 10\\ 12 & 4 & 14 & 6\\ 3 & 11 & 1 & 9\\ 15 & 7 & 13 & 5\end{bmatrix}$$

$$A_4=\begin{bmatrix}14 & 10 & 11 & 15\\ 9 & 3 & 0 & 4\\ 8 & 2 & 1 & 5\\ 13 & 7 & 6 & 12\end{bmatrix}$$

$$I_8=\begin{bmatrix}0 & 48 & 12 & 60 & 3 & 51 & 15 & 63\\ 32 & 16 & 44 & 28 & 35 & 19 & 47 & 31\\ 8 & 56 & 4 & 52 & 11 & 59 & 7 & 55\\ 40 & 24 & 36 & 20 & 43 & 27 & 39 & 23\\ 2 & 50 & 14 & 62 & 1 & 43 & 27 & 61\\ 34 & 18 & 46 & 30 & 33 & 17 & 45 & 29\\ 10 & 58 & 6 & 54 & 9 & 57 & 5 & 53\\ 42 & 26 & 38 & 22 & 41 & 25 & 37 & 21\end{bmatrix}$$

Algorithms 4 and 5 below shows the procedure for performing dithering.

Algorithm 4: Dithering with Bayer Matrices

```
Data : Image array F and its dimension ($N_x$ , $N_Y$).
          Create an output array G for the dithered image
Pick the dithering matrix ($I_2$, $I_4$, or $I_8$)
For pixel (i, j) in array F
      G(i, j) = Apply dithering filter to F(i,j)
      Apply the dithering to neighboring pixels
      $T(i,j) = \frac{I_N((i \bmod 2),(j \bmod 2)) + 0.5}{N^2} \times 255$
         if ( F(i, j) > T(i , j) )
             G(i, j) = 255
         else
             G(i, j) = 0
         end
Output : Save the new image array G to output image.
```

Algorithm 5 : Dithering with A_4 matrix

```
Data : Image array F and its dimension ($N_x$ , $N_Y$).
          Create an output array G for the dithered image
Pick the dithering matrix $A_4$
For pixel (i, j) in array F
      G(i, j) = Apply dithering filter to F(i,j)
      Apply the dithering to neighboring pixels
      $T(i,j) = \frac{A_4(i \bmod 2, j \bmod 2) + 0.5}{4^2} \times 255$
         if ( F(i, j) > T(i , j) )
             G(i, j) = 255
         else
             G(i, j) = 0
         end
Output : Save the new image array G to output image.
```

Algorithm 6: Proposed Dithering method with four Gray-scale levels

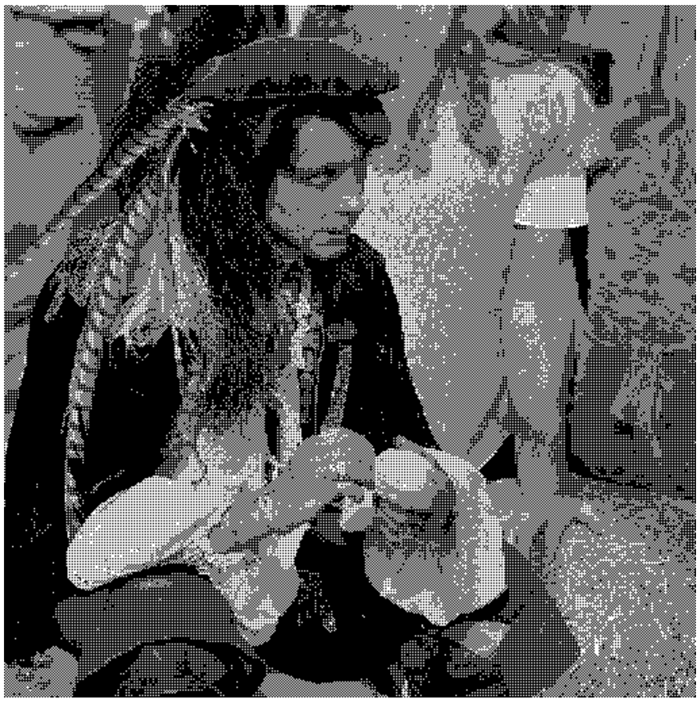
```
Data : Image array F and its dimension ($N_x$ , $N_Y$).
          Create an output array G for the dithered image
For pixel (i, j) in array F
         if ( 0 <= F(i, j) < 42 )
             G(i, j) = 0
         else if ( 42 <= F(i, j) < 127)
             G(i, j) = 85
         else if ( 127 <= F(i, j) < 212)
             G(i, j) = 170
         else
             G(i, j) = 255
         end
Output : Save the new image array G to output image.
```

I used algorithms 4 and 5 for dithering with Bayer matrices and A_4 matrix respectively. I proposed a new algorithm that represents a gray-scale image with only four possible levels (0, 85, 170, 255). Algorithm 6 below explains the approach I used for this multi-value pixel value.

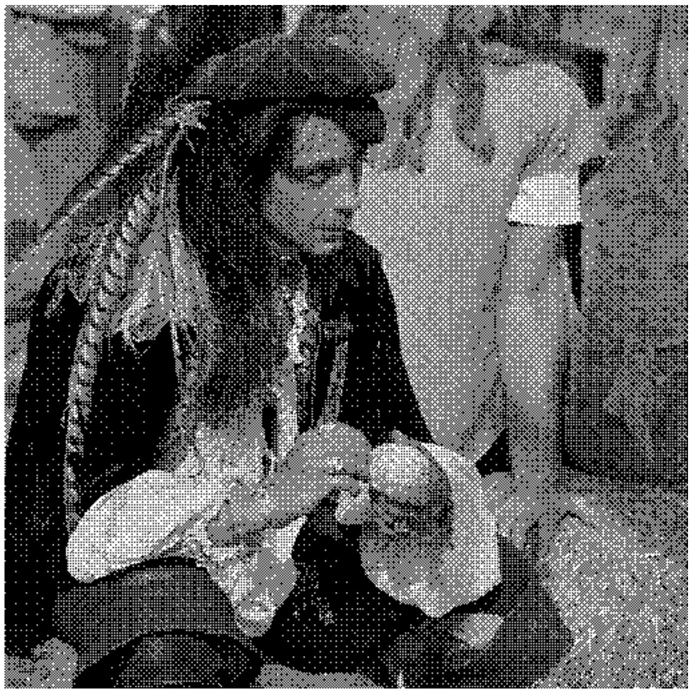
III. Result

Figure (5) below compares the result of using I_2 and I_8 to the man image.

(a). Original man image.



