EE569: Homework #2
Report

ISSUED: 2/4/2018
DUE DATE: 11:59PM, 3/4/2018

ISHAN MOHANTY
USC ID: 4461-3447-18
Email: imohanty@usc.edu

"It is theoretically possible to warp spacetime itself, so you're not actually moving faster than the speed of light, but it's actually space that's moving."
------ Elon Musk

Problem 1: Geometric Image Modification

(a) Geometrical Warping

I. Abstract and Motivation

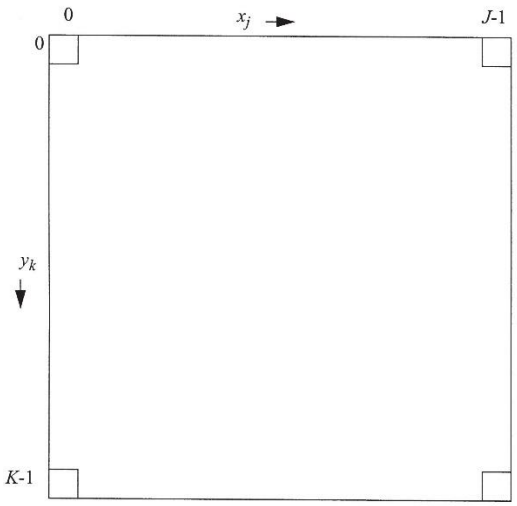
Geometrical modifications in image processing involve spatial movement of pixels using operations like scaling, translation, rotation, non-linear warping and other perspective transforms. In this case, we are motivated to achieve geometrical warping by first studying the relationship between image coordinates and cartesian coordinates. The second concept that we implement is reverse address mapping where-in we first go to the output side and derive the respective pixel intensities from the input image on which the transform was to be performed in the first place. Here we attempt to spatially warp a square image to a disc shaped image.

II. Approach and Procedures

Main Algorithm used here is Non-linear polynomial warping with variables X, Y, X*Y & 1.

The following steps describe the procedure required to perform **geometrical warping**:

1. Read the 24-bit RGB colour image.
2. Perform reverse address mapping, from output to input I.E setup output and grab pixels from input corresponding to pixels in the output.
3. This is done by first converting image coordinates to respective cartesian coordinates and after this for each cartesian coordinate pair we calculate polar radius and then threshold on the fact that the output has a radius r = $\sqrt{x^2 + y^2}$ calculated to be 256.5.



Source:[1]

Image to cartesian coordinates relationship:

$$Xj = j + 0.5 \quad \text{----- 1}$$
$$Yk = K\ (\text{Image Height}) - k - 0.5 \quad \text{----- 2}$$

J = image width, K = Image Height, Xj = cartesian x coordinate, Yk = cartesian y coordinate,

j = image column coordinate, k = image row coordinate.

Cartesian coordinate to image relationship:

$$v = x - 0.5 \quad \text{------3}$$
$$u = K\ (\text{Image Height}) - y - 0.5 \quad \text{-------4}$$

u = image row, v = image column, x = cartesian x coordinate, y = cartesian y coordinate, K = image height.

4. If the radius for the square image is less than 256.5 we perform **non-linear warping** operations else assign output value as zero or black pixels. Given below is the matrix operations that we carry out in-order to achieve non-linear warping.

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} xo \\ yo \\ xo * yo \\ 1 \end{bmatrix} = \begin{bmatrix} xi \\ yi \\ xi * yi \\ 1 \end{bmatrix}$$

Here xo, yo represent output cartesian coordinates and xi, yi represent input cartesian coordinates. The matrix above with a to l parameters represent the non-linear warping matrix.

We calculate this matrix based on 4 points in every quadrant of the on the output cartesian points and find corresponding input cartesian points.

The non-linear warping matrix has coefficients of a, b & c change for different Quadrants whereas d = 0. For example, in quadrant II, a = 0.998871, b = 0.00113 & c = 0.002265 and d=0. This gives us xi from xo that is we now know the exact position of the input pixel based on the corresponding output cartesian value mapped and transformed by the non-linear matrix. The values of a, b, c are given below, these give us xi when multiplied by xo.

Xi = a*Xo + b*Yo + c*Xo*Yo, d = 0

(X < 0 && Y > 0) -- Quadrant II

(0.998871*Xo + 0.00113*Yo + 0.002265*Xo*Yo)

(X >=0 && Y >=0) --- Quadrant I

(0.998871*Xo + 0.001122*Yo + 0.002253*Xo*Yo)

(X > 0 && Y < 0) --- Quadrant IV

(1.00112053*Xo -0.001122*Yo - 0.0022410*Xo*Yo)

(X < 0 && Y < 0) --- Quadrant III

(1.001122*Xo - 0.001126*Yo - 0.002253*Xo*Yo)

Similarly, for Yi = e*Xo + f*Yo + g*Xo*Yo , d = 0

(X < 0 && Y > 0) -- Quadrant II

(0.001128*X +0.998869*Y - 0.002265*X*Y);

(X >=0 && Y >=0) --- Quadrant I

(-0.0011289*X + 1.001122*Y + 0.002241*X*Y);

(X > 0 && Y < 0) --- Quadrant IV

(-0.0011205*X +1.001122*Y + 0.002241*X*Y);

(X < 0 && Y < 0) --- Quadrant III

(0.00112245*X + 0.998873*Y - 0.002253*X*Y);

5. After we have the xi and yi for the corresponding xo and yo, we convert these newly mapped input pixels corresponding to output pixels, to image coordinates.
6. We perform bilinear interpolation on the newly mapped input image coordinates and get the pixel intensity for the warped output image.
7. After this step output the warped image into a file and check in ImageJ.

The following steps describe the procedure required to perform **geometrical dewarping**:

1. Read the warped image.
2. Convert the warped image coordinates to cartesian coordinates.
3. Similarly, like warping, perform the algorithm for dewarping using non-linear warping matrix transform. Now instead of getting input pixel positions, get output pixel locations using reverse address mapping.

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} xi \\ yi \\ xi * yi \\ 1 \end{bmatrix} = \begin{bmatrix} xo \\ yo \\ xo * yo \\ 1 \end{bmatrix}$$

a, b & c values for Xi, Yi & Xi*Yi for 4 different quadrants, Here we get Xo based on values:

1.00056776e+00 , -5.68880703e-04 , -1.13998357e-03, // a, b, c (d = 0)

1.00056888e+00 ,-5.65549579e-04 ,-1.13554790e-03,

9.99434446e-01 , 5.66658513e-04 , 1.13110351e-03 ,

9.99433341e-01 , 5.67767429e-04 , 1.13553921e-03

e, f & g values for Xi, Yi & Xi*Yi for 4 different quadrants, Here we get Yo based on values:

-5.67763072e-04 , 1.00056888e+00 , 1.13998357e-03, // e, f, g (h=0)

5.68880703e-04 , 9.99434450e-01 , -1.13554790e-03,

5.65553919e-04 , 9.99433341e-01 , -1.13110351e-03,

-5.66658513e-04 , 1.00056777e+00 , 1.13553921e-03

4. After getting the mapped output cartesian coordinates, convert them to image coordinates and do bilinear interpolation to get the corresponding pixel intensity.
5. Write the image into a file and analyse the image in ImageJ.

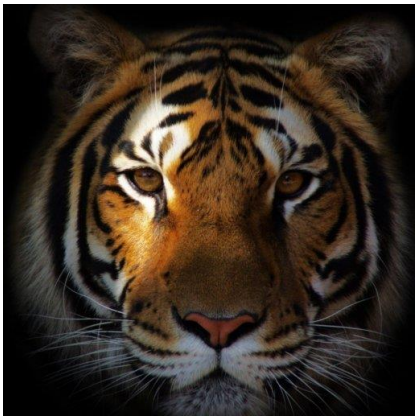
III. Experimental Results

(a) Original Puppy Image

(b) Warped Puppy Image

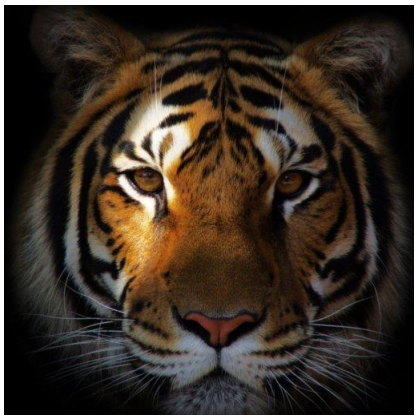
(c) Original Puppy Image

(d) Dewarped Image



(e) Original Tiger Image

(f) Warped Tiger Image



(g) Original Tiger Image

(h) Dewarped Tiger Image

(i) Original Panda Image

(j) Warped Panda Image

(k) Original Panda Image (L) Dewarped Panda Image

Figure 1: Warping and Dewarping

IV. Discussion

After performing second order polynomial warping on the images, we see above all the results obtained after warping and reverse warping or dewarping.

Since second order non-linear polynomial is used, the warped images above have very distinctive and sharp changes that can be easily observed by the naked eye. The spatial transformations are large and these creates some distortions in the images when one tries to dewarp or get the warped image back to the original image. We don't observe any obvious differences between reverse warped images and original images, but if we look closely we can find some distortions in the images (k) and (L) of the panda images where in:

1.The original image has a flat head whereas the dewarped or reverse warped image has a rounder head.

2. eyes are sharper and better quality in the image (k) whereas they are little blurred and not so sharp in the image (L) of the panda.

3. you can also observe the size of the left arm is different in the original panda image and the dewarped panda image.

We can also observe that there is some difference in the puppy image:

1.if you closely observe the images (c) and (d), you can see that there are darker patches at the bottom left and right edge in the pattern of the fur. Like it is more blurred in the (d) or dewarped puppy image and sharper in the original image.

