Name: Deepika Kanade ***Student Id: 5369288614***

EE 569: Homework #2

Table of Contents:

Problem 1- Geometric Image Modification

a) Geometrical Warping

1. Abstract and Motivation

In geometrical warping techniques the position of pixels in the image is transformed by using operations such as translation, rotation and scaling. 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. This concept is mainly used in photo editors, panoramas, designing logos, computer vision applications.
In the first part of the question, 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. In the second part of the question, homographic transformation is used. In this method, the points in a plane are projected on a different plane so as to compensate for the change in viewpoint while viewing both the images. The homographic transformation concept along with image stitching is used to obtain the effect of panorama.

2. Theoretical Approach and Procedure

Task: To convert the given square images into disc images and apply reverse transformation to obtain the square images

In geometrical warping, the co-ordinates are mapped from one image to other. In our case, as we have to convert a square image to a disc image, the points in the square image are mapped into points of the disc image. This can be shown as follows:

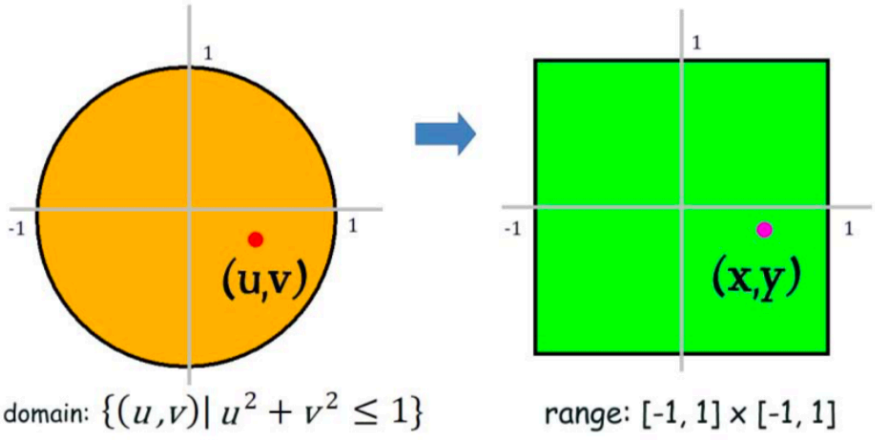


Figure 1: Mapping of points in disc image to square image [2]

The disc has radius 1 whereas the square has length 2. The point (u,v) is in the interior of the circle whereas the point (x,y) is the corresponding point mapped in the square.

The mapping from square to disc and disc to square using radial mapping method is as shown below:

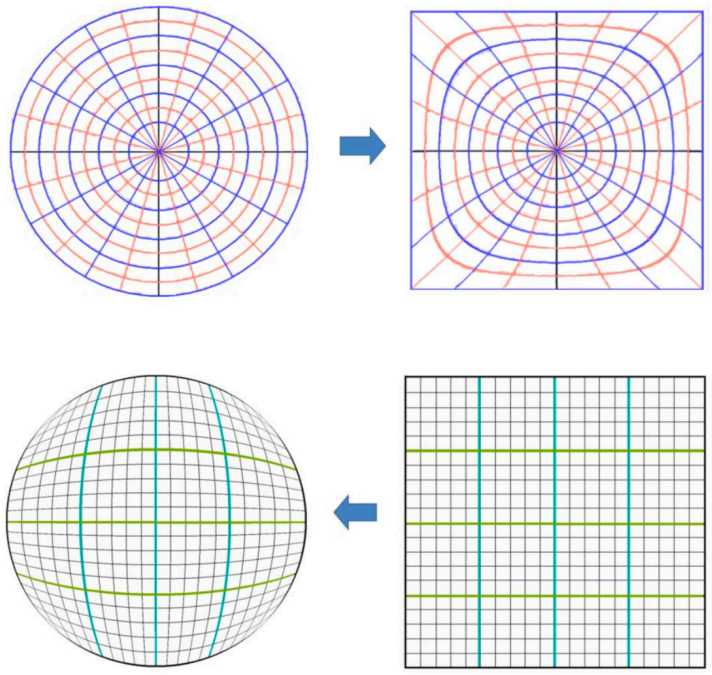


Figure 2: Radial mapping from disc to square and square to disc

The formula to map from square to disc is as given below:

Square to disc mapping:

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

The formula for mapping from disc to square is as given below:

Disc to square mapping:

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

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

Algorithm for geometrical warping in C++:

Step 1: Read the contents of the input images “Puppy.raw”, “Panda.raw” and “Tiger.raw” into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert the 1D input image into a 2D image for R, G, B channels respectively.
Step 3: Conversion from square to disc

Step 3a: Convert the pixel co-ordinates of the image from 0-512 to -1 to 1 as the formulae mentioned above are for these co-ordinates.

Step 3b: Apply the formula mentioned above for mapping from square to disc and obtain the (u,v) co-ordinates for every (x,y) co-ordinates in the square.

Step 3c: Map the co-ordinates from Step 3b back to the range 0-512.

Step 3d: Copy the values of input image into the image with co-ordinates mentioned in Step 3c.

Step 4: Convert the 2D output image from Step 3d into a 1D image to write in a file to be displayed as the output image.
Step 5: Conversion from disc to square

Step 5a: Convert the pixel co-ordinates of the image from 0-512 to -1 to 1 as the formulae mentioned are for these co-ordinates

Step 5b: Apply the formula mentioned for mapping from square to disc and obtain the (u,v) co-ordinates for every (x,y) co-ordinates in the square.

Step 5c: Calculate (xnew,ynew) from the (u,v) co-ordinates obtained in Step 5b by using the formula mentioned above for converting disc image to square.

Step 5d: Map the co-ordinates from Step 4c back to the range 0-512 and round them.

Step 5e: Copy the values of image obtained from square to disc conversion into the image with co-ordinates mentioned in Step 5d.

Step 6: Convert the 2D output image from Step 5e into a 1D image to write in a file to be displayed as the output image.

3. Experimental Results

Figure 3: Original Puppy image

Figure 4: Original Tiger image

Figure 5: Original Panda image

Figure 6: Square to Disc mapping for puppy image

Figure 7: Square to Disc mapping for tiger image

Figure 8: Square to Disc mapping for panda image

Figure 9: Disc to Square mapping for puppy image

Figure 10: Disc to Square mapping for tiger image

Figure 11: Disc to Square mapping for panda image

4. Discussion

The geometrical warping to convert from square image to disc image and the reverse mapping to convert from disc 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 disc and vice versa definitely gives an artistic look to the image.

b) Homographic Transformation and Image stitching

1. Abstract and Motivation

Homographic Transformation concept is used in a lot of computer vision applications. These transformations are used to create panoramas, in photo editors, mobile phones etc. A homographic transformation transforms an image plane P1 into another plane P2. To obtain this transformation, control points have to be selected in the two overlapping images. These control points give us an idea of the orientation of one image with respect to another. After finding the

orientations, image stitching is performed wherein multiple input images are grouped together to get a composite output image such that it has a flow. This grouping of images is done by adding the warped image into the original image. As our line of view is fixed, any image to the left or right of this line of view appears to be tilted. Image warping is used to transform the left or right image with respect to the current image's perspective. Implementing all these concepts mentioned above give us the feel of a panorama.

2. Approach and Procedure

Task: To create a panorama effect using the left, middle and right images.

The transformation from one image to the other image is given by the Homography matrix. The homography matrix takes into account the concept of projective planes wherein given any four points in one reference planes there exists a relationship that transforms those points to four points in other reference plane.

This concept is as shown below:

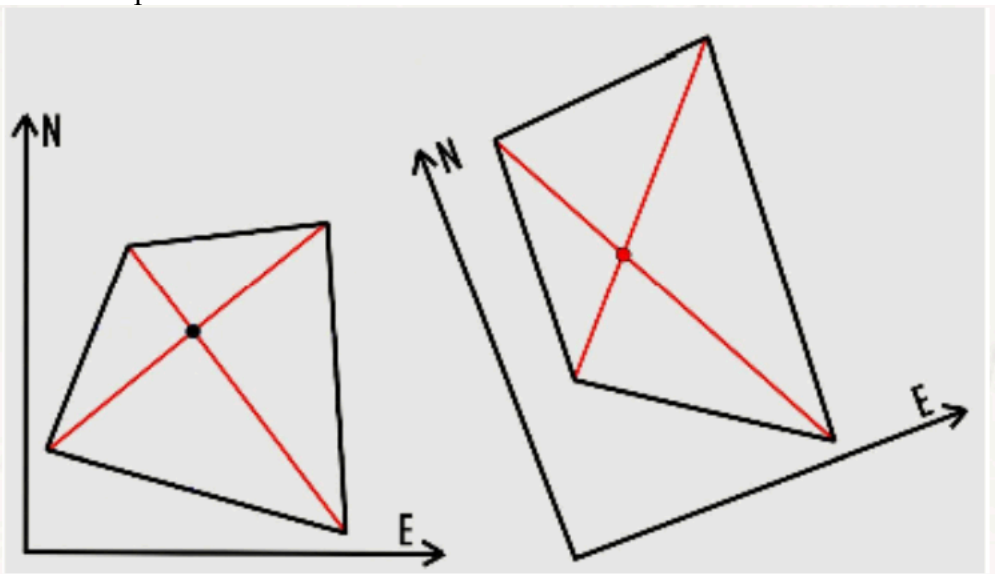


Figure 12: The homography transforms a quadrilateral from one reference system into the corresponding quadrilateral in another reference system.
(Source: http://www.corrmap.com/features/homography_transformation.php)

The Homography transformation procedure is as given below. The H matrix transforms the points in one plane to points in another plane by using the following relation:

$$P2 = H \times P1$$

where, $P1 = Image\ points\ in\ the\ homogenenous\ coordinates\ before\ transform$
$P2 = Image\ points\ in\ the\ homogeneous\ coordinates\ after\ transform$

$H = A\ 3x3\ homographic\ transformation\ matrix$

The transformation matrix is as given below:

$$\begin{bmatrix} x'_2 \\ y'_2 \\ w'_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \text{ and } \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{x'_2}{w'_2} \\ \frac{y'_2}{w'_2} \end{bmatrix}$$

As the points in 3D plane are projected onto a 2D plane, the first two co-ordinates are divided by the weight of the third co-ordinate.

The Homographic transformation matrix can be further simplified as follows:

$$\begin{vmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3Y_3 & -y_3Y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 \end{vmatrix} \cdot \begin{vmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{vmatrix} = \begin{vmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{vmatrix}$$

Figure 13: Homography Transformation matrix
(Source: http://www.corrmap.com/features/homography_transformation.php)

Where,

a = fixed scale factor in X direction with scale Y unchanged
b = scale factor in X direction proportional to Y distance from origin.
c = origin translation in X direction.
d = scale factor in Y direction proportional to X distance from origin.
e = fixed scale factor in Y direction with scale X unchanged.
f = origin translation in Y direction.
g = proportional scale factors X and Y in function of X.
h = proportional scale factors X and Y in function of Y.

The points (x1, y1), (x2, y2), (x3, y3), (x4, y4) denote points in the moving image whereas (X1,Y1), (X2,Y2), (X3,Y3), (X4,Y4) denote points in the fixed image.

The control points were selected in Matlab by using the command cpselect. The H matrix was then computed from the control points in the moving and fixed images.

To get the appropriate control points, I performed the feature extraction of the image in openCV. The Scale Invariant Feature Transform(SIFT) was used to obtain the best control points. An image of these control points extracted is as shown below:

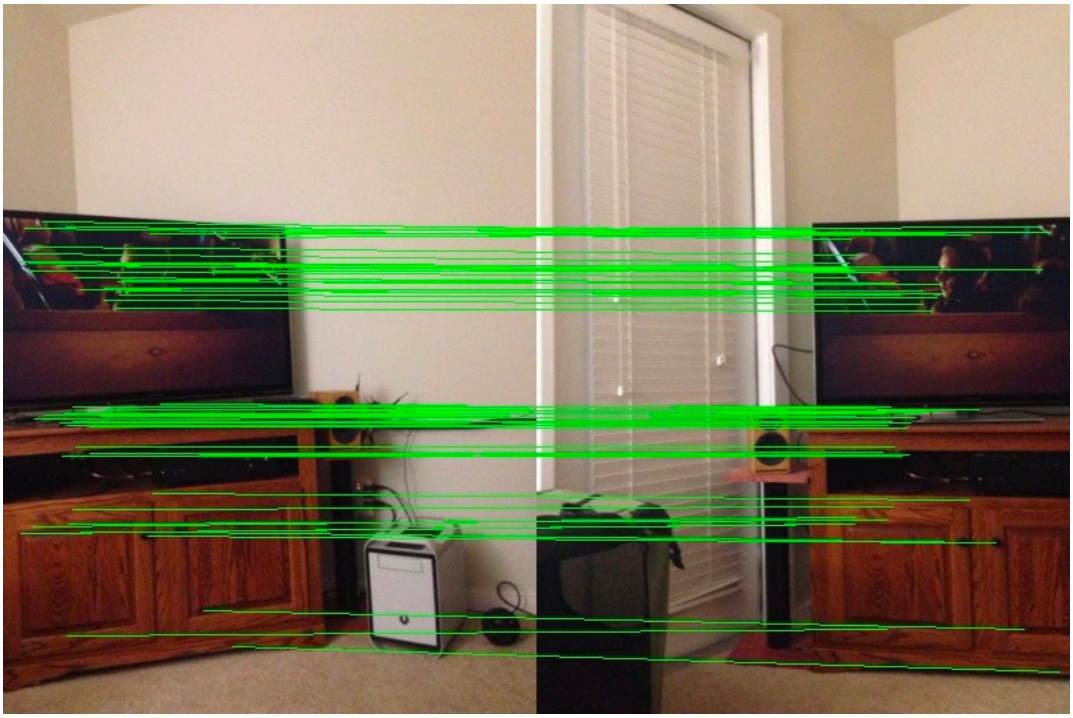


Figure 14: Best control points for the left and middle images

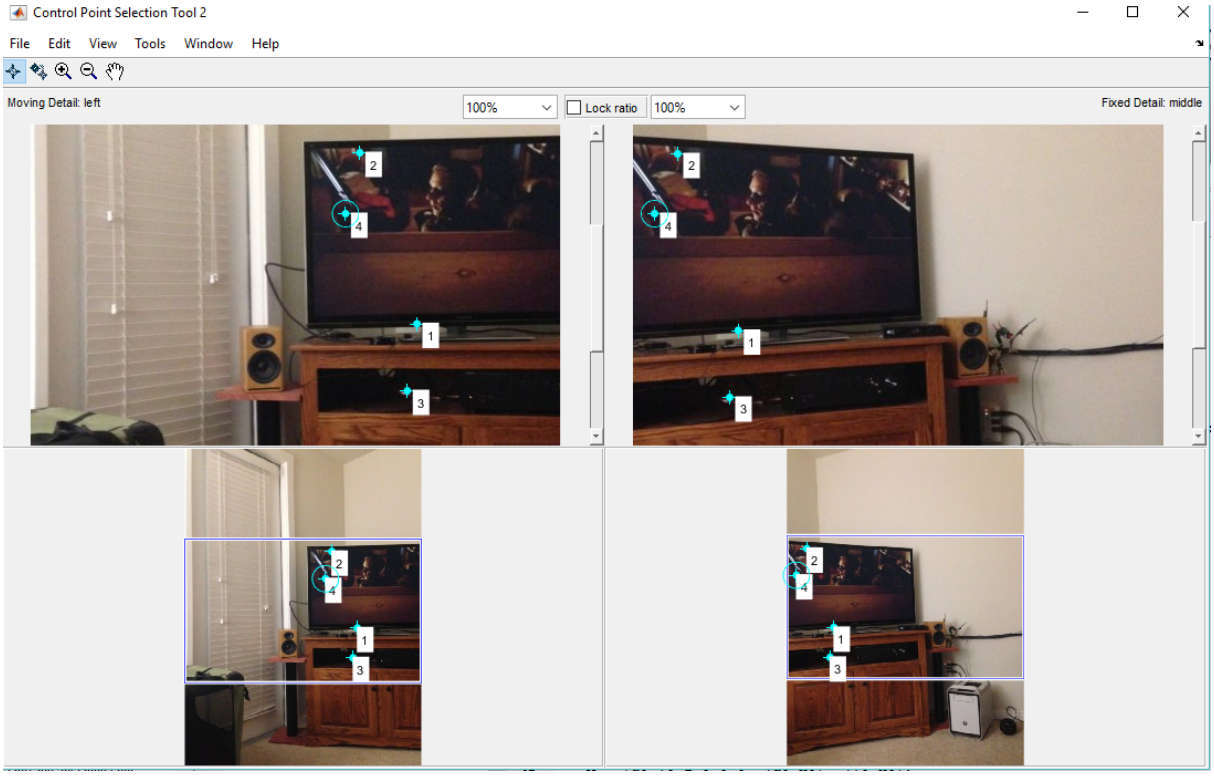


Figure 15: Control points for left and middle image

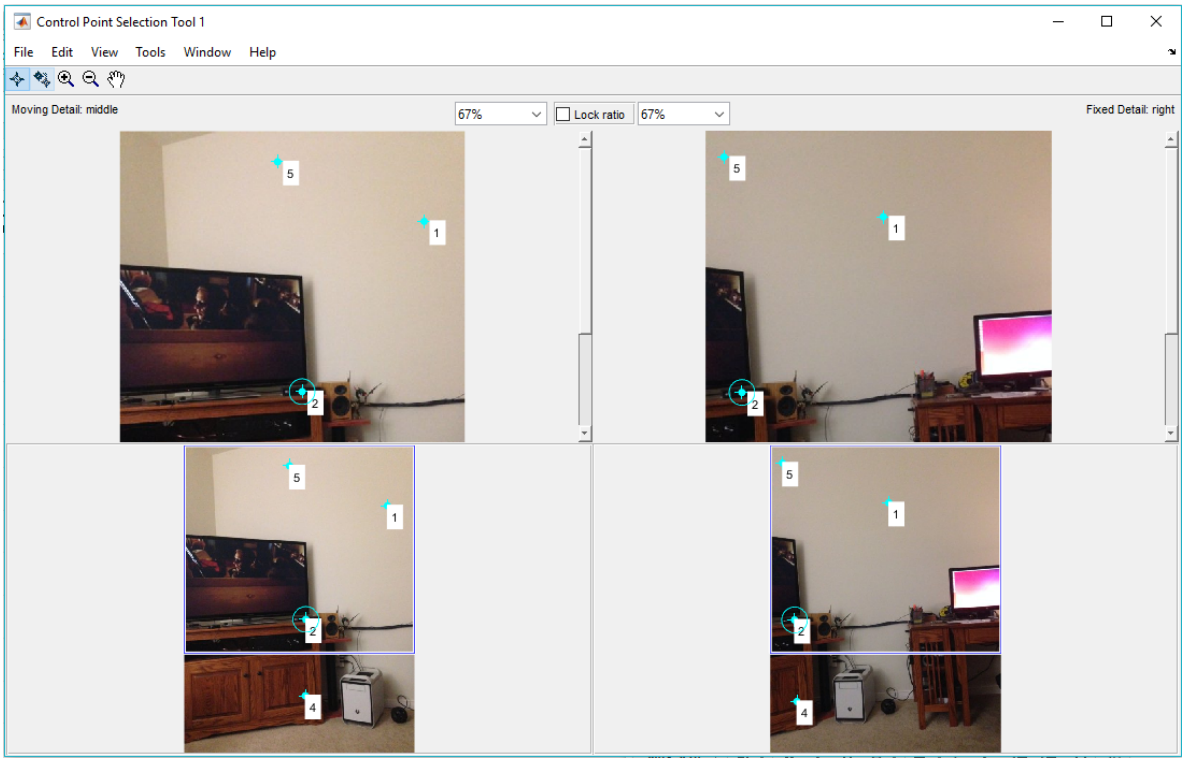


Figure 16: Control points for right and middle image

From these control points, I selected my four control points. As there are 8 unknowns in the H matrix equation given above, atleast 4 points are required in each image.

The four control points for the left and middle images are as given below:
x1 = 473; y1 = 282;
x2 = 424; y2 = 338;
x3 = 362; y3 = 350;
x4 = 204; y4 = 293;

X1 = 482+300; Y1 = 822;
X2 = 424+300; Y2 = 883;
X3= 360+300; Y3 = 895;
X4 = 195+300; Y4 = 831;
The four control points for the right and middle images are as given below:

x1 = 366; y1 = 258;
x2 = 525; y2 = 258;
x3 = 436; y3 = 125;
x4 = 224; y4 = 47;

X1 = 372+300; Y01 = 1264;
X2 = 544+300 Y2 = 1262;
X3 = 436+300; Y3 = 1119;
X4 = 237+300; Y4 = 1049;

After obtaining the control points, the H inverse matrix is found. This matrix gives a mapping from the middle image to the left image/right image. The averaging of intensity values has to be done for the overlapping parts of the left/Right image and middle image.

Bilinear Interpolation is used to approximate the co-ordinates of the left/right image from the H inverse matrix. The concept of bilinear Interpolation is as explained below:

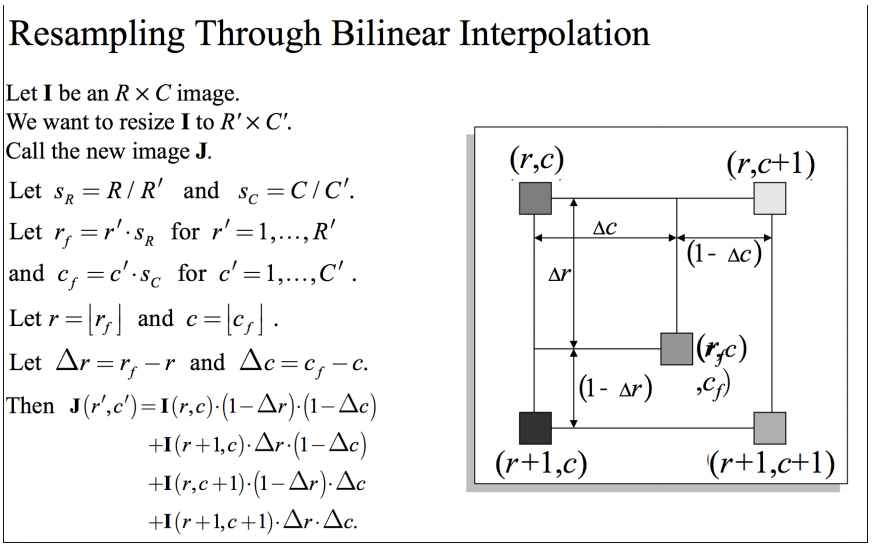


Figure 17: Resampling using Bilinear Interpolation

Where deltaR and deltaC are as mentioned above. Hence the values of the left/right image are decided using bilinear interpolation.

Algorithm for Homography matrix generation in Matlab:

Step 1: Read the contents of the input images “Left.raw”, “Right.raw”, “Middle.raw” into separate 1S arrays whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Set the row offset as 300, column offset as 795 and create a black image of size 1240x2500.
Step 3: Embed the middle image in the black image created in Step 2 at a distance of column offset and row offset from the start.
Step 4: Get the control points in the left and right image and right and middle image by using cpselect.
Step 5: Put the values of the 4 control points of each image into the formula given above for the Homography Transformation matrix and generate the transformation matrix.
Step 6: Compute the H inverse matrix from the H matrix obtained in Step 5.
Step 7: Export the H inverse matrix from Matlab to C++.

Algorithm for Homographic Transformation and Image Stitching:

Step 1: Read the contents of the input images “Left.raw”, “Right.raw”, “Middle.raw” into separate 1D arrays whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Set the row offset as 300, column offset as 795 and create a black image of size 1240x2500.
Step 3: Embed the middle image in the black image created in Step 2 at a distance of column offset and row offset from the start.
Step 4: Embed the left image in the embedded middle image

Step 4a: Input the H inverse matrix obtained in Matlab for the combination of left and middle image and scan the entire embedded image.

Step 4b: Calculate the weight vector which is given as:

W=1/ ((Hinv[2][0]*i) + (Hinv[2][1]*j) + Hinv[2][2])

Where Hinv[2][0], Hinv[2][1], Hinv[2][2] denote the last row of the H inverse matrix.

Step 4c: Generate a vector A which contains the row number, column number and 1.

Step 4d: Multiply H inverse by the vector A generated in Step 4c.

Step 4e: Calculate x coordinate and y co-ordinate by multiplying the resultant vector from Step 4d by the weight vector.

Step 4f: Apply Bilinear interpolation on the image

Step 4f 1: Calculate the floored value for each value of x and y co-ordinate.

Step 4f 2: Check if the mapped pixel index exceeds the height and width of the left image.

Step 4f 3: Calculate delx and dely which will be used to obtain the four adjacent pixels of the input image to be used for Bilinear Interpolation.

Step 4f 4: Find the co-ordinates from the left image to be embedded into the embedded image by using the concept of Bilinear Interpolation.

Step 4g: Average the intensity value of the overlapping regions between left and middle image.

Step 5: Embed the right image in the embedded middle image

Step 5a: Input the H inverse matrix obtained in Matlab for the combination of right and middle image and scan the entire embedded image.

Step 5b: Calculate the weight vector which is given as:

W=1/((Hinv[2][0]*i)+(Hinv[2][1]*j)+Hinv[2][2])

Where Hinv[2][0],Hinv[2][1],Hinv[2][2] denote the last row of the H inverse matrix.

Step 5c: Generate a vector a which contains the row number, column number and 1.

Step 5d: Multiply H inverse by the vector a generated in Step 5c.

Step 5e: Calculate x coordinate and y co-ordinate by multiplying the resultant vector from Step 5d by the weight vector.

Step 5f: Apply Bilinear interpolation on the image

Step 5f 1: Calculate the floored value for each value of x and y co-ordinate.

Step 5f 2: Check if the mapped pixel index exceeds the height and width of the left/right image.

Step 5f 3: Calculate delx and dely which will be used to obtain the four adjacent pixels of the input image to be used for Bilinear Interpolation.

Step 5f 4: Find the co-ordinates from the left image to be embedded into the embedded image by using the concept of Bilinear Interpolation.

Step 5g: Average the intensity value of the overlapping regions between right and middle image.

Step 6: Embed the middle image back into the embedded image already consisting of warped left and right images.

Step 7: The embed image array is then converted into a 1D array for writing into a file to be displayed as the final output image.