(b). Dithering man image (Bayer I_2 matrix)

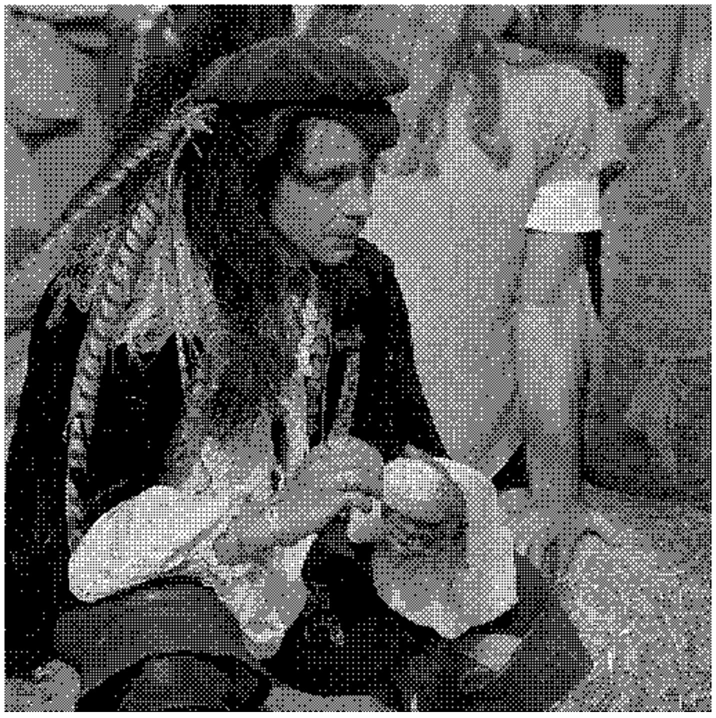


(c). Dithered man image (Bayer I_8 matrix)

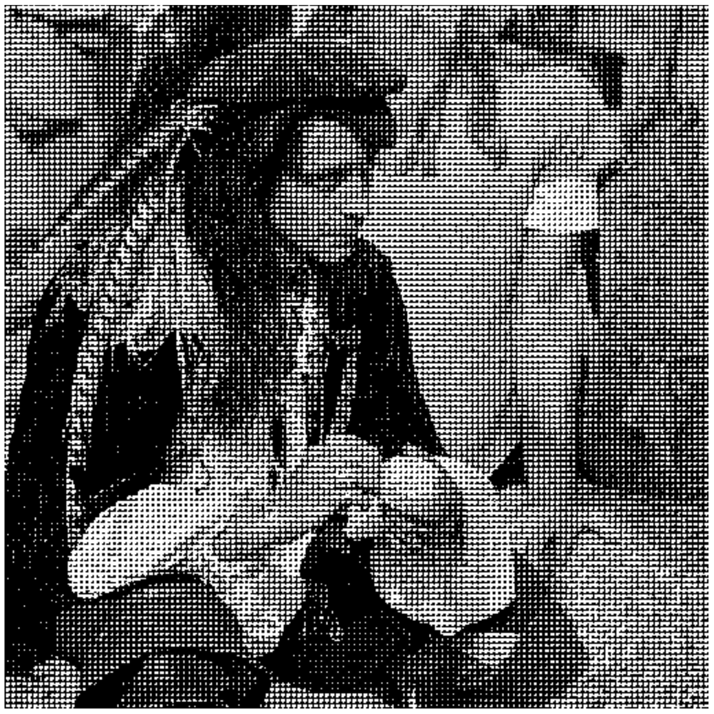
Figure (5) : Image Dithering

Figure (6) below compares the result of using Bayer matrix I_4 and matrix A_4 for dithering.

(a). Original man image.



(b). Dithering man image (Bayer I_4 matrix)



(c). Dithered man image (Bayer A_4 matrix)

Figure (6) : Image Dithering

Figure (7) below shows the result of the proposed method for four gray-scale level encoding.

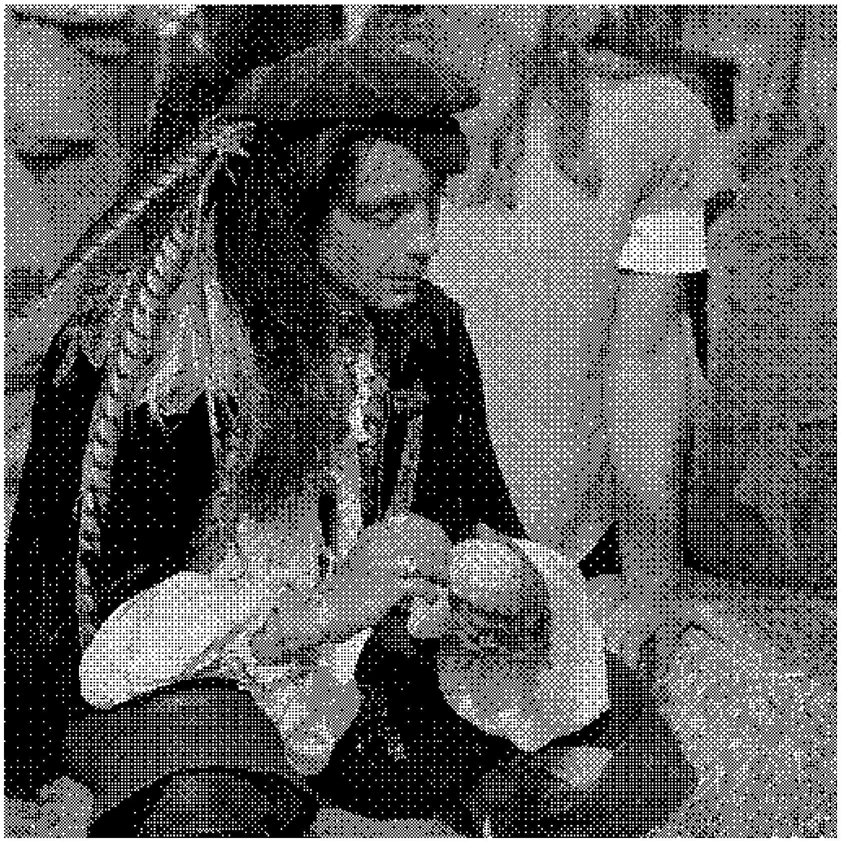


Figure (7) The multi-level coded image with my proposed method.

<u>ANSWERS TO QUESTION</u>

1) Figure (5) shows the result of dithering with I_2 and I_8 matrices.
2) The image produced by dithering with matrix A_4 shows substantial visual artifacts. This is so because the smallest values in A_4 cluster around a small region in the matrix. For the I_4 there is less visual artifacts compared to the A_4 matrix.
3) Algorithm 6 gives the proposed method.