Although these are some not so obvious differences we can see that the problems of the distortions can arise due to many reasons, some may be:

1. We have more equations than unknowns as we take 4 control points for each quadrant and hence 16 control points overall, this leads to an overdetermined system producing some inaccuracies and distortions. Hence our non-linear warping matrix is not very accurate as the parameters lose some information during the calculation process.

2. While doing bilinear interpolation we are only able to estimate the pixel intensity and since we have floating point values in our coordinate system we cannot accurately determine the pixel intensity. Therefore, while warping we do lose some amount of information due to bilinear interpolation and while dewarping or reverse warping there is some additional loss as well due to bilinear interpolation. Hence, we cannot expect a perfect image while recovering a warped image.

3. Polynomial warping or non-linear warping does produce a lot of information loss due to distortions. This could also be a problem during image recovery process.

1(b) Homographic Transformation and Image Stitching

I. Abstract and Motivation

Homographic transformation is used to create panoramas that give a 360 degree view of the object that is captured by a sweeping uncalibrated camera. Depending on the angle far off objects look very tiny, perspective projection concepts when applied corrects the image and makes it look normal. In this section we are motivated to perform image stitching using homographic transformation to create a panorama.

II. Approach and Procedures

1.Read the 3 RGB images: left, middle and right.

2. translate and paste the middle image to a black surface of size 5000x5000.

3. select control points that are very unique and common to both the left and the middle image pasted on the black surface and perform homographic transformation. This transformation gives the new spatial information for the left image with respect to middle pasted image. 11 points common to both middle image pasted on black surface and left image were taken. After the transformation the images looked like they were stitched together at an angle. This can be seen in the results section below.

4. select control points that are very unique and common to both the right and the middle image pasted on the black surface and perform homographic transformation. This transformation gives the new spatial information for the right image with respect to middle pasted image. 11 points common to both middle image pasted on black surface and left image were taken. After the transformation the images looked like they were stitched together at an angle. This can be seen in the results section below.

5. Hence, after performing these results we could see the image stitching happening very efficiently. The inverse homography matrix was calculated via inbuilt MATLAB functions and the control points were chosen manually.

6. we use bilinear interpolation when converting cartesian coordinates to image plane coordinates.

The homographic transformation procedure is stated below. Images of points in a plane, from two different camera viewpoints, under perspective projection (pin hole camera models) are related by a homography:

$$P_2 = HP_1$$

where H is a 3x3 homographic transformation matrix, P_1 and P_2 denote the corresponding image points in homogeneous coordinates before and after the transform, respectively. Specifically, we have

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

To estimate matrix H, you can proceed with the following steps:

- Select four point pairs in two images to build eight linear equations.
- Solve the equations to get the 8 parameters of matrix H.
- After you determine matrix H, you can project all points from one image to another by following the backward mapping procedure and applying the interpolation technique.

Control Points for left and middle image:

Left image coordinates (x,y)	Middle image pasted on black space coordinates (x,y)
337,426	2073,3780
322,30	2050,3381
390,453	2130,3808
392,316	2134,3675
398,430	2141,3787

424,567	2167,3911
277,591	2211,3561
373,560	2118,3918
325,192	2066,3548
460,201	2015,3965
453,596	2194,3940

The inverse homography matrix for left to padded middle image is given by:

$H^{-1}_{L} =$ 0.555512 0.021690 -253.9237

-0.18858 0.76337 -443.5935

-0.000574 0.000017 1.260170

Control Points for Right and middle image:

Right image coordinates (x,y)	**Middle image pasted on black space coordinates (x,y)**
90,587	2280,3939
121,434	2319,3793
29,365	2231,3722
33,414	2235,3768

81,355	2278,3713
61,358	2260,3718
125,436	2319,3796
125,433	2329,3793
168,556	2371,3910
119,431	2311,3783
(52,195)	2254,3561

The inverse homography matrix for right to padded middle image is given by:

H^{-1}_R = 1.159958 -0.011384 -1154.5288

0.208937 1.073586 -1094.069

0.000609 -0.000017 0.404287

III. Experimental Results

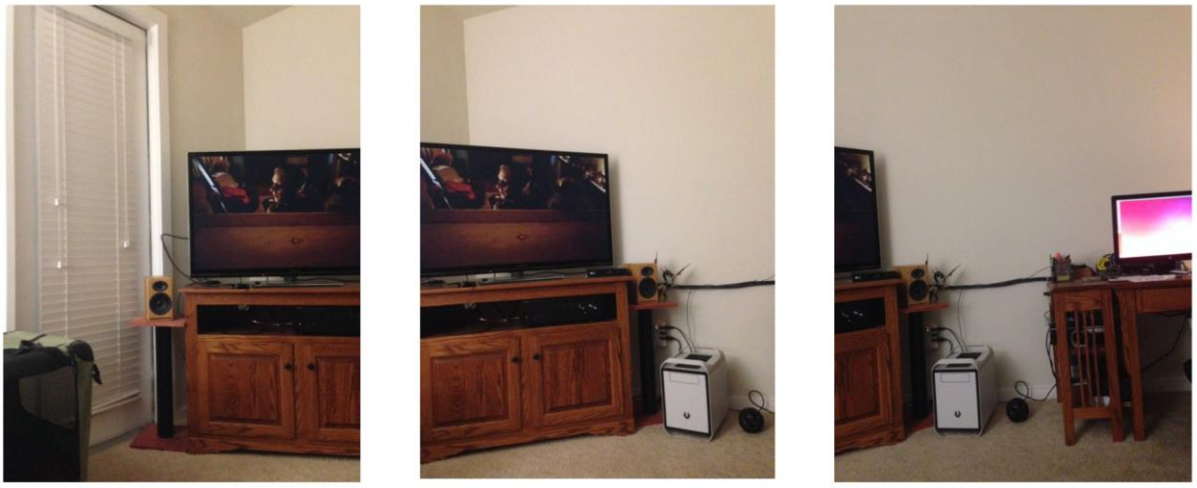


Figure 2: Left, Middle and Right Image

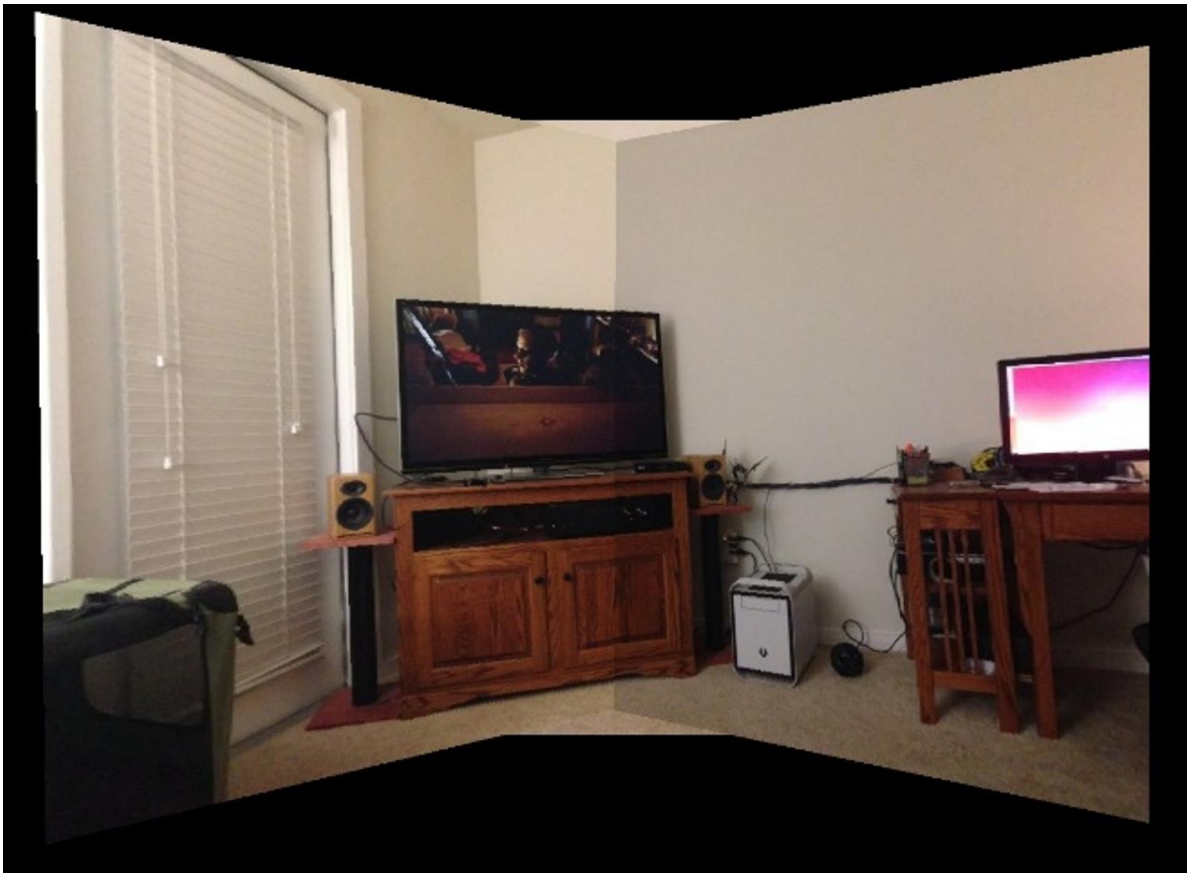


Figure 3: Stitched Image

IV. Discussion

1. In the above image 11 control points each were used for the left and middle pasted image and the right and middle pasted image respectively. More than 4 key points if used produces a more accurate result, else we cannot get a perfect stitch. For this reason algorithms like SIFT and RANSAC use large amount of key points common to both images.

