Muthulakshmi Chandrasekaran

4486-1802-42

muthulac@usc.edu

EE569 Homework #2

March 4 ,2018

Problem 1: Geometric Image Modification

Part A – Geometrical Warping

A.1 Motivation

Geometric Transformations are those transformations which operate on the relationships between the pixel locations. In these operations, a pixel at a location (x, y) is mapped to a new location (x', y') based on some criterion, which can be given as

$$x' = T_x(x, y) \quad \text{and}\; y' = T_y(x, y)$$

where T is the transformation that is applied. This geometric transformation mainly finds application in the correction of geometric distortions and reduction of sampling artifacts. Some basic geometric transformations include Rotation, Scaling, Translation etc. Warping can be defined as a form of geometric modification which is applied to the domain of the image. Here all the geometric properties of the image are modified. The intensity levels of the actual image and the warped image are the same.

A.2 Approach

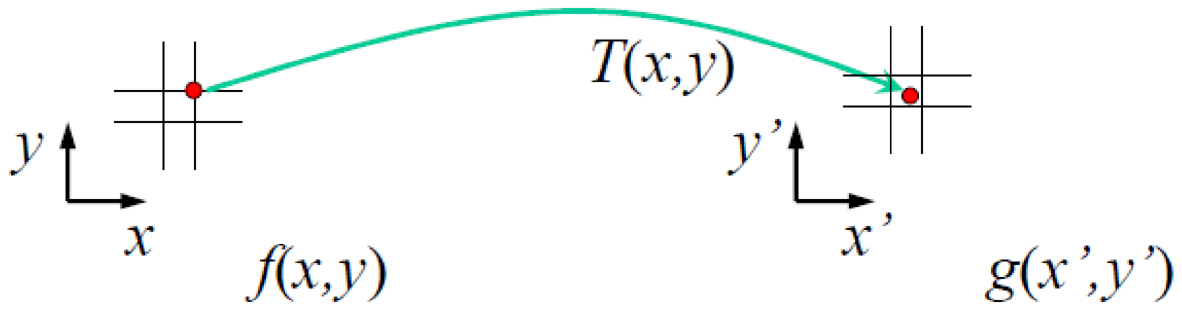
Warping is generally done in two steps :

i. Calculation of the corresponding pixel locations
ii. Resampling the pixel co – ordinates

The image co – ordinates are discrete values. The new image co – ordinates are defined by certain transformation formulae. This mapping is done first. The resulting new image co – ordinates may not be fully discrete values. So, interpolation of the resulting new co – ordinate positions are done. There are two methods to implement warping.

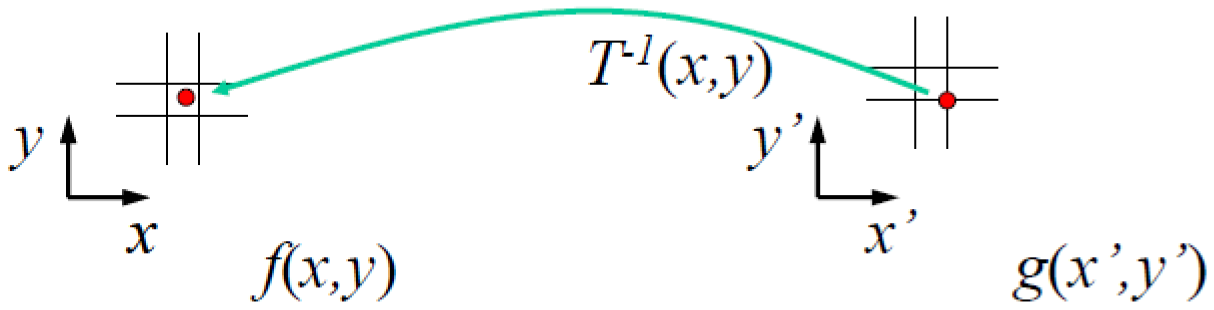
i. Forward Mapping

Here each pixel in the source image is mapped to the destination image.



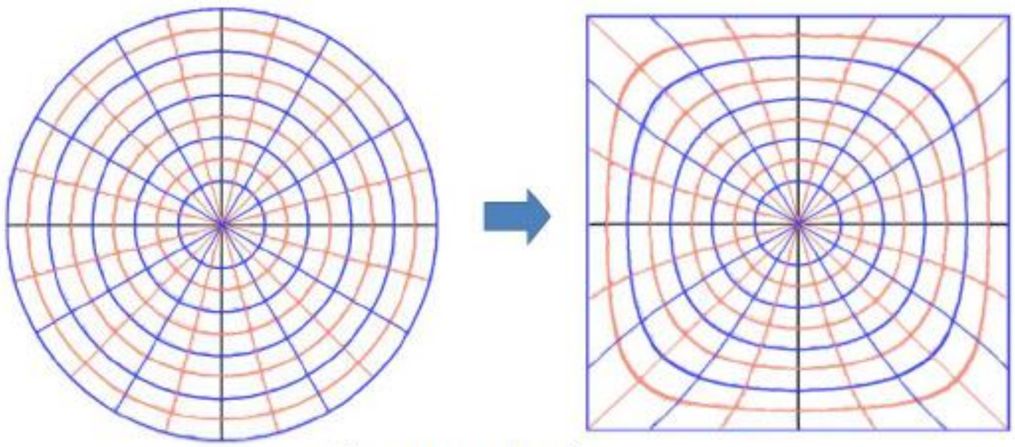
ii. Inverse Mapping

Here, for each pixel location in the destination image, the corresponding source pixel is determined.



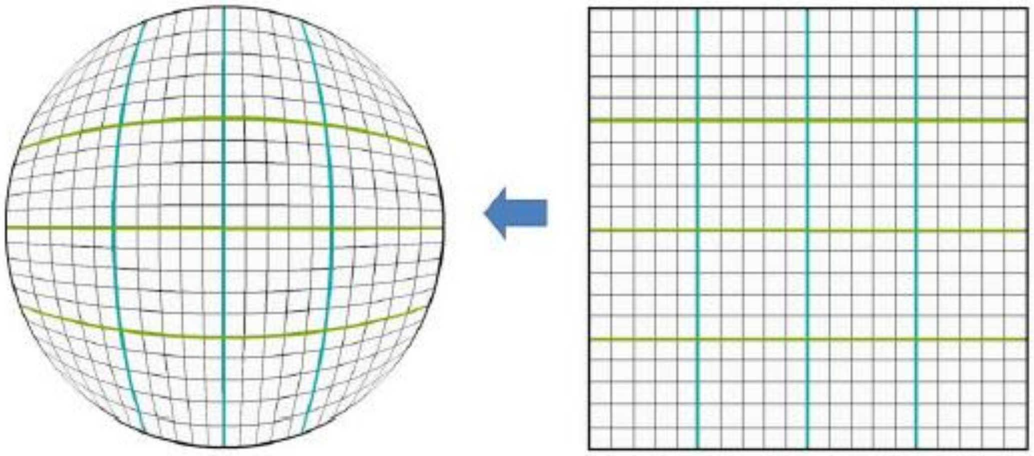
In this problem, geometrical warping must be done to convert a square image into a disc image, and the disc image back to a square image.

Conversion from Square to disc and converting the disc back to a square is done using elliptical mapping as shown below [3].



Disc to square mapping:

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



Square to disc mapping:

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

The following considerations are made during warping:

- The pixels on the boundary of the square are mapped to the boundary of the circle.
- Center of the square image is the center of the circle image.
- One – one mapping of pixels between the square image and the disc image.

The problem is coded using C++ using the following approach.

i. The input image is obtained from the file using the file commands, and stored in an array.
ii. The output image array for storing the warped image is defined.
iii. Here forward mapping is implemented. For every (x, y) in the input Square image, the corresponding pixel locations (u, v) in the warped image is found. If the co – ordinates are floating point numbers, bilinear interpolation is done to round off to the nearest integer values. Then the pixel value at the location (x,y) is copied to the position (u,v).
iv. The array for storing the Recovered Image is defined.
v. For getting back the actual square image from the warped disc image, forward mapping is again implemented. For every $(u1, v1)$ in the disc image, the corresponding pixel locations $(x1, y1)$ in the Recovered Square image is found. If the co – ordinates are floating point numbers, bilinear interpolation is done to round off to the nearest integer values. Then the pixel value at the location (u1,v1) is copied to the position (x1,y1).
Thus warping is implemented.

A.3 Results

The following results are obtained

Figure 1.1(a) shows the actual puppy.raw image which is given as the input. Figure 1.1(b) shows the Warped image. Figure 1.1(c) shows the recovered image.

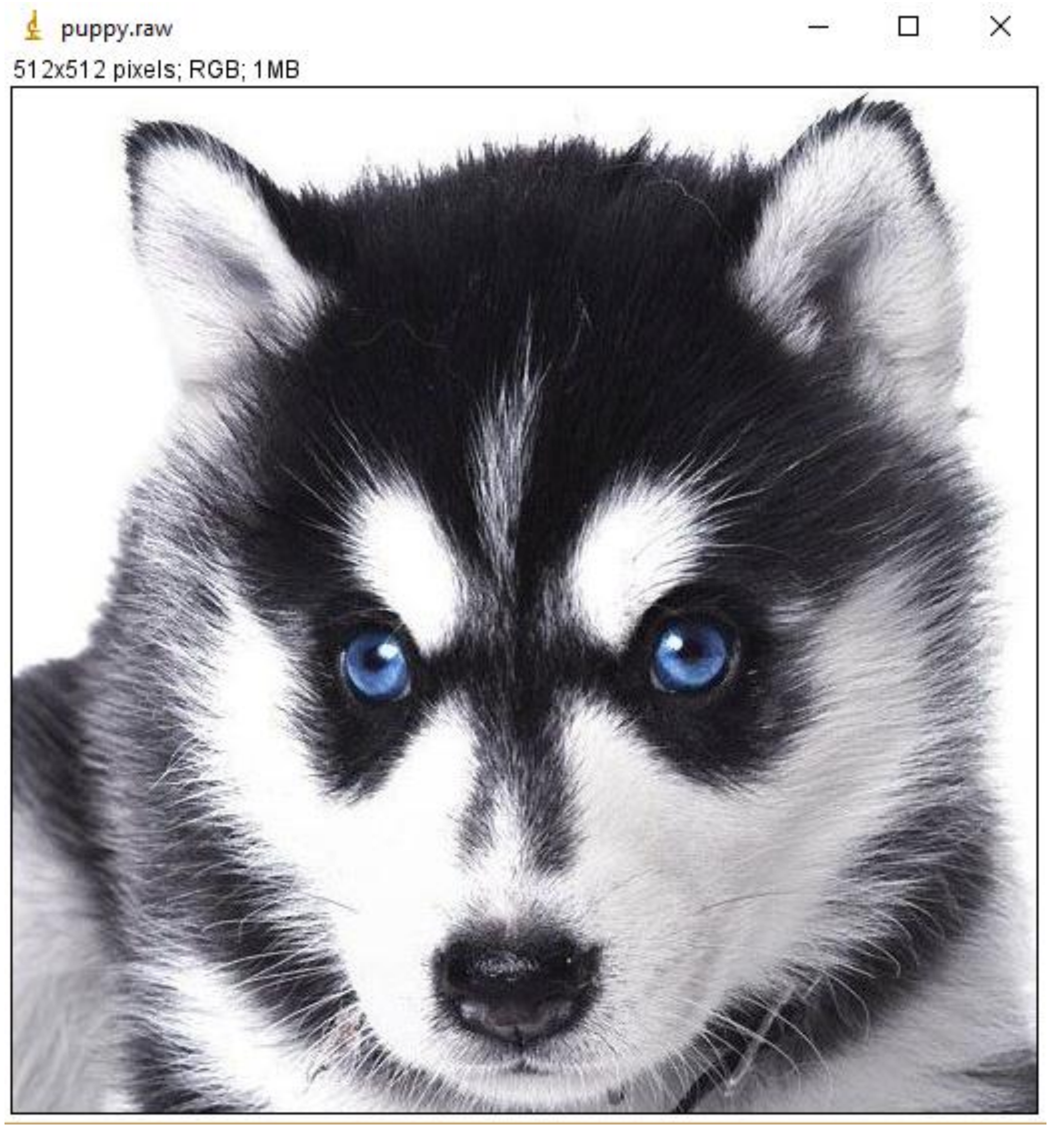
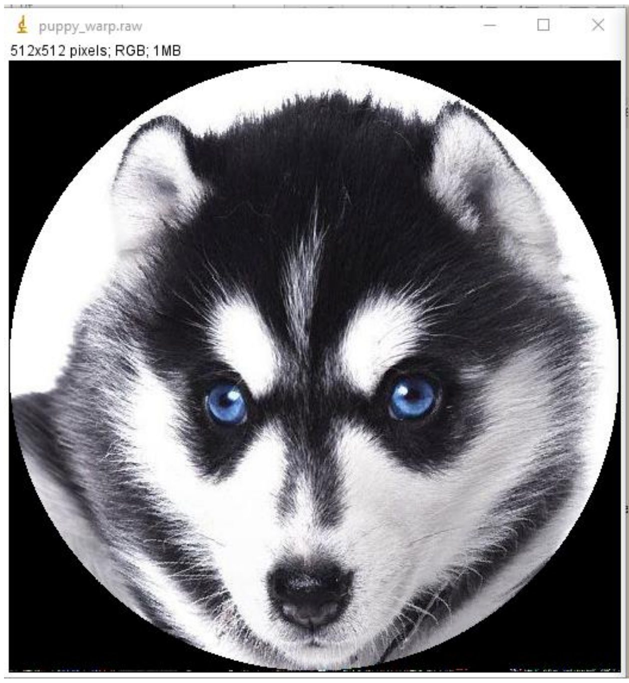


Figure 1.1(a) Input – puppy.raw



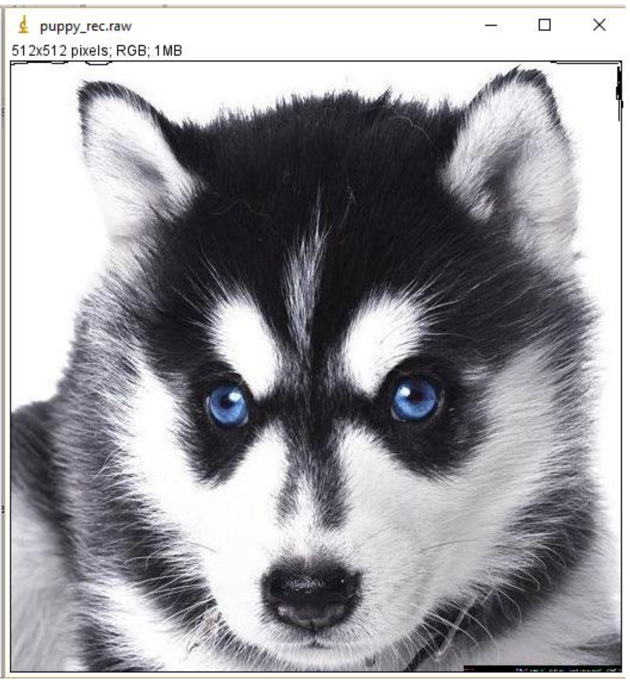


Figure 1.1(b) Warped Image **Figure 1.1(c) Recovered – puppy. raw**

Figure 1.2(a) shows the actual tiger.raw image which is given as the input. Figure 1.2(b) shows the Warped image. Figure 1.2(c) shows the recovered image.

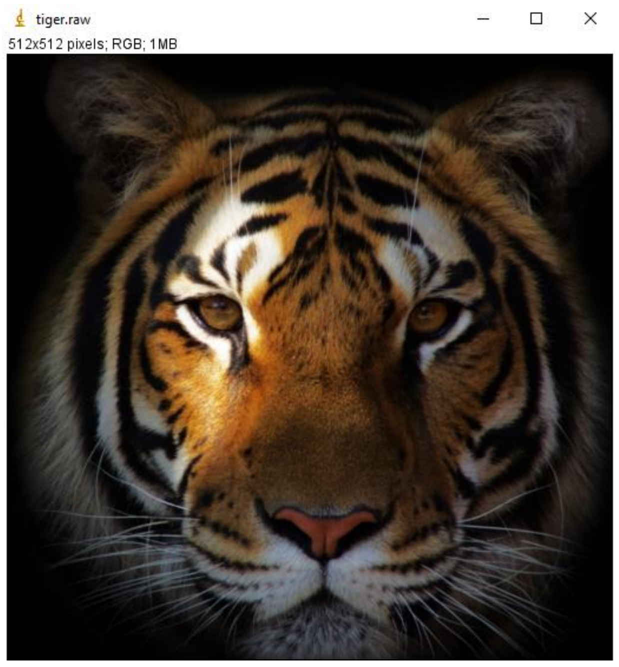
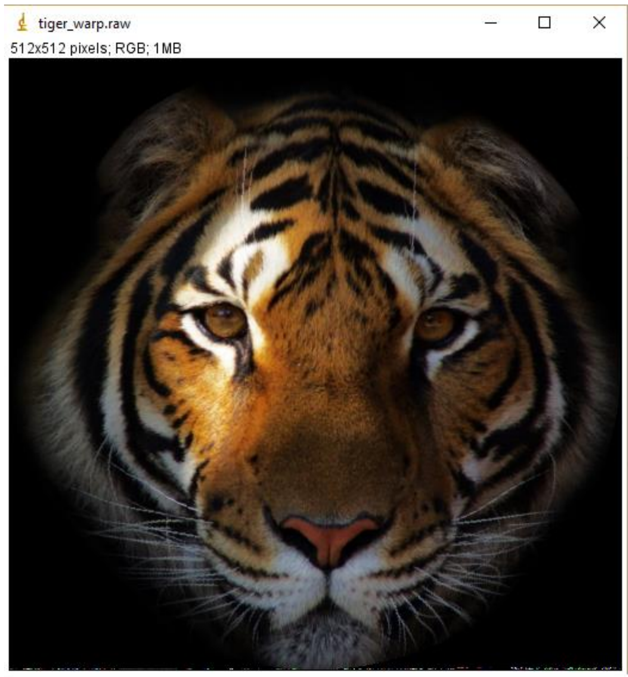


Figure 1.2(a) Input – tiger.raw



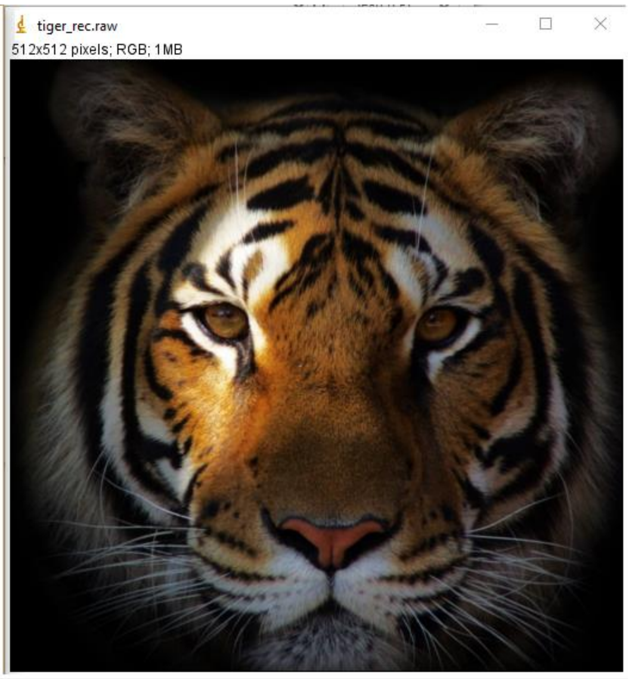


Figure 1.2(b) Warped Image **Figure 1.2(c) Recovered – tiger. raw**

Figure 1.3(a) shows the actual panda.raw image which is given as the input. Figure 1.3(b) shows the Warped image. Figure 1.3(c) shows the recovered image.

