HOMEWORK #2

Issued: 9/16/2016 **Due: 10/9/2016**

By: Hariprasad Ravishankar **USC ID: 8991379333**

Problem 1: Geometric Image Modification

1.1. Abstract and Motivation

Geometric Modification involves image processing techniques to obtain different effects on images such as warping, swirling etc. These effects are achieved by the cascaded effect of 3 basic operations, which are Rotation, Scaling and Translation. Geometric modification can also be used to project one image onto another image so as to modify the plane of the image. This technique is called Homographic Transformation as is used commonly during sports programs such as the Olympics to project the athlete's national flag on swimming pools, race tracks, etc.

In this problem, we perform geometric image modification on 2 kitten images to produce the effect of spatial warping (Diamond shape). We also solve the missing pieces of *Hillary* and *Trump* images. Further, we produce the effect of image projection by projecting the *Trojan* and *Tartan* image on the field.

1.2. Approach and Procedures

To produce the effect of image warping on Kitten_1 and Kitten_2 images, we first divide the input image into eight triangular regions. We then map these eight triangles to corresponding eight triangles in the desired output image. For eg.

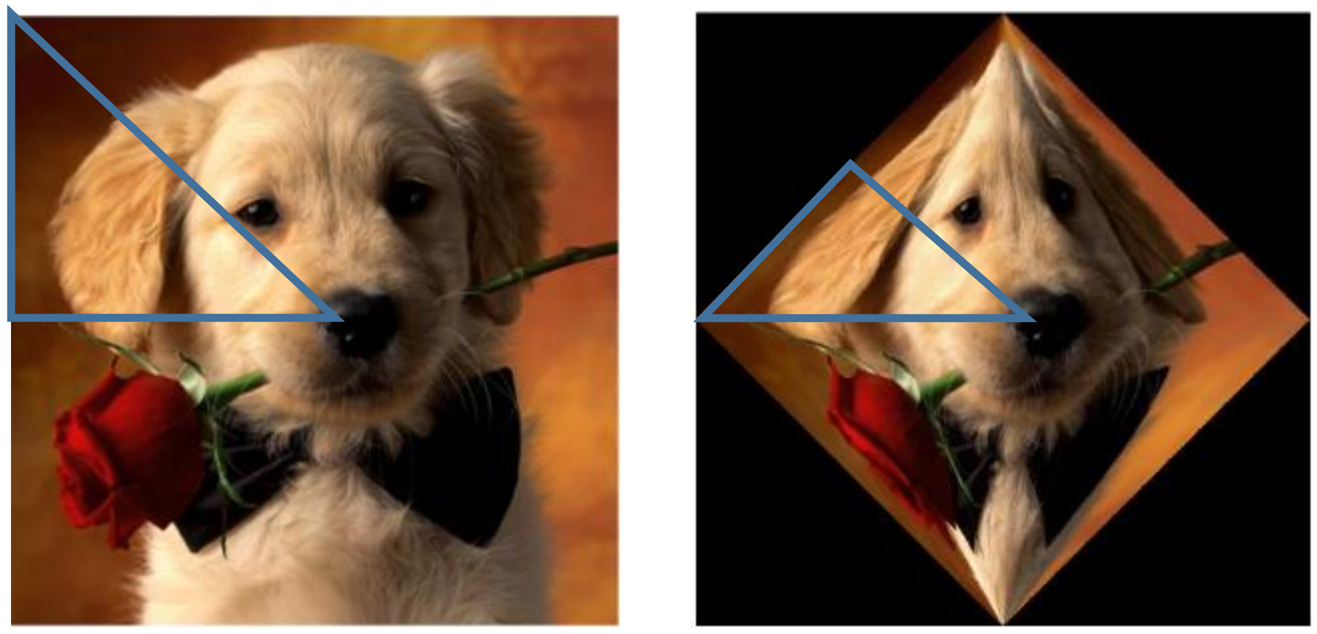


Figure 1: Triangle Mapping for Spatial Warping

Thus, for every triangle, 3 points from input image are mapped to 3 points in the output image. In this example, ***points A(0,0) , B(149,149) , C(149,0)*** in the input image, are mapped to points ***A'(74,74) , B'(149,149) , C(149,0)*** respectively.
Once, the points are mapped, the Linear equations for this mapping is solved. The linear transformation can be written as follows

$$\begin{matrix} x_2 \\ y_2 \\ 1 \end{matrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

Each pair of mapped points will give two equations, and thus 3 pairs of points will give 6 equations using which the 6 parameters can be solved.