2. Strategies for selecting points:
   For the left and Middle image:
   i. We can observe that the TV edges are sharply angled, hence we can find some key points along the edges of the TV.
   ii. The table on which the TV is placed has some sharp transitions, this motivates us to select some more points there common to both images.
   iii. Also the walls meet at a common point in both images, we could take a few key points common to both images.
   iv. There are some shiny pixels in the hollow side of the table under the TV in both images, we could select 2-3 common points there as well.

v. Therefore, we go linearly from top to bottom establishing points in a vertical as well as horizontal direction along the edges. We need to observe extremely unique key points or sharp transition points. That is the Key to selecting good control points.

Problem 2: Digital Half-toning

2(a) Dithering

I. Abstract and Motivation

The halftone technique takes in continuous image form and produces dots to give a gradient effect which is visually attractive and helps printers that cannot reproduce continuous imagery. Dithering uses some form of noise applied to data that randomizes the quantization error hence prevents color banding. It typically converts greyscale images to black and white images in such a way that the black dots gives the average grey level effect in the original image. Dithering techniques have improved printing by a huge margin. In this section we are motivated to study different dithering techniques like average thresholding or fixed thresholding, Random thresholding and dithering using Bayer matrix.

II. Approach and Procedures

Procedure for fixed thresholding:

1. Read 8-bit input image.
2. Perform thresholding by selecting the threshold as 127 and make the pixel intensities either 0 or 255 based on the threshold of 127. Pixel intensity > 127, pixel value = 255 or pixel intensity <= 127, pixel value = 0.
3. Store the result to an output buffer and write to a file location.

Procedure for Random thresholding:

1. Read 8-bit input image.
2. generate a random number in the range 0 – 255 and set this as threshold.
3. Based on this threshold assign 0 if pixel value is less than threshold else output 255.
4. Store these threshold pixel values in an output buffer and write to a file location

Procedure for Dither Matrix thresholding:

1. Read 8-bit input image.
2. Bayer Matrix for I_2 is given, use this to calculate I_4, I_8.
3. Corresponding to I_2, I_4, I_8 calculate threshold matrix T_2, T_4 and T_8.
4. Using the threshold matrix T, assign values to the output based on the threshold value T given in the matrix position T [Img_row % N] [Img_col % N], where N = 2, 4 or 8.
5. Write the output to a file location and analyse with ImageJ.

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

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

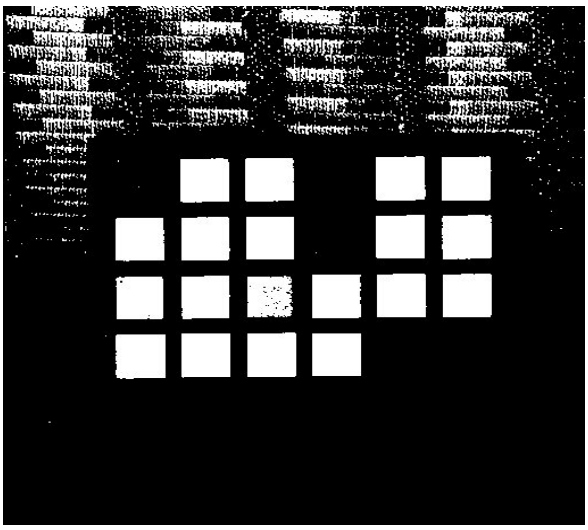
$$T(x,y) = \frac{I(x,y) + 0.5}{N^2}$$

Note: Multiply T(x,y) by 255 to get the value of the threshold

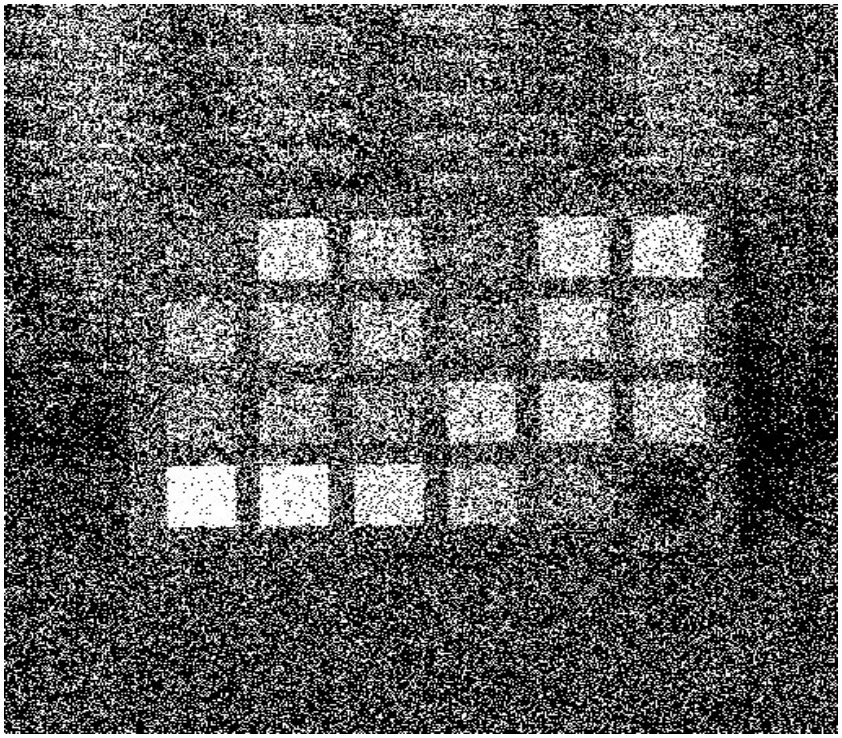
Procedure for Quad level or 4 level thresholding:

1. Read 8-bit input image.
2. Bayer Matrix for I_2 is given, use this to calculate I_4, I_8.
3. Corresponding to I_2, I_4, I_8 calculate threshold matrix T_2, T_4 and T_8.
4. Now we see that if the pixel value is between zero and T/2 we assign the output pixel intensity as 0, if it is between T/2 and T we assign the pixel intensity as 85, if it is between T and (255+T)/2 we assign the pixel value as 170 else if it is between (255+T)/2 and 255 we assign pixel intensity as 255.
5. After thresholding by this method we output the image to a file.