3. Experimental Results

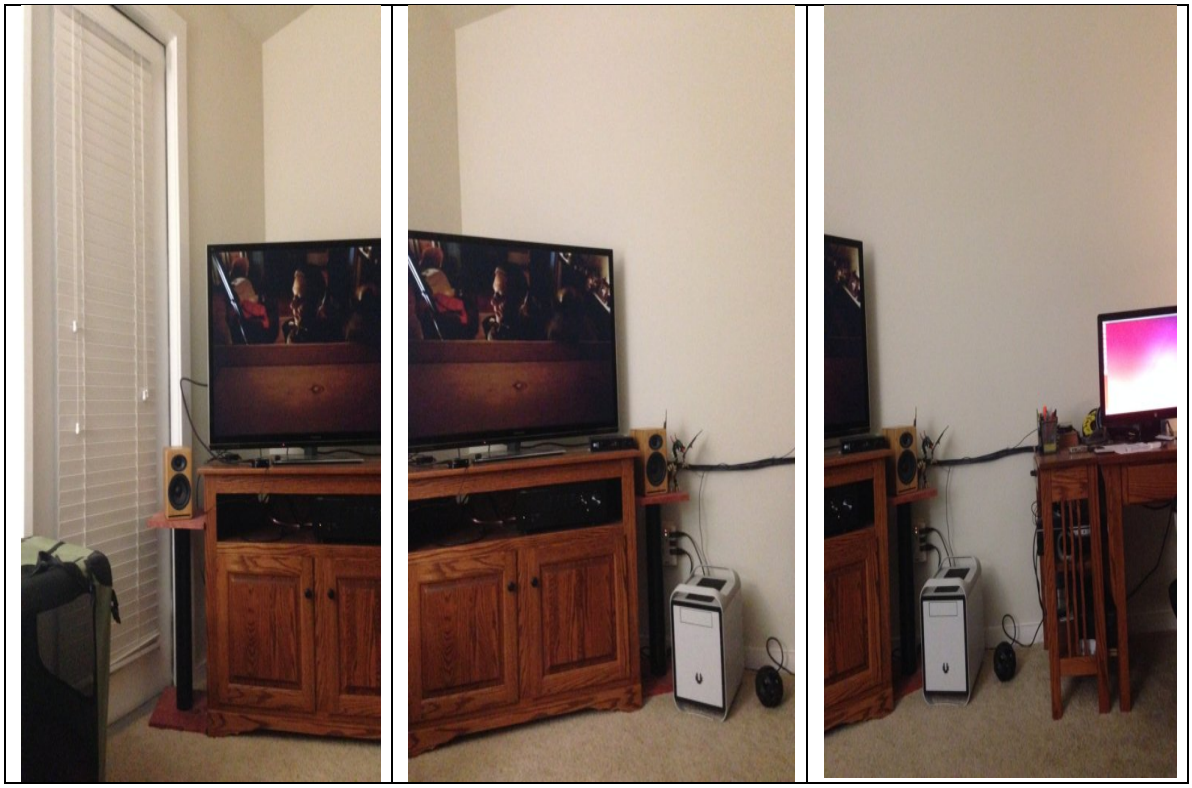


Figure 18: Original Left, Middle and Right Images

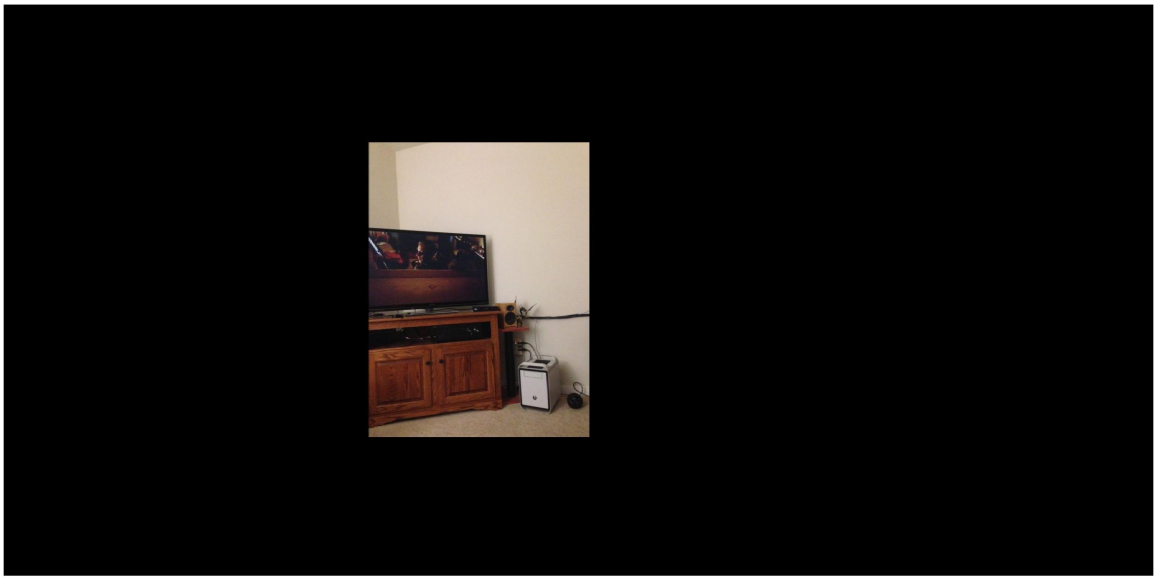


Figure 19: Embedded middle image of size 1240x2500

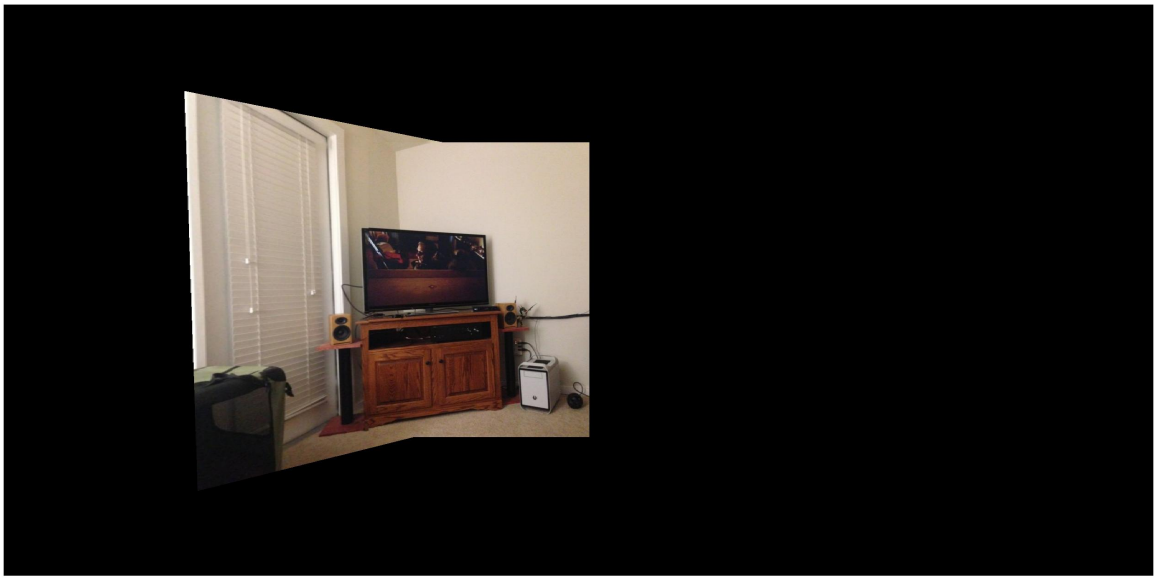


Figure 20: Output after embedding left Image

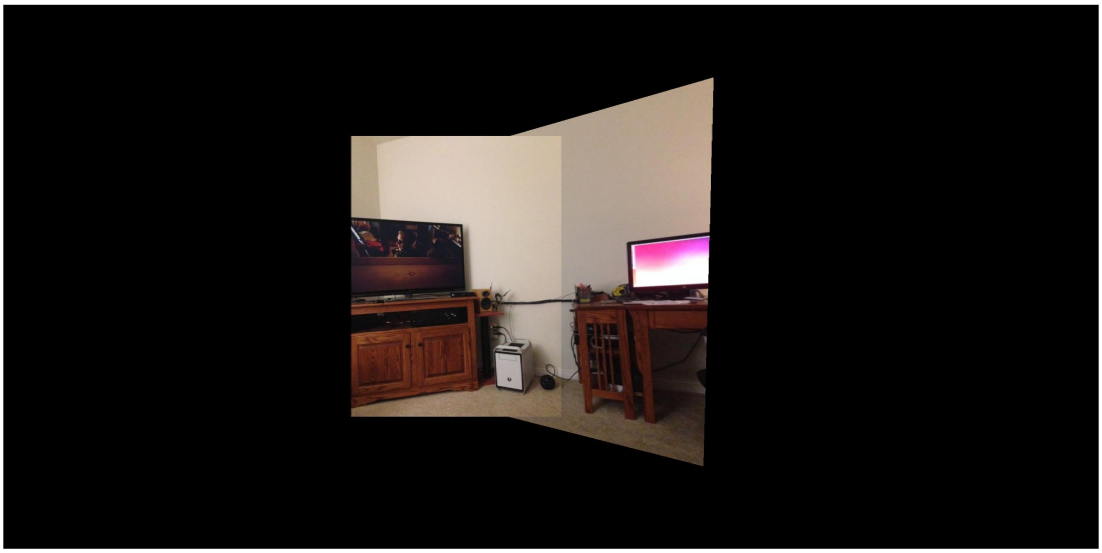


Figure 21: Output after embedding right Image

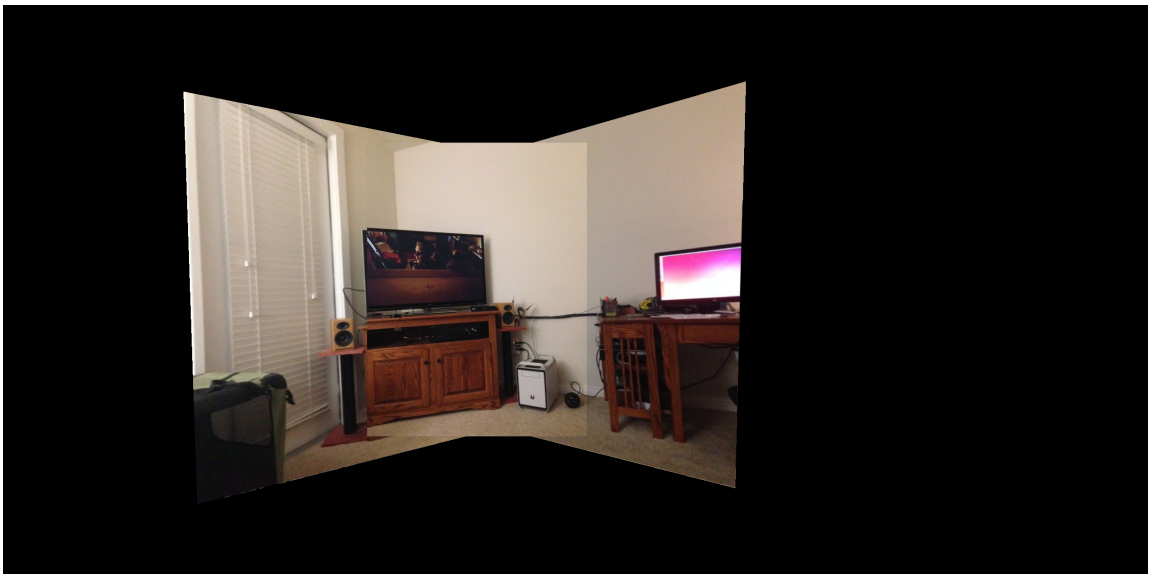


Figure 22: Final Output after embedding left, right, middle images

4. Discussion

Answer a) I used four control points from each of the image to find the transformation matrix. These four control points were used as there are 8 unknowns in the equation given for the transformation matrix. If the number of control points is increased it will be easier to map the points from the left/right image to middle image. This will be very helpful in the warping procedure.

Answer b) The control point selection is the most important part for this transformation. As mentioned earlier, I found the best control points that could be taken by using SIFT method. The entire algorithm for control point selection is already mentioned above.

Problem 2– Digital Half-toning

a) Dithering

1. Abstract and Motivation

Digital Half-toning is a procedure in which the gray-level image is converted into a binary image. This concept is mainly used in printers as the printers can only print black and white pixels in order to print monochrome images. Digital Half-toning takes into account the properties of the human visual system to be able to see a gray scale image even with two pixel intensities present in the image. This is because the human visual system tends to average out the intensities rather than considering each pixel separately, which gives an illusion of a gray scale image.

Digital Half-toning is done by using two methods namely Dithering and Error Diffusion. Random and fixed thresholding can also be used to convert the original image into a binary image.

In Dithering, an index matrix is created and placed on the original image and depending on the pixel values of the original image and the Bayer matrix, the output pixel value is thresholded. When the pixel value in the original image is greater than the Bayer matrix value, a black dot is placed at that value.

There are a variety of Bayer matrices used in practice which mainly consist of 2x2,4x4 and 8x8 matrices.

2. Approach and Procedure

Task: To convert a gray scale image into a binary image using fixed thresholding, random thresholding and Dithering using 2x2,4x4 and 8x8 Bayer's matrices.

Let the input pixel of the original image be denoted as F(i,j) and let the output image be denoted as G(i,j).

For performing fixed thresholding, a threshold of intensity 127 is decided to convert the gray scale image into a binary image. Each pixel is mapped to 0 if is smaller than T and mapped to 255 otherwise.

$$G(i,j) = \begin{cases} 0 & \text{if } 0 \le F(i,j) < T \\ 255 & \text{if } T \le F(i,j) < 256 \end{cases}$$

However, in order to reduce the quantization error obtained due to fixed thresholding, random thresholding is used. This is implemented by generating random uniform numbers between 0 and 255 and comparing the input pixel values with the random numbers. The following decision rule is used for assigning intensity values to the output image:

$$G(i,j) = \begin{cases} 255 & \text{if } rand(i,j) \leq F(i,j) \\ 0 & \text{if } rand(i,j) > F(i,j) \end{cases}$$

In order to reduce the noisy outputs obtained by using random and fixed thresholding, dithering is used.

The input Index 2x2 matrix is given as follows:

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

The values of the index matrix tell the likeliness of a pixel being turned on. For instance, 0 suggests that the pixel is turned on first whereas 3 suggests that the pixel is turned on the last. The 4x4 and 8x8 Index matrices can be constructed from the 2x2 Index matrix by the following formula:

$$I_{2n}(i,j) = \begin{bmatrix} 4 * I_n(x,y) + 1 & 4 * I_n(x,y) + 2 \\ 4 * I_n(x,y) + 3 & 4 * I_n(x,y) \end{bmatrix}$$

The input Index matrix is then converted into the threshold matrix which is used for comparison with the input image. It can be generated from the Index matrix as follows:

$$T(x,y) = \frac{I(x,y) + 0.5}{N^2}$$

Where N^2 is the number of pixels in the input Bayer matrix. As the input image matrix is very large, the threshold matrix is used as a mask on the entire input image. The following decision rule is used to turn the pixel on or off.

$$G(i,j) = \begin{cases} 1 & \text{if } F(i,j) > T(i \bmod N, j \bmod N) \\ 0 & otherwise \end{cases}$$

Algorithm for Dithering in C++:

Step 1: Read the contents of the input image "Colorchecker.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert the 1D input image into a 2D image for R channel as the image is comprised of a single channel.
Step 3: Fixed thresholding function was executed using the decision rule mentioned above to obtain the thresholded matrix.
Step 4: Random numbers were generated between 0-255 and the random thresholding function was invoked to compare the random numbers with input pixel values.
Step 5: The input image was normalized for values between 0-1 to implement the dithering procedure.
Step 6: 2x2 Index matrix was created and stored in a 2D array.
Step 7: 4x4 Index matrices were created from the 2x2 matrix by using the formulae mentioned above.
Step 7: 8x8 Index matrices were created from the 4x4 matrix by using the formulae mentioned above.
Step 8: 2x2,4x4 and 8x8 threshold matrices were obtained from the Index matrix by using the formulae mentioned above.
Step 9: The input matrix and the Threshold matrix were compared and values to the output pixels were assigned using the decision rule mentioned above.
Step 10: The output arrays from step 3,4 and 9 were converted into a 1D array for writing into a file to be displayed as an output image.

3. Experimental Results

The I4 and I8 matrices are generated using the I2 Index matrix. They are as follows:

```
5 9 6 10
13 1 14 2
7 11 4 8
15 3 12 0
```

Figure 23: I4 Index matrix

```
21 37 25 41 22 38 26 42
53 5 57 9 54 6 58 10
29 45 17 33 30 46 18 34
61 13 49 1 62 14 50 2
23 39 27 43 20 36 24 40
55 7 59 11 52 4 56 8
31 47 19 35 28 44 16 32
63 15 51 3 60 12 48 0
```

Figure 24: I8 Index matrix

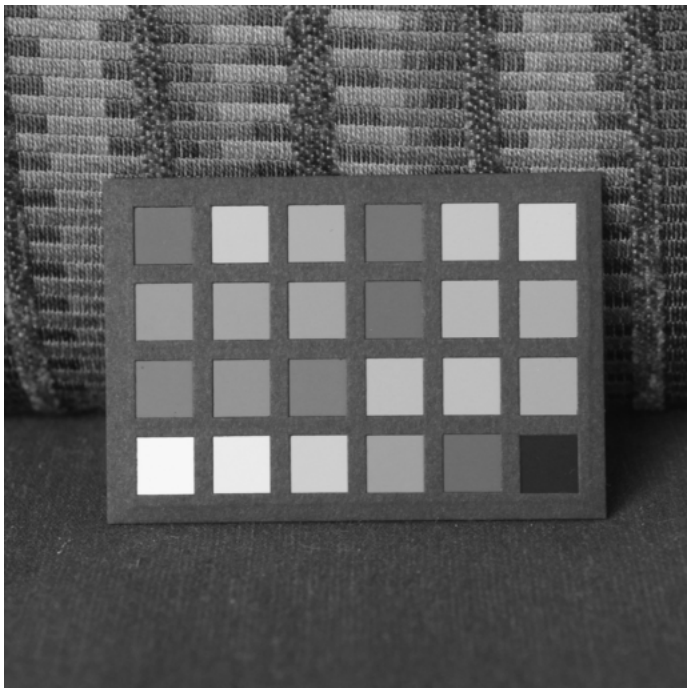


Figure 25: Original Colorchecker image

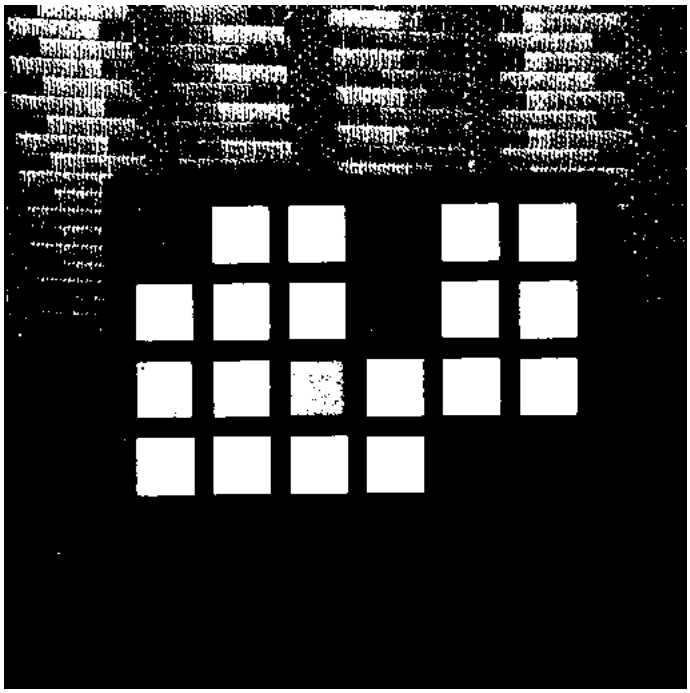


Figure 26: Fixed Thresholded Image

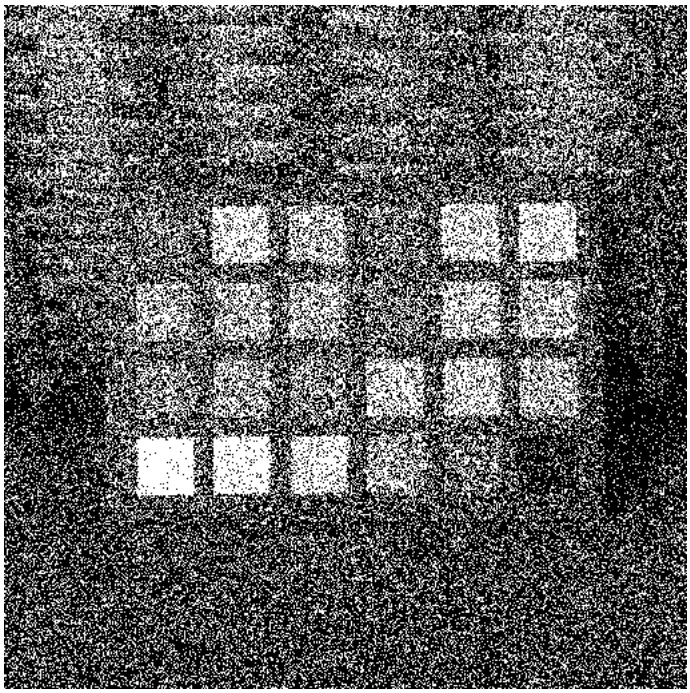


Figure 27: Random Thresholded Image

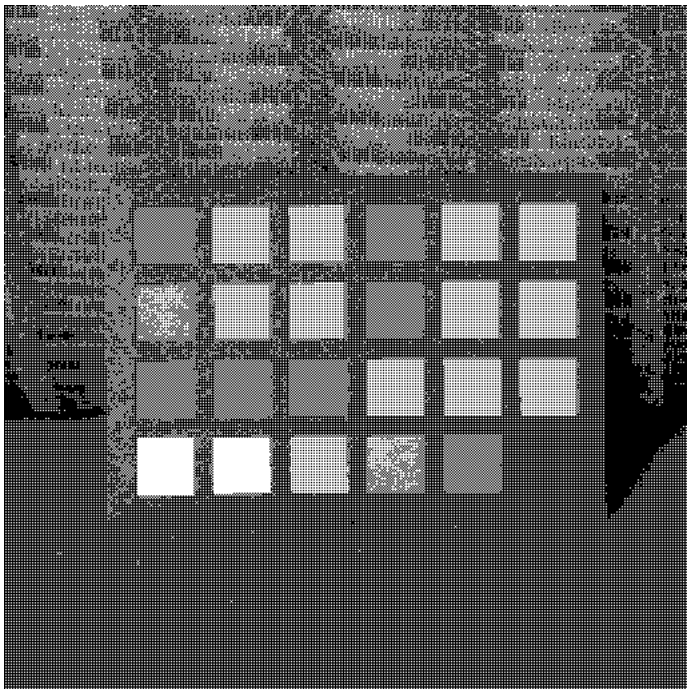


Figure 28: Index matrix I2 Dithered Image

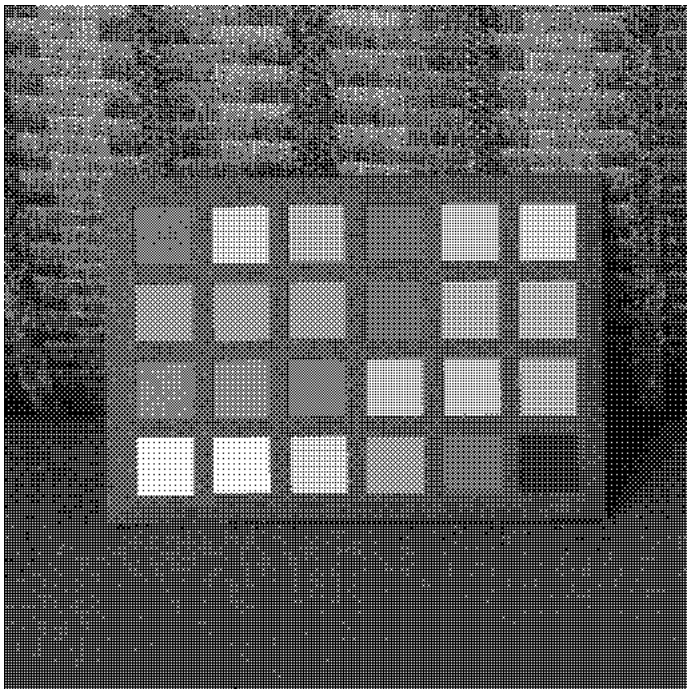


Figure 29: Index matrix I4 Dithered Image

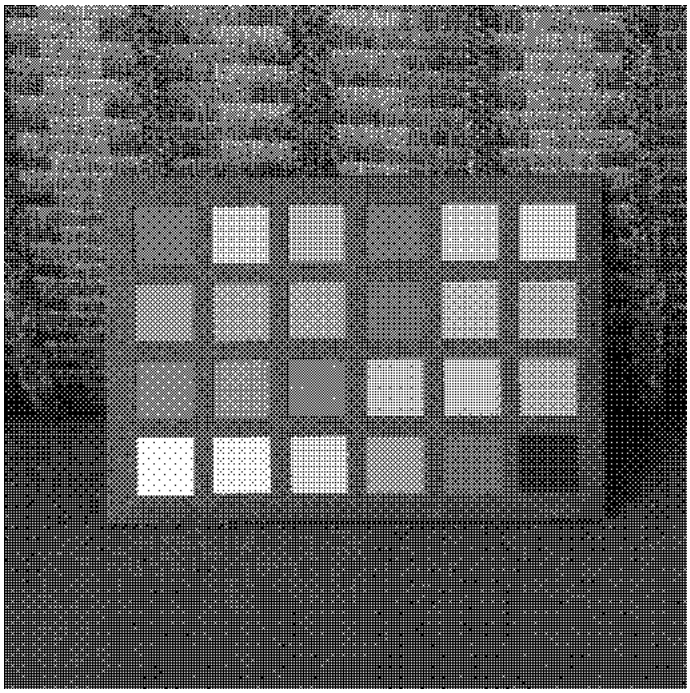


Figure 30: Index matrix I8 Dithered Image

4. Discussion

The fixed thresholded image has only two gray levels and hence can be viewed as a black and white image. In order to reduce the false contours in the fixed thresholded image, random thresholding is used. In random thresholding, the image obtained has only two levels but looks like a gray image from a distance.
In order to reduce the noise contents present in the random thresholded image, dithering is used. As the size of the Index matrix increases, the probability of the black pixel being turned on increases. Hence, the Dithered image using 8x8 index matrix resembles the original gray scale image more closely than the Dithered image using index 4x4 and 2x2 matrices. The I4 and I2 dithered images have lots of dots in them because of the density difference between the black dots.