IV. Discussion.

I noticed that the images produced with the matrices about came with different artifacts. This is so because this dithering approach limited the effect of quantization to only a single pixel. The quantization introduced a level of error to the image but this error was not considered with respect to the neighboring pixels. Another issue is that sometimes, this approach leaves the image a large white space and this deteriorates the quality of the dithered image.

(b). Error Diffusion

I. Motivation

The error diffusion method is the half-toning method that spreads quantization error across pixels that are yet to be processed. This process differs from dithering because dithering is a point operation that applies to only a point but error diffusion is an area operation such that the operation at pixel influences what happens at neighboring pixels. The error diffusion method produces more readable images as compared to others. This makes the method suitable for printing on a white and black laser printer.

II. Approach

The approach for this problem involved applying the error diffusion matrix to the image. For the serpentine approach, if pixels were scanned from left-right for a row of pixels then the next row is scanned from right-left. The algorithms below show the procedure for dithering with noise diffusion. The diffusion matrices are shown below.

$$F = \frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix} \qquad F_s = \frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 7 & 0 & 0 \\ 1 & 5 & 3 \end{bmatrix}$$

$$J = \frac{1}{35}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix} \qquad S_k = \frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

<u>Algorithm 7 :   Floyd-Steinberg's Serpentine approach for Error Diffusion</u>

```
Data : Image array L and its dimension (Nx , N_Y).
          Create an output array G for the dithered image
For pixel (i, j) in array L
         if ( F(i, j) < 127)
            G(i, j) = 0
         else
            G(i, j) = 255
         end
         Compute quantization error E(i ,j) = G(i, j) – L(i, j)
         if  (i % 2 == 0)
             Apply diffusion matrix F to error E(i, j) over L
         else
              Apply the diffusion matrix F_S to error E(i, j) over L
         end
Output : Save the new image array G to output image.
```

<u>Algorithm 8 :   Jarvis, Judice, and Ninke's approach for Error Diffusion</u>

```
Data : Image array L and its dimension (Nx , N_Y).
          Create an output array G for the dithered image
```

```
for pixel (i, j) in array L
    if ( F(i, j)  < 127)
        G(i, j) = 0
    else
        G(i, j) = 255
    end
    Compute quantization error E(i ,j) = G(i, j) – L(i, j)
    Apply diffusion matrix J to error E(i, j) over L

Output : Save the new image array G to output image.
```

<u>Algorithm 8 : Stucki's approach for Error Diffusion</u>

```
Data : Image array L and its dimension ($N_x$ , $N_Y$).
       Create an output array G for the dithered image
For pixel (i, j) in array L
    if ( F(i, j)  < 127)
        G(i, j) = 0
    else
        G(i, j) = 255
    end
    Compute quantization error E(i ,j) = G(i, j) – L(i, j)
    Apply diffusion matrix $S_K$ to error E(i, j) over L

Output : Save the new image array G to output image.
```

III. Results

Figures below shows the result of applying error diffusion for dithering.

(a). Original man image

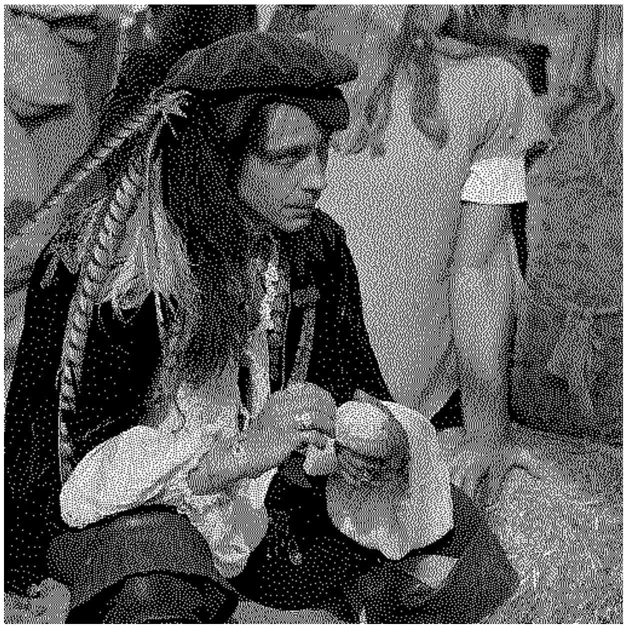
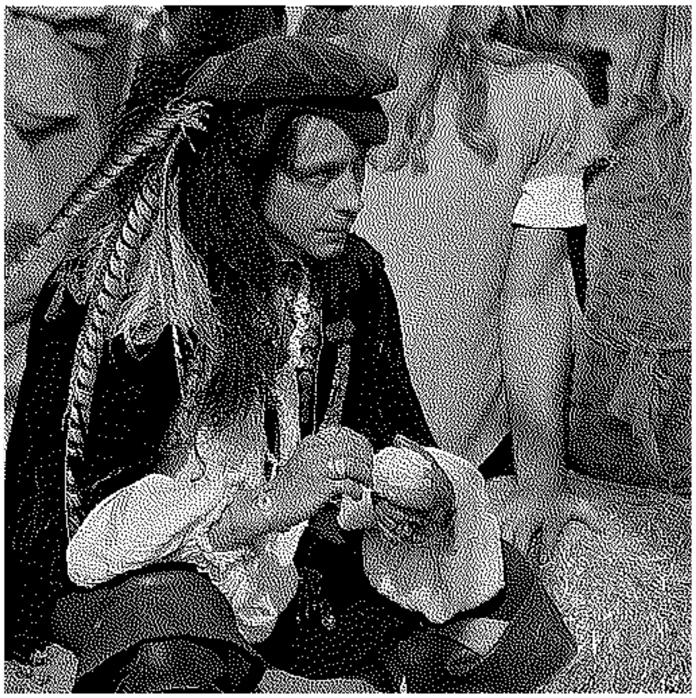
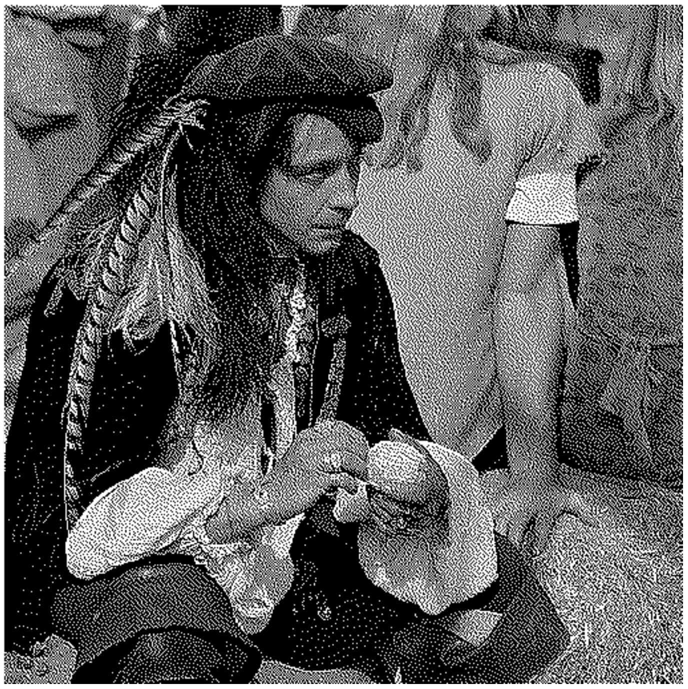