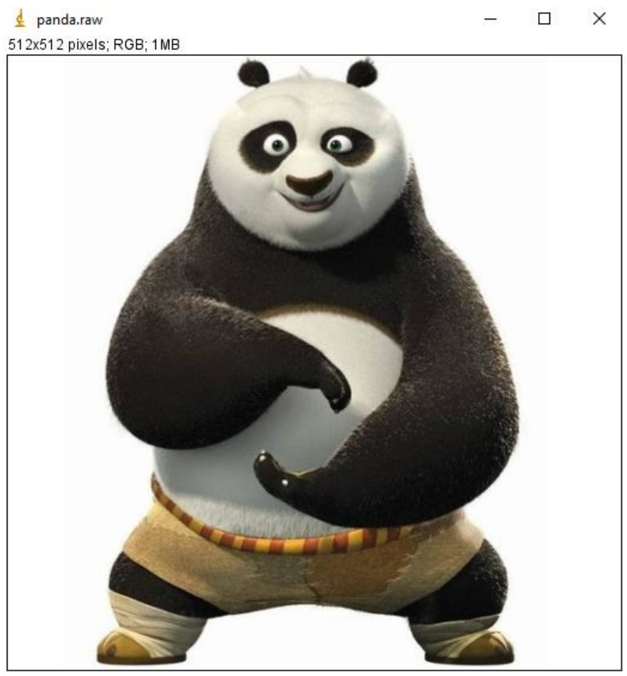
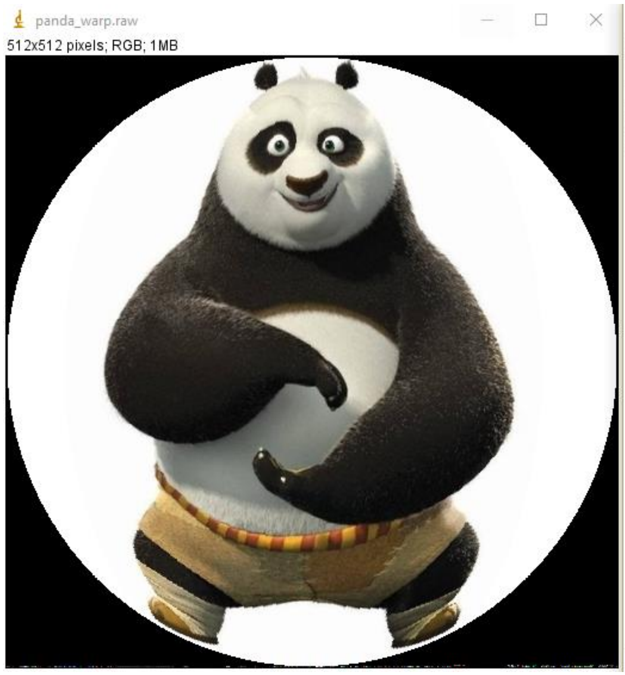


Figure 1.3(a) Input – panda.raw



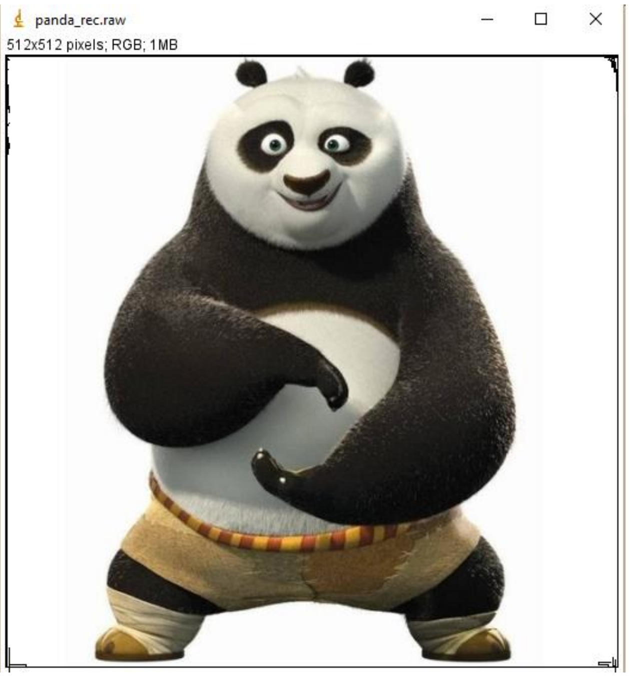


Figure 1.3(b) Warped Image **Figure 1.3(c) Recovered – panda. raw**

A.4 Discussion

Geometrical warping here is implemented using the elliptical method, discussed earlier. The warped images are without any distortions, as seen in the Figure 1.1(b), Figure 1.1(c) and in Figure 1.1(d). This is because the points on the circle are lesser than the points on the square. In getting back the actual image from the warped image, bilinear interpolation is used, since the points on the circle are mapped to points on the square. Even though bilinear interpolation is done, there are some distortions found on the edges of the image. This is because of the interpolation from smaller number of pixels to largest number of pixels. There is also blurring of the details of the image when mapped from circle to square, because of the interpolated pixels.

This could be improved if reverse mapping is used, where in conversion from square to circle will be based on the co-ordinates in the circle image, and vice versa.

Part B – Homographic Transformation and Image Stitching

B.1 Motivation

To get a 360-degree view of any object is difficult. It can be obtained by taking multiple images and stitching them so that the degree of viewing is high. This is obtained by homographic transformation and image stitching. Homographic transformation refers to the transformation of (x,y) co – ordinates in one plane to another plane. A projective plane is a plane that is used here that considers all possible straight lines from the plane to the space. If any four points in a plane are connected in a plane, there always exists a relationship that transforms the, into corresponding four points in the other plane. The projection could be planar or cylindrical. Image stitching relies on feature extraction from the choice of control points.

B.2 Approach

Homographic transformation is done as follows. Images taken at two different angles from a pinhole are related as

$$\mathrm{X_{new}} = \mathrm{H\,X_{old}}$$

Where H is the homographic transformation matrix, $\mathrm{X_{old}}$ is the co – ordinates of the old image (to be warped image) and $\mathrm{X_{new}}$ is the co – ordinates of the new image (fixed image). The homogeneous image co – ordinates are given as $\begin{bmatrix} \mathrm{x1} \\ \mathrm{y1} \\ 1 \end{bmatrix}$

This can be expanded as

$$\begin{bmatrix} \mathrm{x2'} \\ \mathrm{y2'} \\ \mathrm{w2'} \end{bmatrix} = \begin{bmatrix} \mathrm{H0} & \mathrm{H1} & \mathrm{H2} \\ \mathrm{H3} & \mathrm{H4} & \mathrm{H5} \\ \mathrm{H6} & \mathrm{H7} & \mathrm{H8} \end{bmatrix} \begin{bmatrix} \mathrm{x1} \\ \mathrm{y1} \\ 1 \end{bmatrix} \text{ and } \begin{bmatrix} \mathrm{x2} \\ \mathrm{y2} \end{bmatrix} = \begin{bmatrix} \mathrm{x2'/w2'} \\ \mathrm{y2'/w2'} \end{bmatrix}$$

Where $\mathrm{x2', y2', w2'}$ correspond to the new image and $\mathrm{x1, y1}$ correspond to the old image.

To find the values of $H0, H1, \ldots, H8$, usually $H8$ is set to 1, and the rest are found using the formula below. Let the control points in the old image be $(x_i, y_i), i = 1,2,3,4$. Let the control points in the new image be $(X_i, Y_i), i = 1,2,3,4$. The parameters are mapped as $a = H0, b = H1, c = H2, d = H3, e = H4, f = H5, g = H6, h = H7$. They are found using

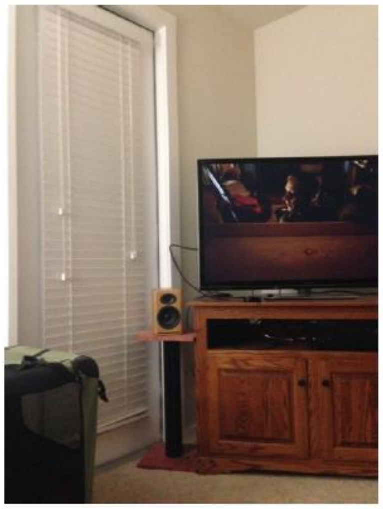
$$\begin{vmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 &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

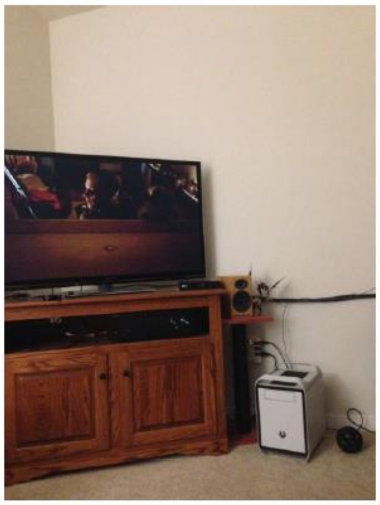
This method is implemented in C++ as follows:

i. The input images are read in separate files.
ii. The output image, which has a bigger dimension (approximately greater than thrice the width of the original image, and approximately twice the height of the actual image) is also defined.
iii. Since there are only three images on which image stitching must be done, the middle image is copied in the bigger image. Then there are two mappings done – first left image is stitched onto the middle image, and next, right image is stitched onto the middle image
iv. For every stitching, Using the control points, the matrix H^{-1} is computed.
v. Using the inverse mapping technique, every pixel in the output image is mapped onto the input image using the formula. Bilinear Interpolation is used to map into the exact pixel.

B.3 Results

Figure 1.4 shows the left, middle and right images used as the input for image stitching.





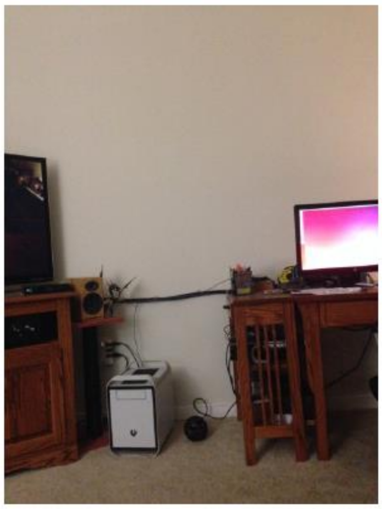


Figure1.4(a) Left.raw **Figure1.4(b)Middle.raw** **Figure 1.4(c) Right.raw**

Figure 1.4 Input images used for Image stitching

Figure 1.5 shows the output image obtained after Image stitching.

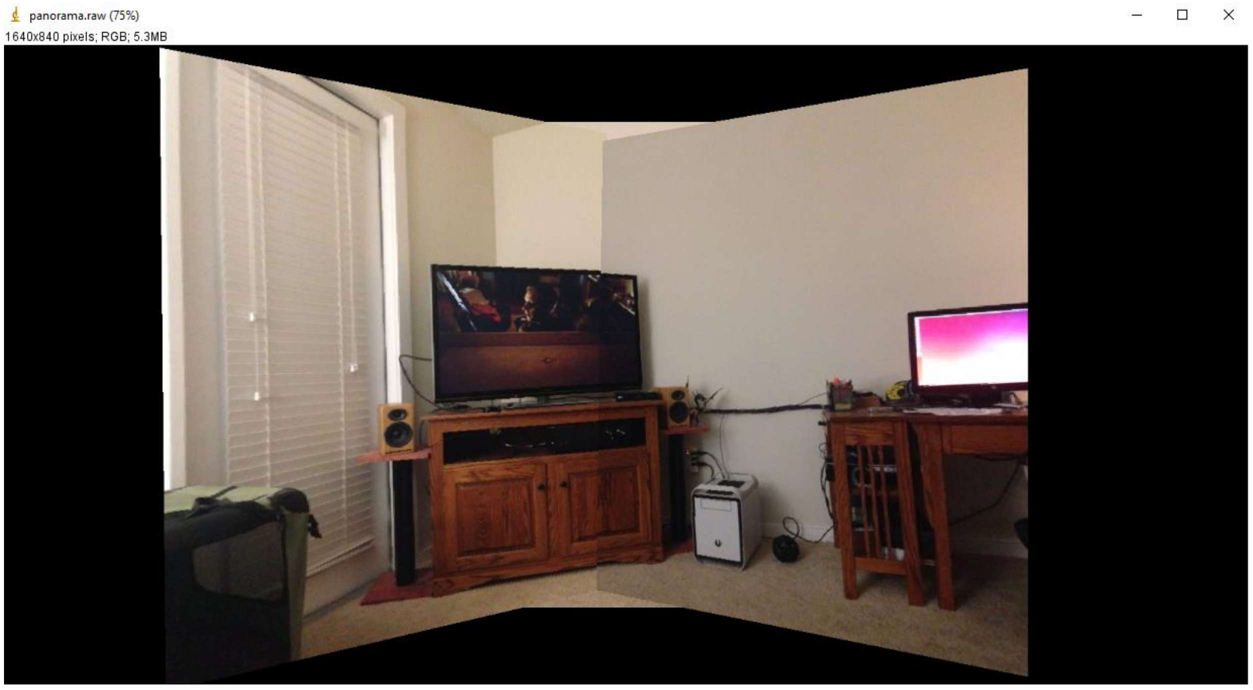


Figure 1.5 Image stitching using Homographic Transformations

The control points used for image stitching are shown in Figure 1.6. Figure 1.6(a) shows the control points used for stitching the left and the right images. Figure 1.6(b) shows the control points used for the right and middle images.

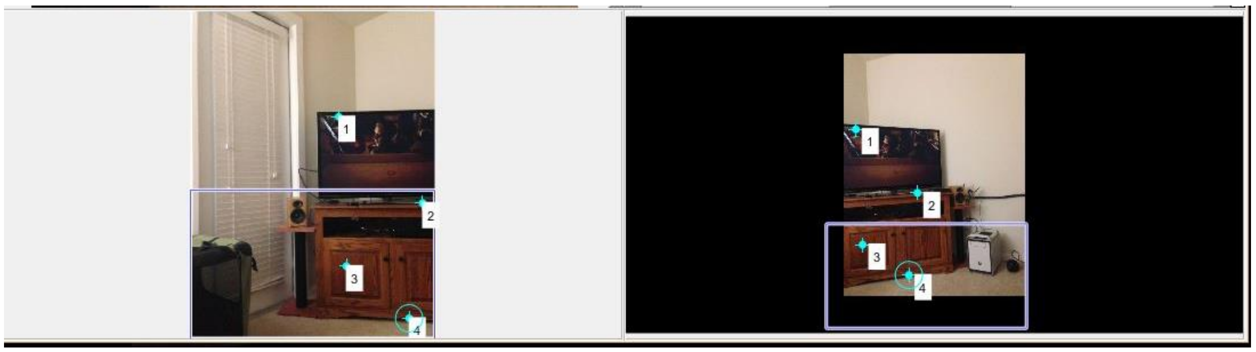


Figure 1.6(a) Control Points for the Left and Middle Images

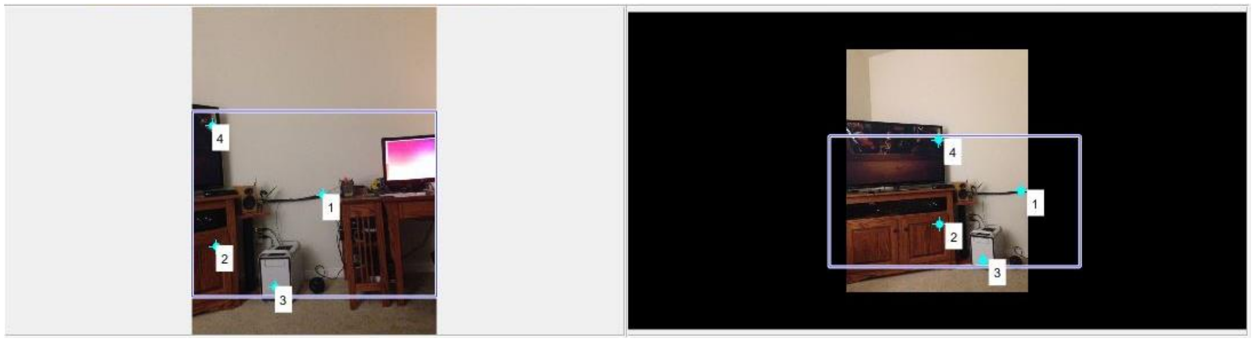


Figure 1.6(b) Control Points for the Right and Middle Images

B.4 Discussion

Here four control points are used for image stitching. The control points are carefully considered such that they span the entire overlap of the two images. The following are some considerations of control points.