Conversion of input image into four intensities:

Task: Input gray scale image has to be converted into four intensity levels- 0, 85, 170, 255.

The basic concept used while converting the input gray scale image into output image is thresholding.
The input image is divided into four bins

a) 0 to 85
b) 86 to 170
c) 171 to 255

If the intensity of the input image is between 0 and 85/2- the intensity value assigned to the pixel is 0.
If the intensity of the input image is between 85/2 and 85- the intensity value assigned to the pixel is 85.
If the intensity of the input image is between 85 and (85+170)/2- the intensity value assigned to the pixel is 85.
If the intensity of the input image is between (85+170)/2 and 170- the intensity value assigned to the pixel is 170.
If the intensity of the input image is between 170 and (170+255)/2- the intensity value assigned to the pixel is 170.
If the intensity of the input image is between (170+255)/2 and 255- the intensity value assigned to the pixel is 255.

Algorithm for converting input image into four intensity values:

Step 1: Read the contents of the input image "Colorchecker.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert the 1D input image into a 2D image for R channel as the image is comprised of a single channel.
Step 3: Divide the input image intensity values in bins as described above.
Step 4: Assign the intensity values to the pixels depending on the method given above.
Step 5: Convert the 2D output image from Step 4 into a 1D image to write in a file to be displayed as the output image.

The output for the four-intensity level image is as follows:

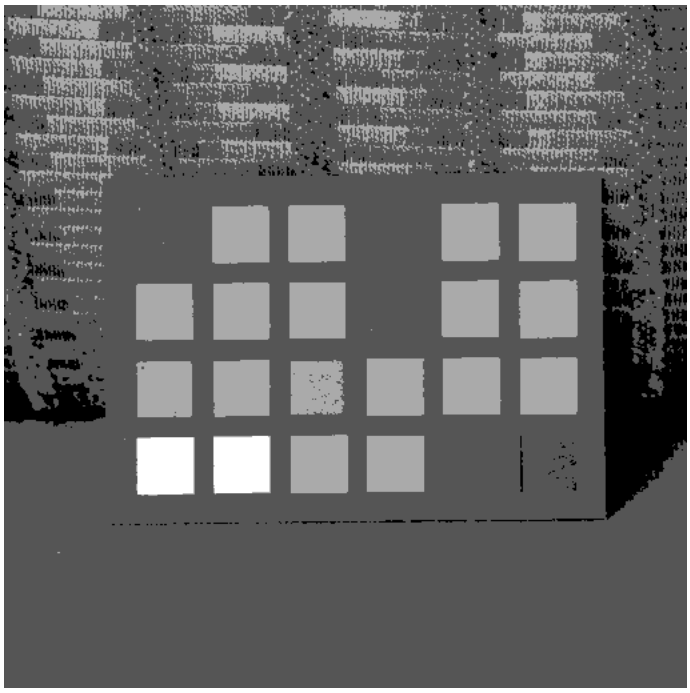


Figure 31: Four intensity level image

A histogram is plotted to show the intensity levels in the output image. The histogram clearly denotes the presence of only four intensities in the image namely 0, 85, 170, 255.

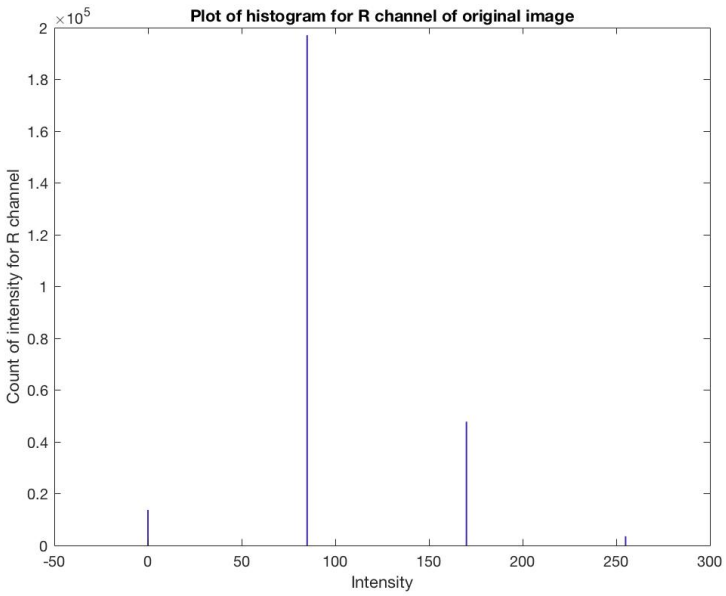


Figure 32: Histogram for four gray level image

b) Error Diffusion

1. Abstract and Motivation

Error Diffusion is another method used for digital Half-toning. In Error diffusion, the error generated at one pixel is diffused to the neighboring pixels. The error is diffused only at the pixels coming after the center pixel while scanning through the image. The pixels before the center pixel are not disturbed. This helps in maintaining the average intensity of the binary image close to the gray scale image.
Error Diffusion is mainly used to convert multi-level image into binary image. Error Diffusion generally enhances the edges in the image. Hence, it is mainly used to enhance text in an image by making it more readable compared to other half-toning techniques like dithering.

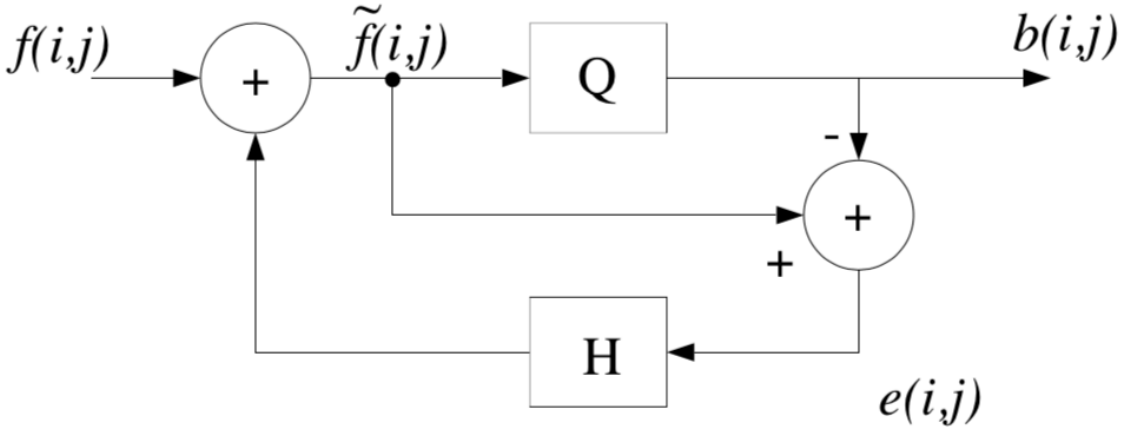


Figure 33: Error Diffusion Algorithm

In this algorithm, the input pixel f(i,j) is modified by adding some past quantization error. This modified pixel is named as f~(i,j). This modified pixel is then quantized to a binary value by using a quantizer with threshold T. The error generated is given as e(i,j) = f(i,j)−b(i,j). This error is then diffused to the future pixels by using several filters namely Floyd and Steinberg, Jarvice Judice and Ninke, Stucki. The filters in each of these methods is given as follows:

$$\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Figure 34: Floyd and Steinberg filter

$$\frac{1}{48}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 7 & 5\\ 3 & 5 & 7 & 5 & 3\\ 1 & 3 & 5 & 3 & 1\end{bmatrix}$$

Figure 35: Judice, Jarvis and Ninke(JJN) filter

$$\frac{1}{42}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 8 & 4\\ 2 & 4 & 8 & 4 & 2\\ 1 & 2 & 4 & 2 & 1\end{bmatrix}$$

Figure 36: Stucki Filter

Since, all the filter coefficient sum upto one, the local average value of the output image is approximately equal to the local average value of the input image. The error diffusion methods implemented make use of serpentine scanning. This helps in diffusing the error evenly in both the sides of the image rather than it accumulating in one corner. In serpentine scanning, for even rows, the scanning is done from left to right whereas for odd rows it is done from right to left. Hence, the masks are also changed accordingly for scanning odd rows.

2. Approach and Procedure

Task: To convert the gray scale image into binary image by implementing Error Diffusion using Floyd-Steinberg, JJN, Stucki matrices.

Each pixel in the input image is thresholded by using some thresholding value which in my case is 127. The error is then calculated by subtracting the original value of the pixel with the new

value of the pixel. The error diffusion is then performed to the neighboring pixels of the center pixel.

Algorithm for Error Diffusion:

Step 1: Read the contents of the input image "Colorchecker.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert the 1D input image into a 2D image for R channel as the image is comprised of a single channel.
Step 3: Perform the Floyd filtering on the input image
Step 3a: If the row is even, assign a value to the output pixel by using thresholding. If the value of input image>127, assign a value of 255 or else assign a value of 0.
Step 3b: Calculate the error by subtracting new value assigned to the pixel after thresholding from the old value.
Step 3c: Diffuse the error to the neighboring pixels as shown in the matrices above.
Step 4: Repeat the procedure for scanning of odd row.
Step 5: Repeat the procedure with JJN and Stucki matrices.
Step 6: Combine the outputs from the Floyd, JJN, Stucki arrays into separate 1D arrays to be displayed as an output image.

3. Experimental Results

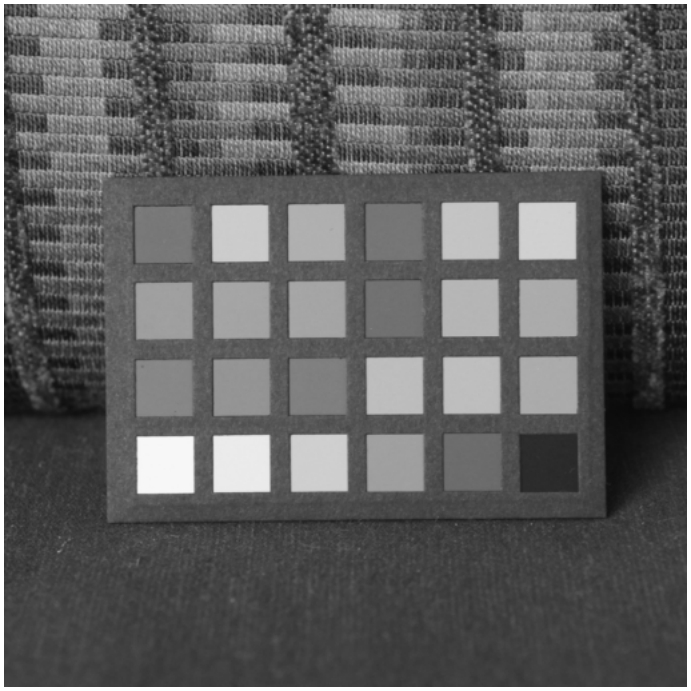


Figure 37: Original Colorchecker image

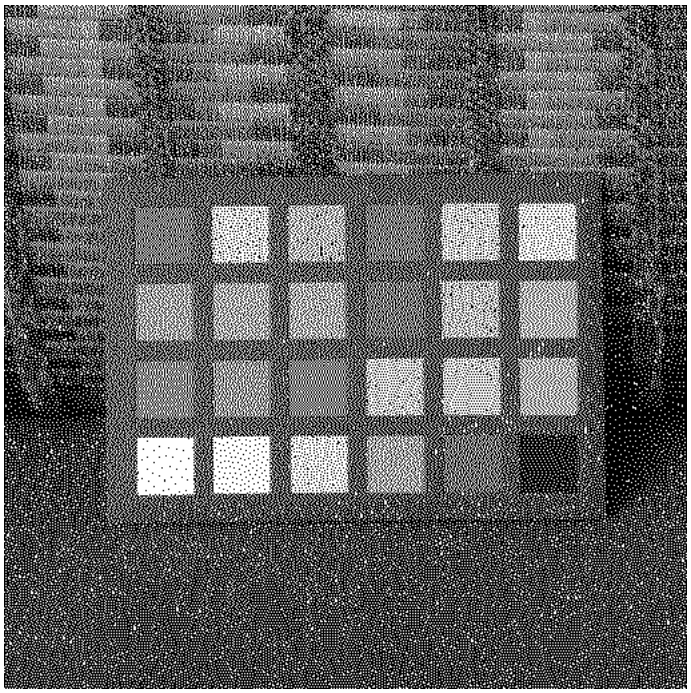


Figure 38: Error Diffusion using Floyd Steinberg matrix

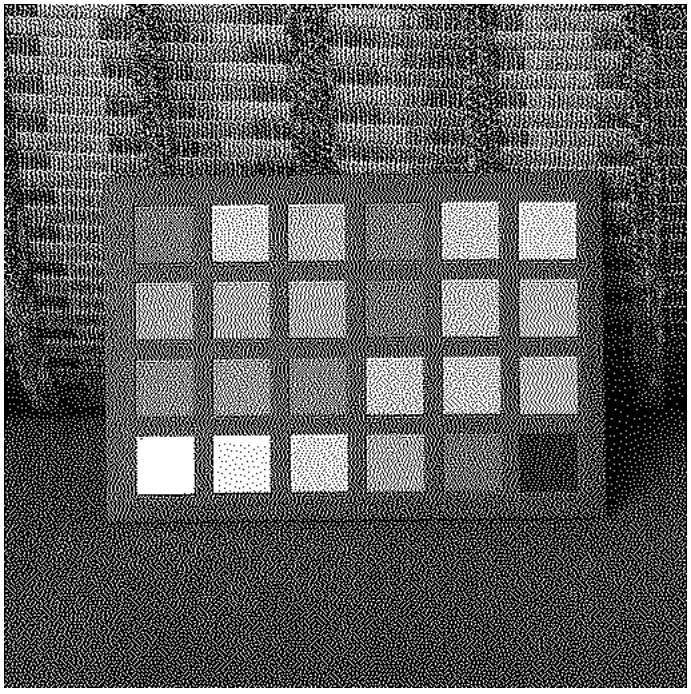


Figure 39: Error Diffusion using Jarvis,Judice and Ninke(JJN) matrix

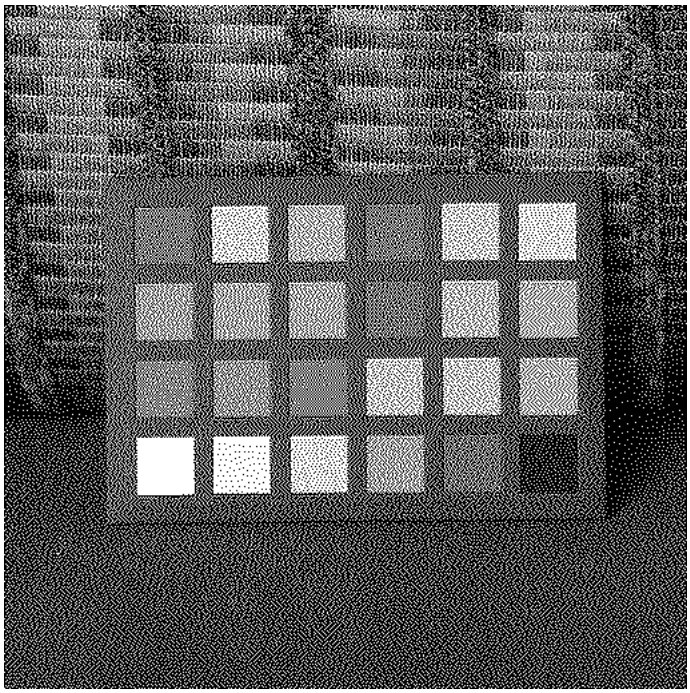


Figure 40: Error Diffusion using Stucki matrix

4. Discussion

According to me, the Error Diffusion method gives a better contrast performance as compared to the Dithering method of Half-toning. The Floyd-Steinberg matrix has a 3x3 mask whereas the JJN, Stucki matrices have 5x5 masks. Hence, as a larger neighborhood is taken into consideration for JJN and Stucki matrices, the error is distributed to more number of neighboring pixels. Whereas in case of Floyd-Steinberg, the error is diffused in the immediate neighbors and hence a large number of white dots are observed in the output.