(b). Man image (Floyd-Steinberg's matrix)

Figure (8) : Effect of Floyd-Steinberg's matrix error diffusion on man image.

(a) Original man image



(b). Man image (JJN error diffusion)



(c). Man image (Stucki diffusion error)

Figure (9): Effect of Jarvis, Judice, and Ninke (JJN) and Stucki diffusion error matrices.

IV. Discussion

I prefer the result produced by Stucki diffusion error because the image is the sharpest and clearest. This is so because the matrix is bigger and the quantization error travels faster in this case compared to the JJN matrix or Floyd-Steinberg’s error diffusion matrix. The Floyd-Steinberg’s matrix produced a fine grained dithered image. The diffusion error method appeared to have performed better than the dithering matrices. The error diffusion methods produced less visible artifacts when compared with the dithering matrices.
My idea to improve the quality of the image produced by this error diffusion is informed by picking a matrix that diffuses the error equally along vertical and horizontal axes. This will help prevent the possibility of visible artifact . Another point to consider is to systematically reduce amount of error diffusion to the neighboring pixels. This will reduce the amount of speckle on the output image. My proposed error diffusion matrix P is shown below.

$$P=\frac{1}{9}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Problem 3: Morphological Processing (30%)

(a) Shrinking

I. Motivation

Shrinking the process of reducing solid object with a hole to a single white pixel located at the center of mass. This process also reduces an object with a hole to a ring located at equal distance from the hole and the closest boundary. This is a method for counting the number of solid objects on an image. The solid objects are reduced to a single lonely white pixel at the center of mass and we can use the lonely filter to count the number of such spots.

II. Approach

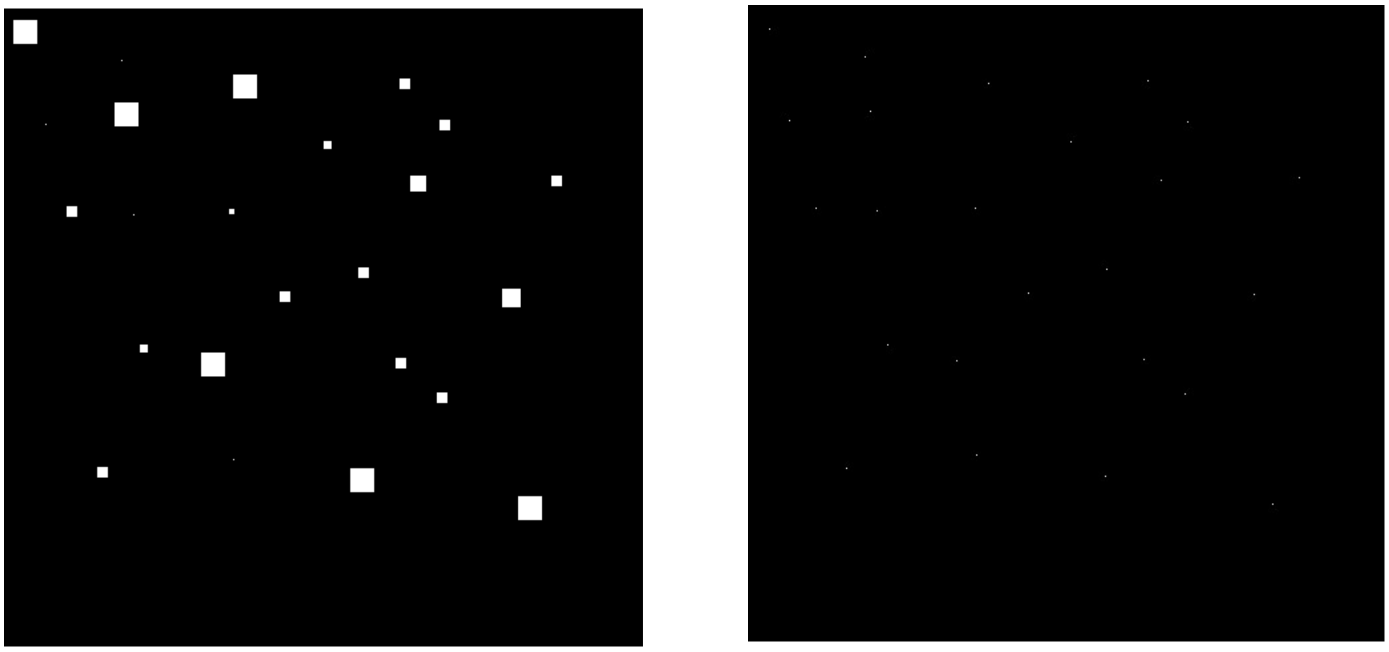
The approach involves applying the conditional mark pattern filter for shrinking. If there is a hit, the pixel is marked. After applying the filter, we then apply the unconditional mark pattern filters to the mark filters. If there is a hit with the unconditional filter, update the pixel value. Repeat this process until the shrinking is complete. Algorithm 9 states the procedure for shrinking.

<u>Algorithm 9 : Binary Image Shrinking</u>

```
 Data : Image array F with dimension ($N_x$ , $N_Y$).
           Create a boolean array M with dimension ($N_x$ , $N_Y$)
           Create an output array G with dimension ($N_x$ , $N_Y$)
Augment image matrix F
Augment image matrix M with padded zero rows and columns
Read the number of white dots in the image
while (Shrinking is not done)
     for pixel (i, j) in array L
          if (There is a hit with shrinking conditional mark pattern)
             M(i, j) = true
          else
             M(i, j) = false
          end
    for pixel (i, j) in array L
          if  (There is a hit with shrinking unconditional mark pattern)
             P = true
          else
             P = false
          end
          $val = P \cup M(i,j)^C$
          if (val == true)
              G(i, j) = X(i, j)
          else
               G(i, j) = 0
          end

    Update the number of dots in the image
    Check if shrinking is done

Output : Save the new image array G to output image.
```

III. Result

Figure (10) below shows the result of shrinking on the squares image. Figure (11) below shows the histogram of the squares. The square size corresponds to the number of iterations



(a). Squares image (Before Shrinking)

(b). Squares image (After Shrinking)

Figure (10) : Effect of shrinking on squares image.