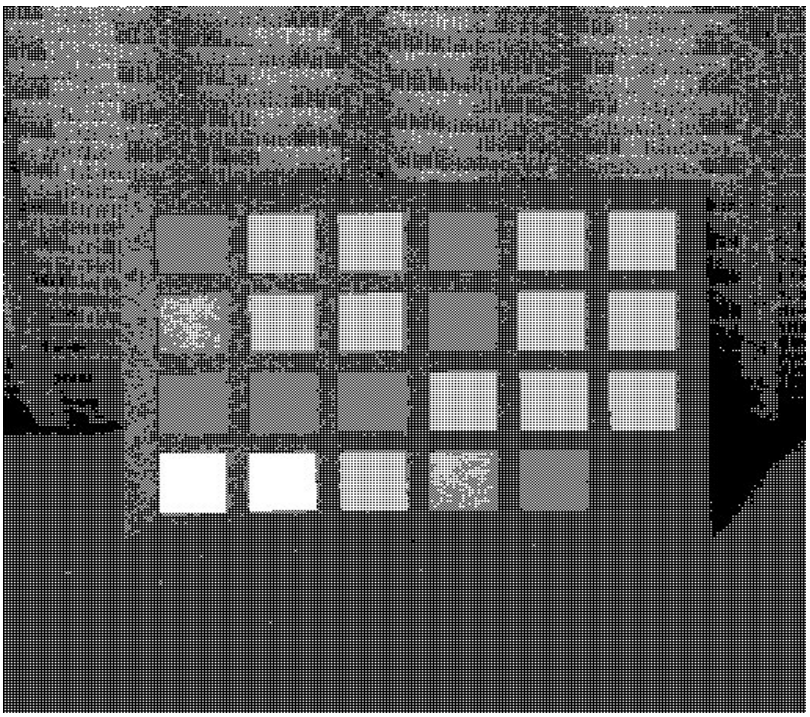
III. Experimental Results



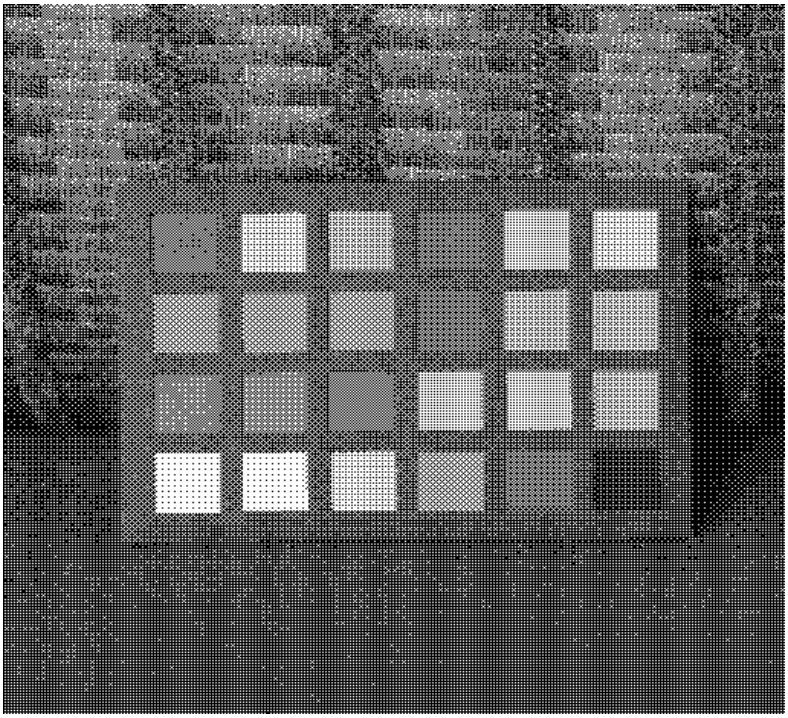
(a) Fixed threshold



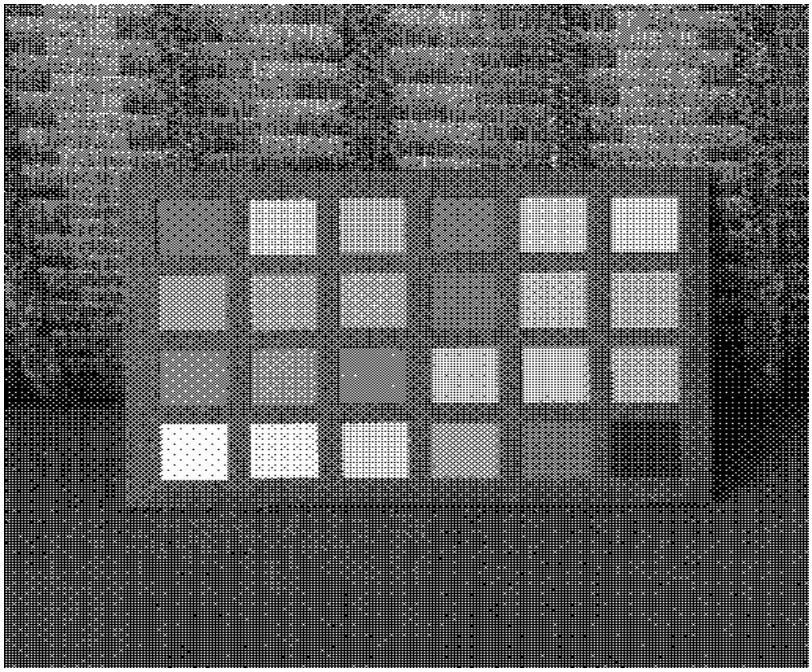
(b) Random Threshold



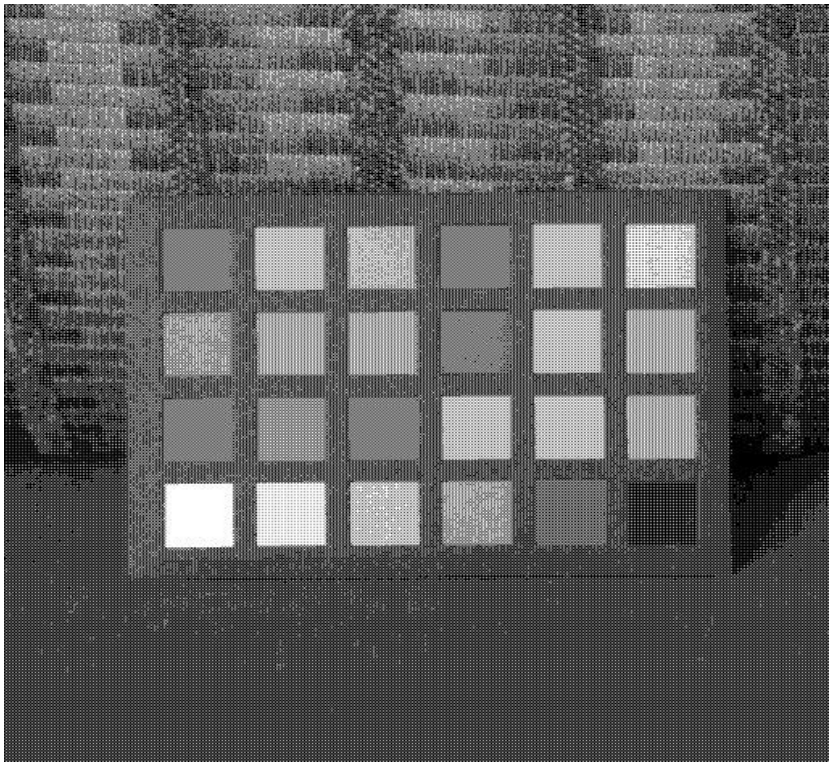
(c) Bayer Dither Matrix N= 2



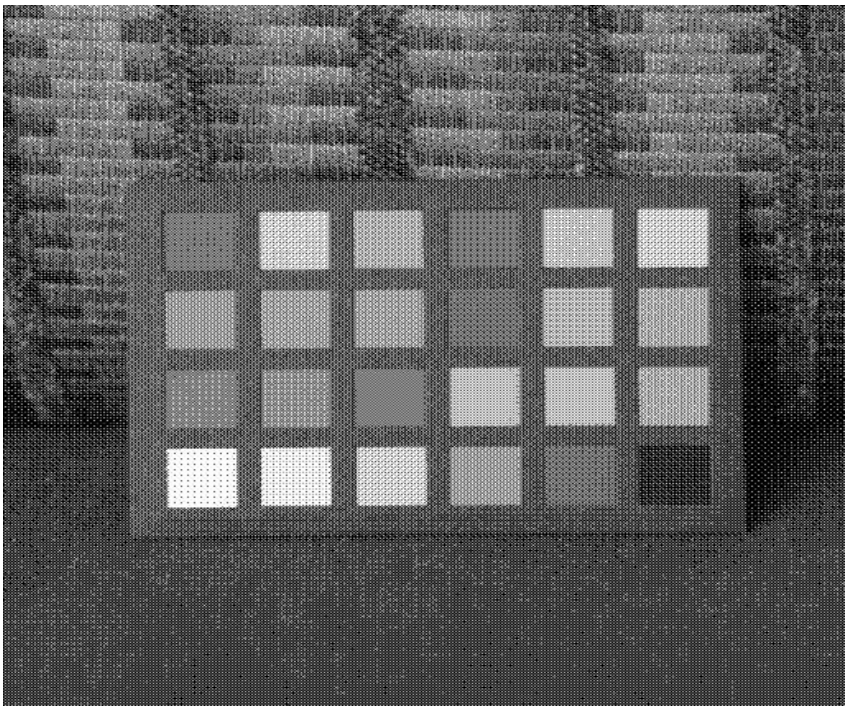
(d) Bayer Dither Matrix N = 4



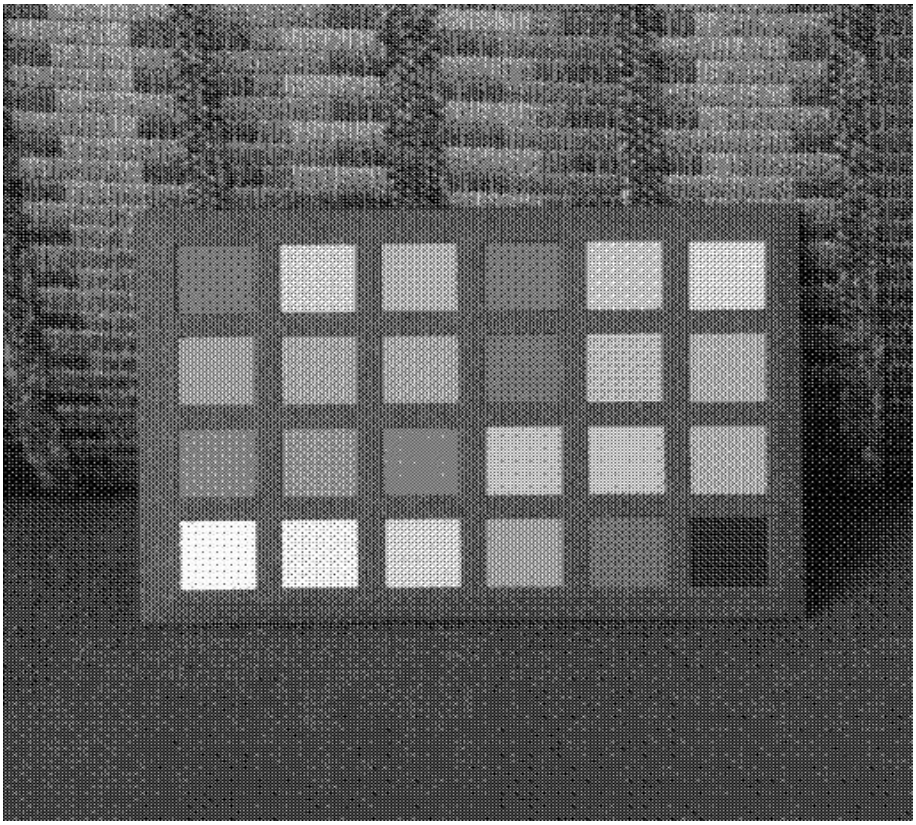
(e) Bayer Dither Matrix N = 8



(f) four level bayer dither Matrix N = 2



(g) four level bayer dither Matrix N = 4



(h) four level bayer dither Matrix N = 8

Figure 4: Dithering methods : Fixed, Random, Bayer Dither Matrix Method, Quad level Bayer Matrix Method.

IV. Discussion

Comparing these images we see that the fixed level threshold image (a) has the worst result in terms of image perception as this results in huge loss of detail. (b) the random threshold method gives better results than the fixed level threshold algorithm as it produces randomness in the image and does not produce artifacts, but due to lot of noise the image details become very messy. The bayer matrix method in (c) , (d) and (e) are clearly visually better than fixed and random methods but the disadvantage is that they produce patterns like cross-hatches. Among (c) ,(d) and (e) the bayer matrix with N=8 has the best visual appeal as it uses the threshold matrix T_8 in a very effective manner. The best out of all of these are the quad level or four level bayer matrix (f), (g) and (h), we can see that it is better than the normal bayer matrix method as the bayer matrix method thresholds between 0 and 255 and hence has sharp contrasts. The image in the normal bayer matrix method has a lot of white pixels in the surrounding areas but this is made better in the quad level bayer matrix method as it has 4 colors to express the image rather than 2. Among (f), (g) and (h), the quad level bayer matrix with N=8 looks the best as it produces a nice halftone which is soothing to the eye.

The design idea and algorithm for the four level Bayer Dither Matrix method with four gray-levels (0, 85, 170, 255) is explained above in the approach and procedures part.