- Control Points are chosen based on the distinctive features. The features that are distinct and overlapping are considered.
- Control Points are chosen such that they do not lie on the same row or same column, since this effects the stitching because of planar projection.

- The overlapping pixels are chosen accurately, with almost no approximations, so that the features are extracted in precision.
- If the control points are close to each other, only that area is concentrated during stitching of the image. This explains the robustness of the decision of feature points (or control points) during geometrical transformations.
- The control points are chosen as a big quadrilateral, with maximum area as possible, and the endpoints not along the same row or column, so that the image is properly stitched.

One such example of control points is shown in Figure 1.6.

If there are more control points chosen, the quadrilateral is with more control points. The H – matrix will still have only 9 values, the computation of H values becomes complex. The choice of more control points results in better stitching. Some other methods to improve the quality of image stitching include using modified Hough transform, seed points etc. [2].

References:

[1] William Pratt, “Digital Image Processing”

[2] Margarita N. Favorskaya, Lakhmi C. Jain,” Computer Vision in Control Systems-2: Innovations in Practice”

[3] Chamberlain Fong, “Analytical Methods for Squaring the Disc”, ICM 2014

Problem 2: Digital Half toning

Part A – Dithering

A.1 Motivation

Halftoning is a method of representation of the different gray scale levels using black and white levels. It is extensively used in printing, wherein all gray levels cannot be explicitly used. These black pixels on white paper become almost invisible to the human eye when they are close enough together, because of the low pass spatial frequency properties of the eye. Digital halftoning is a method wherein computer algorithms are used to half – tone an image. Some common methods of half toning are

- Thresholding
- Dithering
- Error – Diffusion etc.

A.2 Approach

The aim of half – toning is to convert the input image into an image that consists of only black and white pixels. The different methods of doing half – toning is discussed below. Let $\mathrm{F(i,j)}$ be a pixel of the input image, and $\mathrm{G(i,j)}$ be a pixel of the output image.

1. **Fixed Thresholding**

Here a threshold value T is fixed. Every output pixel $\mathrm{G(i,j)}$ is obtained by comparing the input $\mathrm{F(i,j)}$ with the T value. Here the input image has 256 gray scale values, from 0 to 255. Hence the threshold value is chosen to be the mean of the values i.e. $\frac{0+255}{2} = 127$. The comparison with the threshold value T = 127 can be given as

$$\mathrm{G(i,j)} = \begin{cases} 255, & \mathrm{F(i,j)} > 127 \\ 0, & \text{otherwise} \end{cases}$$

The approach in C++ is given as:

i. The input image is read into an array Imagedata using file operations.

ii. The output array ImageOutput is created, which is of the same size as that of Imagedata.
iii. Each pixel $G(i,j)$ is found by comparing the corresponding $F(i,j)$ with the threshold value T.
The value is determined accordingly.
Hence, the array ImageOutput contains only the black and white pixels.

2. Random Thresholding

Here the threshold value for half – toning is random. Since there are 256 gray scale vales, from 0 to 25, a random number R is generated in the range (0,255) using the command rand () in the header file math.h. Every output pixel $G(i,j)$ is obtained by comparing the input $F(i,j)$ with R.

$$R = \mathrm{rand}(0{,}255) \rightarrow \text{to generate a random number in the range } (0{,}255)$$

$$G(i,j) = \begin{cases} 255, & R \le F(i,j) \\ 0, & \text{otherwise} \end{cases}$$

The approach in C++ is given as:

i. The input image is read into an array Imagedata using file operations.
ii. The output array ImageOutput is created, which is of the same size as that of Imagedata.
iii. A random number is generated for each pixel using the command $R = \mathrm{rand}()\ \%\ 256$. Each pixel $G(i,j)$ is found by comparing the corresponding $F(i,j)$ with the threshold value R.
The value is determined accordingly.
Hence, the array ImageOutput contains only the black and white pixels.

3. Using Dithering Matrices

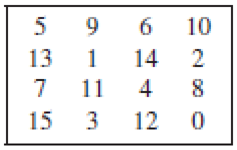
Dithering can be described as a noise that is added intentionally to an image, that will help in reducing the randomly occurring quantization errors. Dithering can be considered a form of quantization. There are many kinds of dithering – we do ordered dither here. Each pixel in a constant gray level window is converted to 1 (or Black) in some order specified by the dithering index matrix. For example, the dithering index matrix I_2 is given by

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

Where the pixel location in 0 is the most likely to be turned to 1 first, and the pixel location 3 the least. These dithering matrices are also called Bayer matrices. The other Bayer matrices are found recursively using the formula

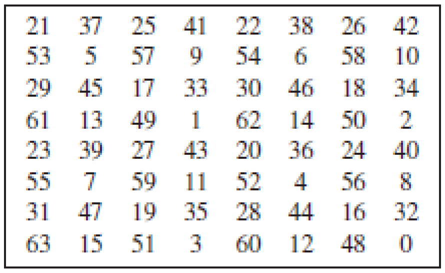
$$I_{2*n} = \begin{bmatrix} 4 * I_n(x,y) + 1 & 4 * I_n(x,y) + 2 \\ 4 * I_n(x,y) + 3 & 4 * I_n(x,y) + 0 \end{bmatrix}$$

Calculating them,



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

4×4



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

8×8

The index matrix is then transformed to a threshold matrix (or a screen matrix) using the following formula.

$$\text{Threshold matrix } T(x,y) = \frac{I(x,y) + 0.5}{N * N}$$

Where N*N denotes the total number of pixels in the image.

This small matrix is repeatedly done over the entire image according to the formula

$$G(i,j) = \begin{cases} 255, & F(i,j) > T(i \bmod N, j \bmod N) \\ 0, & \text{otherwise} \end{cases}$$

The approach in C++ is given as:

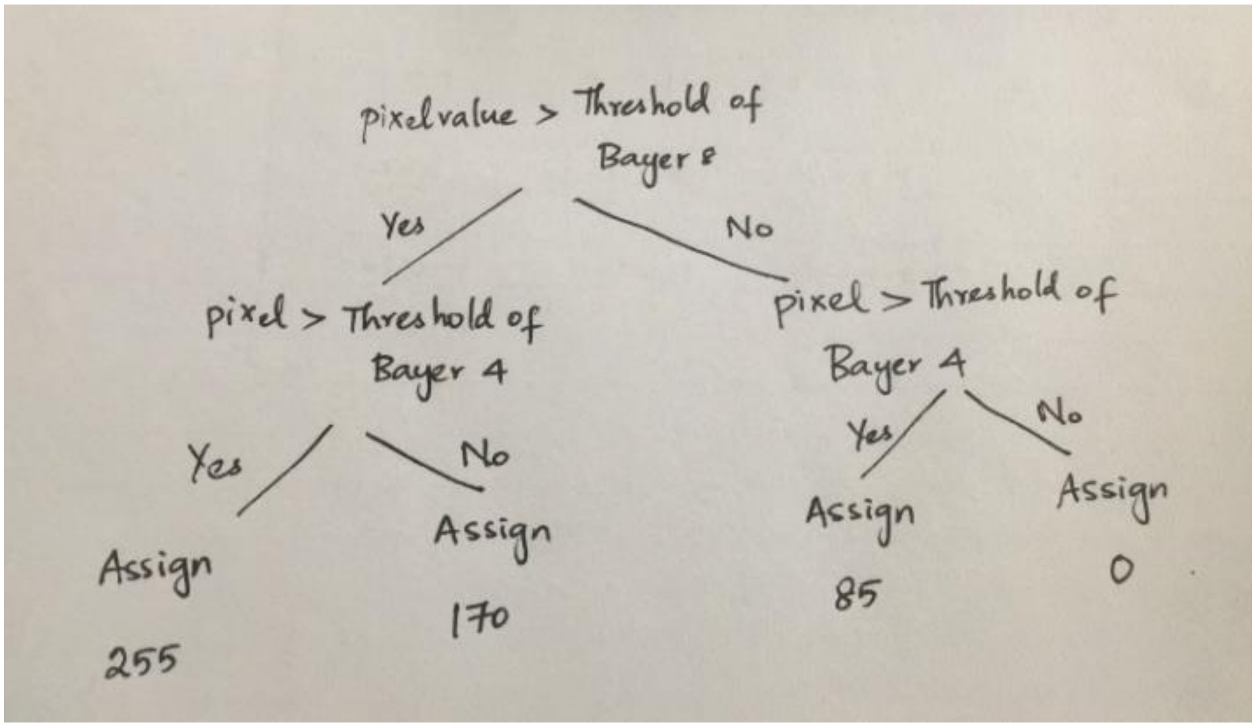
i. The input image is read into an array Imagedata using file operations.

ii. The output array ImageOutput is created, which is of the same size as that of Imagedata.
iii. Bayer Matrix of Index $2 * 2$ is first created. Then Bayer Index Matrices of size $4 * 4$ and $8 * 8$ are created recursively using the formula.
iv. For each Bayer Matrix, the threshold matrix is computed, and the corresponding output is obtained.
    Hence, the array ImageOutput contains only the black and white pixels.

If the screen can display only four intensity levels, the following four levels are chosen – 0, 85, 170 and 255. Initially the image is checked using Bayer8 matrix, and then using Bayer4 matric. A simple explanation is as follows.

Assume thres4 and thres8 are the threshold matrices of Bayer Matrix 4 and Bayer Matrix 8 respectively. Assume pixel is each pixel of the image.

i. Calculate the Bayer Matrices B4, of size 4*4 and Bayer Matrix B8, of size 8*8.
ii. Calculate the corresponding threshold matrices thres4 and thres8 respectively.
iii. For each input pixel, do the following.


pixelvalue > Threshold of Bayer 8
Yes
No
pixel > Threshold of Bayer 4
pixel > Threshold of Bayer 4
Yes
No
Yes
No
Assign 255
Assign 170
Assign 85
Assign 0

iv. Assign the corresponding gray scale level.

The following algorithm is implemented to display four intensity levels.

The approach in C++ is given as:

i. Get the Input image in the array named InputData using file operations.
ii. Get the Bayer Matrices of size 4*4 and 4*8.
iii. Calculate the corresponding threshold matrices.
iv. Assign the new value to each pixel in the output image ImageOutput (either of 0 , 85, 170 or 255) to the pixel value from Imgedata based on the above algorithm.

Thus, the ImageOutput array consists of only 4 values.

A.3 Results

The actual input image for Half toning is the colorchecker.raw image, as shown in Figure 2.1.

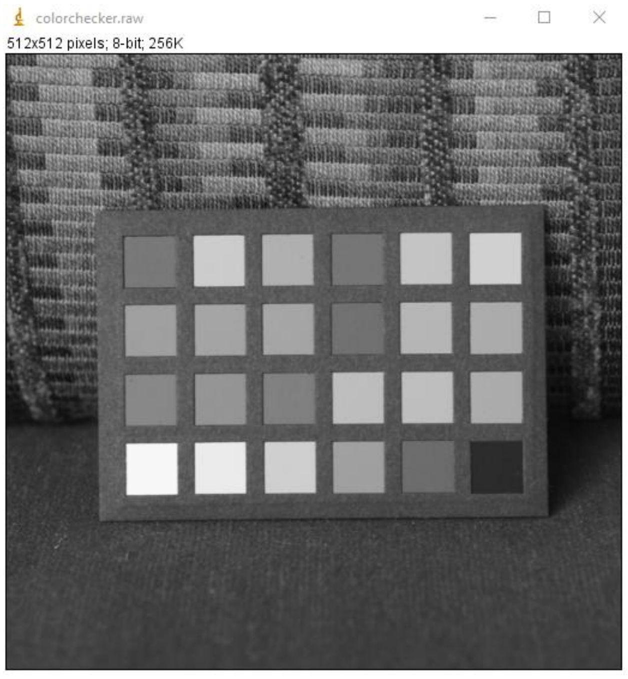


Figure 2.1 Original colorchecker.raw image

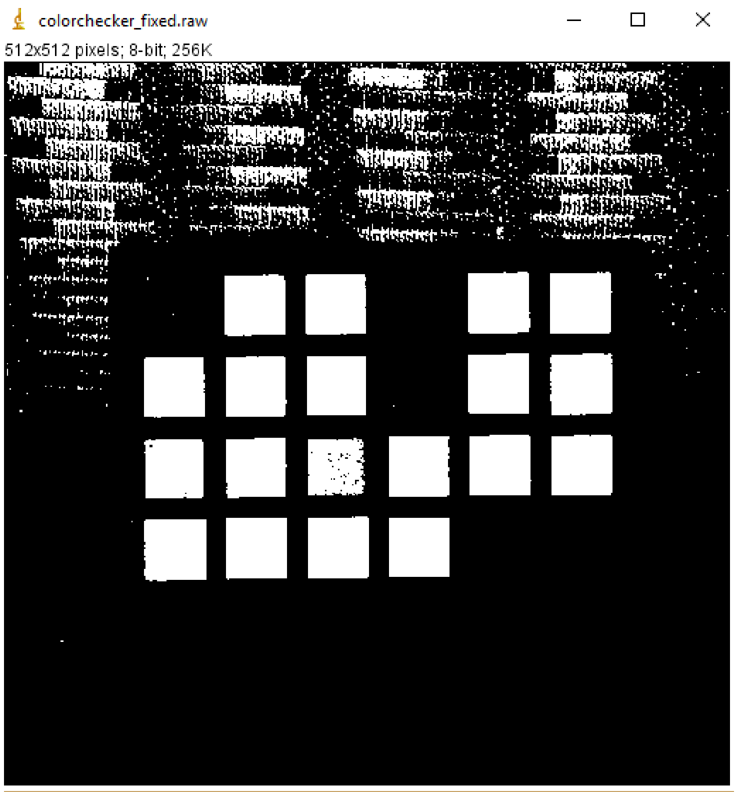


Figure 2.2 Output image of Fixed Thresholding

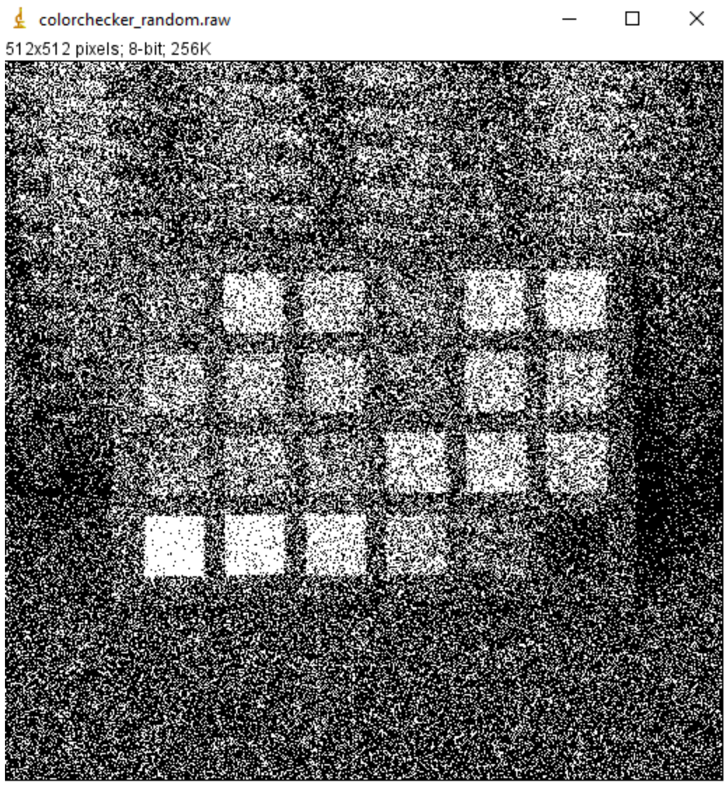


Figure 2.3 Output Image of Random Thresholding

The threshold value is chosen as T=127 and fixed thresholding is done. The output obtained is shown in Figure 2.2. In random thresholding, the threshold value is random. The output image is obtained as in Figure 2.3.

Dithering is an improved version of thresholding. Figure 2.4 shows the output images obtained after dithering. Figure 2.4(a) shows the output for dithering using Bayer Matrix I_2. Figure 2.4(b) shows the output for dithering using Bayer Matrix I_4. Figure 2.4(c) shows the output for dithering using Bayer Matrix I_8.