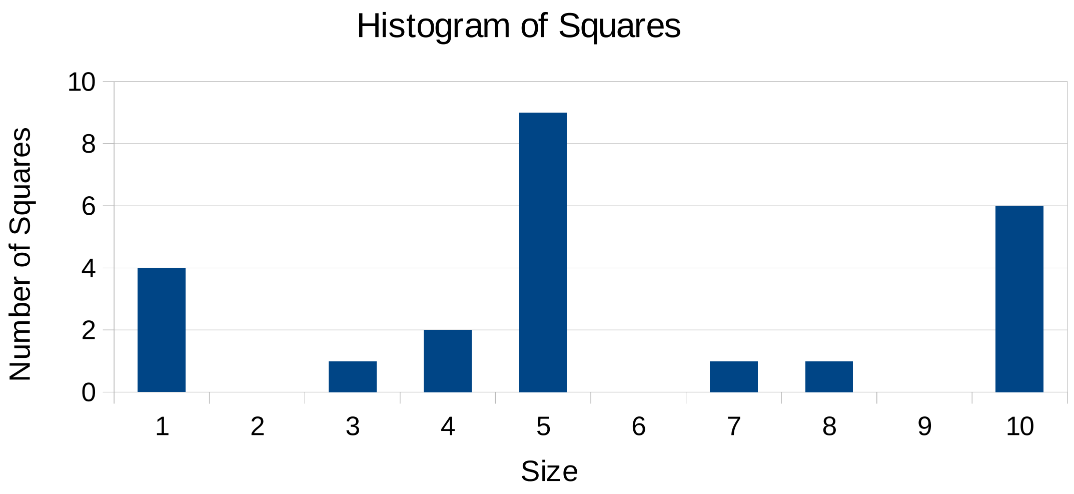


Figure (11) : Histogram of square size on the square images.

ANSWER TO QUESTIONS

1) The total number of squares in the image is 24.
2) There are 7 different square sizes in the image. Table 2 below shows the frequency. The size corresponds to the number of iterations need to shrink the square to a single white pixel.

Square Size	**Frequency**
1	4
3	1
4	2
5	9
7	1
8	1
10	6

IV. Discussion

The shrinking method can be easily applied to count the number of solid objects in an image. This is so because the solid objects are reduced to a white single pixel that can be easily identified and counted. On the other hand, it is quite difficult to use shrinking to count the number of hollow objects in an image. This is so because hollow objects are reduced to a ring of different shapes. The non-uniformity of the rings after shrinking makes it difficult to find a filter for counting the single pixels.
The mask pattern for counting the single white dots is shown below.

$$W = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 255 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

There is a dot if mask pattern W hits a pixel value in the image. It is almost impossible to come up with a single mask that hits all possible rings for hollow objects.

(b) Thinning

I. Motivation

Thinning is the process of reducing a solid object to a minimally connected line located at equal distance from the nearest outer boundaries. The output is different for hollow objects. Thinning reduces hollow objects to a ring at equal distant from the nearest outer boundaries. This method can be used to identify different types of shape on an image.

II. Approach

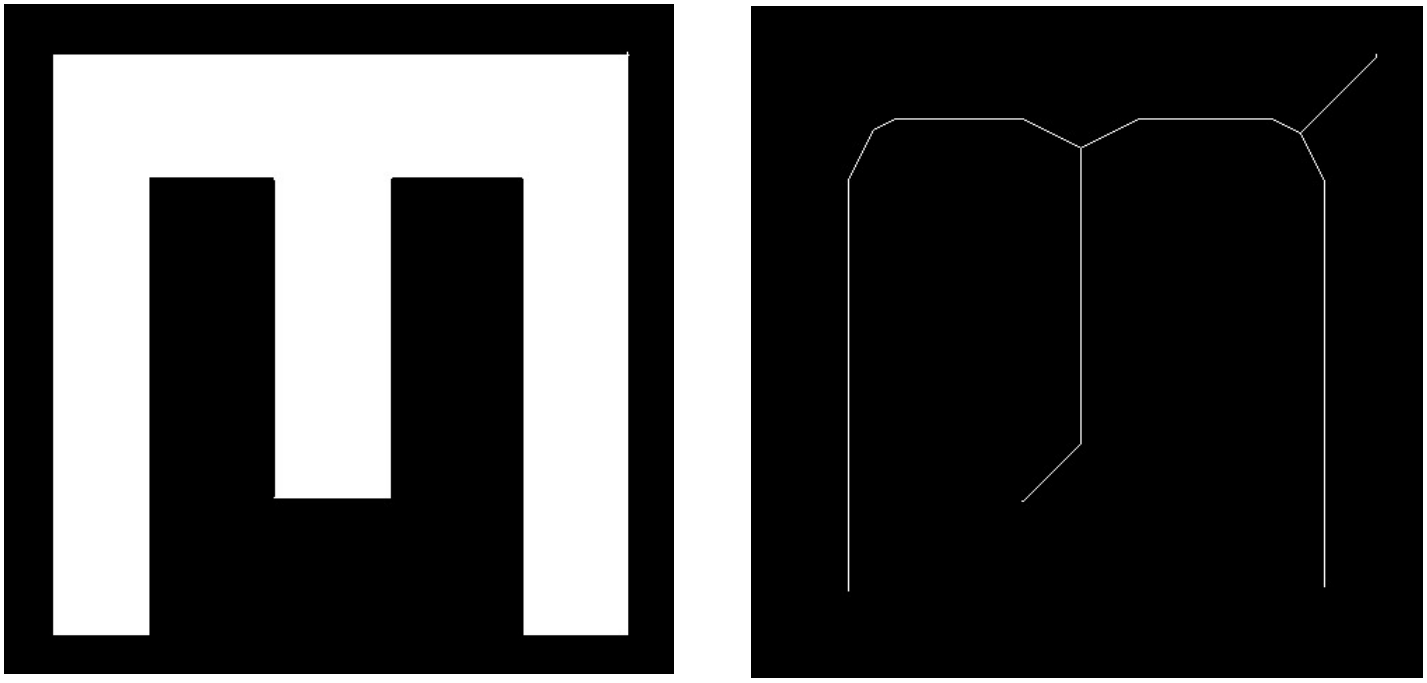
This involves applying the conditional mark pattern filter for thinning. A pixel is marked if any mark pattern hits the pixel. After applying the filter, we then apply the unconditional mark pattern filters to the marked pixels. If there is a hit with the unconditional filter, update the pixel value. Repeat this process until the thinning is complete. Algorithm 10 states the procedure for thinning.