MATLAB was used to solve the set of equations for the parameters using the command x = A\b

Here, the points x1, y1 were set to the points in the output image, i.e these were A', B', C' and points x_2, y_2 were set to points in the input image, i.e A, B, C. This directly gives the reverse mapping from output to input.

Bilinear interpolation is then performed to compute the interpolated value of the pixel from the input image that must be copied onto the output image. The formula for Bilinear interpolation is as follows

x and y needn't be integers as mapped values may be a fraction. Thus we can calculate the closest integer to x and y to be x' and y'. We then define parameters a and b as follows

a = x – x' and b = y – y'

Knowing parameters x', y', a and b we can estimate the value of pixel I using the formula.

$$I(i,j) = (1-b)[(i-a)F(x',y') + aF(x'+1,y')] + b[(1-a)F(x',y'+1) + aF(x'+1,y'+1)]$$

We then replace every pixel (i,j) inside the triangle in the output image with I(i,j) computed after interpolation. In order to identify if a pixel lies inside the triangular region mapped by the three coordinates, the line equation for every pair of points were computed, then every coordinate (i,j) is substituted into the line equation to decide which side of the line, the point lies. Only points that lie inside the triangular region are reversed mapped and interpolated from input image.

This procedure is repeated for all eight triangles to produce a spatially warped image.

Puzzle Matching

In the puzzle matching problem, the C++ program first renders the coordinates of the corners in Hillary and Trump image in clockwise order starting from the top-left corner. The program also outputs of the corners of the puzzle piece image provided in the ***piece.raw*** file.

For the *Hillary and Trump* images, the program scans the image and looks for the first 255 value. The first 255 value in all three channels, corresponds to the top left corner coordinate of the missing puzzle piece. Further, the last 255 value in all three channels corresponds to the top-right corner of the missing puzzle piece in the same row. Similarly, the bottom left and bottom right corner can be detected when the neighborhood has 3 non 255 valued R, G and B pixels around its 3 x 3 neighborhood.

In the *piece.raw* image, the program scans first row-wise, and then column wise.

During the row-wise scanning, the program checks for the first non 255 value on all three channels and puts the threshold such that the pixel value must be less than 200 to be detected as a corner. This row-wise scanning detects two corners in the piece image for Hillary and Trump. A set of flags is used to differentiate between *Trump and Hillary* images separated between multiple rows of white pixels. To detect the bottom corner, the program checks for the first row of white pixels after the detection of a non-white pixel in the previous row.

The column-wise scanning mimics the row-wise scanning algorithm and detects the first nonwhite pixel along the column while scanning from left to right. The other corner coordinate is computed when the first column of all white pixels is detected. Thus all four corner coordinates of both the Hillary and Trump puzzle pieces are obtained.

Once all the four corner coordinates for *Hillary and Trump* piece images and the original coordinates are obtained, a set of 8 linear equations are solved to compute 6 parameters that are the result of the cascaded effects of Translation, Scaling and Rotation of the puzzle pieces. The mapping matrix relation is shown below.

$$\begin{matrix} x_2 \\ y_2 \\ 1 \end{matrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

Here, the point $P(x_1,y_1)$ is chosen to be the corner coordinate in the original Hillary image and $P(x_2,y_2)$ is the corresponding mapped coordinate of the corner in the *piece* image. This avoids the inverse matrix computation. Thus every pair of mapped coordinates gives two equations. Any three mapped coordinate corners can be used to obtain the set of equations. MATLAB is used to solve the set of equations and obtain the parameters using the command x = A\b

Once all the parameters a1 to a6 are computed, the coordinates corresponding to a white pixel in all three channels are reverse mapped to coordinates in the *piece* image by multiplying with the transformation matrix. **Bilinear interpolation** is used to compute the pixel value from pixel image to be copied to the Hillary image at the mapped coordinate.

Since there exists a **white boundary** around the mapped image, a **median filter** is applied only to those pixels that lie on the boundary. A neighborhood size **of 3 x 3 was chosen for the median filter**.

Homographic Transformation

Homographic transformation is performed by projecting one image onto another image such that the image lies on another plane. To implement this four points were mapped from the *Trojan/Tart*ans image to four corresponding points on the *field.raw* image. The four points that were chosen in the *Trojan* and *Tartan* images were **A (16,12), B (15,341) , C(143,343) , D(144,12)**. These points were mapped to the corresponding points **A' (554,411), B' (447,542) , C'(451,600), D'(546,516)** in the field image where the *Trojan/Tartan* images have to projected onto.

One these points were obtained, they were substituted in the projection formula

$$P_2 = HP_1$$

Where H is a 3 x 3 transformation matrix, points P_1 and P_2 are points before and after projection.