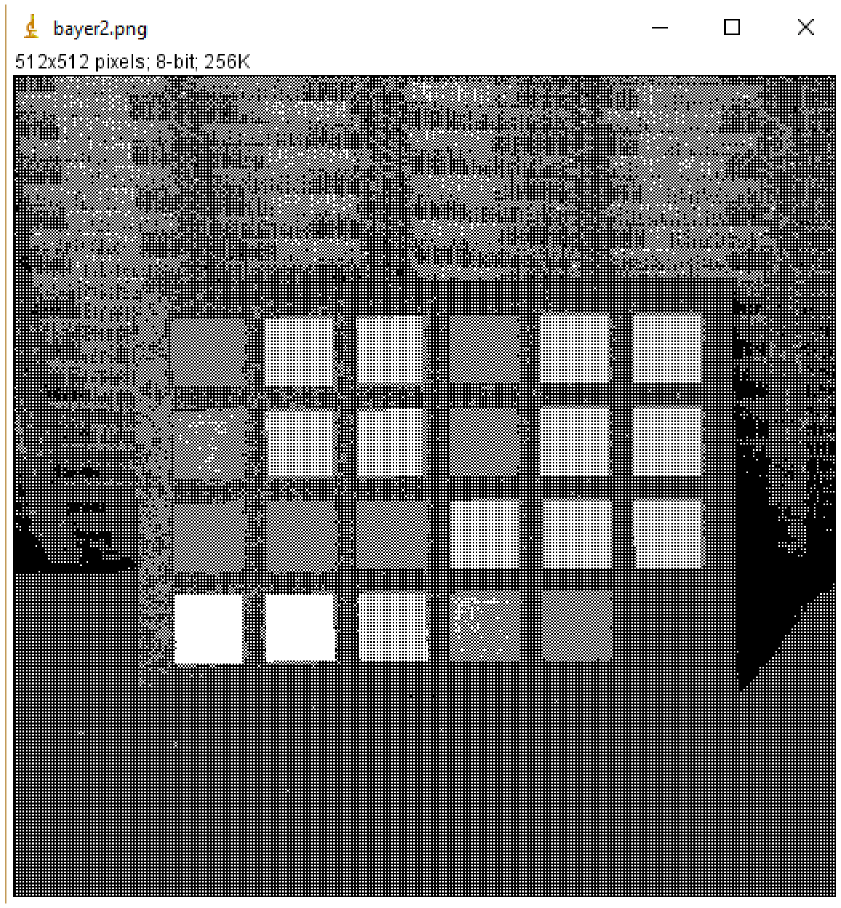


Figure 2.4(a) Using Bayer Matrix I_2

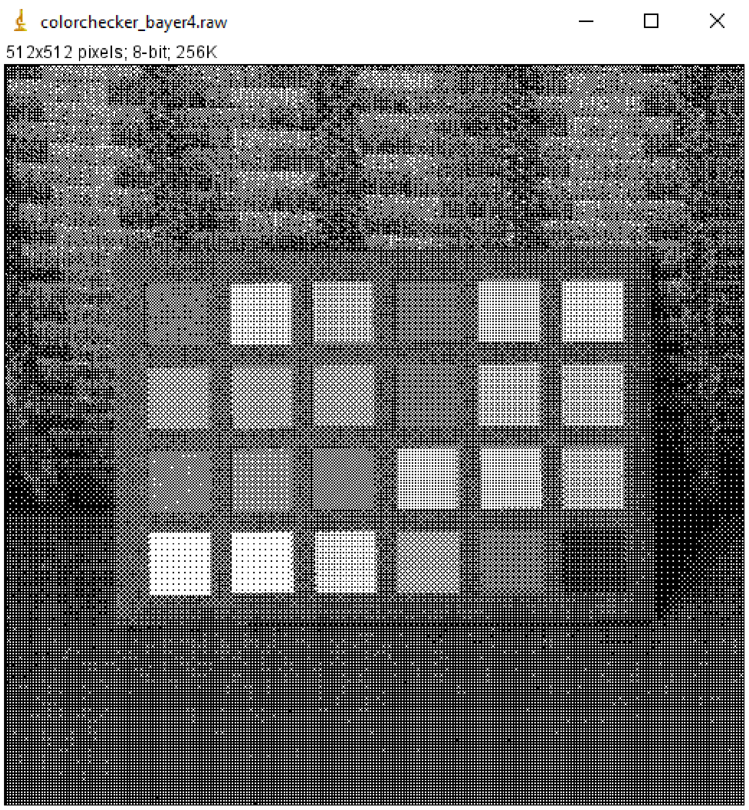


Figure 2.4(b) Using Bayer Matrix $\mathbf{I}_4$

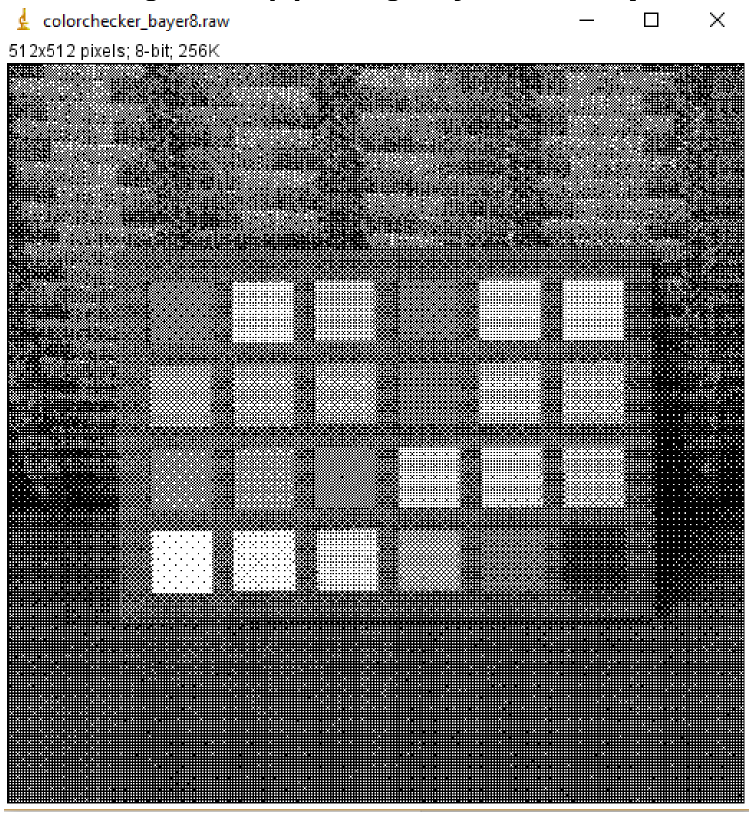


Figure 2.4(c) Using Bayer Matrix $\mathbf{I}_8$

If the screen can display only four intensity levels, the output image is obtained as in Figure 2.5.

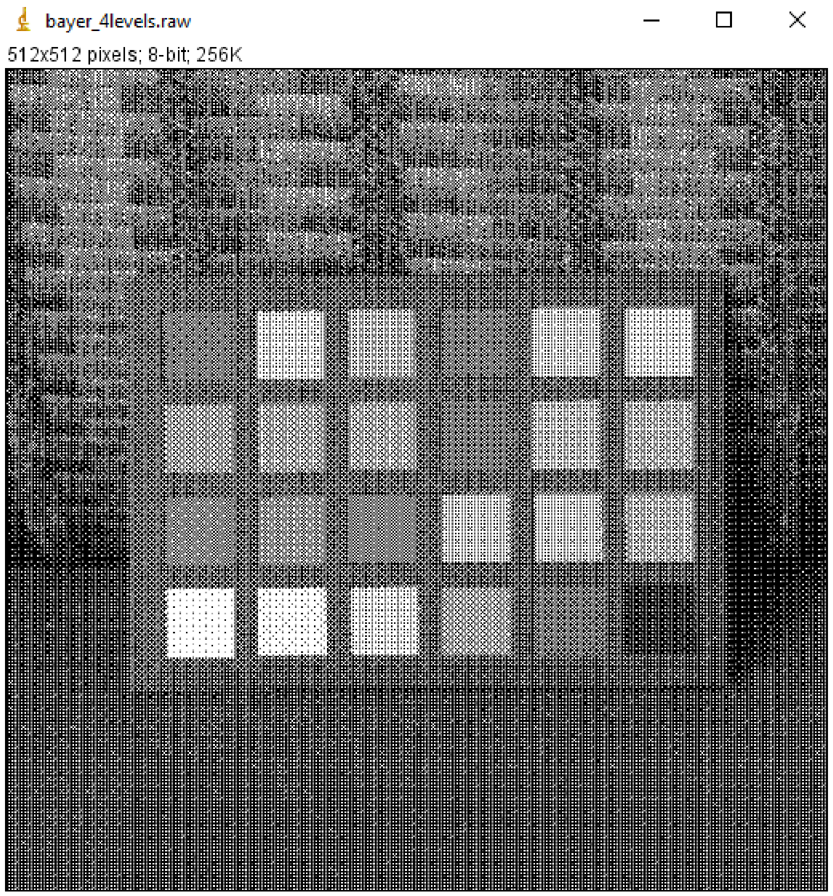
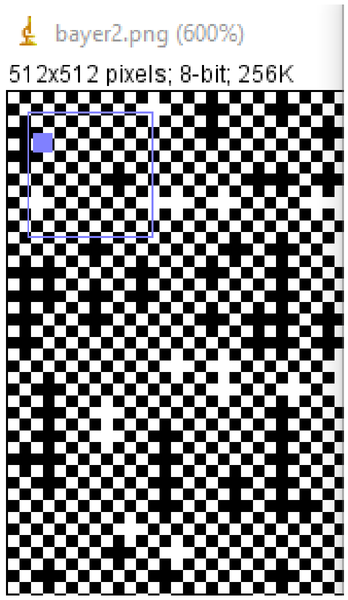
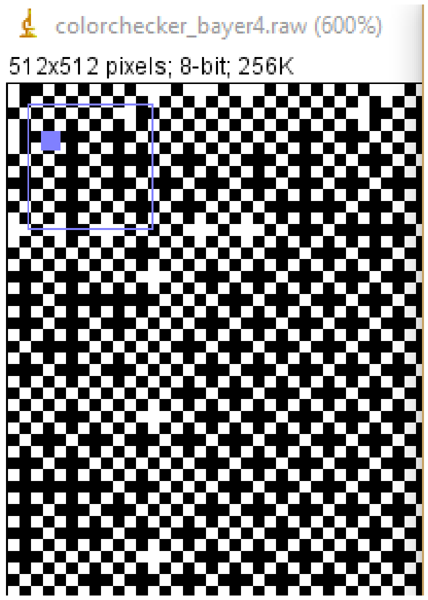


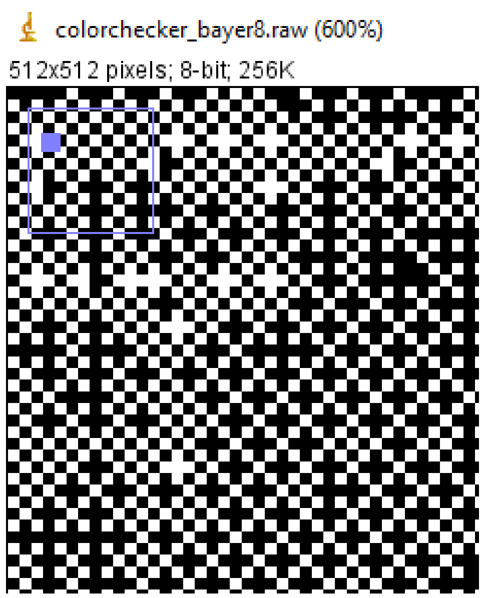
Figure 2.5 Output image with four intensity levels

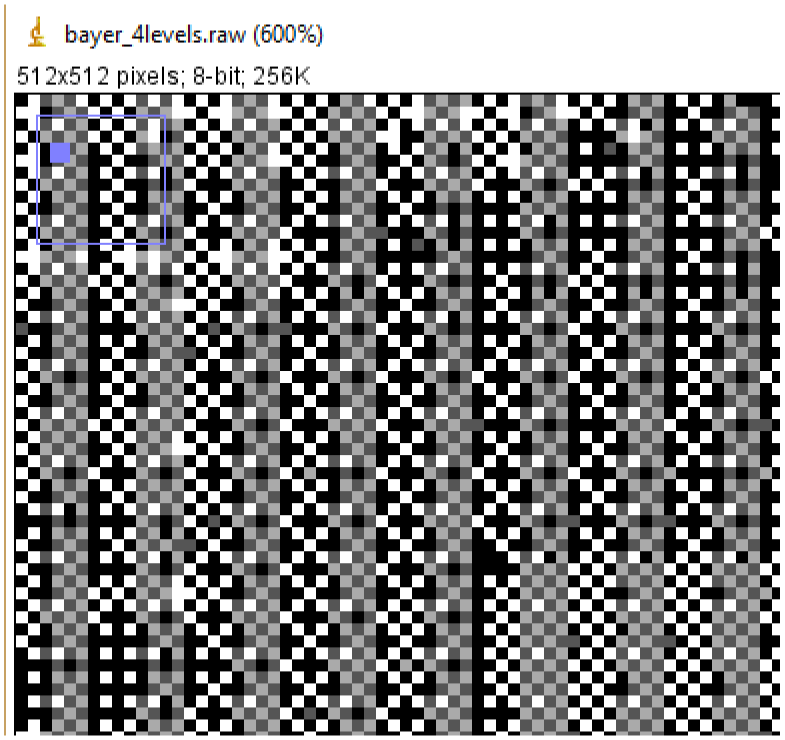
A.4 Discussion

The four dithering matrices are obtained using recursion, and the corresponding output images are displayed. The visual quality of the images is checked. The images obtained using thresholding give poor results compared to the Bayer matrices. This is checked by examining certain portions of the image by zooming in, as follows.









The following image compares the results obtained using all the above discussed half toning algorithms. It is observed that if there are more gray scale values, the image is better in terms of visual quality. The presence of only two levels gives a comparatively abstract image. The pixel density is also found to increase, thus giving a proper definition to the image. Thus, the image with four intensity levels is better. This can be compared by using some performance evaluation metrics like PSNR, MSE etc.

Part B – Error Diffusion

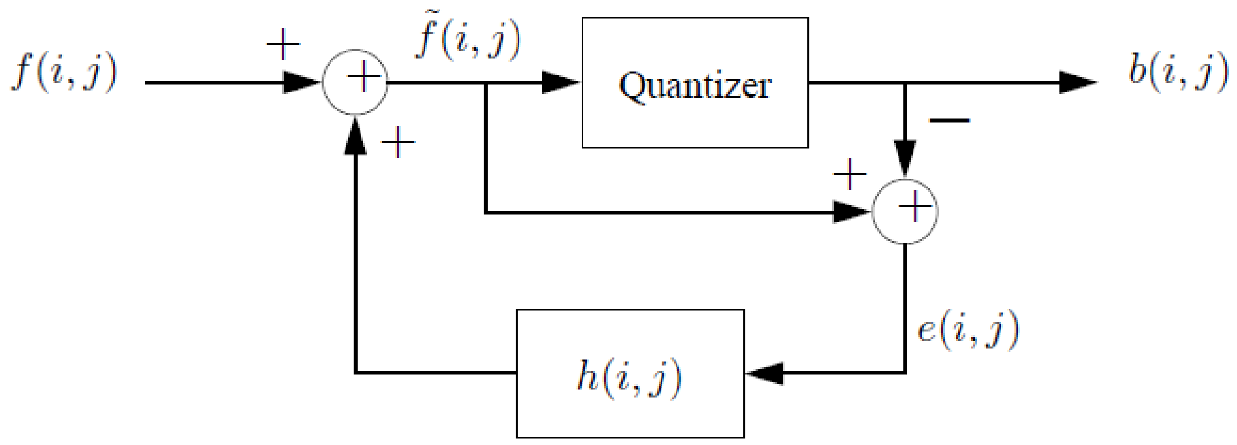
B.1 Motivation

Halftoning is a method of representation of the different gray scale levels using black and white levels. Digital halftoning is a method wherein computer algorithms are used to half – tone an image. Some common methods of half toning discussed in Part (A) are Thresholding and Dithering. These are point – based operations, where in each output pixel is computed as a function of the input pixel, using some masks and threshold values. Analyzing the output images, it is found that the output image is not very desirable, since there are many discontinuities in the image. Error diffusion is a neighborhood based operation, wherein each output pixel is computed as a function of the pixels in the N*N neighborhood.

B.2 Approach

In Error Diffusion, each pixel is quantized by the neighbors in the N*N neighborhood. Here each pixel is quantized, and the quantization error is diffused into the next pixel. This process of error diffusion can be visualized as shown below, where $f(i, j)$ is the input pixel, and $b(i, j)$ is the output pixel.

0where wh



Where

$$b(i,j) = \begin{cases} 255 & \text{if } \tilde{f}(i,j) > T \\ 0 & \text{otherwise} \end{cases}$$

$$e(i,j) = \tilde{f}(i,j) - b(i,j)$$

$$\tilde{f}(i,j) = f(i,j) + \sum_{k,l \in S} h(k,l)e(i-k,j-l)$$

The common algorithm to do error – diffusion is as follows.

i. Initialize $\tilde{f}(i,j) = f(i,j)$

ii. For each pixel, compute

$$b(i,j) = \begin{cases} \tilde{f}(i,j), & \tilde{f}(i,j) > \text{Threshold} \\ 0, & \text{otherwise} \end{cases}$$

iii. Diffuse the error using the following method. Assuming the error diffusion matrix is

	X	h1
h2	h3	h4

Error $e = \tilde{f}(i,j) - b(i,j)$.

$$\tilde{f}(i,j+1) = \tilde{f}(i,j+1) + h1 * e$$
$$\tilde{f}(i+1,j-1) = \tilde{f}(i+1,j-1) + h2 * e$$
$$\tilde{f}(i+1,j) = \tilde{f}(i+1,j) + h3 * e$$
$$\tilde{f}(i+1,j+1) = \tilde{f}(i+1,j+1) + h4 * e$$

iv. This is done for all pixels of the image, and the output is given by $b(i,j)$.

Some of the commonly used error diffusion matrices are given in Table 2.1.