2(b) Error Diffusion

I. Abstract and Motivation

Error diffusion is one of many types of half-toning that distributes error quantities that are calculated by quantizing residuals. These error values are distributed to neighbouring pixels and these change in values influence thresholding at future stages. Error diffusion is mainly used in converting multi-level images to binary images.

II. Approach and Procedures

Procedure for Floyd-Steinberg, Jarvis, Judice, and Ninke (JJN) and Stucki:

1. Read 8-bit image.
2. Select threshold as 127.
3. We perform serpentine scanning i.e for every odd value of rows we move in the reverse direction and for even rows we move in the regular fashion.
4. If the original pixel value is greater than 127 assign the output image to 255 else assign a 0.
5. Now the new pixel value is the thresholded one and is stored in a variable.
6. We then calculate error as (original pixel value – new pixel value).
7. This error is distributed to neighbouring pixels by multiplying diffusion error with the corresponding weights present in the 3x3 matrix of Floyd-Steinberg, 5x5 matrix of JJN and 5x5 matrix of Stucki. This product is summed up with the neighbouring pixel values and updated in the buffer image.
8. Repeat this process for all pixel values present in the image.
9. Output the image into a file location after the first 8 steps are completed.

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

Floyd Steinberg Error Diffusion Matrix

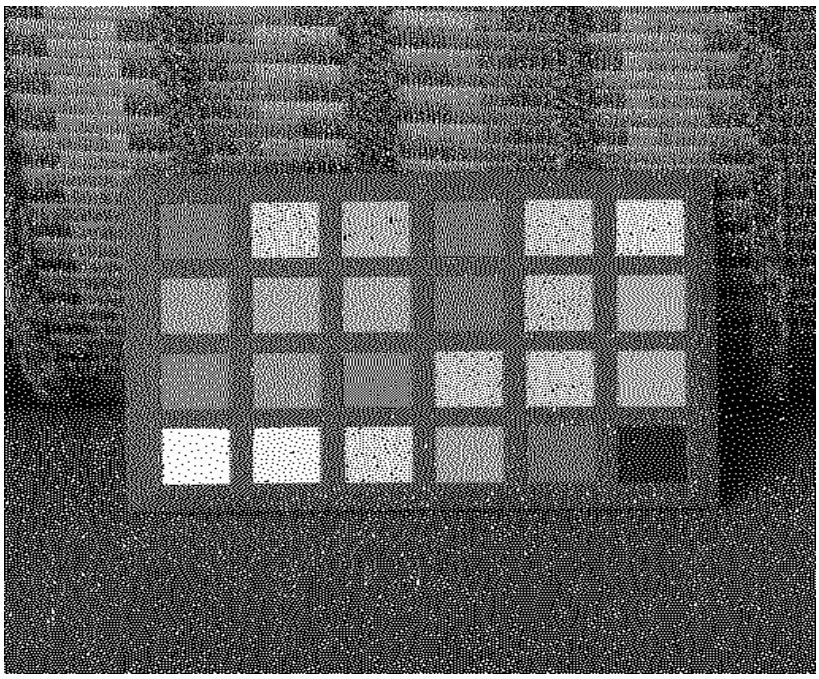
$$\frac{1}{48}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

JJN Error Diffusion Matrix

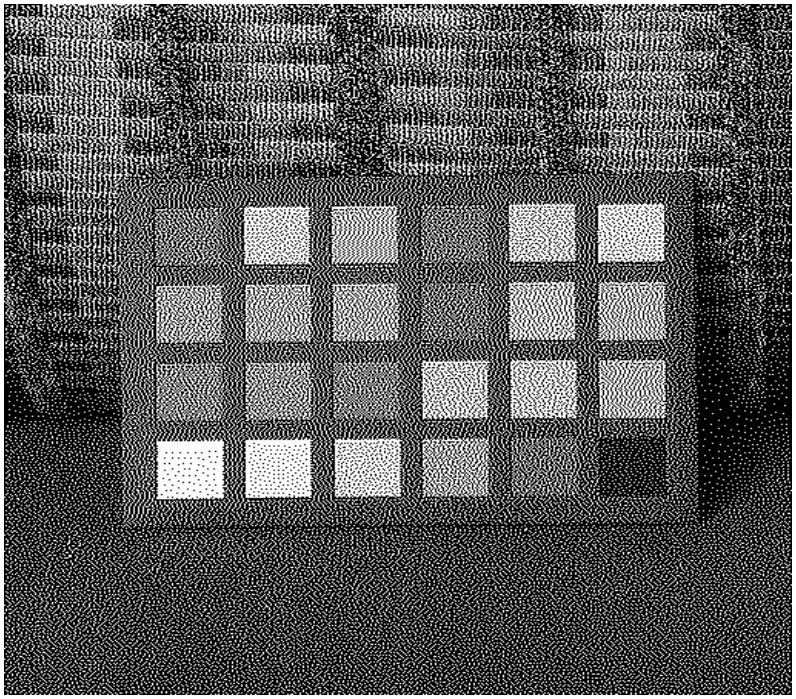
$$\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

Stucki Error Diffusion Matrix

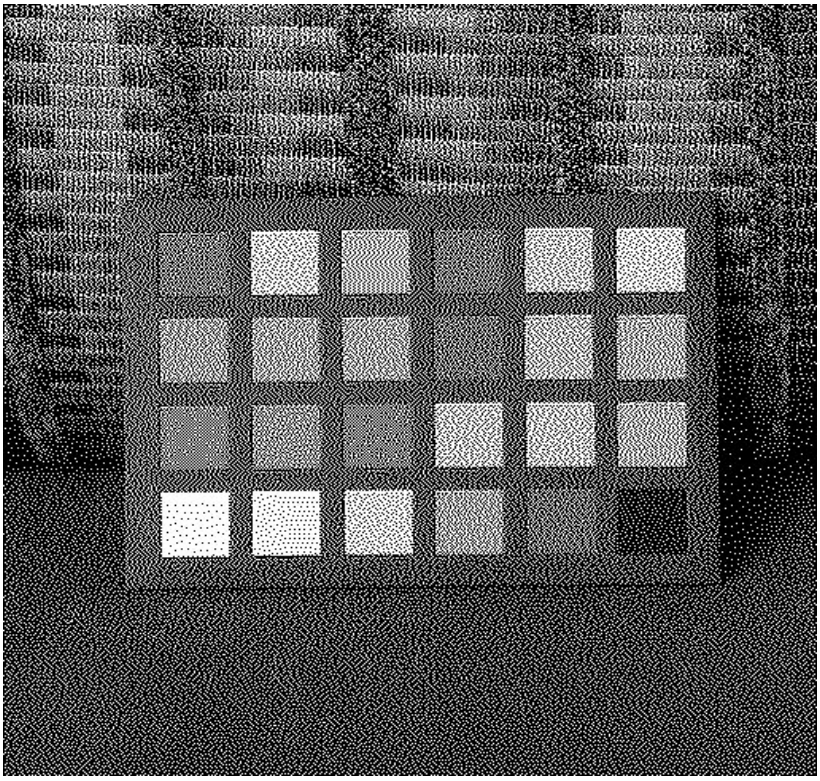
III. Experimental Results:



(a) Floyd-Steinberg Error Diffusion



(b) Jarvis, Judice and Ninke (JJN) Error Diffusion



(c) Stucki Error Diffusion

Figure 5: Floyd-Steinberg, JJN and Stucki Error Diffusion

IV. Discussion

Comparing the error diffusion methods amongst each other first we observe that the Floyd Steinberg method produces a very fine-grained image with lot of sharp white pixels in the neighbourhood. This does not seem very pleasing to the eye although it does remove some artifacts. The JJN method produces a coarser image that looks visually better than the Floyd Steinberg method as it applies the matrix weights on a larger neighbourhood and decresses visual artifacts. The Stucki is somewhat similar to JJN method as it also has a error diffusion matrix that is 5x5, although the weights in the matrix has been cleverly designed so that we get a clear and sharp output. Therefore Stucki > JJN > Floyd-Steinberg method.

Now comparing these results with the dithering bayer matrix method, we see that the error diffusion methods are much better than the dithering bayer matrix method. Reasons being:

1. Visually the error diffusion methods look more continuous, coarser and sharper.
2. Dithering matrix is ordered type that is the neighbouring pixels are not affected by the matrix whereas in the error diffusion method, the quantized residual is diffused to the neighbours hence, influencing thresholding decision values. This process is better as it reduces patterned artifacts and hence more visually appealing.

My idea to get better results would be to add some amount of gaussian noise to introduce randomness and then perform the Stucki or JJN error diffusion. I think this may give better results as the randomness in the noise coupled with fixed weights that are multiplied by neighbours will produce an altogether random pattern that would reduce more artifacts. Hence, I feel this method would be most effective.

2(c) Color Halftoning with Error Diffusion

I. Abstract and Motivation

Color halftoning is basically error diffusion applied to the three color planes independently. In this section we are motivated to learn about separable error diffusion which is the error diffusion algorithm applied to the 3 color planes separately and also MBVQ or minimum brightness variation criterion algorithm which thresholds a pixel based on the vertices of a tetrahedron.

II. Approach and Procedures

Procedure for separable error diffusion:

1. Read the 24-bit RGB image.
2. Construct a color map that contains CMYK & RGBW as the vertices of the cube.
3. Convert to CMY color plane.

4. Perform serpentine scanning and during this time calculate the Euclidean distance between each of the vertices of the color cube with respect to the pixel value.
5. Select the vertex which is nearest to the pixel intensity value.
6. Assign that representative pixel intensity of the color cube vertex to the new pixel intensity. In this process we threshold based on the closest distance of the pixel value to the vertices of the cube.
7. We then perform Floyd Steinberg error diffusion to the three color planes and threshold the pixel values which are sent to the intermediate output buffer.
8. We then convert the thresholded CMY image back to RGB and write to the output.
9. Hence, we obtain the separable error diffusion method on the RGB image.