<u>Algorithm 10 : Binary Image Thinning</u>

```
Data : Image array F with dimension (Nx , N_Y).
       Create a boolean array M with dimension (Nx , N_Y)
       Create an output array G with dimension (Nx , N_Y)
Augment image matrix F
Augment image matrix M with padded zero rows and columns
Read the number of white dots in the image
while (Thinning is not done)
   for pixel (i, j) in array L
       if (There is a hit with thinning conditional mark pattern)
         M(i, j) = true
       else
         M(i, j) = false
       end

   for pixel (i, j) in array L
       if (There is a hit with thinning unconditional mark pattern)
         P = true
       else
         P = false
       end
       val = $P \cup M(i,j)^C$
       if (val == true)
          G(i, j) = X(i, j)
       else
          G(i, j) = 0
       end

   Update the number of dots in the image
   Check if thinning is done

Output : Save the new image array G to output image.
```

III. Results

Figure (12) below shows the result of thinning on the letterE image.



(a). LetterE image (Before Thinning) (b). LetterE image (After Thinning)

Figure (12) : Effect of Thinning on LetterE image.

IV. Discussion

The thinning process reduces a circle or square to a single white pixel. It reduces a rectangle or ellipse to a straight line. This eventual shape after thinning differs depending on the This method can be employed to distinguish different shapes in a given image. Unlike the shrinking method, the final state of thinning can be observed to determine the initial object shape.

(c) Skeletonizing

I. Motivation

Skeletonizing reduces an image to a stroke at equal distance from the nearest boundaries. This is similar to the thinning process. For skeletonizing, the images are reduced to their medial axis skeleton. This makes the eventual shape after skeletonizing more distinctive than we have for thinning. Unlike the thinning process, the eventual shape a circle and square after skeletonizing are different.

II. Approach

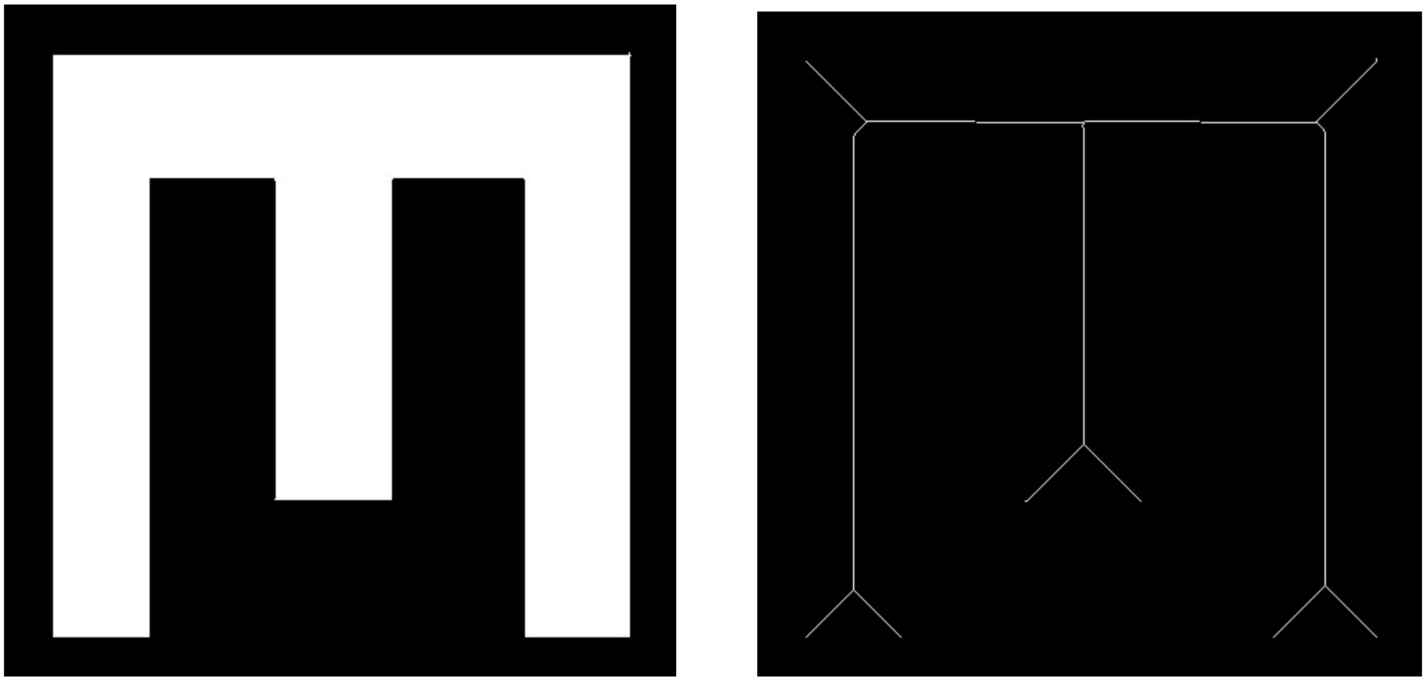
The approach involves applying the conditional mark pattern filter for skeletonizing. If there is a hit, the pixel is marked. After applying the filter, we then apply the unconditional mark pattern filters to the mark filters. If there is a hit with the unconditional filter, update the pixel value. Repeat this process until the skeletonizing is complete. Algorithm 11 states the procedure for skeletonizing.

<u>Algorithm 11: Binary Image Skeletonizing</u>

```
Data : Image array F with dimension ($N_x$ , $N_Y$).
      Create a boolean array M with dimension ($N_x$ , $N_Y$)
      Create an output array G with dimension ($N_x$ , $N_Y$)
Augment image matrix F
Augment image matrix M with padded zero rows and columns
Read the number of white dots in the image
while (Skeletonizing is not done)
   for pixel (i, j) in array L
       if (There is a hit with skeletonizing conditional mark pattern)
         M(i, j) = true
       else
         M(i, j) = false
       end

   for pixel (i, j) in array L
       if (There is a hit with skeletonizing unconditional mark pattern)
         P = true
       else
         P = false
       end
       $val = P \cup M(i,j)^C$
       if (val == true)
          G(i, j) = X(i, j)
       else
          G(i, j) = 0
       end

   Check if skeletonizing is done

Output : Save the new image array G to output image.
```

III. Results

Figure (13) shows the result of skeletonizing on letterE image.