Table 2.1 Commonly used Error Diffusion Matrices

Floyd _ Steinberg Matrix	$\frac{1}{16}\begin{bmatrix}0 & 0 & 0\\0 & 0 & 7\\3 & 5 & 1\end{bmatrix}$
Jarvis – Judice and Ninke (JJN)	$\frac{1}{48}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 7 & 5\\3 & 5 & 7 & 5 & 3\\1 & 3 & 5 & 3 & 1\end{bmatrix}$
Stucki	$\frac{1}{42}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 8 & 4\\2 & 4 & 8 & 4 & 2\\1 & 2 & 4 & 2 & 1\end{bmatrix}$

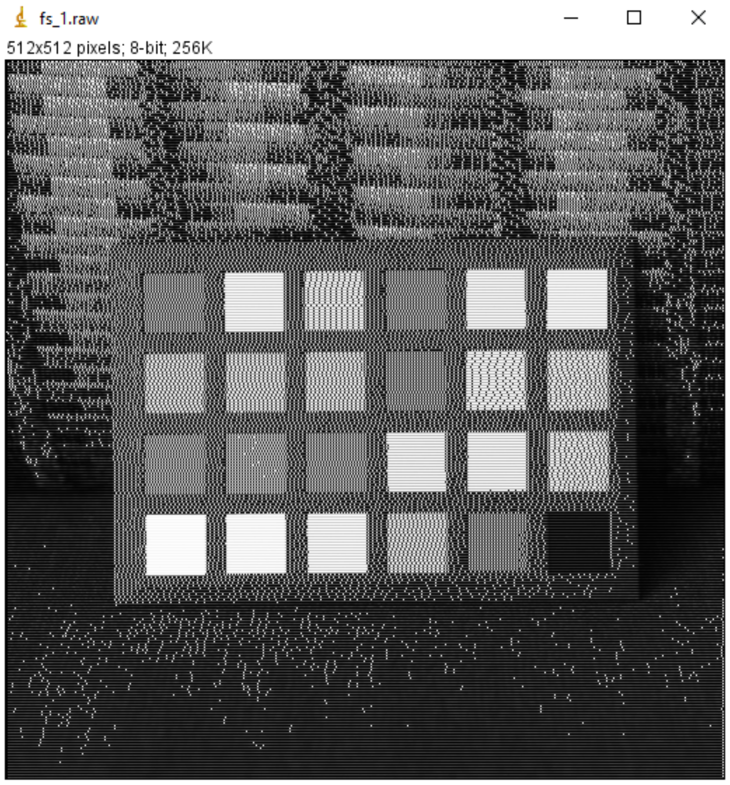
The approach in C++ is as follows:

i. The input image is read into an array Imagedata using file operations.
ii. The output array ImageOutput is created, which is of the same size as that of Imagedata.
iii. Using each of the error diffusion matrices, the output image is obtained according to the approach described above.
Hence error diffusion is implemented.

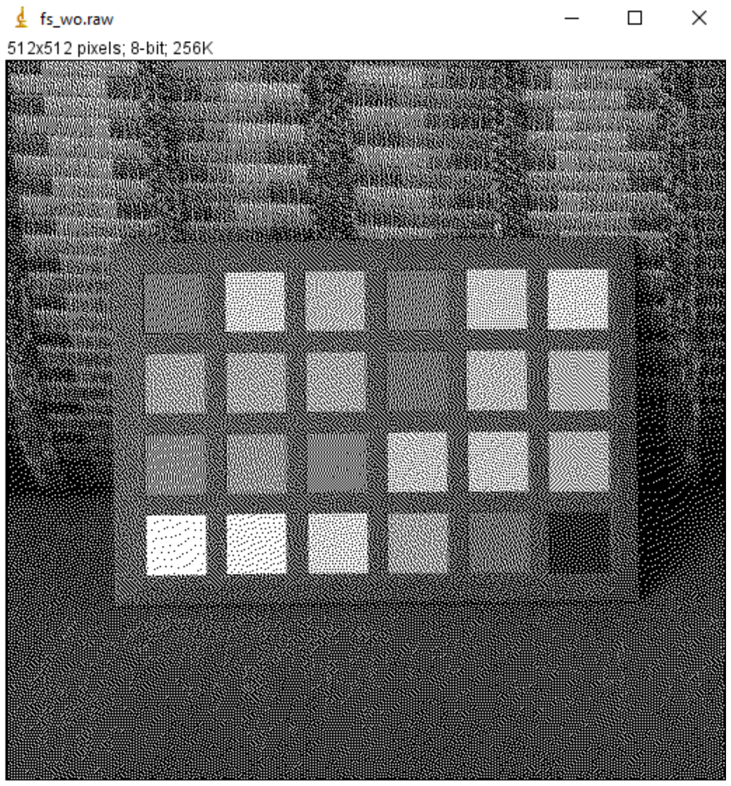
B.3 Results

Figure 2.6 gives the images obtained by using the different error diffusion matrices.

Figure 2.6(a) gives the output obtained when Floyd – Steinberg matrix is used. Figure 2.6(b) gives the output when JJN error diffusion matrix is used. Figure 2.6(c) gives the output image when Stucki matrix is used.



Serpentine Scanning



Without Serpentine Scanning

Figure 2.6(a) Using Floyd – Steinberg matrix

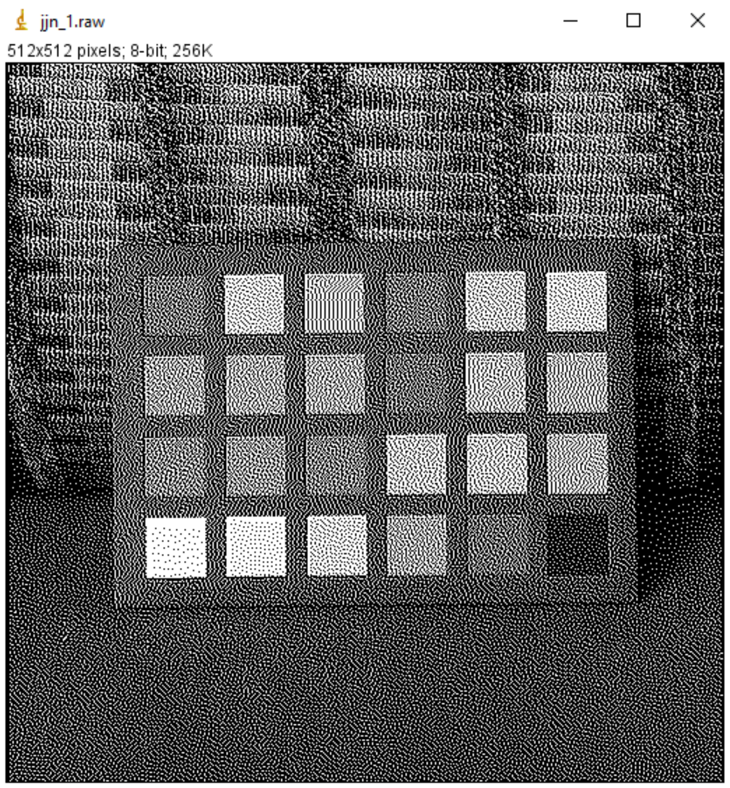


Figure 2.6(b) Using JJN Matrix

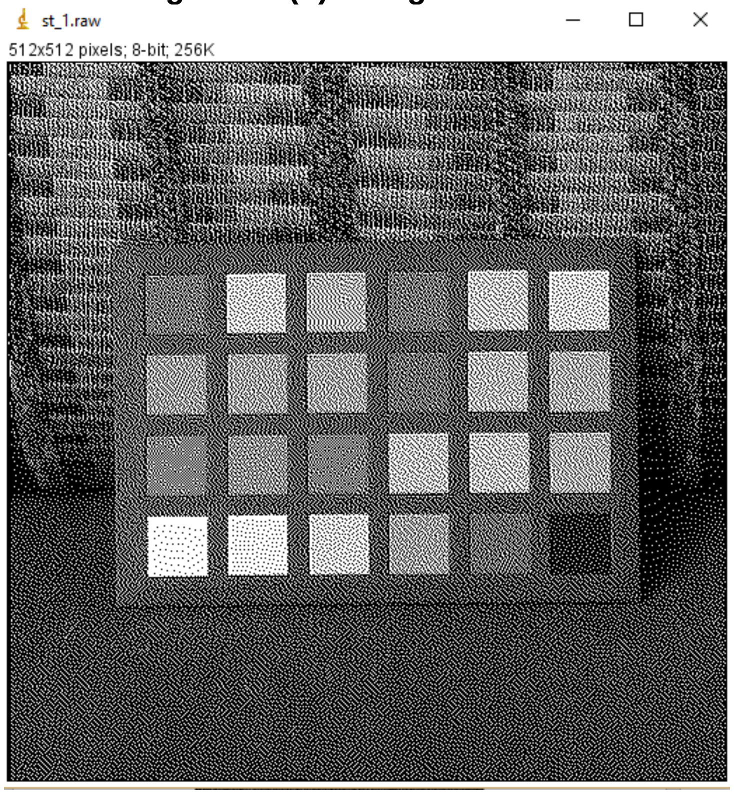


Figure 2.6(c) Using Stucki Matrix

Figure 2.6 Output of error diffusion method of half – toning

The following Figure 2.7 gives the zoomed version of the error -diffusion methods.

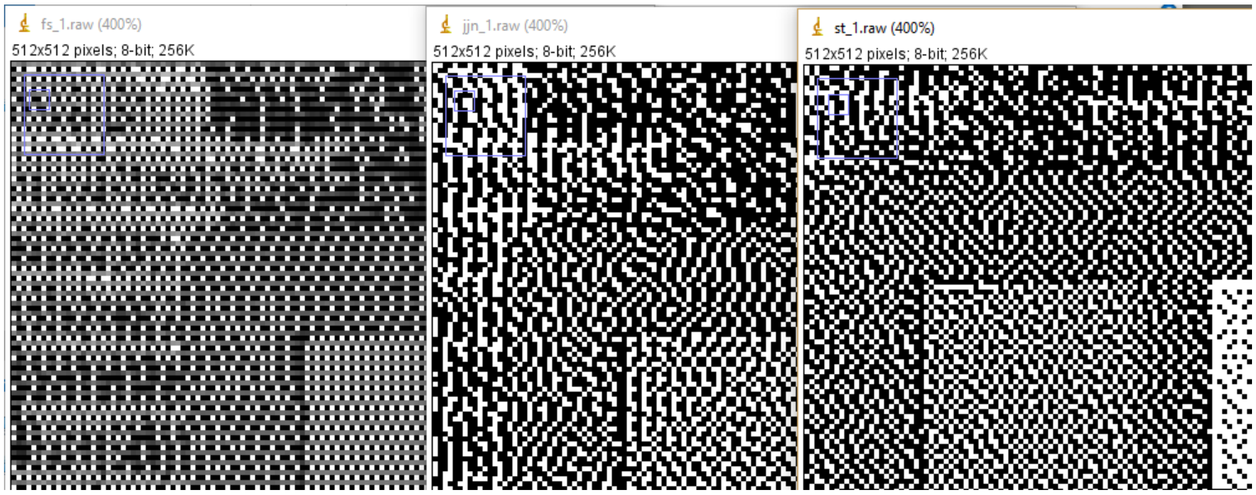

fs_1.raw (400%)
512x512 pixels; 8-bit; 256K
jjn_1.raw (400%)
512x512 pixels; 8-bit; 256K
st_1.raw (400%)
512x512 pixels; 8-bit; 256K

Figure 2.7 Zoomed in images of different error diffusion methods

B.4 Discussion

The images obtained using the error diffusion matrices are shown in Figure 2.6. The main disadvantage of gray scale half toning is that there are artifacts, which cannot be avoided. Comparing the results of dithering and half – toning, dithering is found to increase the sharpness of the image. In error diffusion, the quantization error is distributed across the neighboring pixels using the corresponding error diffusion matrices. Error diffusion methods are found to produce better results. Serpentine scanning is found to give netter results compared to one without serpentine scanning. Floyd – Steinberg matrices diffuses error to the neighboring pixels only. Hence this results in fine grains, that increases sharpness. The error diffusion using JJN Matrix is slower, since there are 12 pixels involved. Stucki also distributes error to 12 pixels, but the output is clear. Hence Stucki matrix is preferred. [2]

To further improve the quality of the halftoning methods, the following must be considered:

- Several ways of calculating and distributing errors i.e. change in co – efficients of the error diffusion
- Dynamically changing the threshold during each iteration such that the adaptive threshold reduces the quantization errors.

The designing of the new error diffusion matrix incorporates the following:

- The filter co – efficients sum up to 1.
- Could be adaptive, that decides threshold in run – time
- This matrix tries to reduce the quantization error i.e. error between the input and output pixels in a weighted fashion
- Higher priority to lower frequencies
- Input value is bound by the output value – hence making the matrix more stable.
- Using the more generally used weight matrix distribution

$$\frac{1}{2} \times \begin{array}{|c|c|}\hline \mathbf{x} & 1 \\ \hline 1 & \\ \cline{1-1}\end{array}$$

This new matrix will give better results because of the adaptive threshold, stability and the importance to the lower frequency terms, that contain more information.

Part C – Color Halftoning with Error Diffusion

C.1 Motivation

Of all the digital – halftoning methods, error diffusion is better in terms of performance, since the quantization error is diffused into future pixels. This half – toning is generally used in conversion of a gray – scale image into an image with black and white pixels only. The error diffusion methods available use the matrix form of error diffusion – like Floyd – Steinberg, JJN and Stucki. Various performance measures have been introduced to improve the accuracy of gray – scale half toning. This method of half – toning can be extended to color images too. In binary images, there is only two gray scale values – wherein the brightness of the image cannot be observed. In color half – toning algorithms, which are an extension of the binary half – toning algorithms, this effect can also be observed. Some improved methods reduce this artifact.

C.2 Approach

There are two approaches to color half toning – one is the simple separable color diffusion, and the other is the MBVQ technique.

1. Simple Error Diffusion

In grayscale – halftoning, the given input image is converted to only one channel, that has only black or white pixels at the output. In this simple error diffusion, each color band is considered as a separate channel. Each R, G, B channel is taken as a separate channel, Floyd – Steinberg error diffusion algorithm is applied separately on each channel. The resulting color space will be either of the following eight colors

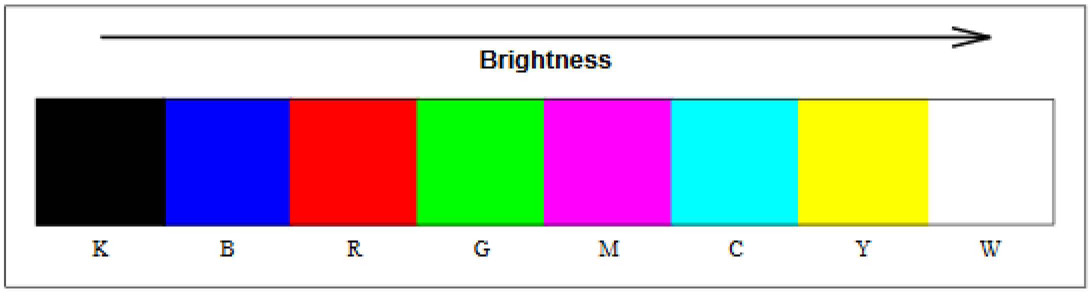
$$W = (0,0,0),\ Y = (0,0,1),\ C = (0,1,0),\ M = (1,0,0),$$
$$G = (0,1,1),\ R = (1,0,1),\ B = (1,1,0),\ K = (1,1,1)$$

The approach in C++ is as follows

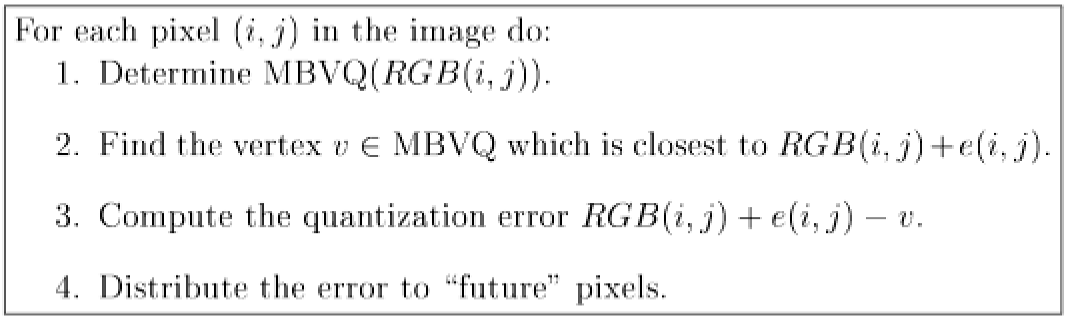
i. The input image is obtained as the input Image array ImageData.
ii. The output array of the same size ImageOutput is defined.
iii. For each channel of the array, i.e. for each color band, Floyd – Steinberg method is applied.
iv. This is applied for all the color bands.
v. The resulting array is now digitally half toned using error diffusion matrices.