JJN and Stucki matrices give a better visual quality as compared to the Floyd matrix. There isn't much of a difference between the outputs of JJN and Stucki matrices as they both have 5x5 masks but the weights associated in both of them is different. JJN matrix can be used in cases where preservation of edge is important as this method preserves the edges in the image. The amount of blurring in the Floyd matrix is more compared to JJN and Stucki matrices.

The Error Diffusion methods gives an output image close to the original gray scale image if looked from a distance. Hence, the error diffusion methods are used a lot in digital Half-toning. However, error diffusion methods tend to display streaking artifacts also known as worm patterns. Hence, other methods have to be employed in place of Error Diffusion.

In order to get better results, Atkinson developed a dithering algorithm named as Atkinson Dithering. In this type of dithering, only a small portion of the error is propagated to the future pixels. Normally only three quarters of the error is diffused to the future pixels. By propagating

only a part of the error, the speckles in the image are reduced, but the continuous dark and light portions of the image may become blur as well. Other methods such as Sierra, Burkes dithering can also be used to improve the results. The matrix used for Atkinson dithering method is as given below:

$$\begin{matrix} & X & 1 & 1 \\ 1 & 1 & 1 & \\ & 1 & & \end{matrix} \quad (1/8)$$

c) Color Half-toning with Error Diffusion

1. Abstract and Motivation

The error diffusion algorithm was earlier restricted to gray scale images but has been extended to the color images in recent years. There are two ways in which color error diffusion can be performed- Separable Error Diffusion and Minimal Brightness Variance Quadruples. The Separable Error Diffusion method involves diffusing error in the three planes i.e. R, G, B separately, whereas in the case of Minimal Brightness Variance Quadruples method, the characteristic of human color perception is taken into consideration.

While converting the gray scale image into a binary image (consisting of Black and White dots), the brightness variations cannot be incorporated which leads to the worm effect in the error diffused images. The main skill in performing color error diffusion is to average out values in the error diffused image to get an image close to the original image. Hence, MBVQ method achieves this better than the Separable Error Diffusion method.

1) Separable Error Diffusion

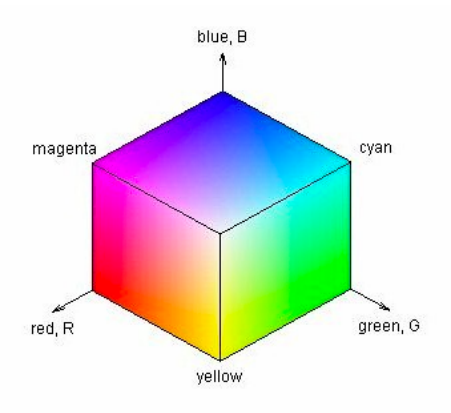


Figure 41: RGB Cube

(Source: http://www.peterstone.name/Maplepgs/colours.html)

As shown in the figure above, every pixel in the input image can be approximated to one of the vertices in the RGB cube. In Separable Error Diffusion method, each channel (C, M, Y) is quantized separately and the error diffusion also takes place individually on each channel. The resulting channels are then combined to give an output image which approximates the input image.

2) Minimal Brightness Variation Quadruples

Separable Error Diffusion gives an output that has a large number of visible artifacts in it. Hence, in order to reduce them, MBVQ method is used. The human Visual System is more sensitive to brightness variations than changes in Chrominance. The MBVQ method takes this into consideration and divides the RGB cube into 6 quadrants. These six quadrants have minimum brightness variation within them and hence are useful in getting an image closer to the input image. The Brightness Variation between the 8 basic colors is given as follows:

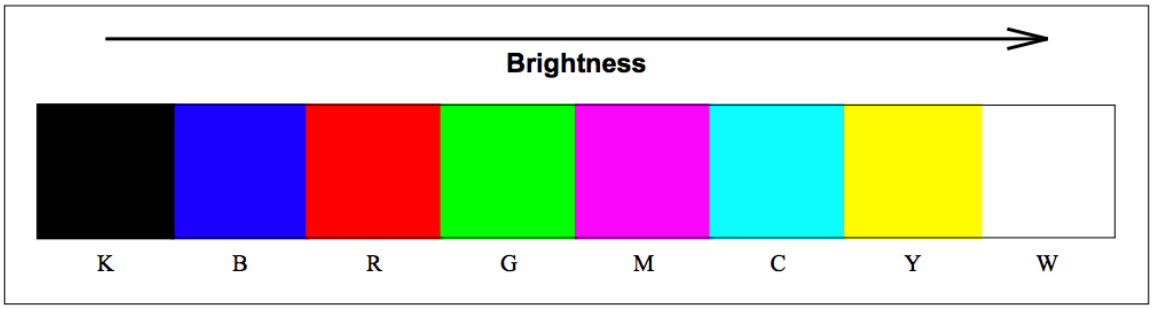


Figure 42: Brightness Variation in 8 colors of the R, G, B cube

(Source: https://pdfs.semanticscholar.org/c67f/37a2ab36bab46bb632b65dc8dc3866f7c80e.pdf)

The MBVQ method makes sure that the White and Black dots are used less whereas the saturated color dots are used more. Hence, the color saturation in the MBVQ method is better.

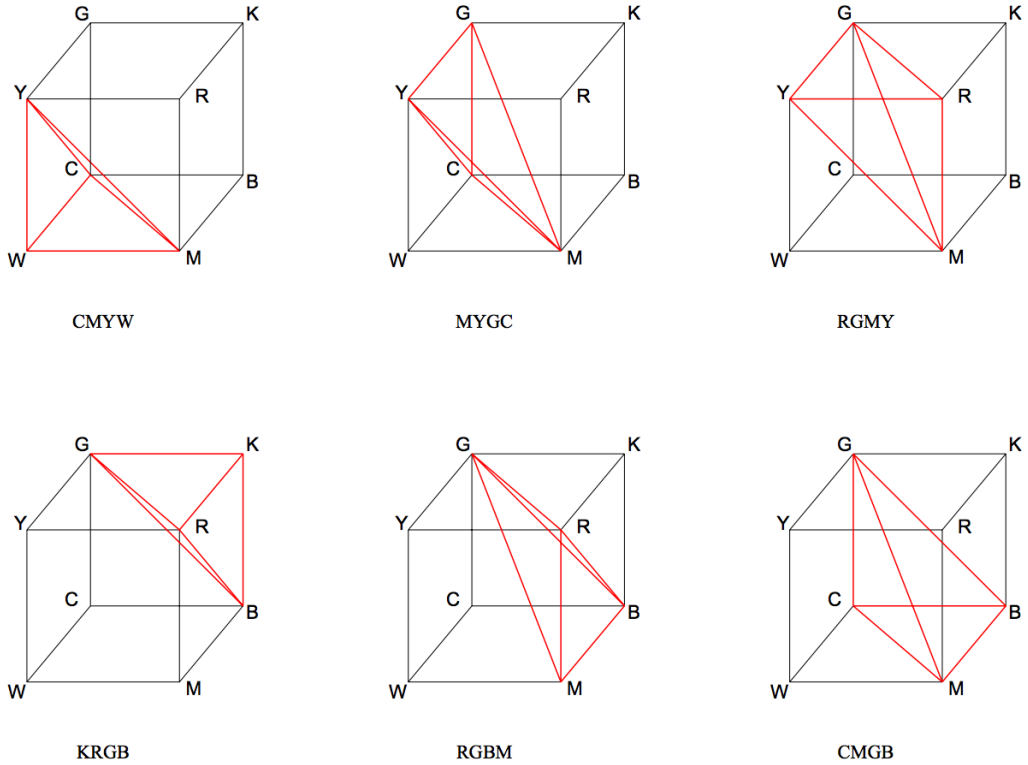


Figure 43: Six Tetrahedrons used for MBVQ method

(Source: https://pdfs.semanticscholar.org/c67f/37a2ab36bab46bb632b65dc8dc3866f7c80e.pdf)

The original image is converted to C, M, Y co-ordinates in order to diffuse the error. However, the decision regarding which quadrant the input pixel belongs to is done by using R, G ,B co-ordinates of the image. The Decision Rule to determine the Quadrant is as given below:

$$R+G\begin{cases} >255\begin{cases} G+B>255\begin{cases} R+B+G>510 \rightarrow CMYW \\ R+B+G<=510 \rightarrow MYGC \end{cases} \\ G+B<=255 \rightarrow RGMY \end{cases} \\ <=255\begin{cases} G+B<=255\begin{cases} R+B+G<=255 \rightarrow KRGB \\ R+B+G>255 \rightarrow RGBM \end{cases} \\ G+B>255 \rightarrow CMGB \end{cases} \end{cases}$$

The distance of the input image pixel with the vertices of the tetrahedron it belongs to is determined, and the value of the minimum distance vertex is assigned to the output pixel. The error is then calculated for each channel separately and diffused to the future pixels.

The o-ordinates of C, M, Y, R, G, B, W, K are as given below:

C (0,255,0) M (255,0,0) Y (0,0,255) R (255,0,255) G (0,255,255) B (255,255,0)

W (0,0,0) K (255,255,255)

2. Approach and Procedure

Task: To convert the given "Flower" image into quantized image using Separable and MBVQ error diffusion methods.

Algorithm for Separable Error Diffusion:

Step 1: Read the contents of the input image "Flower.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D arrays for R, G, B channels respectively.
Step 3: Convert the 2D arrays for R, G, B channels into C, M, Y channels by using the formula

C=255 - R
M=255 - G
Y=255 – B

Step 4: Apply the Floyd Error Diffusion algorithm

Step 4a: If the row is even, assign a value to the output pixel by using thresholding. If the value of input image>127, assign a value of 255 or else assign a value of 0.

Step 4b: Calculate the error by subtracting new value assigned to the pixel after thresholding from the old value.

Step 4c: Diffuse the error to the neighboring pixels as shown in the matrices above.

Step 4d: Repeat the steps for odd numbered rows in the image.

Step 5: Convert the C, M, Y channels of the new image obtained by thresholding into R,G,B channels using the formula mentioned in Step 3.
Step 6: Combine the R, G, B channels into a 1D array which will be displayed as an output image.

Algorithm for MBVQ based error diffusion:

Step 1: Read the contents of the input image "Flower.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D arrays for R, G, B channels respectively.
Step 3: Convert the 2D arrays for R, G, B channels into C, M, Y channels by using the formula

C=255 - R
M=255 - G
Y=255 – B

Step 4: Apply the MBVQ Error Diffusion algorithm

Step 4a: If the row is even, determine the quadrant of the input image pixel from the decision rule mentioned above.

Step 4b: Compute the distance of the M,C,Y values of the input pixel with the vertices of the decided tetrahedron from Step 4a.

Step 4c: Quantize the output pixel depending on the minimum distance vertex value.

Step 4d: Compute the error for the C, M, Y channels and diffuse the error using Floyd matrix

Step 4d 1: If the row is even, assign a value to the output pixel by using thresholding. If the value of input image>127, assign a value of 255 or else assign a value of 0.

Step 4d 2: Calculate the error by subtracting new value assigned to the pixel after thresholding from the old value.

Step 4d 3: Diffuse the error to the neighboring pixels as shown in the matrices above.

Step 4d 4: Repeat the steps for odd numbered rows in the image.
Step 5: Convert the C, M, Y channels of the new image obtained by thresholding into R,G,B channels using the formula mentioned in Step 3.
Step 6: Combine the R, G, B channels into a 1D array which will be displayed as an output image.

3. Experimental Results

Figure 44: Original Flower Image

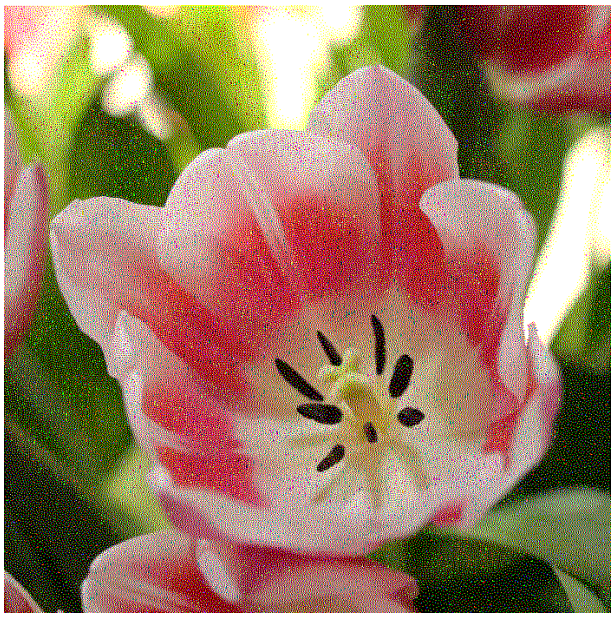


Figure 45: Flower Output by using Separable Error Diffusion

Figure 46: Flower Output by using MBVQ Error Diffusion method

4. Discussion

In Separable Error Diffusion method, each of the C, M, Y channels are diffused separately. Hence, while distributing the error to the three channels separately, it compensates for the error distribution. But when the channels are combined together, the error increases manifold and hence visible artifacts are seen in the image. This can be seen in terms of dots in the output of Separable Error Diffusion. The image obtained by Separable Error Diffusion is noisy compared to the original image but is sharper at the same time. The edges are preserved by using error diffusion.

By using the MBVQ method for error diffusion, the output obtained is visibly better than the output from separable method. This is because in MBVQ method the halftone set consists of only four colors at a time. Also, all the channels are simultaneously considered while applying the error diffusion algorithm and hence the obtained image is closer to the input image provided. Considering a small set of halftones reduces the noise considerably and the brightness content of the image is adjusted in a better way.

Problem 3- Morphological Image Processing

1. Abstract and Motivation

Morphological image processing is a method in which the image is modified depending on its shape. 'Morph' means shape and hence the shape of the element is a factor that is considered while performing these operations. 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.

As morphological image processing techniques are employed on binary images, the gray scale to binary image conversion has to be performed. This is done by applying a threshold to the image. The threshold generally taken into consideration is 127. After thresholding, the image consists of two intensities i.e. 0 and 1. The 0 intensity pixels are considered as background pixels whereas the 1 intensity pixels are considered foreground pixels.

Morphological image processing uses a structuring element which normally is 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.

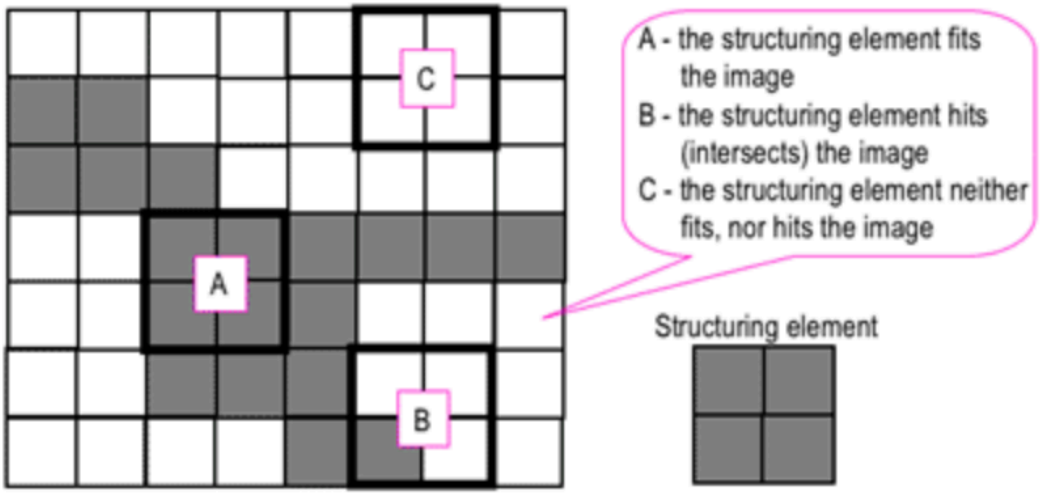


Figure 47: Structuring Element Concept to depict hit or miss.

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. For instance, in the above image, part A exactly matches the structuring element and hence the values in part A are set to foreground value.

Each center pixel has two types of connectivity with its neighboring elements- 4 connectivity and 8 connectivity. In four connectivity, the pixels to the right, up, down and left of the center pixel are assigned weight 2 while in the 8 connectivity, the diagonal right, diagonal left, diagonal down and diagonal up elements are assigned weight 1. This assignment of weight is used to calculate the bonds for the center pixel.