These below represent the color cube map :

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

Procedure for MVBQ:

1. Read the 24-bit RGB image.
2. Construct color tables for the following tetrahedrons which are MVBQ types: CMYW, MYGC, RGMY, KRGB, RGBM & CMGB.
3. Convert to CMY color plane.
4. Perform serpentine scanning and during this time select the MVBQ type based on the algorithmic diagram shown below and calculate the Euclidean distance between each of the 4 vertices of the tetrahedron with respect to the pixel value.
5. Select the vertex which is nearest to the pixel intensity value.
6. Assign that representative pixel intensity of the tetrahedron vertex to the new pixel intensity. In this process we threshold based on the closest distance of the pixel value to the vertices of the cube.
7. We then perform Floyd Steinberg error diffusion to the three color planes and threshold the pixel values which are sent to the intermediate output buffer.
8. We then convert the thresholded CMY image back to RGB and write to the output.
9. Hence, we obtain MVBQ thresholded RGB image.

```
pyramid MBVQ(BYTE R, BYTE G, BYTE B)
{
  if((R+G) > 255)
    if((G+B) > 255)


      if((R+G+B) > 510)     return CMYW;
      else                  return MYGC;
    else                    return RGMY;
  else
    if(!((G+B) > 255))
      if(!((R+G+B) > 255)) return KRGB;
      else                  return RGBM;
    else                    return CMGB;
}
```

Figure 5: Algorithm to determine MBVQ type [2]

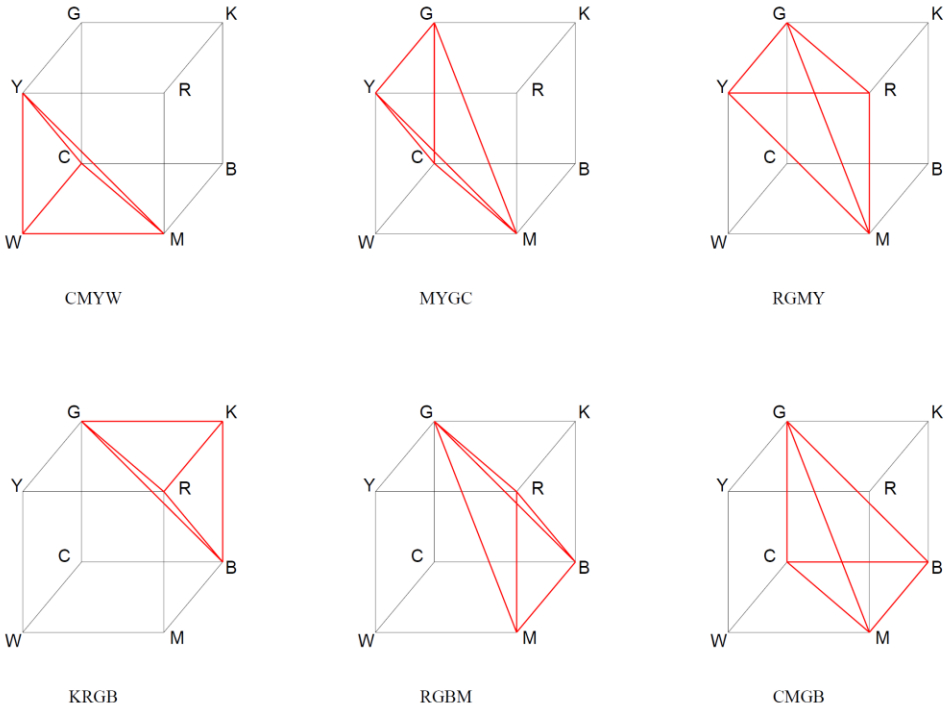
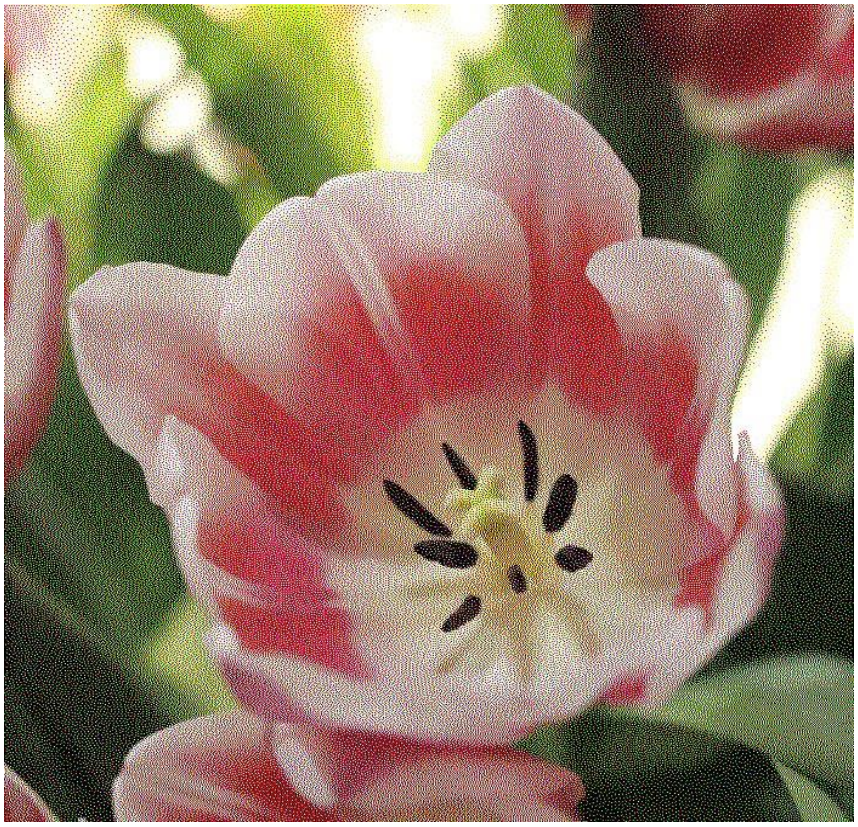


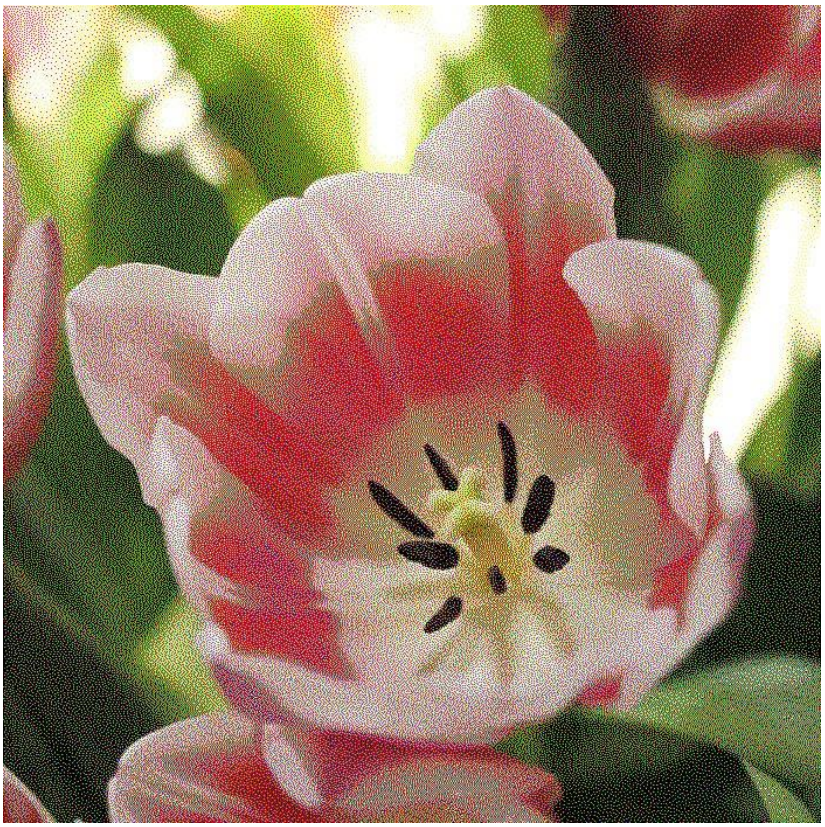
Figure 6 : MBVQ Types[2]

III. Experimental results:

(a) Original Flower Image



(b) Separable Error Diffusion



(c) MBVQ Based Error Diffusion

Figure 7: Separable and MBVQ baed error diffusion

IV. Discussion

Comparing the Separable diffusion image and MBVQ based diffusion image, we observe that the image quality is better in MBVQ Based Error Diffusion. The differences in the two methods are as follows:

In MBVQ method we select the best tetrahedron halftone set based on the halftone noise and this helps in deciding which color to render based on minimal brightness. If we reduce the brightness near the pixel and choose the nearest least bright pixel intensity in the tetrahedron, we can reduce halftone noise and hence artifacts.

In the Separable diffusion method we do not select any halftone set to minimize the halftone noise in the image, we merely perform error diffusion independently on the three color planes. This results in sharp undesirable contrasts and noisy artifacts.

Therefore, MBVQ caters to the human perception as it makes the image more uniform and hence reduces high frequency noise whereas separable error diffusion does not help in this regard.

Problem 3: Morphological Processing

(a) Shrinking
(b) Thinning
(c) Skeletonizing

I. Abstract and Motivation

Morphological processing involves structure manipulations of objects based on masking patterns. Mainly thresholding of images is done to obtain them in binary form and then non-linear operations are performed to transform the shape or morphology of the foreground objects with respect to the background. Fundamental operations that are carried out are erosion and dilation. In this section we are motivated to use pattern table masks in order to carry out shrinking, thinning and skeletonizing processes.