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

Thus, 4 pairs of mapped coordinates will give 8 equations to solve 9 variables. We know $H_{33} = 1$. Thus we use 8 equations to solve for 8 variables using MATLAB. The equations can be rearranged in the form of Ax = b as follows **[1].**

Let small x and y denote P_1 and Capital X and Y denote P_2.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & y_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ 0 & 0 & 0 & y_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 \\ 0 & 0 & 0 & y_3 & y_3 & 1 & -x_3Y_3 & -y_3Y_3 \\ 0 & 0 & 0 & y_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}$$

Thus, x is computed as A\b. Once all 8 parameters are obtained, every point inside the Trapezoid in the field image is reverse mapped to the Trojan image, by multiplying with the inverse of the Transformation matrix is follows.

Let P_1 now be a point in the Field image and P_2 be its projected point on Logo image. H^{-1} is computed using MATLAB

$$P_2 = H^{-1}P_1$$

Since the points P_2 can be floating point number, Bilinear interpolation is used to compute the Pixel value representing P_2. The pixel Value is then copied to location P_1 in the Field Image.

Further, the interpolated value is checked so that it does not equal to 0 or 255 to avoid copying the background for both Trojan and Tartan images.

1.3. Experimental Results

Shown below are the Kitten_1 and Kitten_2 warped images

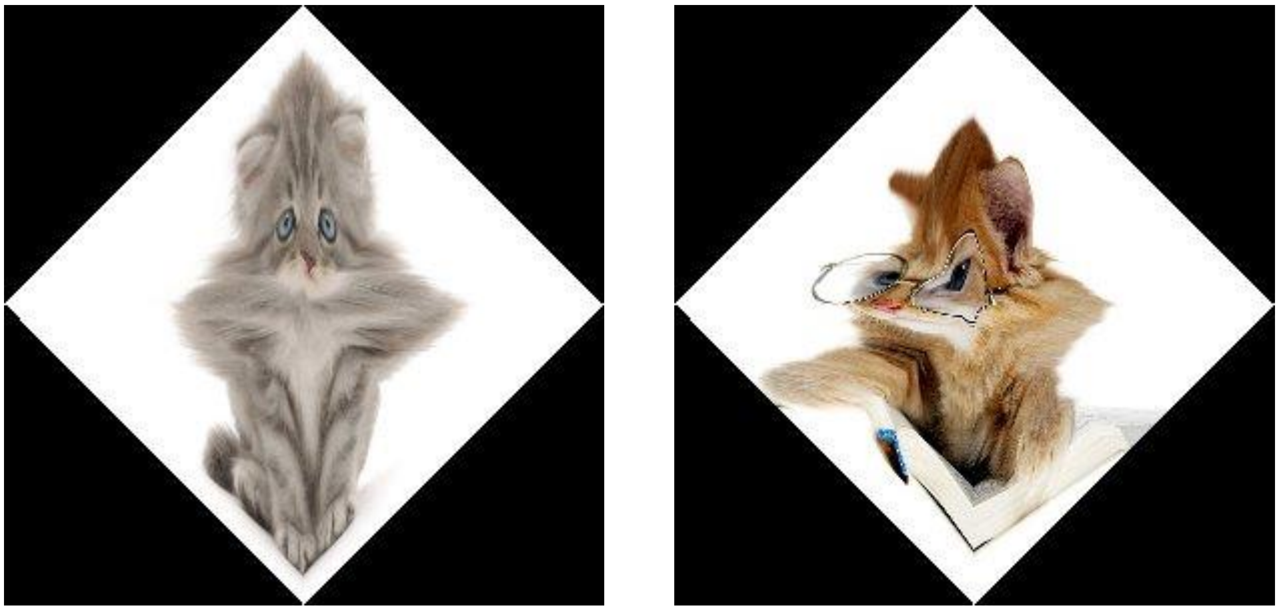
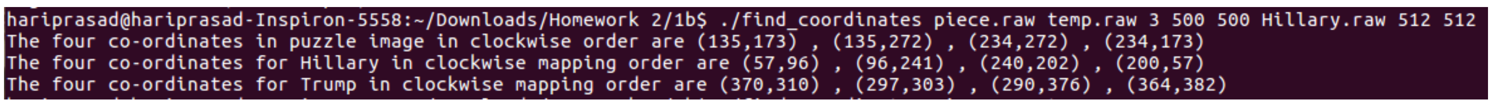