(a). LetterE image (Before Skeletonizing) (b). LetterE image (After Skeletonizing)

Figure (13): Effect of image Skeletonizing on letterE image.

IV. Discussion

Skeletonizing is a better method for distinguishing different shapes. This method reduces an image to its medial axis skeleton. Hence the eventual shapes are highly distinctive. This makes it easier to identify unique shapes than we have in thinning. We can apply this method to identifying the conductive paths on a printed circuit board (PCB). This method can also be applied to counting the number of components on a designed PCB.

(d) Counting Game

I. Motivation

The goal of this task is to identify the number of objects on a given image and different types of object on the image. It could be employed to identify specific targets in the image. This can be achieved by employing different types of morphological processing operations and logical operations to achieve this. There is no single approach or sequence of steps that works for the counting game. You have to examine the image and then decide the steps and procedure that works for counting objects on the image.

II. Approach

The image has hollow circular holes, solid circular shapes, and hollow rectangular shapes. In this case, it quite difficult to use employ any of the morphological processes first because of the difficulty of reading the eventual images. I had to settle for something else.
The first thing is to take the complement of the image and this converts the holes to a solid shape. Then count the number of holes by applying shrink. Identify the position of the holes and apply addition filter to remove the holes. Now we have no hole on the image. We can then shrink the image to count the number of solid shapes in the image.
After that, I inspected the corner of the shapes to see if there it is a square or circle. If the corner pixel is 255, then the shape must be square. Otherwise, the shape is circle. Algorithm 12 below explains the procedure for the counting game. Figure (14) below shows the flow-chart for solving the counting game.

Algorithm 12: Binary Image Skeletonizing

```
Data : Image array F with dimension (Nx , N_Y).
       Create a array M with dimension (Nx , N_Y)
       Create an output array G with dimension (Nx , N_Y)

for pixel (i, j) in array L
    if (F(i, j) == 255)
        G(i, j) = 0
    else
        G(i, j) = 255
    end

while (Shrinking is not done)
    for pixel (i, j) in array G
        Apply conditional mark pattern for shrinking to produce M(i, j)
        Apply the unconditional mark pattern for shrinking
        Update the image array

    Check if the shrinking is over

Count the number of  black holes = number of white single spots
Remove the black holes centered at white single spots on F.

while (Shrinking is not done)
    for pixel (i, j) in array F
        Apply conditional mark pattern for shrinking
        Apply the unconditional mark pattern for shrinking
        Update the image array

    Check if the shrinking is over

Count the number of  white object = number of white single spots
Remove the black holes centered at white single spots on F.
Count the number of squares in the image

Output : Save the new image array G to output image.
```

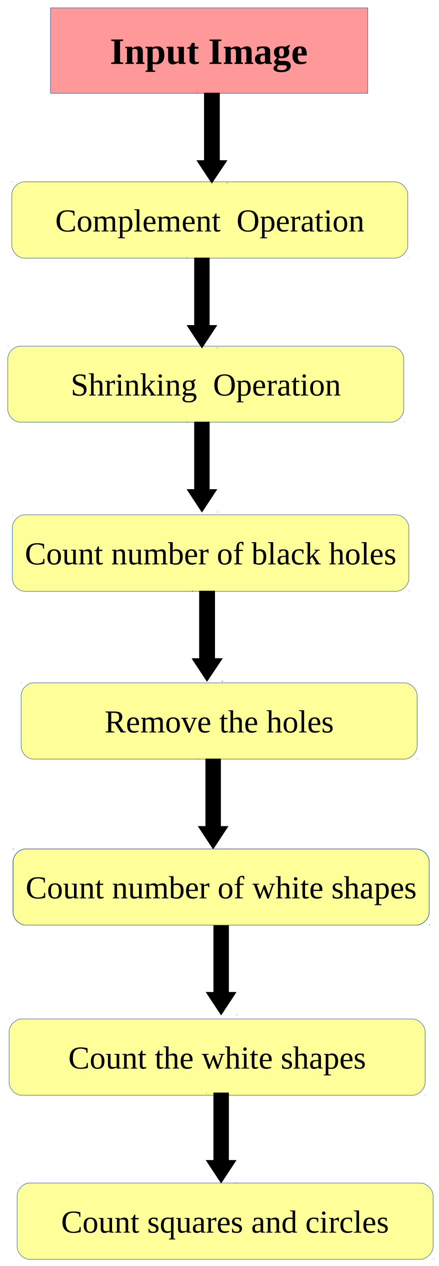
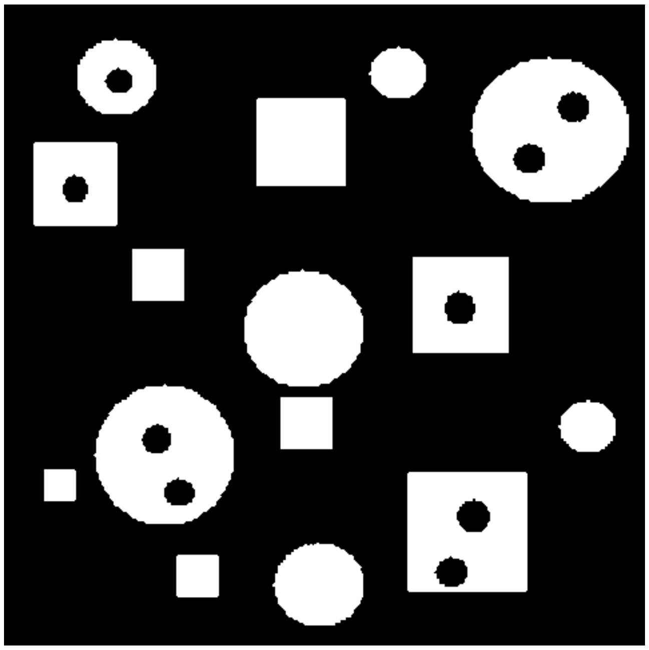


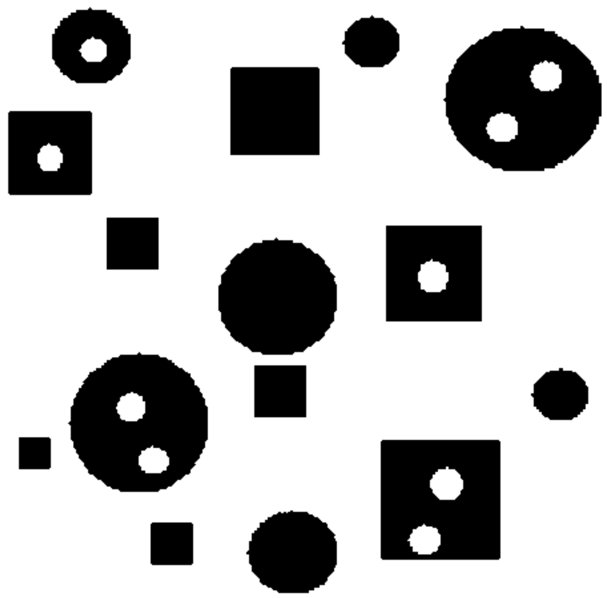
Figure (14) : Flow-chart for the counting game.