2. Minimum Brightness Variation Criterion

Some shortcomings of the error diffusion method include – Presence of excessive fine grain noise in the image, local color is not the desirable color etc. These shortcomings could be reduced by implementing the Minimum Brightness Variation Criterion. The brightness variation considered needs only eight color sets as shown below

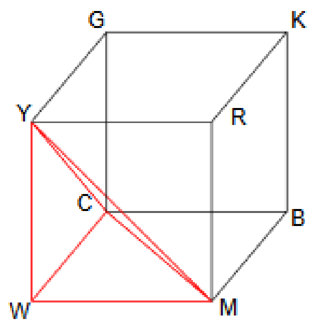


The algorithm can be given as

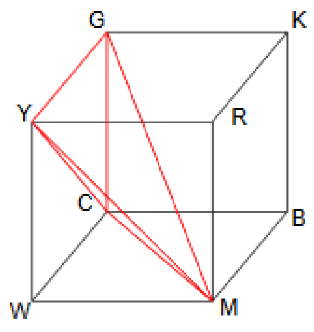


```
For each pixel $(i, j)$ in the image do:
    1. Determine MBVQ$(RGB(i, j))$.
    2. Find the vertex $v \in$ MBVQ which is closest to $RGB(i, j) + e(i, j)$.
    3. Compute the quantization error $RGB(i, j) + e(i, j) - v$.
    4. Distribute the error to “future” pixels.
```

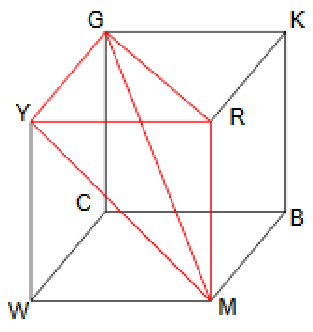
The difference of this algorithm from the separable error diffusion method is in step (2) where in the vertex closest to the pixel is calculated, instead of the eight vertices of the cube. The six MBVQ planes are



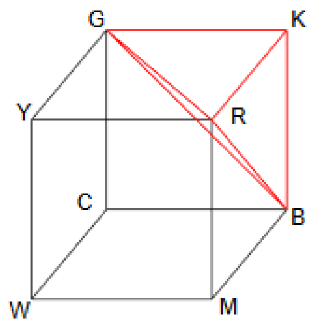
CMYW



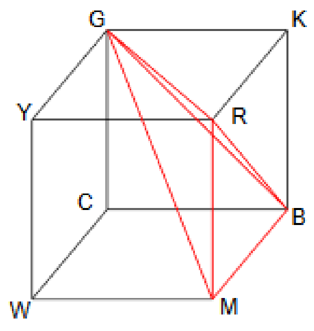
MYGC



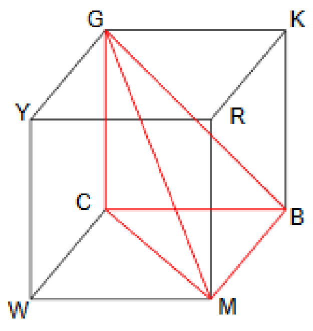
RGMY



KRGB



RGBM



CMGB

The only possible improvement could be the method of error diffusion implemented. The approach in C++ is as follows.

i. The input image is obtained as Imagedata.
ii. The output array is defined to be ImageOutput.
iii. For every point in the pixel, the MBVQ plane is determined.
iv. The closest vertex to that is determined. The error is calculated and the error is distributed to the neighboring pixels.

The output is now color – halftoned.

C.3 Results

The following Figure 2.6(a) shows the actual flower.raw image. Figure 2.6(b) shows the output obtained using the separable error diffusion.

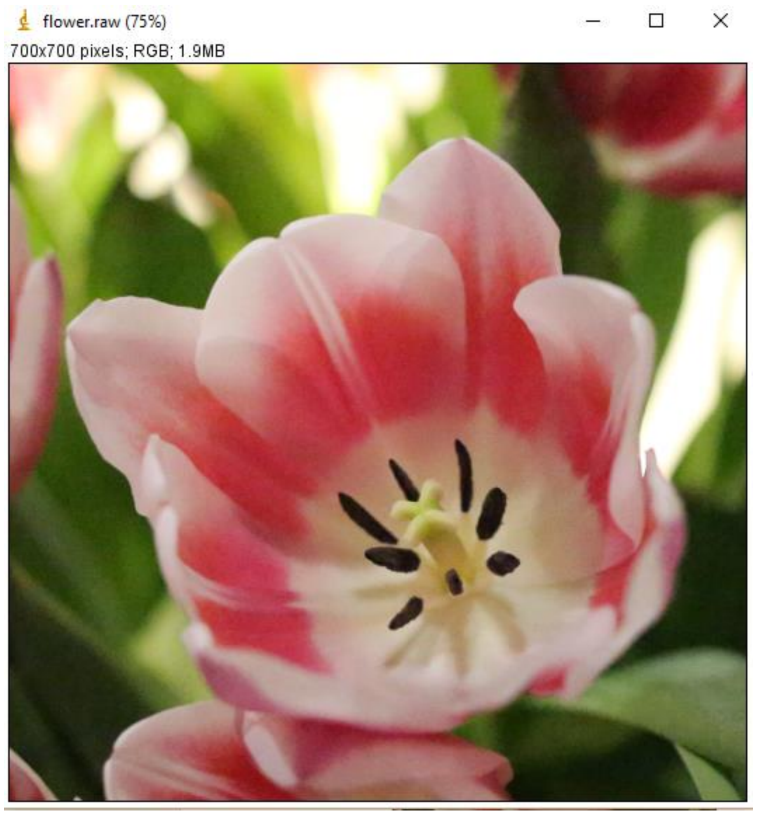


Figure 2.6(a) Actual flower.raw

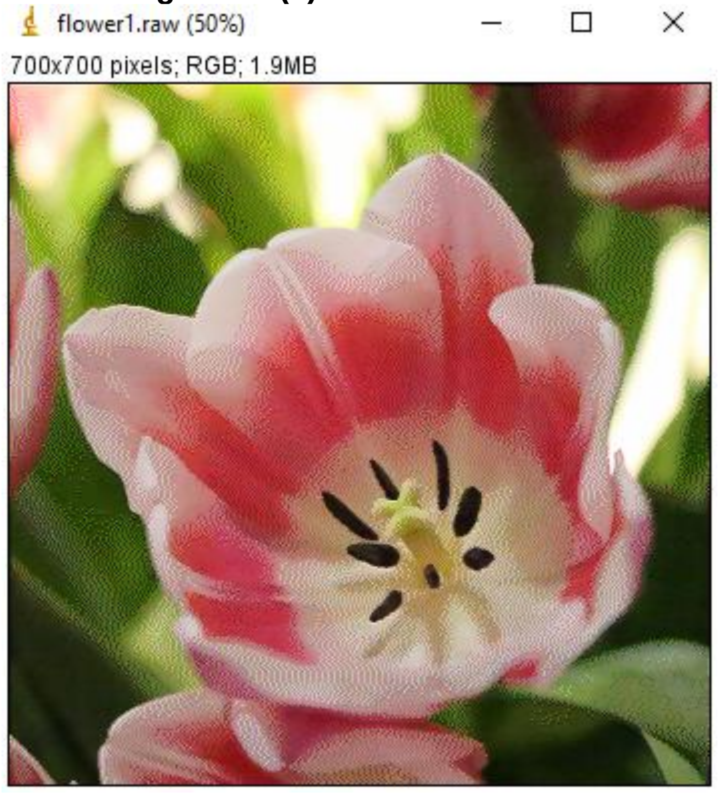


Figure 2.6(b) Output of Separable Error Diffusion

The following Figure 2.6(c) gives the output of the image by MBVQ.

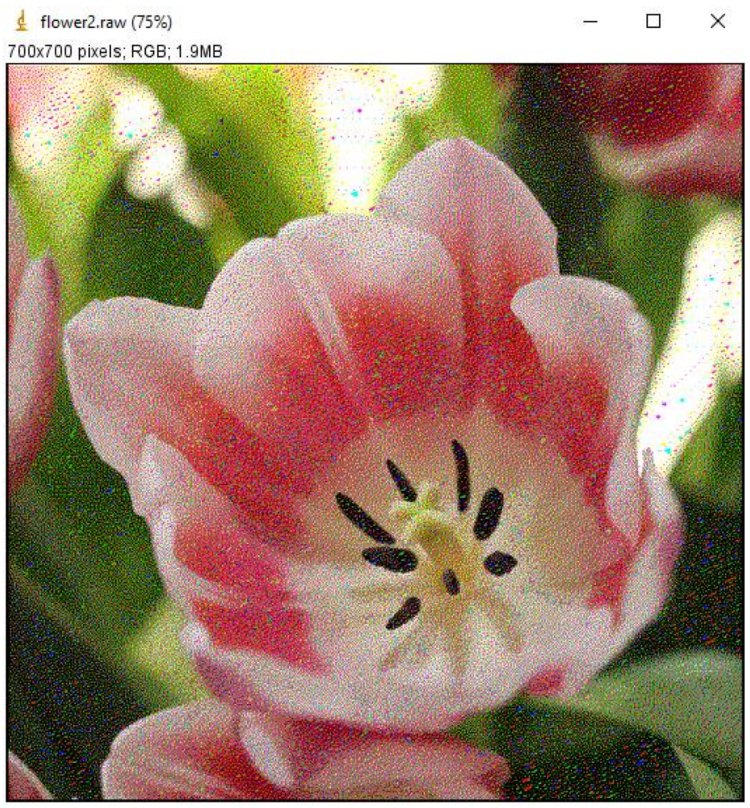


Figure 2.6(c) output obtained by MBVQ

C.4 Discussion

Comparing the outputs of separable color diffusion and MBVQ, the results of MBVQ turn out to be better. This is because in separable color diffusion, the points are decided just based on the threshold values. There are also fine grain artifacts in the image, which reduces the visual quality. This MBVQ is found to have certain advantages

- The pattern pf positioning of dots is visually unnoticeable.
- The local average color is not that different from its neighbors.
- The brightness variation is minimal.

Though the same error diffusion method is used here (Floyd – Steinberg) for both types of color – halftoning, it does not really matter because the decision rule is different. Error diffusion just helps in the distribution of error intensity across the pixels. The metric that achieves this half – toning is the distance parameter. This could be further improved if the decision algorithm is improved.

References:

[1] William Pratt, “Digital Image Processing”

[2] https://en.wikipedia.org/wiki/Dither

[3] D. Shaked, N. Arad, A. Fitzhugh, I. Sobel, "Color Diffusion: Error-Diffusion for Color Halftones”, HP Labs Technical Report, HPL-96-128R1, 1996.

Problem 3: Morphological Processing

Part A – Shrinking

A.1 Motivation

Morphological Processing refers to the non – linear operations that are carried out on the shape or features of the image. These operations rely on the relative ordering of the pixel values, rather than the actual pixel values. There are two sets of elements needed for this operation – one is the basic binary image, and the other is the structuring element that performs some operations on the image. Some common morphological operations are shrinking, thinning and skeletonizing. Shrinking is particularly useful in counting the number of distinct regions in an image.

A.2 Approach

Shrinking can be defined as the process of identifying the distinct regions in an image, and shrinking them to a point, so that the number of distinct objects can be identified. In general terms, shrinking refers to erasing black pixels such that object without holes shrinks to a single pixel, at or near its center of mass, and the object with holes erodes to a connected ring lying midway between each hole and connected boundary. Here shrinking is carried out using a Hit – and – Miss transform.

The Hit – and – Miss algorithm for shrinking is for two stages as follows:

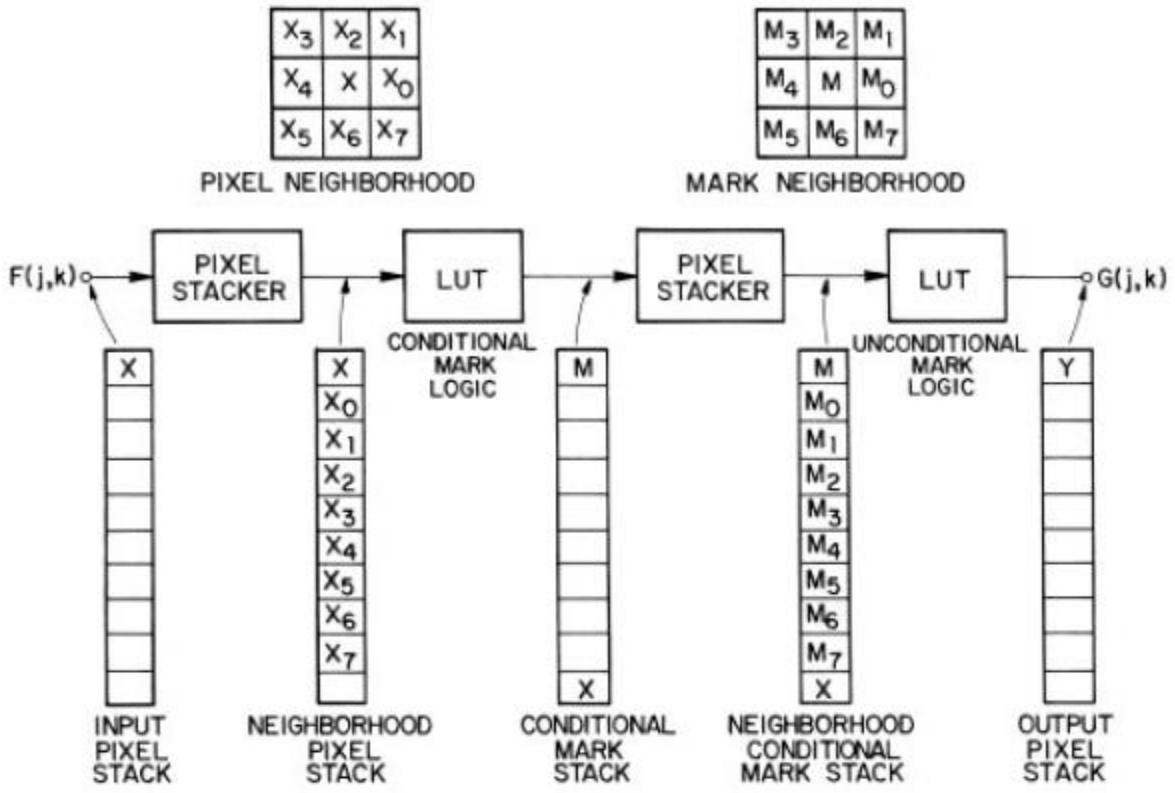
- In the first stage, the eight neighbors of every pixel are gathered in an array. The conditional mask M is created.
- In the second stage, the center pixel X and the conditional masks in a $3 * 3$ neighborhood are examined to create the output pixel.

This is performed until there are no erasures.

The conditional mask for shrinking (Stage 1) is given in Appendix A [1].

The unconditional mask for Shrinking (Stage2) is given in Appendix B. [1].

The general Hit and Miss Transform can be given as follows [1].



The approach in C++ is given below.

i. The input image is obtained in an array Imagedata using file operations.
ii. The output image array is defined as ImageOutput, which is the same size of the input.
iii. For each input white input pixel, the bond is calculated and the conditional mask pattern is obtained as the first stage of processing.
iv. In the second stage, the conditional mask pattern is obtained using the look – up tables. Based on this, the output pixel value is determined.
v. Steps iii and iv are repeated until there are no further erasures.

   Thus, every star in the image is shrinked to a single pixel. Now counting the number of white pixels in the image gives the count of the number of stars in the image.
vi. To find the distribution of the number of varied sizes of stars
   - Find the number of iterations that are required for each star to shrink to one pixel – give each new star a new label only if hasn't been assigned a label before.
   - Store the number of iterations in another array that has the same size as that of the input array.
   - After all iterations, count the number of unique labels. This gives the count of the different star sizes present in the image.
   - Plot the histogram to get a better observation of results.

A.3 Results

Shrinking is done on the stars.raw image. The actual input image is shown in Figure 3.1. Figure 3.2 shows the image that shows the stars shrinked to pixels.