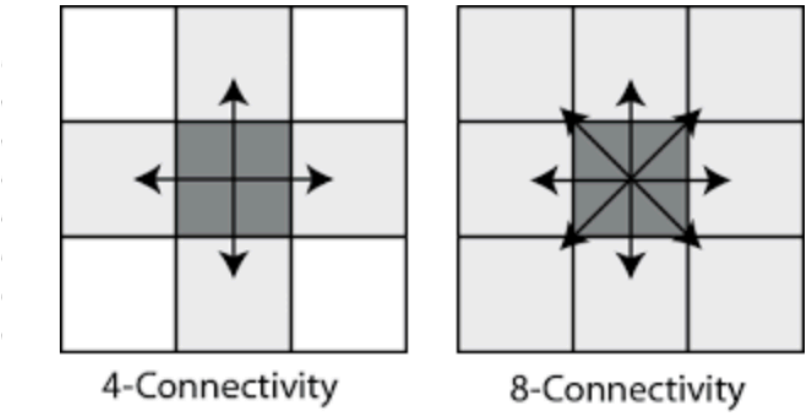


Figure 48: 4 connectivity and 8 connectivity

The bond values are used to find the unconditional and conditional masks to be applied for the given morphological operation.
The conditional and unconditional masks are being defined as follows:

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

Type	Bond	Patterns
S	1	001 100 000 000 010 010 010 010 000 000 100 001
S	2	000 010 000 000 011 010 110 010 000 000 000 010
S	3	001 011 110 100 000 000 000 000 011 010 010 110 110 010 010 011 000 000 000 000 100 110 011 001
TK	4	010 010 000 000 011 110 110 011 000 000 010 010
STK	4	001 111 100 000 011 010 110 010 001 000 100 111
ST	5	110 010 011 001 011 011 110 011 000 001 000 010
ST	5	011 110 000 000 011 110 110 011 000 000 110 011
ST	6	110 011 011 110 001 100
STK	6	111 011 111 110 100 000 000 001 011 011 110 110 110 110 011 011 000 001 000 100 110 111 111 011
STK	7	111 111 100 001 011 110 110 011 001 100 111 111
STK	8	011 111 110 000 011 111 110 111 011 000 110 111
STK	9	111 011 111 111 111 110 100 001 011 011 111 111 110 110 111 111 011 111 100 001 110 111 111 111
STK	10	111 111 111 101 011 111 110 111 111 101 111 111
K	11	111 111 110 011 111 111 111 111 011 110 111 111

Figure 49: Conditional Mask patterns [1]
Where S-shrinking, T-Thinning, K-sleletonizing

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 4-connection	0	0	0	0	0	0							
	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 Offset	0	M	M	M	M	0	0	M	0	0	0	M	
	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 Cluster	0	A	M	M	B	0	0	0	M	M	0	0	
	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 Cluster	M	M	D										
	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 Branch	D	M	0	0	M	D	D	0	M	M	0	D	
	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

Table 14.3-3 Skeletonize Unconditional Mark Patterns

Spur	0	0	0	0	0	0	0	0	M	M	0	0	
	0	M	0	0	M	0	0	M	0	0	M	0	
	0	0	M	M	0	0	0	0	0	0	0	0	
Single 4-connection	0	0	0	0	0	0	0	0	0	0	M	0	
	0	M	0	0	M	M	M	M	0	0	M	0	
	0	M	0	0	0	0	0	0	0	0	0	0	
L Corner	0	M	0	0	M	0	0	0	0	0	0	0	
	0	M	M	M	M	0	0	M	M	M	M	0	
	0	0	0	0	0	0	0	M	0	0	M	0	
Corner Cluster	M	M	D	D	D	D							
	M	M	D	D	M	M							
	D	D	D	D	M	M							
Tee Branch	D	M	D	D	M	D	D	D	D	D	M	D	
	M	M	M	M	M	D	M	M	M	D	M	M	
	D	D	D	D	M	D	D	M	D	D	M	D	
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 Branch	D	M	0	0	M	D	D	0	M	M	0	D	
	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

Figure 50: Unconditional Mask Patterns [1]

Shrinking, thinning and skeletonizing are performed in two stages. In the first stage the boundary of the image is acquired by using the conditional masks. In the second stage the neighboring pixels of the center pixel are taken and compared with the unconditional masks to decide where to erode the image. The algorithm for the three morphological operations is summarized below:

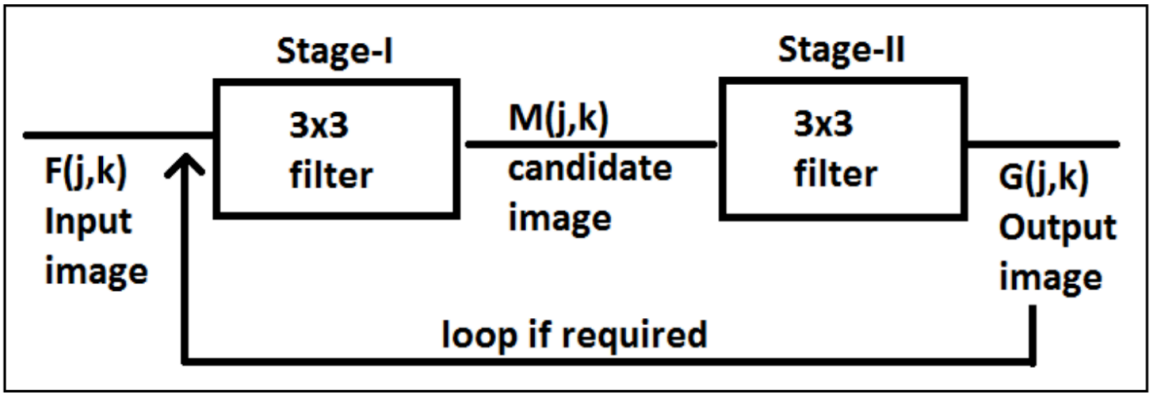


Figure 51: Algorithm for Thinning, Shrinking and Skeletonizing

A combination of the 3x3 masks from the first and the second stage gives a combined effect of a 5x5 mask and hence is useful to obtain the desired morphological operation.

a) Shrinking

1. Abstract and Motivation

Shrinking is a process in which the number of white pixels in the image is reduced and the number of black pixels in the image is increased. If the object is without holes, the result of shrinking algorithm gives a dot, whereas if the image has a hole, the result of shrinking operation is a ring.

2. Approach and Procedure

Task: To apply shrinking algorithm on a binary image "Stars.raw" and find the size and frequency of the stars in that image.

As mentioned above, in the shrinking algorithm, a two stage procedure is used. In the first stage, the conditional masks are checked and the boundary of the image to be changed is obtained. In the second stage, the unconditional masks are used to peel of the boundary obtained in the first stage. Hence, in every iteration the size of the image reduces.

Algorithm for shrinking and counting number of stars:

Step 1: Read the contents of the input image "Stars.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D array for R channel as there is only one channel in the image (it is a gray image).
Step 3: The gray scale image is thresholded to obtain a binary image.
If value of input pixel<=127, output value is 0
If value of input pixel>127, output value is 255
Step 4: Shrinking algorithm is applied on the binary image-Stage 1
Step 4a: If the value of the input binary image is zero, the Marray is assigned value 0
Step 4b: If the value of the input binary image is 1, the 3x3 neighborhood of the center element is stored in a mask. The 9-array elements are then converted to string for ease of comparison.
Step 4b 1: The bond value is calculated for the mask and depending on the bond value, the conditional masks for shrinking are applied to the 9-element mask from step 4b.
Step 4b 2: If the 9-element mask obtained from input binary image matches with any of the conditional masks for shrinking, the value of Marray is set as 1, else it is set to 0.
Step 5: Shrinking algorithm is applied on the binary image-Stage 2
Step 5a: If the Marray value for considered pixel is 0, the output image has same value as that of input image for that pixel.
Step 5b: If the Marray value for the considered pixel is 1, a 3x3 mask of neighboring elements is created.

Step 5b 1: If the mask of 9 elements matches any of the unconditional masks for shrinking, the value of output image pixel is same as the input image pixel, else the value of output pixel is set to 0.
Step 5: The algorithm is run iteratively till all the stars in the image are shrunk to a single value.
Step 6: The output image is converted back to 0-255 values using the same concept mentioned in step 3.
Step 7: The entire output image is scanned and checked for values of 255. The number of pixels having value 255 is stored in a variable count. This variable depicts the number of white dots in the image obtained after shrinking.
Step 8: The 2D array is then combined into a 1D array for displaying as an output image.

Algorithm for counting stars and counting size and frequency of stars:

Step1: For every iteration in the shrinking algorithm, the output image is scanned and converted to a mask of size 9.
Step 2: If the center element of the mask generated from step 1 has center value 1, the count variable is incremented. This is how for every iteration the number of white dots is counted. The value of count for every iteration is stored in an array.
Step 3: To get unique number of stars in every iteration, the value of count for every iteration is subtracted from the count for previous iteration.
Step 4: Transfer the iteration index and the count values into a file and read this file in Matlab to plot the histogram of the size of stars and their frequencies.

This is how the number of stars reduced to as single dot in every iteration increases. **The iteration number gives the size of stars and the number of stars reduced in that iteration gives the frequency.**
A total number of 15 iterations are required to shrink the image into dots.

3. Experimental Results

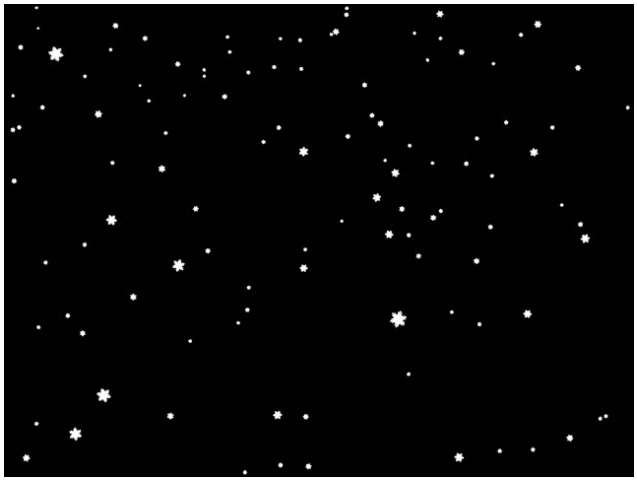


Figure 52: Original Stars Image

Figure 53: Shrinked image after 5 iterations

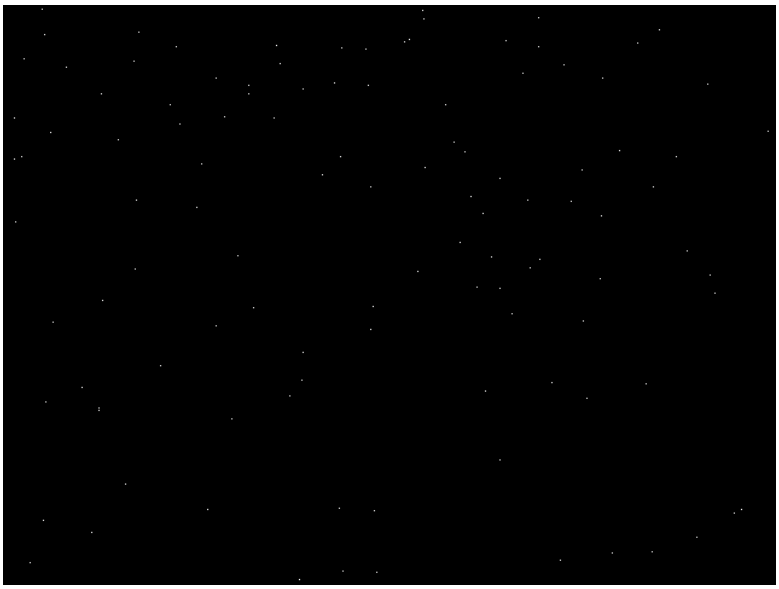


Figure 54: Final shrinked image after 15 iterations

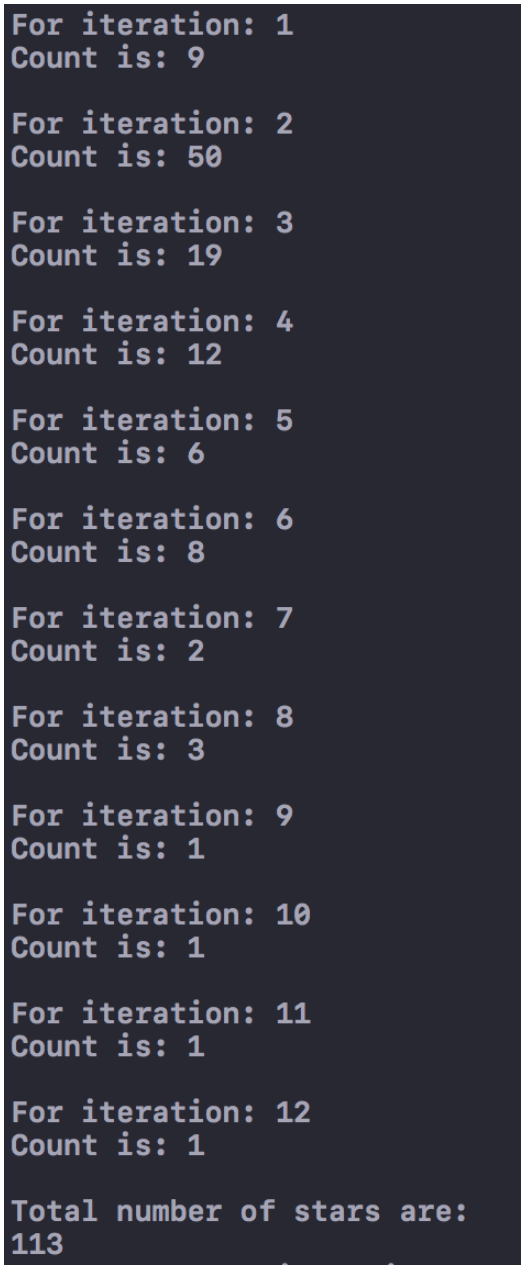


Figure 55: Output showing total number of stars and frequency of stars at each iteration

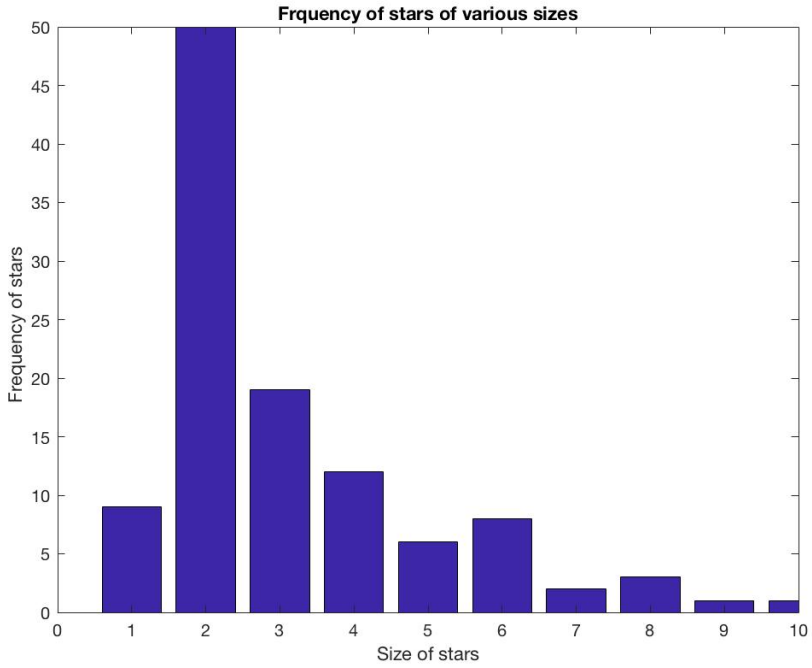


Figure 56: Histogram showing size and frequency of stars

4. Discussion

The shrinking algorithm was implemented on the star image. In 15 iterations, all the stars in the image were reduced to a white dot and hence the size and frequency of the image could be calculated. With every iteration, the size of the stars reduces. This can be seen from Figure where after 5 iterations, the size of stars has reduced considerably.

There a total number of 112 stars in the image. However, one star in the image gets split at the 5^{th} iteration and hence after applying the shrinking algorithm, the total number of stars are counted as 113.

The size and frequency of the stars are as mentioned below:

Size of Stars (Iteration number)	Frequency of stars
1	9
2	50
3	19
4	12
5	6
6	8
7	2
8	3
9	1
10	1
11	1
12	1
Total	**113**

b) Thinning

1. Abstract and motivation

Thinning is a process in which the number of white pixels in the image is reduced and the number of black pixels in the image is increased. It is a process in which the black pixels are erased such that an object without holes erodes to a minimally connected stroke located equidistant from its nearest outer boundaries, and an object with holes erodes to a minimally connected ring midway between each hole and its nearest outer boundary. [1] Hence, the final output obtained is a thin line.

2. Approach and Procedure

Task: To apply thinning algorithm on a binary image "Jigsaw_1.raw" and study the output.

As mentioned above, in the thinning algorithm, a two stage procedure is used. In the first stage, the conditional masks are checked and the boundary of the image to be changed is obtained. In the second stage, the unconditional masks are used to peel of the boundary obtained in the first stage. Hence, in every iteration the size of the image reduces. As the image given as input has white as background and black as foreground, it has to be inverted to obtain white as foreground and black as background as the algorithm and the masks are designed for the latter.