II. Approach and Procedures

The following steps describe the approach and procedures required to perform shrinking, thinning and skeletonizing:

1. Read the 8-bit image.
2. Perform thresholding by selecting the threshold as 127 and make the pixel intensities either 0 or 255 based on the threshold of 127. Pixel intensity > 127, pixel value = 255 or pixel intensity <= 127, pixel value = 0.
3. After this create a window coordinate map that considers the 8 neighbour positions around the foreground pixel.
4. Iterate through the image which is stored in a local buffer and when you find the foreground pixel, store all the pixel intensity values of the neighbours in the window pattern array based on the window coordinate map that is created.
5. Once the window pattern has all the values of the 8 neighbours, then search through the conditional maps of shrinking, thinning or skeletonizing based on which operation you want to perform. If the neighbourhood in the foreground stored in the window pattern matches the S, T or K conditional pattern map, mark it as 255 (1) else mark it as 0 and store it in another Buffer known as “MBuffer”. This is the stage 1 of the process.
6. In stage 2, we perform operations on the “Mbuffer”. We extract the neighbours of the foreground pixel and store it again in the window pattern. Now compare the window pattern value which is 8 bits of storage with the value present in the unconditional pattern map of S,Tor K depending on the operation used.
7. If the window pattern value matches with the value present in the unconditional pattern map of S,Tor K, mark it as 255 (or 1) else mark it as 0. Repeat this for all foreground pixel values and store the result in the output buffer.
8. Do a comparison between the Local Buffer which initially has the input image details with the output buffer. If the comparison results in a perfect match break from the iteration loop and write the output image to the file and analyse with ImageJ, else store

the output buffer pixel values into the Local Buffer and repeat steps 4, 5, 6, 7 & 8 for the required iterations.

9. After all these operations, we obtain the shrunk, thinned or skeletonized image.
10. For the shrinking case count all the stars once all iterations have reduced the stars to point values of 255 pixel intensity.
11. For star sizes and counts we store the stack size value in our depth search algorithm mentioned below in 3d and print out the max stack size value whenever we perform a foreground operation. We then count the similar sizes and plot the histogram for it.

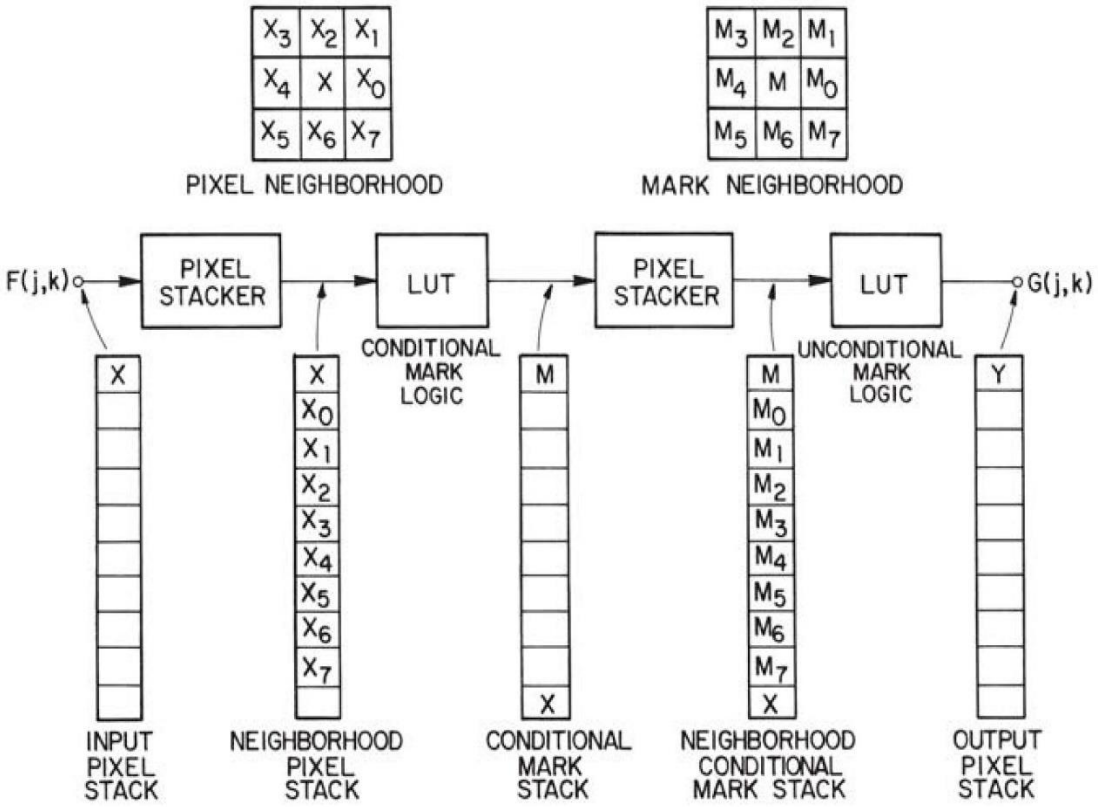
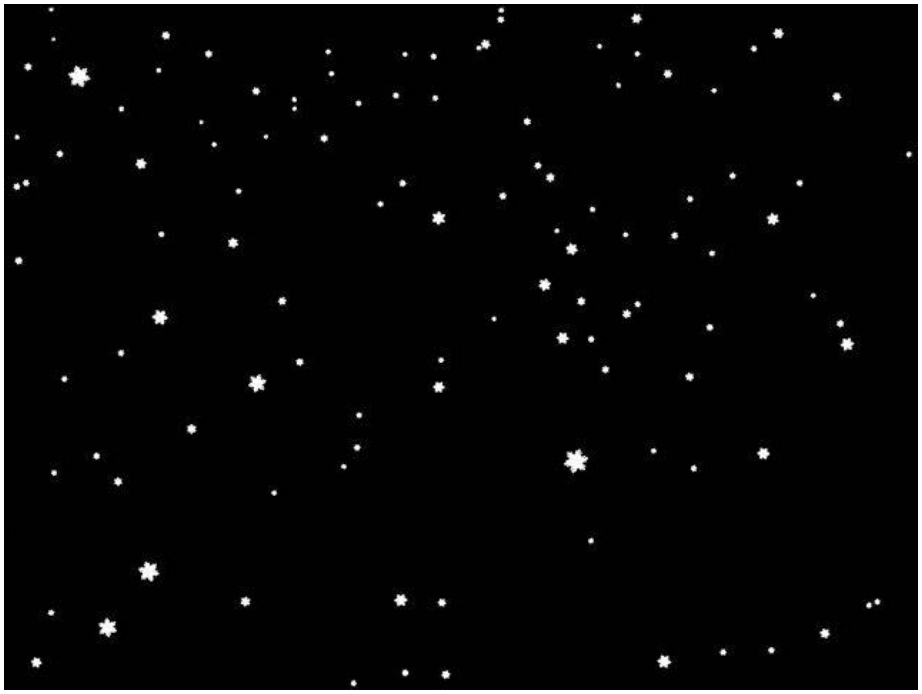
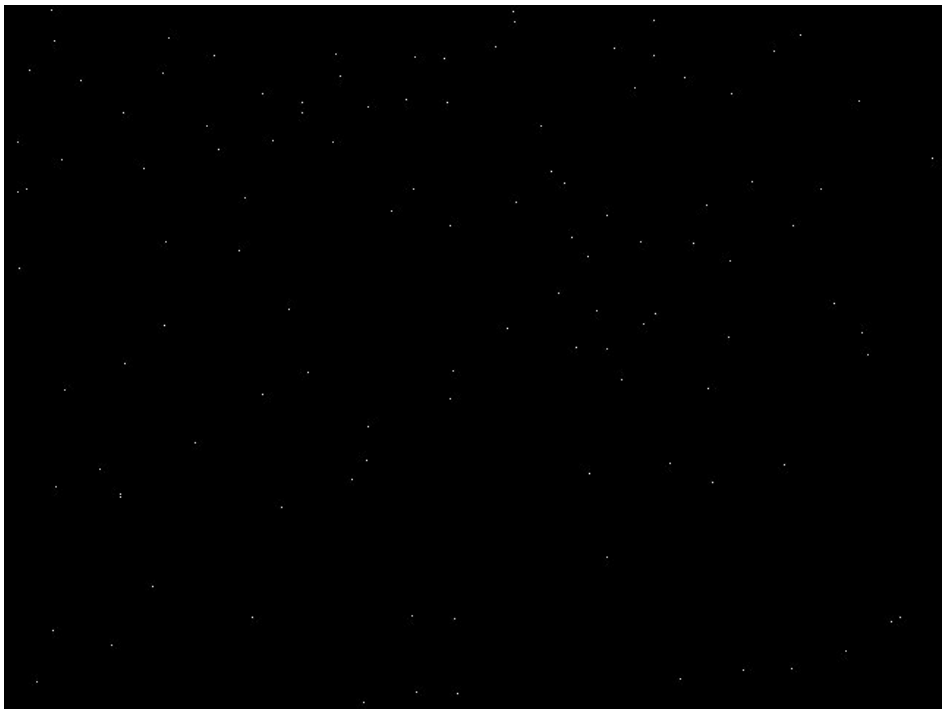


Figure 8 : Algorithm for STK[1]

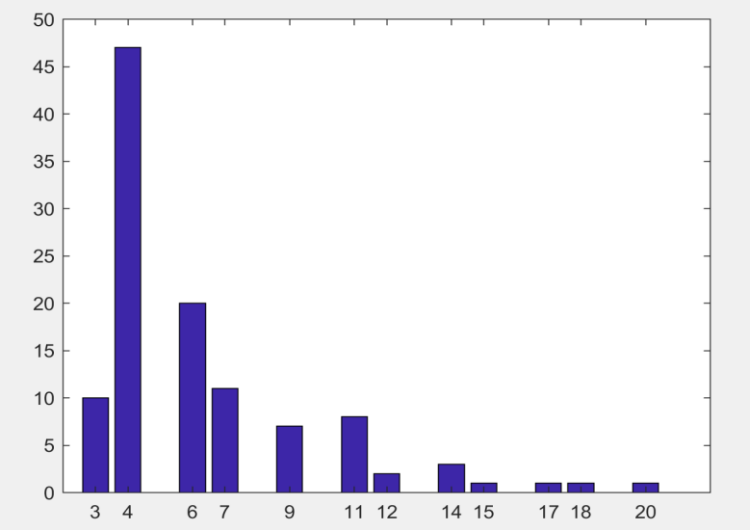
III. Experimental Results:



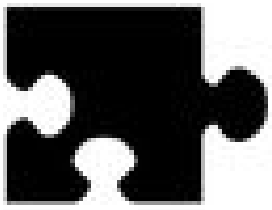
(a) Stars.raw (640x480)



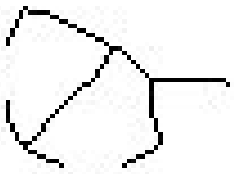
(b) Shrink operation on Stars.raw



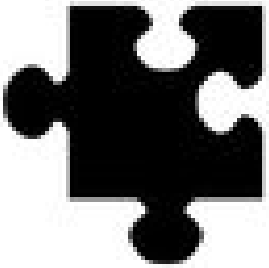
(c) Histogram , x-axis: star size vs y-axis: count



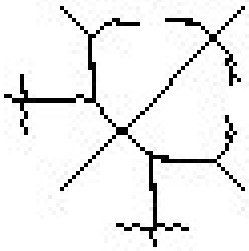
(d) Original Jigsaw1 image



(e) Thinned Jigsaw1 image



(f) Original Jigsaw2 image



(g) Skeletonized Jigsaw2 image

Figure 9: shrink, thin & skeletonize

IV. Discussion

(a) Shrinking

After implementing the shrinking filter, we observed that the stars of various sizes shrunk to single dots of pixel intensity 255. Thus, the algorithm used shows expected behaviour and using this the following results were found:

1. Total count of the stars: 112
2. Star Size: 3 number of stars : 10
   Star Size: 4 number of stars : 47
   Star Size: 6 number of stars : 20
   Star Size: 7 number of stars : 11
   Star Size: 9 number of stars : 7
   Star Size: 11 number of stars : 8
   Star Size: 12 number of stars : 2
   Star Size: 14 number of stars : 3
   Star Size: 15 number of stars : 1
   Star Size: 17 number of stars : 1
   Star Size: 18 number of stars : 1
   Star Size: 20 number of stars : 1

The histogram for the star sizes with respect to frequency in shown above in the experimental results.

(b) Thinning

After implementing the thinning filter to the jigsaw_1, we obtain the output shown in the experimental results. We observe that the thinned lines are minimally connected and equidistant from its nearest outer boundaries. Line breaks were observed when threshold was considered to be 127 gray value but after trial and error a threshold of 103 gray value produced no line breaks in the image thinning.

(c) Skeletonizing

After implementing the skeletonizing filter to the jigsaw_2, we obtain the output shown in the experimental results. We observe that we have produced the skeleton of the jigsaw piece that branches of at the edges. This behaviour is expected and hence we have obtained the right results. Line breaks were observed when threshold was considered to be 127 gray value but after trial and error a threshold of 104 gray value produced no line breaks in the image thinning.

3(d) Counting game

I. Abstract and Motivation

In this problem we are motivated to count the number of jigsaw's in the board image by using a certain algorithm called "**connected component labelling**" and also attempt to count the number of unique pieces using rotation, flipping and comparing by extracting segments of the image and using it for the unique counting process.

II. Approach and Procedures

Procedure used for counting total number of jigsaw's in the board:

1. Read the 8-bit board image.
2. Perform thresholding by selecting the threshold as 127 and make the pixel intensities either 0 or 255 based on the threshold of 127. Pixel intensity > 127, pixel value = 255 or pixel intensity <= 127, pixel value = 0.
3. Make a buffer called labelled image of the same size as that of the input image and set all it's pixel values to 255.
4. Start iterating through the image, set the connected component variable and labelled component variable to 0.
5. If the pixel value of the image is the foreground and if the labelled pixel value is not set (meaning its value is 255), increment the connected component variable and assign it to the labelled component variable. After this push it into a stack and then go to step number 7.
6. If it is a background pixel or if it is already labelled, then repeat 5 for the next pixel value in the image.
7. Pop the first element in the stack and traverse through its neighbours. If the neighbour is a foreground pixel and is not already labelled, assign it the current label value from the labelled component variable and add it to the stack and repeat this step till there are no other elements in the stack.
8. Then go back to 5 and for the next foreground pixel in the for loop repeat the process by incrementing connected component variable and then assign this incremented value to the labelled component variable.
9. After traversing in this fashion, we get the connected component variable count which gives us the total number of components present. Output this to the display.

Procedure used for counting unique jigsaw pieces:

1. 1. Read the 8-bit board image.
2. Perform thresholding by selecting the threshold as 127 and make the pixel intensities either 0 or 255 based on the threshold of 127. Pixel intensity > 127, pixel value = 255 or pixel intensity <= 127, pixel value = 0.
3. Segment out the board image into 16, 93*93 square images as 372/4 = 93. We should get 16 images through this.

4. Now run 2 loops that compares the original image with the other images. For example, the first image is compared with the other 15 images first by rotating the other image by 90, 180 and 270 and checking the matching percent by comparing pixel by pixel the value of the original jigsaw image with the other jigsaw images.
   The matching percent is stored in a variable called Max_Matching_Percent.
5. After rotation, flipping is performed on the compared image and then similarity is checked with the original image. the matching percent is calculated and then if the matching percent is higher than the matching percent in rotation we update the Max_Matching_Percent.
6. We check if the Max_Matching_percent is greater than 96.5% (threshold) , if the degree to which it matches is satisfied by the above condition we record count of matches as 1.
7. We repeat this process for the second original piece and 14 other pieces as we have already taken the first original jigsaw piece into account. This continues for all other pieces and computationally the comparison results in 120 comparisons.
   (15*(15+1)/ 2 = 120).
8. We find the total matches after this comparison and subtract by total number of jigsaws, this gives us number of unique pieces in the board image.
   No_unique_jigsaws = (total_jigsaws - total_matches).

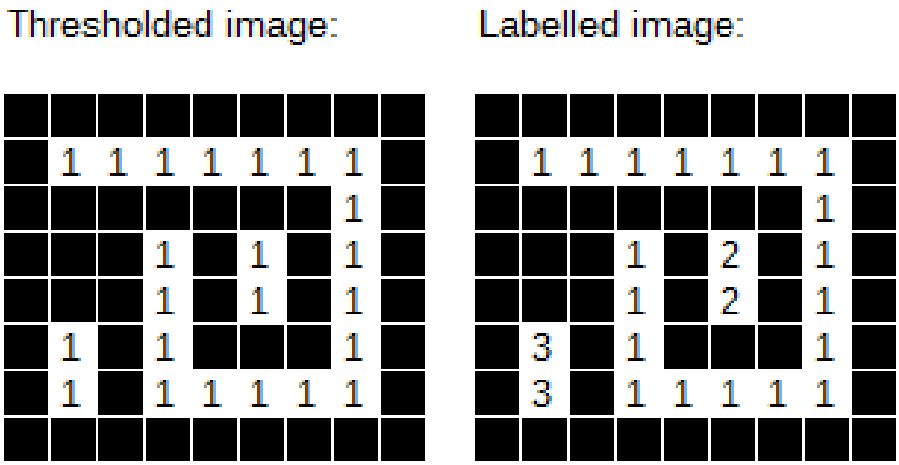


Figure 10: Connected Component Labelling [3]

III. Experimental Results:

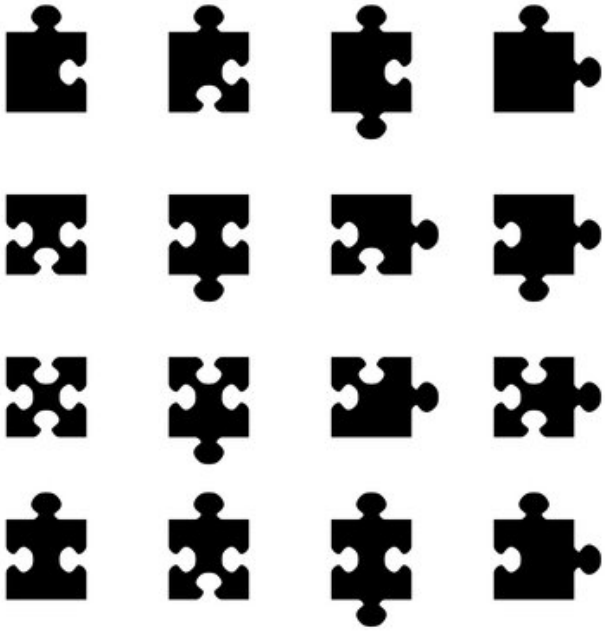


Figure 11: Board.raw (372x372)

From the above procedures we find that:

1. The total number of jigsaw's: 16
2. Number of unique jigsaw's: 10

IV. Discussion

To count the number of jigsaw's the connected component labelling was utilized and the procedure to perform this is mentioned above. With this procedure, the total number of jigsaw's was found to be 16. To count the number of unique jigsaw's operations like rotation and flipping was used and the detailed procedure is mentioned above. With this procedure the number of unique jigsaw's was found to be 10.

References:

[1] Digital Image Processing 4th Edition - William K Pratt

[2] "Color Diffusion: Error-Diffusion for Color Halftones", Doron Shaked*, Nur Arad†, Andrew Fitzhugh‡, Irwin Sobel†† HP Laboratories Israel* HPL-96-128(R.1) May, 1999

[3]https://www.codeproject.com/Articles/825200/An-Implementation-Of-The-Connected-Component-Label