III. Results

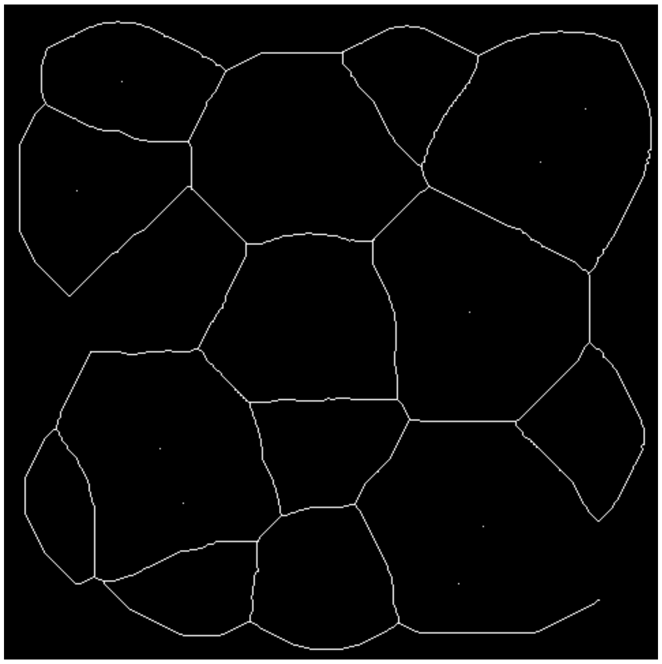
Figure (15) shows the initial, intermediate, and final images for the counting game.



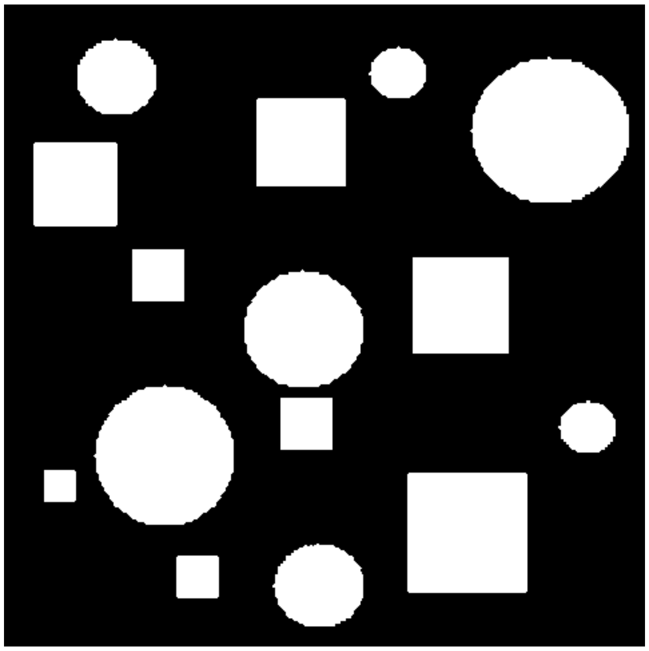
(a). Original Board



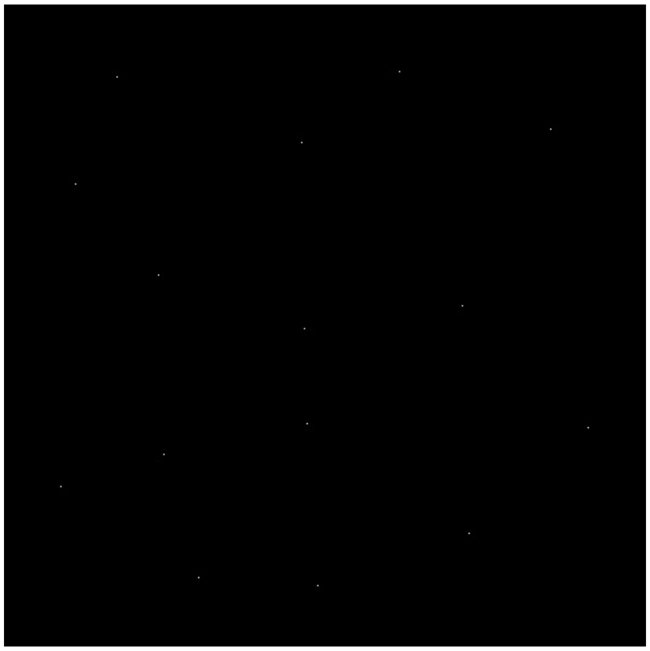
(b). Board (After Complement Operation)



(c). Board (After shrinking black holes)



(d). Board (After removing the holes)



(e). Board (After shrinking the white objects)

Figure (16): Counting game on the board image.

<u>ANSWERS TO QUESTION</u>

1) Number of white objects in the image is 9
2) Number of holes (black circular holes within white objects) in the image is 15
3) Number of white square objects (with or without holes) in the image is 7
4) Number of white circle objects (with or without holes) in the image 8.

IV. Discussion

The counting problem is a bit complicated because there is no single correct solution. You have to examine the shapes of interest and come up with the necessary operations to count the shapes. For the board image, the most challenging task was the filling of holes in the board image. I identified the center of the holes and I filled-up the holes. The filling of holes could be more difficult if the size of holes differs greatly. I took advantage of the similarity in size of holes to easily fill the holes.
I also had to come up with an easy way to identify the shapes in the image. One way to differentiate a square from a circle is by calculating the circularity of the shapes. This is quite complicated and the computational complexity is high. Therefore I had to use an easier approach. What I did was to test the pixels at the corners of this shape. For a square, there is a white pixel at the corners and for circle there is no white pixel at the corners. In fact, checking one corner is enough to settle the dichotomy between these two shapes.
Finally, there is no guarantee that this approach will work for any other board image. The proposed solution was developed based on the configuration of the board.