Bridging

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. The algorithm used for bridging is as follows:

$$G(j,k) = X \cup [P_1 \cup P_2 \cup \cdots \cup P_6]$$

where

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

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

$$P_3 = \bar{X}_0 \cap \bar{X}_6 \cap X_7 \cap [X_2 \cup X_3 \cup X_4]$$

$$P_4 = \bar{X}_0 \cap \bar{X}_2 \cap X_1 \cap [X_4 \cup X_5 \cup X_6]$$

$$P_5 = \bar{X}_2 \cap \bar{X}_4 \cap X_3 \cap [X_0 \cup X_6 \cup X_7]$$

$$P_6 = \bar{X}_4 \cap \bar{X}_6 \cap X_5 \cap [X_0 \cup X_1 \cup X_2]$$

and

$$P_Q = L_1 \cup L_2 \cup L_3 \cup L_4$$

$$L_1 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap \bar{X}_7$$

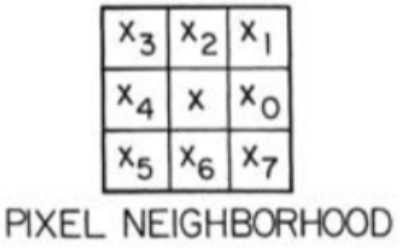
$$L_2 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap \bar{X}_7$$

$$L_3 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap X_7$$

$$L_4 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap X_7$$

Figure 57: Algorithm for bridging operation [1]

G (i,j) is the output of the bridging process and the mask selected from the input image is as given below:



Algorithm for thinning:

Step 1: Read the contents of the input image "Jigsaw_1.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D array for R channel as there is only one channel in the image (it is a gray image).
Step 3: The gray scale image is thresholded to obtain a binary image.
If value of input pixel<=127, output value is 0
If value of input pixel>127, output value is 255
Step 4: Invert the normalized input image from Step 3 by applying following logic
If input pixel value is 0, output value is 1
If input pixel value is 1, output value is 0
Step 5: Thinning algorithm is applied on the binary image-Stage 1
Step 5a: If the value of the input binary image is zero, the Marray is assigned value 0.
Step 5b: If the value of the input binary image is 1, the 3x3 neighborhood of the center element is stored in a mask. The 9-array elements are then converted to string for ease of comparison.
Step 5b 1: The bond value is calculated for the mask and depending on the bond value, the conditional masks for thinning are applied to the 9-element mask from step 4b.
Step 5b 2: If the 9-element mask obtained from input binary image matches with any of the conditional masks for thinning, the value of Marray is set as 1, else it is set to 0.
Step 6: Thinning algorithm is applied on the binary image-Stage 2
Step 6a: If the Marray value for considered pixel is 0, the output image has same value as that of input image for that pixel.
Step 6b: If the Marray value for the considered pixel is 1, a 3x3 mask of neighboring elements is created.
Step 6b 1: If the mask of 9 elements matches any of the unconditional masks for thinning, the value of output image pixel is same as the input image pixel, else the value of output pixel is set to 0.
Step 7: Apply the bridging algorithm mentioned above to the output obtained from thinning in Step 6b.
Step 8: The output image is converted back to 0-255 values using the same concept mentioned in step 3.
Step 9: The output image is inverted using the logic explained in Step 4.
Step 10: The 2D array is then combined into a 1D array for displaying as an output image.

3. Experimental Results

Figure 58: Thinning after 5 iterations

Figure 59: Thinning after 10 iterations

Figure 60: Thinning after 15 iterations

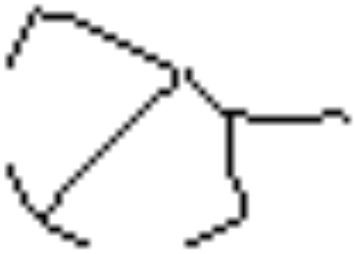


Figure 61: Final Thinning output

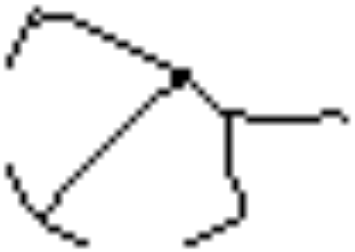


Figure 62: Thinning output after bridging

4. Discussion

With every iteration, the size of the jigsaw puzzle reduces. This can be seen from the outputs for 5,10,15,20 iterations. The thinning algorithm is implemented using a lot of conditional and unconditional masks. A total of 20 iterations are required for the output to be converted into a thin line.
The best part of the algorithm is that after the image has been reduced to a line, even if the number of iterations applied increases, the output does not change.
As there was a gap in the output obtained after thinning operation, I implemented the bridging operation. Due to this, the breaks in the thinning output got eliminated and the output was composed of one stroke.

c) Skeletonizing

1. Abstract and Motivation

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. Skeletonizing is a type of thinning where we erase black pixels such that an object without holes erodes to a minimally connected rooted stroke located equidistant from its nearest outer boundaries, and an object with holes erodes to a minimally connected ring midway between each hole and its nearest outer boundary. [1]

2. Approach and Procedure

Task: To apply skeletonizing algorithm on a binary image "Jigsaw_2.raw" and study the output.

As mentioned above, in the skeletonizing algorithm, a two stage procedure is used. In the first stage, the conditional masks are checked and the boundary of the image to be changed is obtained. In the second stage, the unconditional masks are used to peel of the boundary obtained in the first stage. Hence, in every iteration the size of the image reduces. As the image given as input has white as background and black as foreground, it has to be inverted to obtain white as foreground and black as background as the algorithm and the masks are designed for the latter.

Bridging

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. The algorithm used for bridging is as follows:

$$G(j,k) = X \cup [P_1 \cup P_2 \cup \cdots \cup P_6]$$

where

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

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

$$P_3 = \bar{X}_0 \cap \bar{X}_6 \cap X_7 \cap [X_2 \cup X_3 \cup X_4]$$

$$P_4 = \bar{X}_0 \cap \bar{X}_2 \cap X_1 \cap [X_4 \cup X_5 \cup X_6]$$

$$P_5 = \bar{X}_2 \cap \bar{X}_4 \cap X_3 \cap [X_0 \cup X_6 \cup X_7]$$

$$P_6 = \bar{X}_4 \cap \bar{X}_6 \cap X_5 \cap [X_0 \cup X_1 \cup X_2]$$

and

$$P_Q = L_1 \cup L_2 \cup L_3 \cup L_4$$

$$L_1 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap \bar{X}_7$$

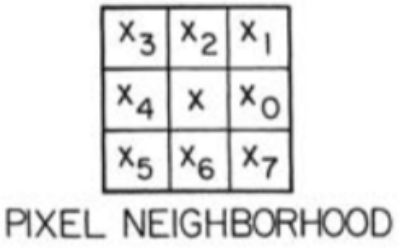
$$L_2 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap \bar{X}_7$$

$$L_3 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap X_7$$

$$L_4 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap X_7$$

Figure 63: Algorithm for bridging operation [1]

G (i,j) is the output of the bridging process and the mask selected from the input image is as given below:



Algorithm for skeletonizing:

Step 1: Read the contents of the input image "Jigsaw_2.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D array for R channel as there is only one channel in the image (it is a gray image).
Step 3: The gray scale image is thresholded to obtain a binary image.
If value of input pixel<=127, output value is 0
If value of input pixel>127, output value is 255
Step 4: Invert the normalized input image from Step 3 by applying following logic
If input pixel value is 0, output value is 1
If input pixel value is 1, output value is 0
Step 5: Skeletonizing algorithm is applied on the binary image-Stage 1
Step 5a: If the value of the input binary image is zero, the Marray is assigned value 0
Step 5b: If the value of the input binary image is 1, the 3x3 neighborhood of the center element is stored in a mask. The 9-array elements are then converted to string for ease of comparison.
Step 5b 1: The bond value is calculated for the mask and depending on the bond value, the conditional masks for skeletonizing are applied to the 9-element mask from step 4b.
Step 5b 2: If the 9-element mask obtained from input binary image matches with any of the conditional masks for skeletonizing, the value of Marray is set as 1, else it is set to 0.
Step 6: Skeletonizing algorithm is applied on the binary image-Stage 2
Step 6a: If the Marray value for considered pixel is 0, the output image has same value as that of input image for that pixel.
Step 6b: If the Marray value for the considered pixel is 1, a 3x3 mask of neighboring elements is created.
Step 6b 1: If the mask of 9 elements matches any of the unconditional masks for skeletonizing, the value of output image pixel is same as the input image pixel, else the value of output pixel is set to 0.
Step 7: Apply the bridging algorithm mentioned above to the output obtained from skeletonizing in Step 6b.
Step 8: The output image is converted back to 0-255 values using the same concept mentioned in step 3.
Step 9: The output image is inverted using the logic explained in Step 4.
Step 10: The 2D array is then combined into a 1D array for displaying as an output image.

3. Experimental Results

Figure 64: Skeletonizing after 5 iterations

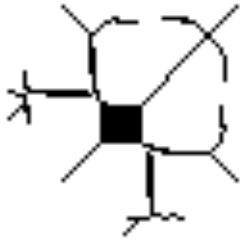


Figure 65: Skeletonizing after 10 iterations

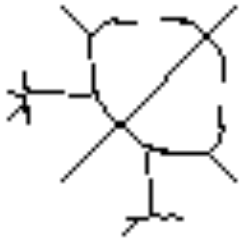


Figure 66: Skeletonizing after 15 iterations

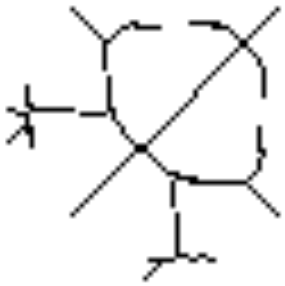


Figure 67: Final Skeletonizing Output

Figure 68: Final Skeletonizing output after bridging

4. Discussion

With every iteration, the size of the jigsaw puzzle reduces. This can be seen from the outputs for 5,10,15,30 iterations. The skeletonizing algorithm is implemented using a lot of conditional and unconditional masks. A total of 30 iterations are required for the output to be converted into a skeleton.
The best part of the algorithm is that after the image has been reduced to a skeleton, even if the number of iterations that are applied increases, the output does not change.
As there was a gap in the output obtained after skeletonizing operation, I implemented the bridging operation. Due to this, the spaces in the skeletonizing output got eliminated and the output looked better.

d) Counting Game

1. Abstract and Motivation

This is a very interesting game in which the number of unique objects in the image are to be counted. Morphological processing is a very important tool used in object recognition and counting. The three morphological methods mentioned in the earlier parts along with some new logical operations are used to count the unique objects. The jigsaws given in the original image are rotated and flipped versions of each other. Hence, these operations also have to be performed on the jigsaws in the image.

2. Approach and Procedure

There are two parts in this question:

1) Finding the total number of objects in the image
2) Finding the unique number of objects in the image

Task: To find the total number of jigsaws in the image and also find the unique jigsaws out of them.

The algorithm for each part is as given below

1) Counting total number of objects in the image

Three concepts namely erosion, dilation and shrinking are used to obtain the total number of objects in the image.

Erosion:

Erosion is a morphological operation that is used on binary images as well as gray scale images. In erosion, the white pixels are eroded in such a way that the boundary of foreground pixels is reduced. The number of holes in the image increases whereas the area of foreground pixel reduces.

Erosion operation has two inputs. One is the input image and other is the structuring element. When the structuring element matches the input image exactly, the pixel underlying it is set to 1. Erosion is only done on foreground pixels i.e. pixels having intensity value 1 in the original image.

A typical structuring element used for erosion is as follows:

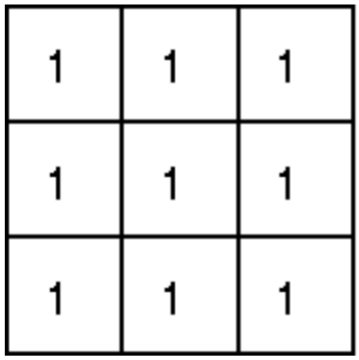


Figure 69: 3x3 Structuring element used for erosion

The erosion operation can be depicted as follows:

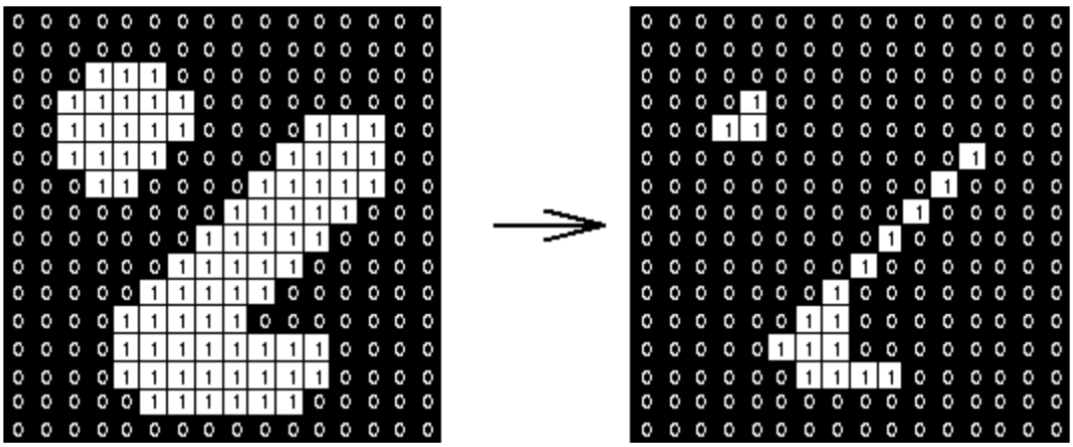


Figure 70: Erosion operation on input binary image
(Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm)

Dilation:

Dilation is a morphological operation that is used on binary images as well as gray scale images. In erosion, the white pixels are dilated in such a way that the boundary of foreground pixels is increased. The number of holes in the image reduces whereas the area of foreground pixel increases.

Dilation operation has two inputs. One is the input image and other is the structuring element. When the structuring element matches any pixel of the input image, the pixel underlying it is set

to 1. Dilation is only done on foreground pixels i.e. pixels having intensity value 1 in the original image.

A typical structuring element used for dilation is as follows:

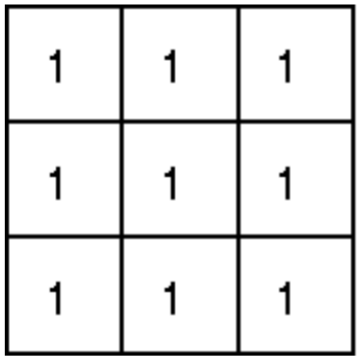


Figure 71: 3x3 Structuring element used for dilation

The dilation operation can be depicted as follows:

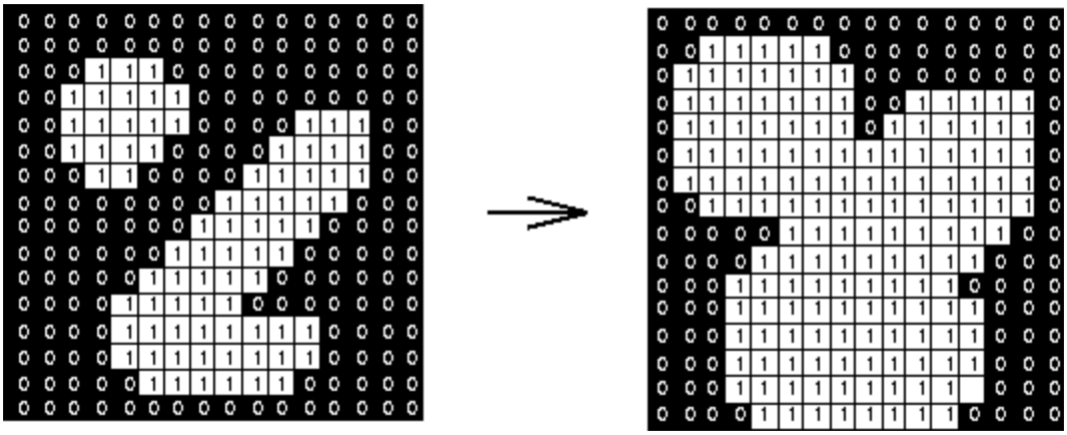


Figure 72: Dilation operation on input binary image
(Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm)

As mentioned in the previous section, in the shrinking algorithm, a two stage procedure is used. In the first stage, the conditional masks are checked and the boundary of the image to be changed

is obtained. In the second stage, the unconditional masks are used to peel of the boundary obtained in the first stage. Hence, in every iteration the size of the image reduces.