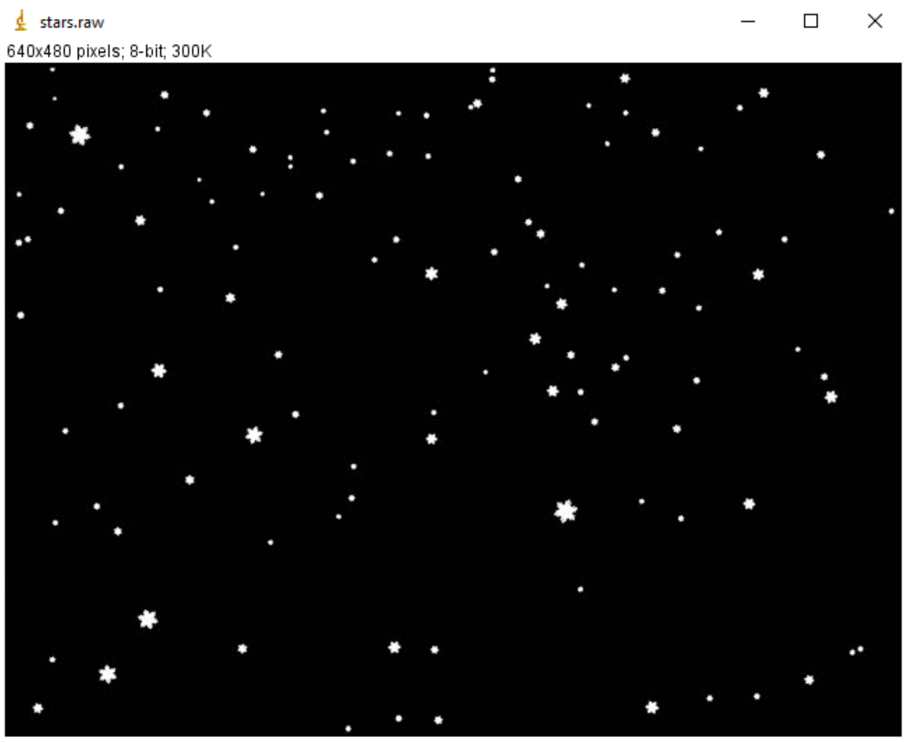


Figure 3.1 Actual stars.raw image

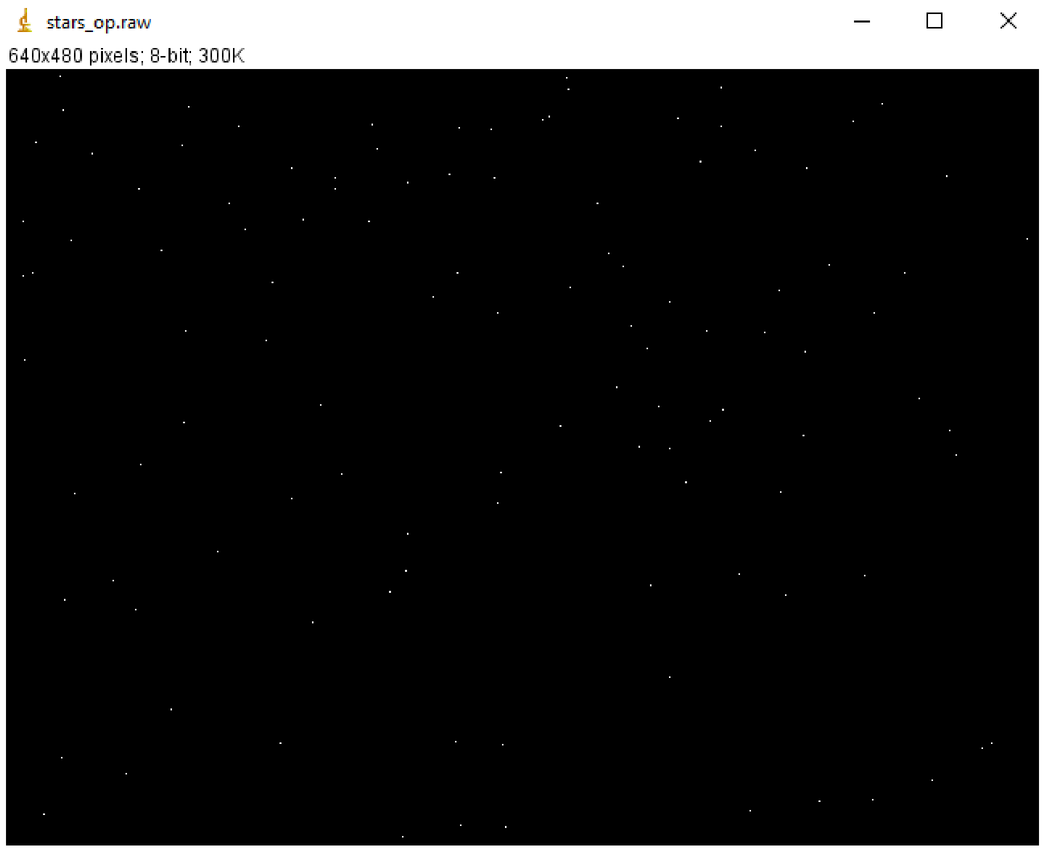
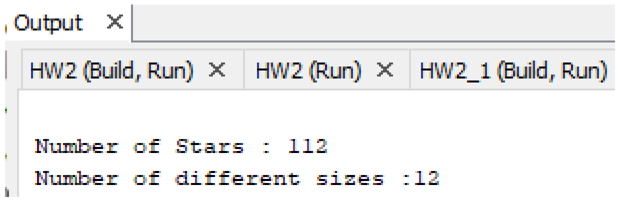


Figure 3.2 Shrinked image

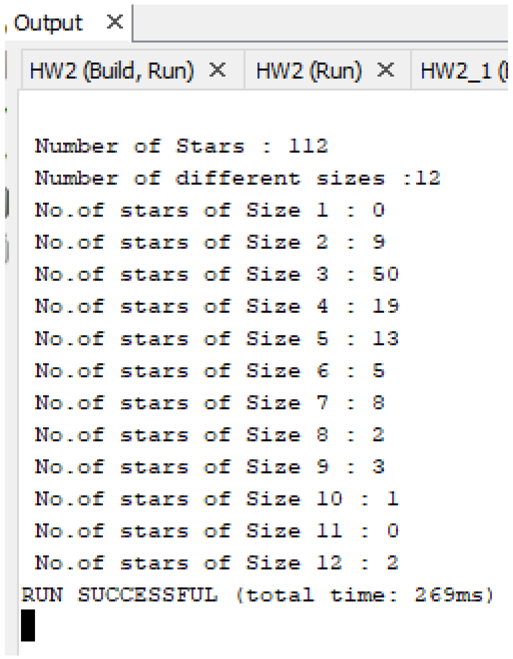
Figure 3.3 shows the count of the number of stars present in the image



```
Number of Stars : 112
Number of different sizes :12
```

Figure 3.3 Count of the stars in the image.

The number of varied sizes of the star image is shown in Figure 3.4(a)



```
 Number of Stars : 112
 Number of different sizes :12
 No.of stars of Size 1 : 0
 No.of stars of Size 2 : 9
 No.of stars of Size 3 : 50
 No.of stars of Size 4 : 19
 No.of stars of Size 5 : 13
 No.of stars of Size 6 : 5
 No.of stars of Size 7 : 8
 No.of stars of Size 8 : 2
 No.of stars of Size 9 : 3
 No.of stars of Size 10 : 1
 No.of stars of Size 11 : 0
 No.of stars of Size 12 : 2
RUN SUCCESSFUL (total time: 269ms)
```

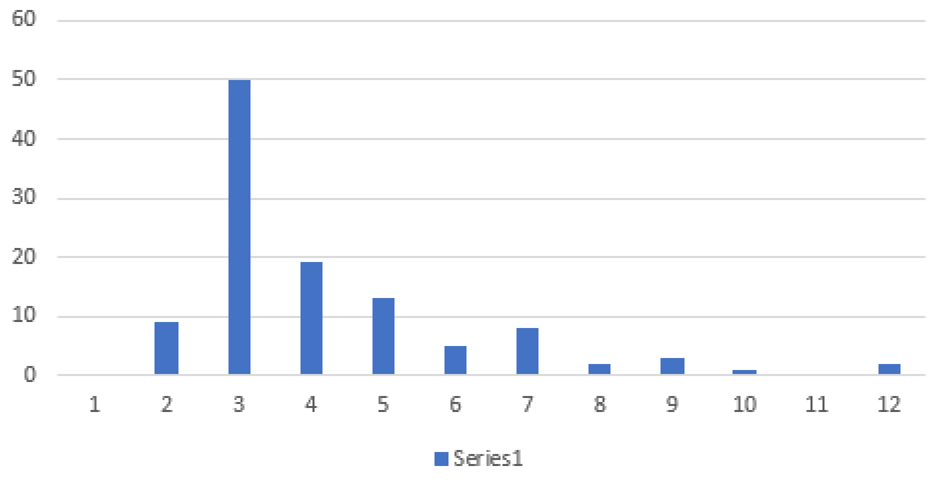


Figure 3.4(a) Number of different square sizes

A.4 Discussion

Shrinking is applied to the given input image, and the output image is obtained as in Figure 3.2. Every star is shrunk to a single white pixel. The total number of stars in the image comes to 112. The count of the stars is obtained using the Hit – and – Miss transform discussed earlier. To find the number of different star sizes, neighborhood is scanned and the size is decided based on that. The number of unique star sizes is found to be 12. This shrinking operation incorporates certain assumptions:

- The input image consists of only two values – BLACK and WHITE. If it contains any other gray scale value, it is thresholded to any of the two values.
- The output image has the same size of the input image, but the foreground pixels are reduced in number during each iteration.

The number of iterations that determine the number of stars in the image depend on the static assignment of arrays (in the implementation part). The algorithm is implemented such that there are no further erasures once the condition is not satisfied. Hence this part is fine. The number of starts of each size greatly depends on the threshold value that is given in the initial conversion of the image to Binary. These results are obtained when the threshold is 127. This varies if the threshold is set to 140. The algorithm to determine the number of stars of varied sizes can be improved, since this relies only on the values of the 8 – connected and 4 – connected pixels.

Part B – Thinning

B.1 Motivation

Morphological Processing refers to the non – linear operations that are carried out on the shape or features of the image. Some common morphological operations are shrinking, thinning and skeletonizing. Thinning is particularly useful in extracting the shape of the image. The foreground pixels are erased until there remains a minimally connected region, which is at an equal distance from all it boundaries. For example, a circular object with a hole in the center thins down to a ring.

B.2 Approach

The approach follows a similar procedure to that of shrinking. The thinning procedure follows a two – stage transformation as explained.

The Hit – and – Miss algorithm for shrinking is for two stages as follows:

- In the first stage, the eight neighbors of every pixel are gathered in an array. The conditional mask M is created.
- In the second stage, the center pixel X and the conditional masks in a $3 * 3$ neighborhood are examined to create the output pixel.

This is performed until there are no erasures.

The conditional mask for thinning (stage 1) is given Appendix A [1].

The unconditional mask for thinning (Stage2) is given Appendix B 1].

The approach in C++ is given below.

i. The input image is obtained in an array Imagedata using file operations.
ii. The output image array is defined as ImageOutput, which is the same size of the input.
iii. For each input white input pixel, the conditional mask pattern is obtained as the first stage of processing.
iv. In the second stage, the conditional mask pattern is obtained using the look – up tables. Based on this, the output pixel value is determined.
v. Steps iii and iv are repeated until there are no further erasures.
   Now the image is modified using thinning, wherein the final image has the structure of the actual image.

B.3 Results

Figure 3.5(a) shows the actual jigsaw_1.raw image used for thinning operations. Figure 3.5(b) shows the output of the image after thinning.

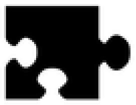


Figure 3.5(a) Input jigsaw_1.raw image

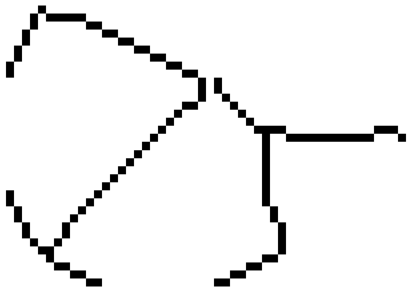


Figure 3.5(b) Image after thinning operation

B.4 Discussion

Thinning operation is particularly useful in extracting the structure of an image. It will be useful in extraction of the structure of the image. As observed in Figure 3.5(b), the actual bumps and lows in an image could be identified. Some advantages of thinning are

- identification of the actual structure of the image
- presence of stokes equidistant from the boundaries etc.

Though thinning offers several advantages, there are some disadvantages like

- The thinning stokes obtained for an object may not be always unique.
- There may be breaks in the thinning stokes, which needs additional special bridging algorithms.
- We need large iterations for images with prominent foreground pixels.

Though thinning has some disadvantages, it is found to be efficient when applied to the given input image, whose output is the thinning stoke obtained in Figure 3.5(b). This thinning structure is observed to follow the bumps and lows of the actual image. This could be improved if the bridging algorithms are implemented effectively.

Part C – Skeletonizing

C.1 Motivation

Morphological Processing refers to the non – linear operations that are carried out on the shape or features of the image. Some common morphological operations are shrinking, thinning and skeletonizing. Skeletonizing can be defined as obtaining the stick figure representation of the image. Thinning can also be done to obtain the structure of the image, but that will not be always unique. The skeleton contains less number of foreground pixels than the actual image. The skeleton represents the local object symmetries and the topology of the foreground pixels [2].

C.2 Approach

Skeletonizing of a digital image gives information about the features that are mostly based on neighborhoods or regions. Skeletonizing could be two types – topological skeletons (wherein the topology of the original object is retained) or geometrical skeletons (wherein the skeleton is at the middle of the image – inclusive of rotation, scaling and transformation). Skeletonizing is carried out using the two – stage Hit – and – Miss transform as in shrinking and thinning. The steps are explained as follows.

- In the first stage, the eight neighbors of every pixel are gathered in an array. The conditional mask M is created.
- In the second stage, the center pixel X and the conditional masks in a $3 * 3$ neighborhood are examined to create the output pixel.

This is performed until there are no erasures.

The conditional mask for skeletonizing (stage 1) is given in Appendix A [1].

The unconditional mask for skeletonizing (Stage2) is given Appendix B [1].

The approach in C++ is given below.

i. The input image is obtained in an array Imagedata using file operations.
ii. The output image array is defined as ImageOutput, which is the same size of the input.
iii. For each input white input pixel, the conditional mask pattern is obtained as the first stage of processing.
iv. In the second stage, the conditional mask pattern is obtained using the look – up tables. Based on this, the output pixel value is determined.
v. Steps iii and iv are repeated until there are no further erasures.

Now the image is modified using skeletonizing, wherein the final image has the structure of the actual image.

C.3 Results

Figure 3.6(a) shows the actual jigsaw_2.raw image used for skeletonizing operations. Figure 3.6(b) shows the output of the image after Skeletonizing.

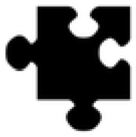


Figure 3.6(a) Input jigsaw_2.raw image

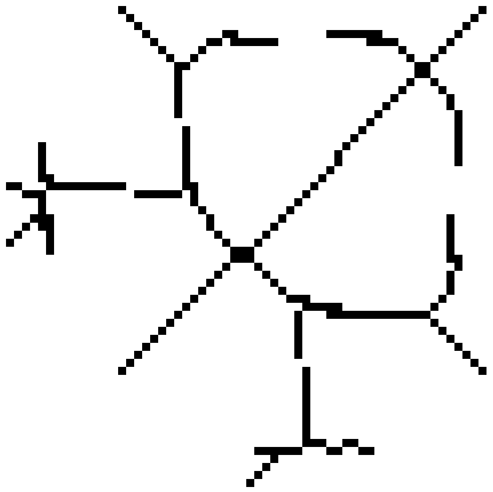


Figure 3.6(b) Image after Skeletonizing operation

C.4 Discussion

Skeletonizing is preferred to obtain the structure of the image. In simple terms, reducing the redundant pixels in an image. There are different forms of skeletonizing that could be implemented effectively to get the actual structure of the image. Some advantages of skeletonizing are

- Unique skeletons are obtained for a pattern
- Different skeletons can be obtained for the same image.

Though thinning offers several advantages, there are some disadvantages like

- Over skeletonizing such that the pattern gets distorted
- Sometimes, special bridging algorithms are needed to bridge the gaps between the skeletons.

Thus, skeletonized image obtained in Figure 3.6(b). This could be improved if the bridging algorithms are implemented effectively.

<u>Part D – Counting Game</u>

D.1 Motivation

Morphological processing can be defined as the non – linear operations that are carried out on the features of the image. Some common examples of morphological processing techniques are

- Shrinking
- Skeletonizing
- Rotation
- Scaling
- Translation
- Erosion
- Dilation
- Opening
- Closing etc.

Many of the morphological processing methods are used collectively for implementing various real – time applications. Some applications include detection and characterization of edges in medical applications, detecting defects in industrial objects, thumb – print detection, detection of holes in fibers, feature detection in moving object analysis and game theory, radar detection etc. Here a counting game is taken as an example, where in different forms of morphological processing is used,

D.2 Approach

This problem can be solved in many ways. The given image consists prominently of two colors – BLACK and WHITE. So, operations are done mostly with these two gray levels. The background here is white, and each piece of the jigsaw puzzle is black. Each puzzle has bumps and holes, that could be resulting in unique or different pieces. The objective of this problem is

- To find the number of pieces in the given puzzle
- To find the number of unique pieces in the given puzzle.

Step I – Finding the number of pieces in the given puzzle.

To find the number of pieces in the jigsaw puzzle, the following approaches could be employed.

i. Each piece of the jigsaw puzzle is a black region (in abstract terms). Every black pixel is shrunk to a single black pixel. Basically, a shrink operation is done for a certain number of times, until there are no further erasures. Each piece in the puzzle is now shrunk to a point.
   Then counting the number of points in the resulting image gives the number of pieces in the image.
ii. Using Connected Component Analysis:
   Connected Component Analysis is done to find the number of connected components in the image. Each connected component is assigned the same label. The number of labels gives the number of pieces in the image.

The connected component analysis approach is as follows.

- The entire image is scanned and the background and foreground pixels are identified.
- For each object pixel, two conditions are checked:

    - If all the neighboring pixels are background, and it hasn't been assigned a label yet, a new label is assigned to the pixel.
    - Check is any of the neighboring pixels is a black pixel. The minimum label of all is assigned as the label for the pixel.
- This is done for multiple iterations until every connected component is assigned the same label.

The approach in C++ is as follows.

i. The input image is obtained in the array Imagedata using file operations.
ii. The labels for each pixel is stored in the array Imagelabel.
iii. Every pixel is scanned, if it's a black pixel, it is either assigned a new label or assigned a minimum of all labels using the Connected Component Analysis Approach discussed above.
iv. After all connected components are identified, the number of unique labels are identified, this gives a count of the number of pieces in the jigsaw puzzle.

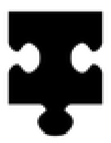
Step II – Finding the number of unique pieces in the jigsaw puzzle.

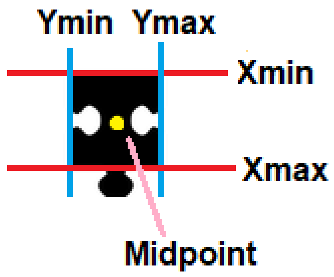
The following approach is employed to find the number of unique pieces in the image. The detailed explanation is as follows.

i. The input image is obtained.
ii. On the input image, Connected Component Analysis is done. That gives the number of unique pieces of the image.
iii. For every connected component, the midpoint of the connected component is obtained, by finding out the boundary values.
An example is shown below, wherein the image on the left is the actual piece of the jigsaw and the image on the right gives the calculation of the midpoint of each connected component.

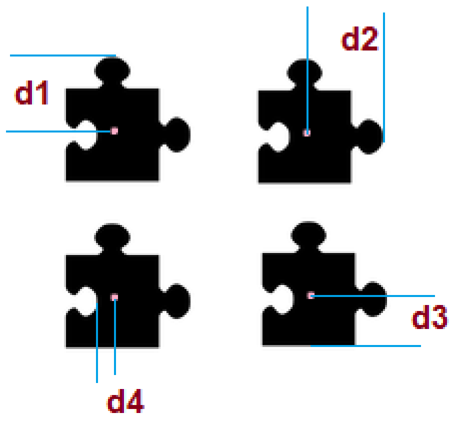
$$\text{Midpoint} = (X, Y) = \left(\frac{Xmin+Xmax}{2}, \frac{Ymin+Ymax}{2}\right)$$

The distance d of the mid point from the boundaries are also found.





iv. Thus, the midpoints of all connected components are identified. Then the distance of the farthest pixel in the connected components in the 4 – connected space is found. Let $d1, d2, d3\ and\ d4$ be the distance of the farthest connected component in each direction. An example is given below.