Hence, **erosion with 11 as the size of the structuring element, dilation with 27 as the size of the structuring element and shrinking is used to obtain the number of objects in the image.**

Algorithm for Counting the number of objects in the image:

Step 1: Read the contents of the input image "Board.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D array for R channel as there is only one channel in the image (it is a gray image).
Step 3: The gray scale image is thresholded to obtain a binary image.
If value of input pixel<=127, output value is 0
If value of input pixel>127, output value is 255
Step 4: Invert the normalized input image from Step 3 by applying following logic
If input pixel value is 0, output value is 1
If input pixel value is 1, output value is 0
Step 5: Apply Erosion algorithm
Step 5a: Scan the entire input image and place 9 elements surrounding the center element of the input array in a mask.
Step 5b: Check how many elements in the input image have a value 1.
Step 5c: If all the elements in the mask obtained from Step 5a have value 1, assign a value 1 to the output image.
Step 6: Apply Dilation algorithm
Step 6a: Scan the entire output image obtained from erosion operation and place 9 elements surrounding the center element of the input array in a mask.
Step 5b: Check how many elements in the input image have a value 1.
Step 5c: If at least one of the element in the mask obtained from Step 5a has value 1, assign a value 1 to the output image.
Step 6: Shrinking algorithm is applied on the binary image obtained from dilation operation-Stage 1
Step 6a: If the value of the input binary image obtained from dilation is zero, the Marray is assigned value 0
Step 6b: If the value of the input binary image obtained from dilation is 1, the 3x3 neighborhood of the center element is stored in a mask. The 9-array elements are then converted to string for ease of comparison.
Step 6b 1: The bond value is calculated for the mask and depending on the bond value, the conditional masks for shrinking are applied to the 9-element mask from step 4b.
Step 6b 2: If the 9-element mask obtained from input binary image matches with any of the conditional masks for shrinking, the value of Marray is set as 1, else it is set to 0.
Step 7: Shrinking algorithm is applied on the binary image-Stage 2
Step 7a: If the Marray value for considered pixel is 0, the output image has same value as that of input image for that pixel.

Step 7b: If the Marray value for the considered pixel is 1, a 3x3 mask of neighboring elements is created.

Step 7b 1: If the mask of 9 elements matches any of the unconditional masks for shrinking, the value of output image pixel is same as the input image pixel, else the value of output pixel is set to 0.

Step 8: The algorithm is run iteratively until all the objects in the image are shrunk to dots.

Step 9: The output image is converted back to 0-255 values using the same concept mentioned in step 3.

Step 10: The entire output image obtained from shrinking is scanned and checked for values of 255. The number of pixels having value 255 is stored in a variable count. This variable depicts the number of white dots in the image obtained after shrinking. This displays the number of objects in the image.

Step 11: The 2D array is then combined into a 1D array for displaying as an output image.

Connected Component Analysis:

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.

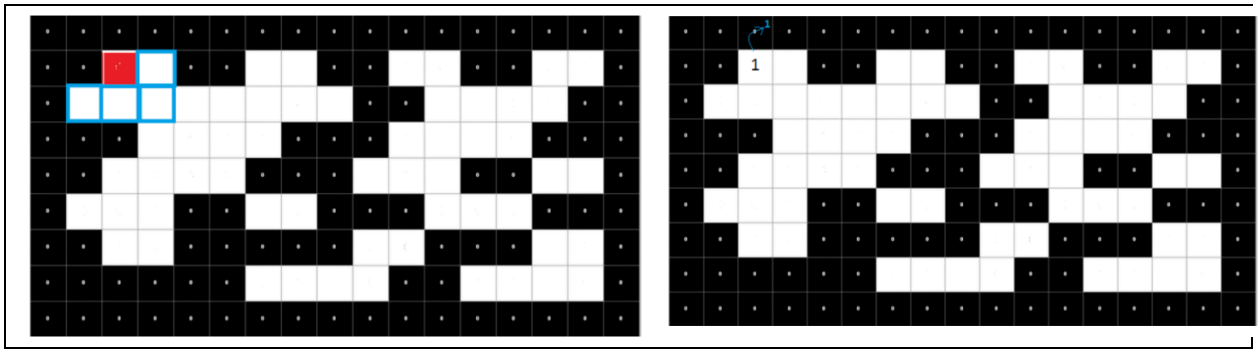
In connected Component Analysis, the entire image is canned from top to bottom and left to right to identify pixels having connectivity i.e. same intensity values. Connected Component Analysis is mainly used in Automated Image analysis applications.

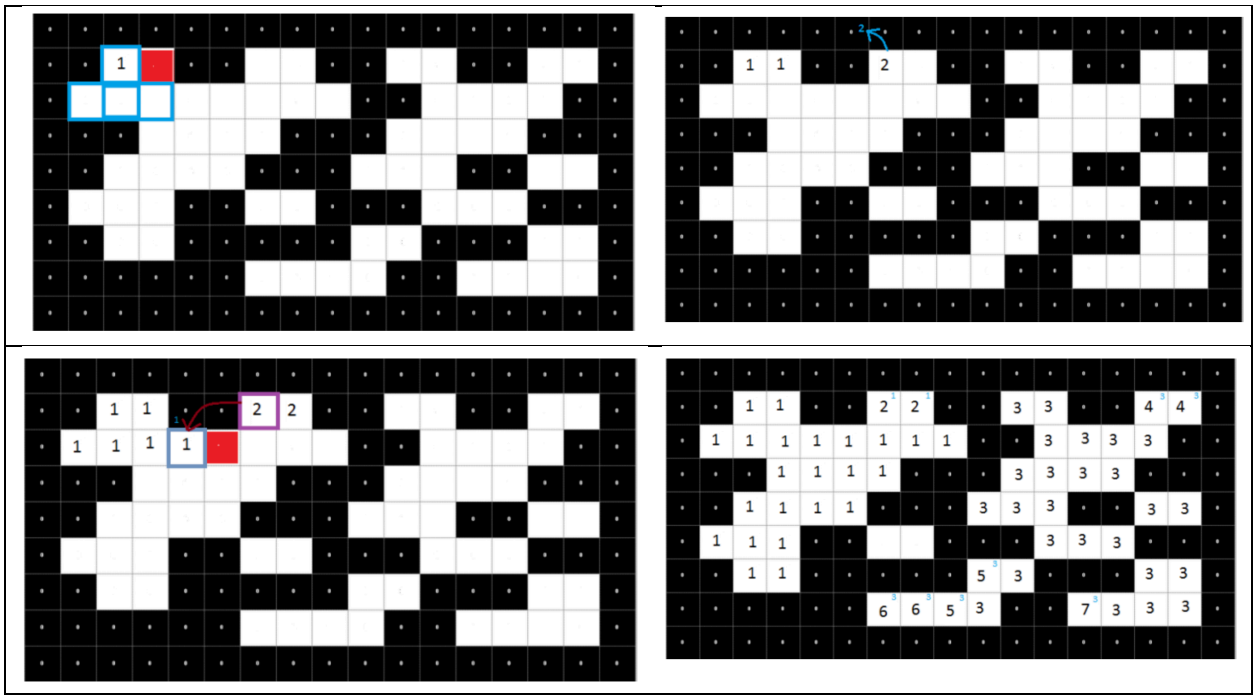
While labelling the elements in any scan for the connected component analysis, four neighbors of the center pixel are considered. Those are the one above the center element, left one and left diagonal and right diagonal terms. The labels are then assigned as follows:

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 neighborhood 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.





In the above example, the first pass of scanning in the Connected Component analysis has been illustrated.

The main components of this image are the holes and protrusions. These holes and protrusions can be used to find the unique objects in the image.

Algorithm for counting unique number of objects in the image:
Step 1: Read the contents of the input image "Board.raw" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step 2: Convert the 1D input image into a separate 2D array for R channel as there is only one channel in the image (it is a gray image).
Step 3: The gray scale image is thresholded to obtain a binary image.
If value of input pixel<=127, output value is 0
If value of input pixel>127, output value is 255
Step 4: Invert the normalized input image from Step 3 by applying following logic
If input pixel value is 0, output value is 1
If input pixel value is 1, output value is 0
Step 5: Apply the connected Component Analysis Algorithm-First pass
Step 5a: Check for the intensity value of the input image. If it's one, proceed.
Step 5b: Take four neighboring elements of the center pixel namely left, right, diagonal left, diagonal right into an array.
Step 5c: Check if all the elements of the array generated in step 5b are 0. If Yes, assign a new label to the element.

Step 5d: If the elements of the array generated in Step 5b are non-zero, store the positive elements and apply bubble sort to find the minimum value of the array. Assign the minimum value to the center pixel.

Step 6: Apply the connected Component Analysis Algorithm-Second pass

Step 6a: Apply recursion on the Input array from the first pass to label the objects depending on their connectivity.

Step 6b: Store the elements of output array from Step 6a into another output array to be used in the later stages.

Step 7: Run an outer for loop for the 16 labels obtained in the Connected Component analysis output from Step 6b.

Step 7a: From the output array of labels, find the minimum y value for every label by using bubble sort.

Step 7b: For every min y value for each label find the corresponding minimum x value. This will give the starting point of each object in the image.

Step 7c: Construct a square of size 45x45 from the starting x and y point obtained from steps 7a and 7b. The square should have intensity value 1.

Step 8: XOR the original inverted image with the square array to obtain the holes and protrusions of the image.

Step 8a: The holes and protrusions can be found out by finding the center of each jigsaw puzzle using the minimum x and minimum y obtained for each object in Step 7.

Step 8b: The holes are protrusions are found out in each direction namely up, down, left and right by counting the number of steps having intensity value 0 in a variable count.

If count<23 which is distance from center of square, it is a hole.

If count=23 or 24, it is protrusion

Else it is neither a hole nor a protrusion.

Holes are labelled as 1, protrusions as -1 and plain surfaces as 0.

Hence an array of 4 elements is created for every object in the image which has the number of holes and protrusions.

Step 9: The holes and protrusion array obtained in Step 8b are then rotated and flipped and compared with all other objects to find if the original, rotated, flipped or flipped and rotated versions are similar to any other object. This is done by comparing each row and all it's possible combinations with all other rows. If a match is found, the count is incremented.

Step 9a: A vector is deleted from the array if it resembles any combination mentioned above and hence the number of unique objects is obtained.

3. Experimental Results

Experimental Results for counting number of objects

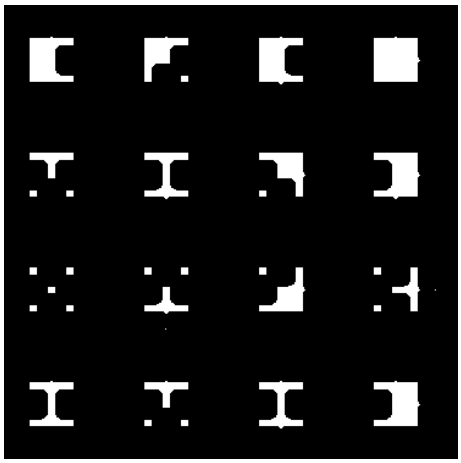


Figure 73: Image after Erosion operation

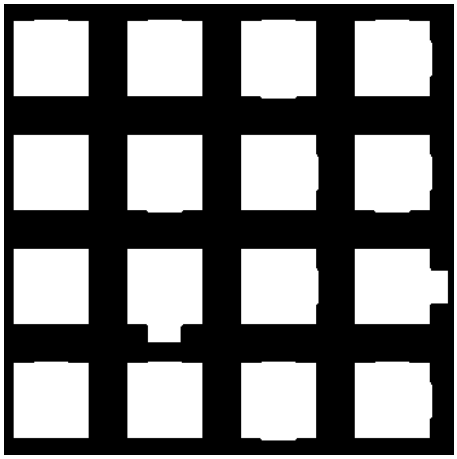


Figure 74: Image after Dilation operation

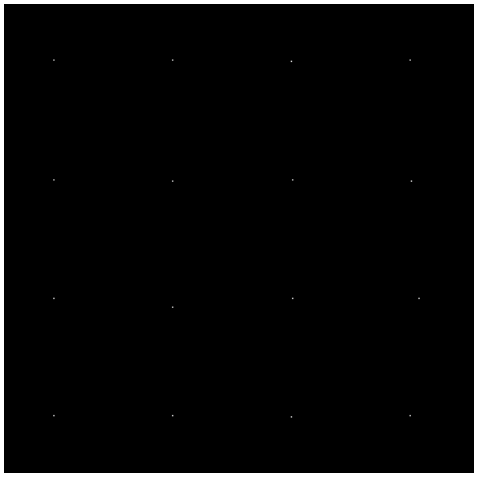


Figure 75: Image after shrinking operation

```
Total number of objects in the image are:
16
```

Figure 76: Output of total number of objects in the image

Experimental Results for counting number of unique objects

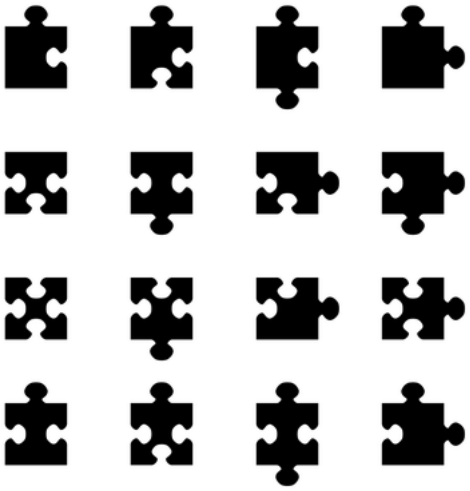


Figure 77: Original Board Image

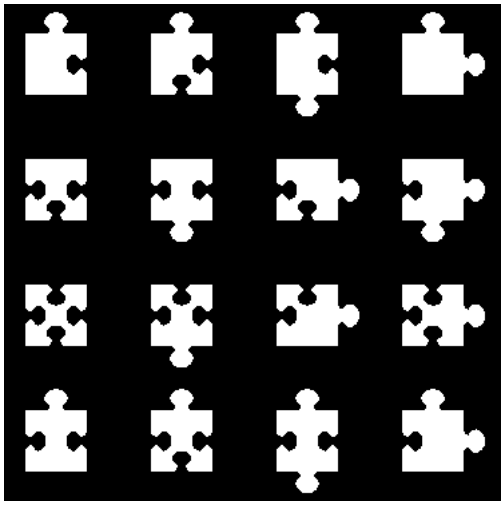


Figure 78: Inverted Board Image

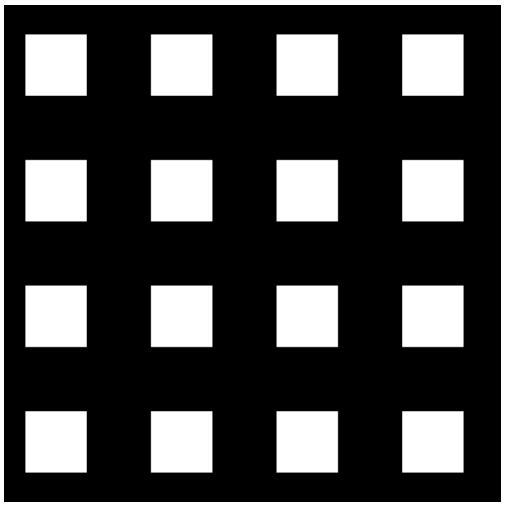


Figure 79: Squares output image

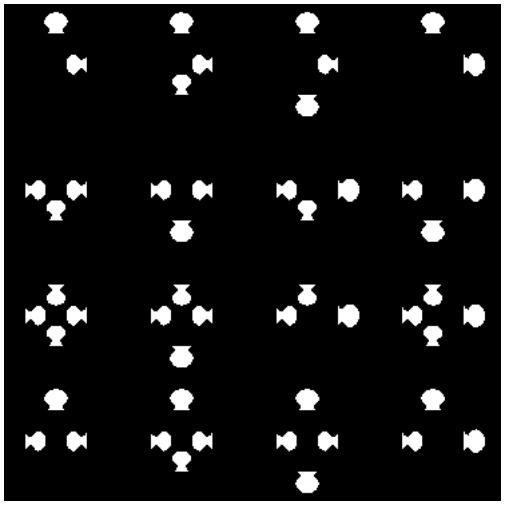


Figure 80: Image with holes and protrusions (XOR image)

```
X and Y starting indices for each object:
22 16
22 110
22 204
22 298
116 16
116 110
116 204
116 298
210 16
210 110
210 204
210 298
304 16
304 110
304 204
304 298
```

Figure 81: Starting index for X and Y for each object

```
Holes and Protrusions array:
-1 1 0 0
-1 1 1 0
-1 1 -1 0
-1 -1 0 0
0 1 1 1
0 1 -1 1
0 -1 1 1
0 -1 -1 1
1 1 1 1
1 1 -1 1
1 -1 0 1
1 -1 1 1
-1 1 0 1
-1 1 1 1
-1 1 -1 1
-1 -1 0 1
```

Figure 82: Array of holes and protrusions

```
Number of unique and matched pieces in every iteration:
0
1 6 10
2
3
4
5 12
6 10
7 15
8
9 11 13
10
11 13
12
13
14
15
```

Figure 83: The object status at each iteration

```
Total Number of unique objects in the image is:
10
```

Figure 84: Final output for total number of unique objects in the image

4. Discussion

1. The total number of objects in the image obtained by running the algorithm is 16 which matches the total number of objects actually present in the image. Initially I used just the shrinking algorithm, but that led to one object being split up. Hence, I came up with the idea of using erosion, dilation and then shrinking to count the number of objects.
2. The unique number of objects was obtained using the concept of connected component analysis along with logical operation of XOR. The unique number of pieces came out to be 10 which coincides with the actual unique objects in the image. The object present in the image are flipped and rotated versions of each other. Hence, all the possible combinations had to be checked and the concept of rotation and flipping had to be used on the images.

REFERENCES

[1] William K. Pratt -Digital Image Processing, Fourth Edition

[2] "Analytical Methods for Squaring the Disc"- Chamberlain Fong- Seoul ICM 2014