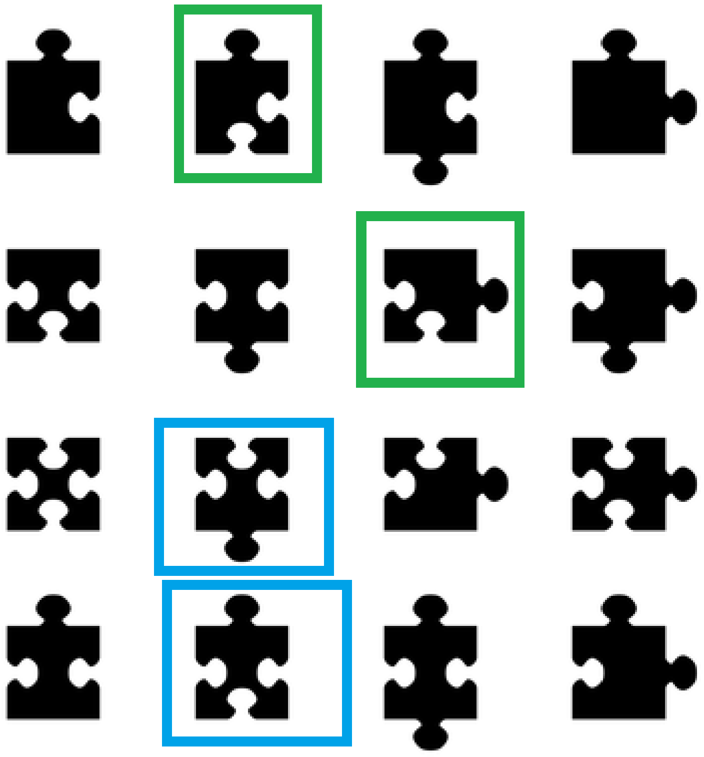
Now check if each of $d1, d2, d3, d4$ is less than or equal to d .

If $di < d$, then a value of -1 is assigned, if $di = d$, then a value of 0 is assigned, if $di > d$, then a value of 1 is assigned.

So, for the above example, this vector v would be [1 1 0 -1]. This vector v is found for every connected component.

v. Now to find the number of unique pieces in the puzzle, the following morphological techniques are considered – Rotation and Flipping.

vi. Now, each of the connected component vector v is compared with rotated versions of all other vectors from the connected components. If a vector is like the rotated version of a vector, then both the components are assumed to be the same. For example, the pieces in green are rotated version of one another. So, they are counted as one unique piece. The pieces in blue are flipped versions of one another, so they are counted only once.



vii. Now, flipping is done – both horizontal and vertical flipping. Each vector v is compared with the flipped version of the connected component. Then rotation is done to incorporate all the possible combinations. If a match is found, the components are not considered unique.

viii. This is done for all connected components.

The final count gives the number of unique pieces in an object.

The following approach is employed in C++.

i. The input image is obtained in the array Imagedata using file operations.

ii. The labels for each pixel is stored in the array Imagelabel.

iii. The number of connected components are identified using Connected Component Analysis.

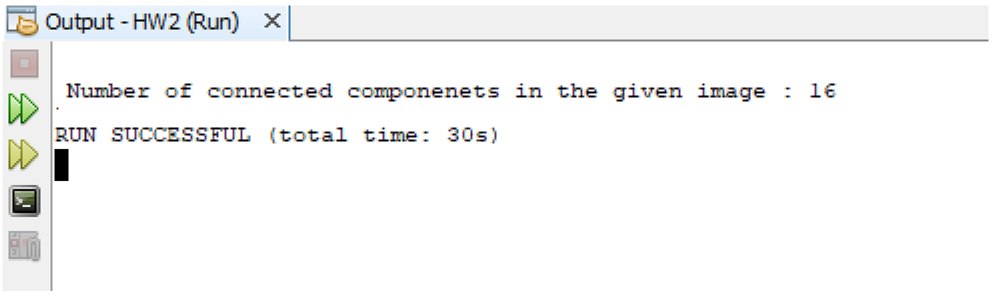
iv. The vector v of each connected component is compared with the rotated and flipped versions of each of the other component.

v. The number of unique components is identified by the variable $unique\_count$, every time when a connected component matches any of the rotated or flipped versions, the count is decremented.

Thus, the variable unique_count gives the number of unique pieces in an image.

D.3 Results

The number of pieces in the Jigsaw – Puzzle is obtained by connected component analysis. The number of pieces are found to be 16, as shown in Figure 3.7.



```
 Number of connected componenets in the given image : 16
RUN SUCCESSFUL (total time: 30s)
```

Figure 3.7 Output showing the number of connected components.

This output is also compared by using the code used in Part (A) for shrinking. The output is obtained as in Figure 3.8. The number of pieces are again found to be 16.

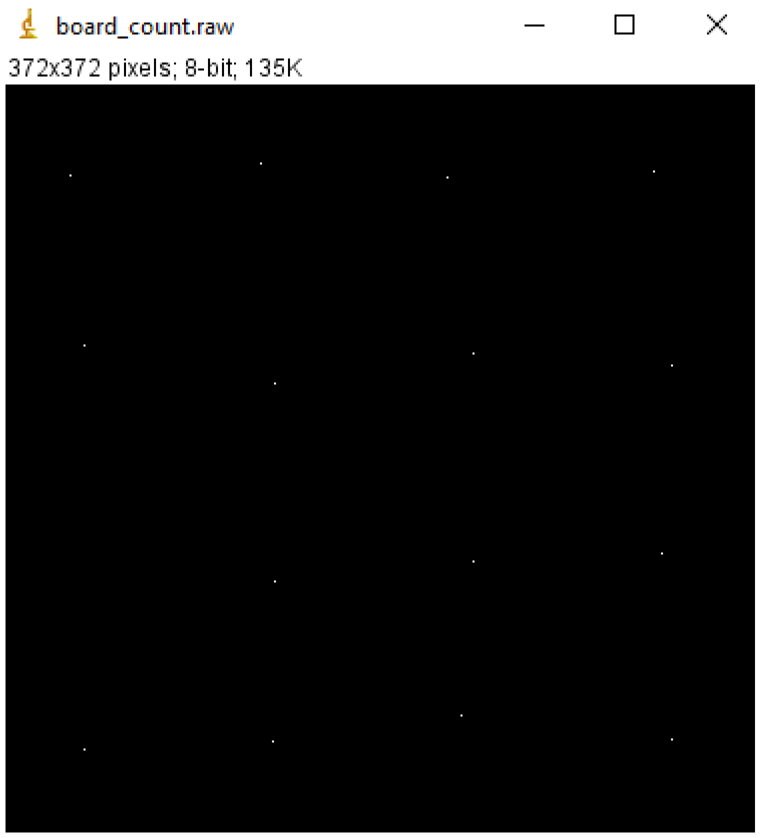
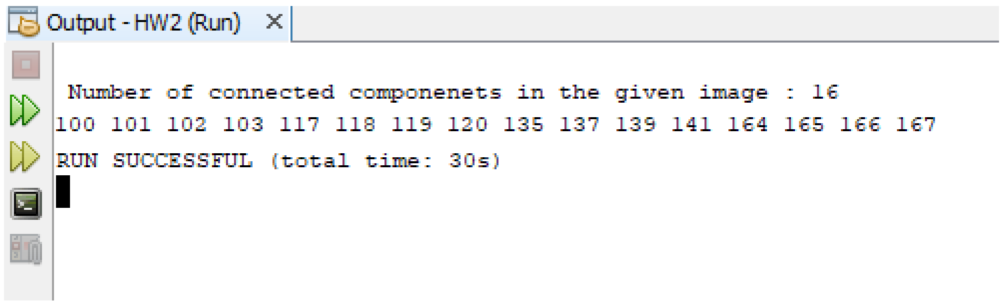


Figure 3.8 Output obtained by shrinking

The following output Figure 3.9 shows the different class labels that are assigned to different connected components. These are in unsigned char components, each of which corresponds to a gray scale level. The image obtained by assigning different class labels, corresponding to Figure 3.9 is shown in Figure 3.10.



```
 Number of connected componenets in the given image : 16
100 101 102 103 117 118 119 120 135 137 139 141 164 165 166 167
RUN SUCCESSFUL (total time: 30s)
```

Figure 3.9 Different Class Labels assigned

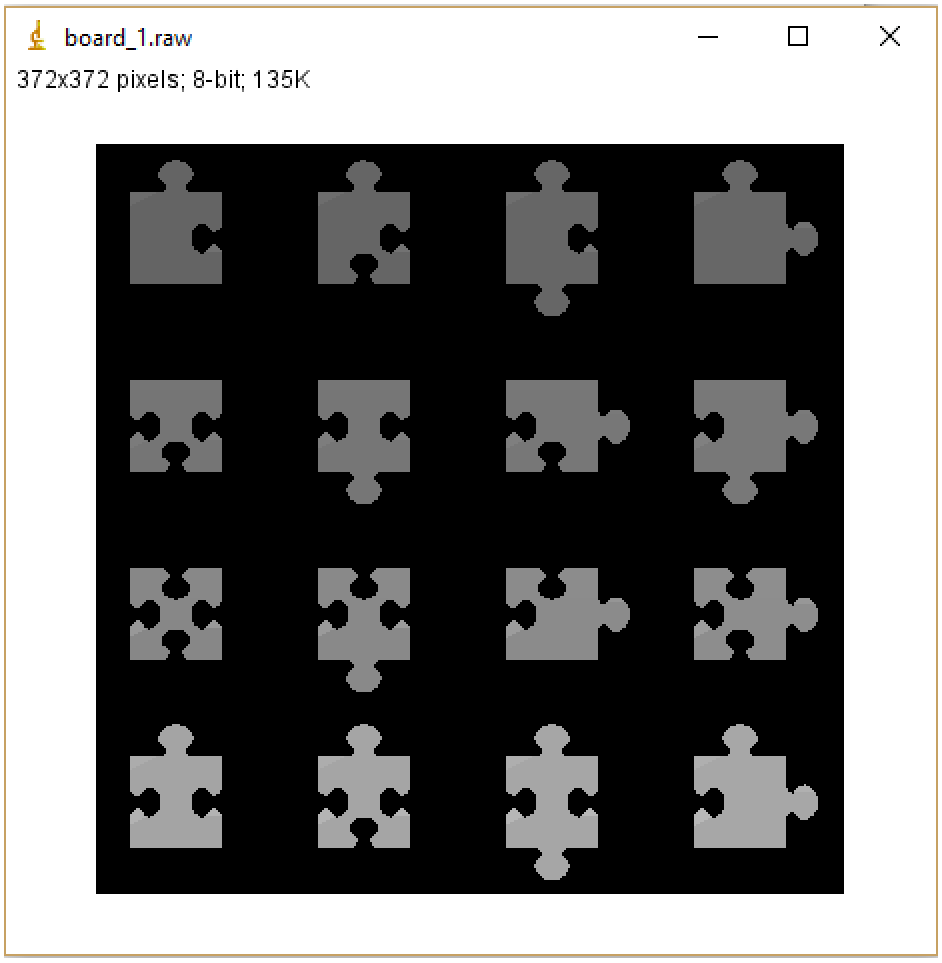
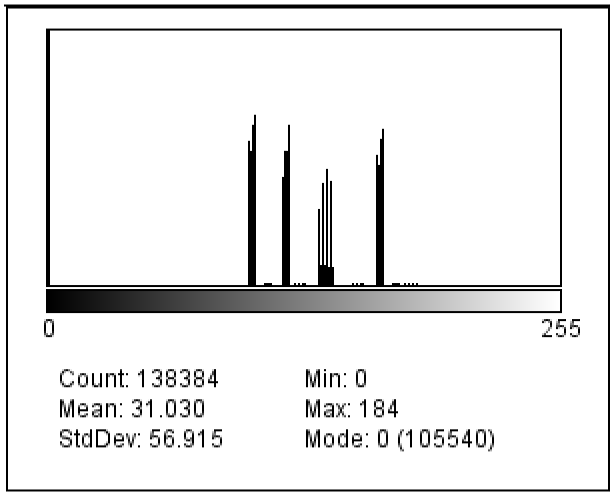


Figure 3.10 Output of Connected Component Analysis

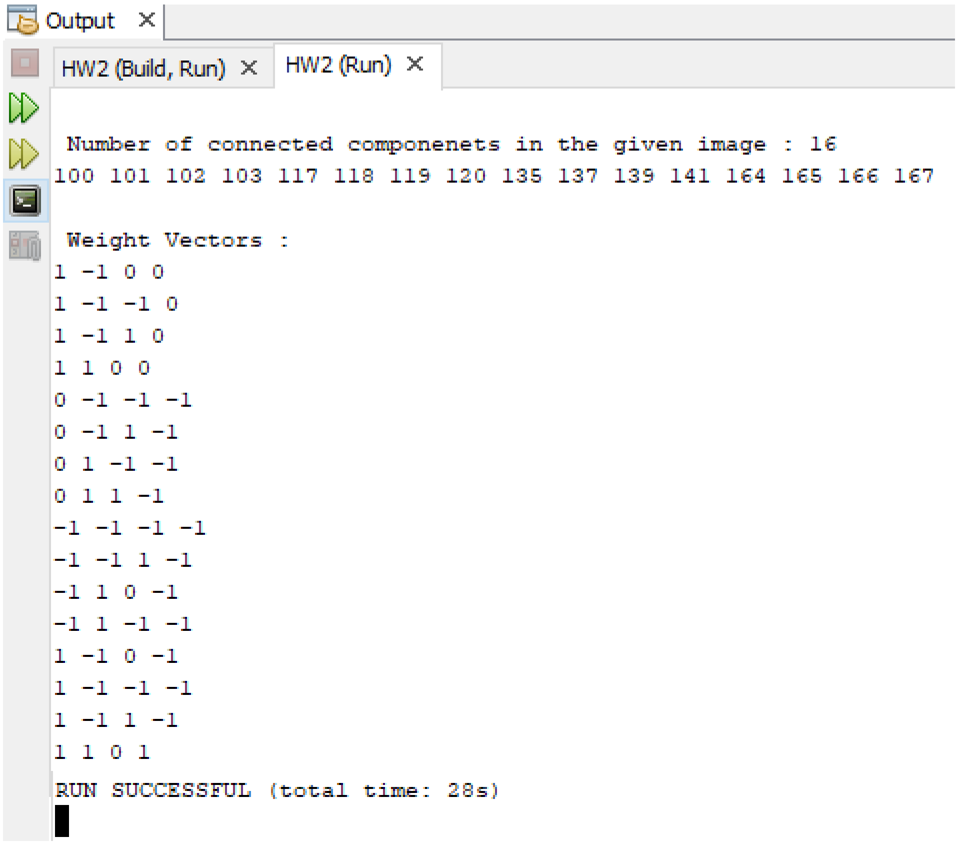
Figure 3.11 shows the histogram of the connected component analysis, wherein the pixels are also in the range of about (100,200).



```
Count: 138384      Min: 0
Mean: 31.030       Max: 184
StdDev: 56.915     Mode: 0 (105540)
```

Figure 3.11 Histogram of the obtained CCA image

Now, counting the number of unique features, the weight vectors v of each of the components are found, as shown in Figure 3.12.

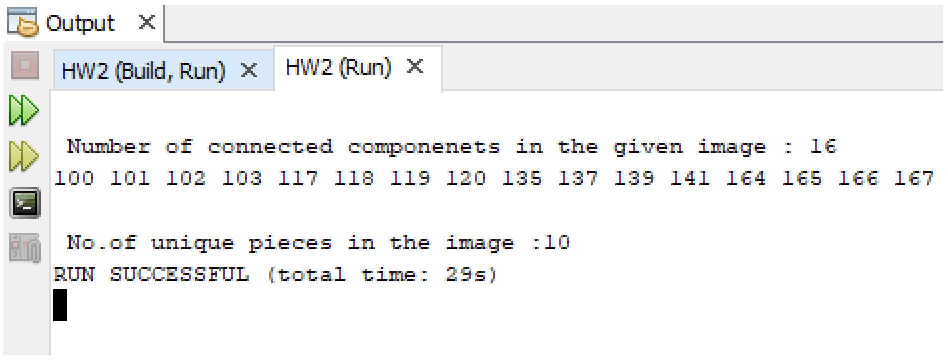


```
 Number of connected componenets in the given image : 16
100 101 102 103 117 118 119 120 135 137 139 141 164 165 166 167

 Weight Vectors :
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
RUN SUCCESSFUL (total time: 28s)
```

Figure 3.12 Weight vectors of each of the jigsaw pieces.

After rotation and flipping, the number of unique images is obtained as in Figure 3.13.



```
 Number of connected componenets in the given image : 16
100 101 102 103 117 118 119 120 135 137 139 141 164 165 166 167

 No.of unique pieces in the image :10
RUN SUCCESSFUL (total time: 29s)
```

Figure 3.13 Number of unique pieces

D.4 Discussion

There are many approaches of solving this problem. Here the number of pixels from the center of each jigsaw is taken as the criteria. Then the normal morphological techniques like rotation and flipping are done. In whatever method the algorithm is done, it involves the checking conditions of rotation ad flipping. The number of unique pieces is found to be 10 here – since flipping considers two pieces to be the same – either the horizontal way or the vertical way. This algorithm can be optimized if connected component analysis method is improved.

References

[1] William Prat, Digital Image Processing

[2] http://www.inf.u-szeged.hu/~palagyi/skel/skel.html