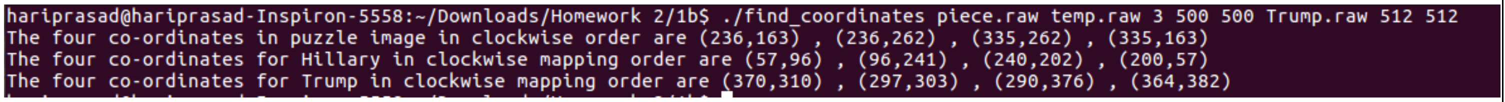
Figure 2: Kitten_1 and Kitten_2 warped images

Problem 1)b Puzzle Matching

Shown below is the terminal screen output to determine the coordinates for *Hillary*, *Trump* and piece images



```
hariprasad@hariprasad-Inspiron-5558:~/Downloads/Homework 2/1b$ ./find_coordinates piece.raw temp.raw 3 500 500 Hillary.raw 512 512
The four co-ordinates in puzzle image in clockwise order are (135,173) , (135,272) , (234,272) , (234,173)
The four co-ordinates for Hillary in clockwise mapping order are (57,96) , (96,241) , (240,202) , (200,57)
The four co-ordinates for Trump in clockwise mapping order are (370,310) , (297,303) , (290,376) , (364,382)
```



```
hariprasad@hariprasad-Inspiron-5558:~/Downloads/Homework 2/1b$ ./find_coordinates piece.raw temp.raw 3 500 500 Trump.raw 512 512
The four co-ordinates in puzzle image in clockwise order are (236,163) , (236,262) , (335,262) , (335,163)
The four co-ordinates for Hillary in clockwise mapping order are (57,96) , (96,241) , (240,202) , (200,57)
The four co-ordinates for Trump in clockwise mapping order are (370,310) , (297,303) , (290,376) , (364,382)
```

Figure 3: Terminal screen output of the Coordinates computed for Trump, Hillary and Piece images

Shown below are the matched puzzle images of *Hillary* and *Trump* images.

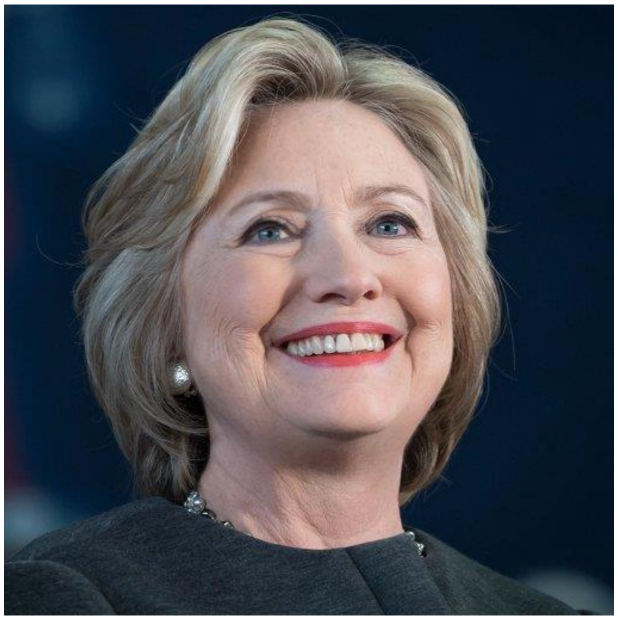
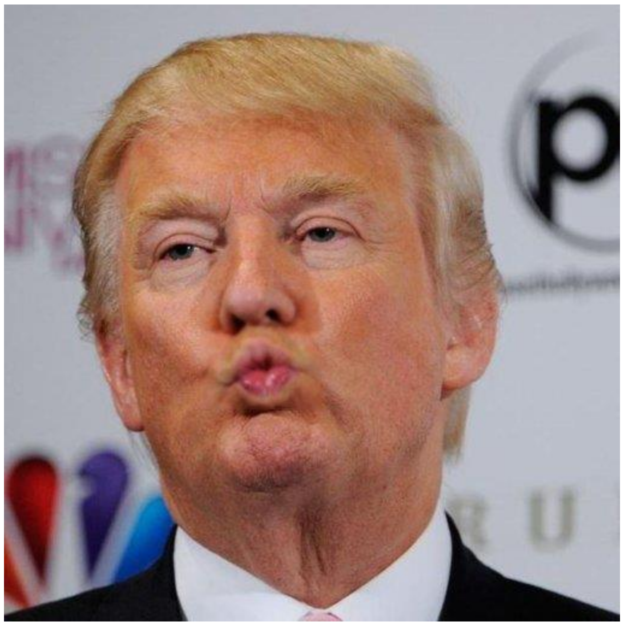
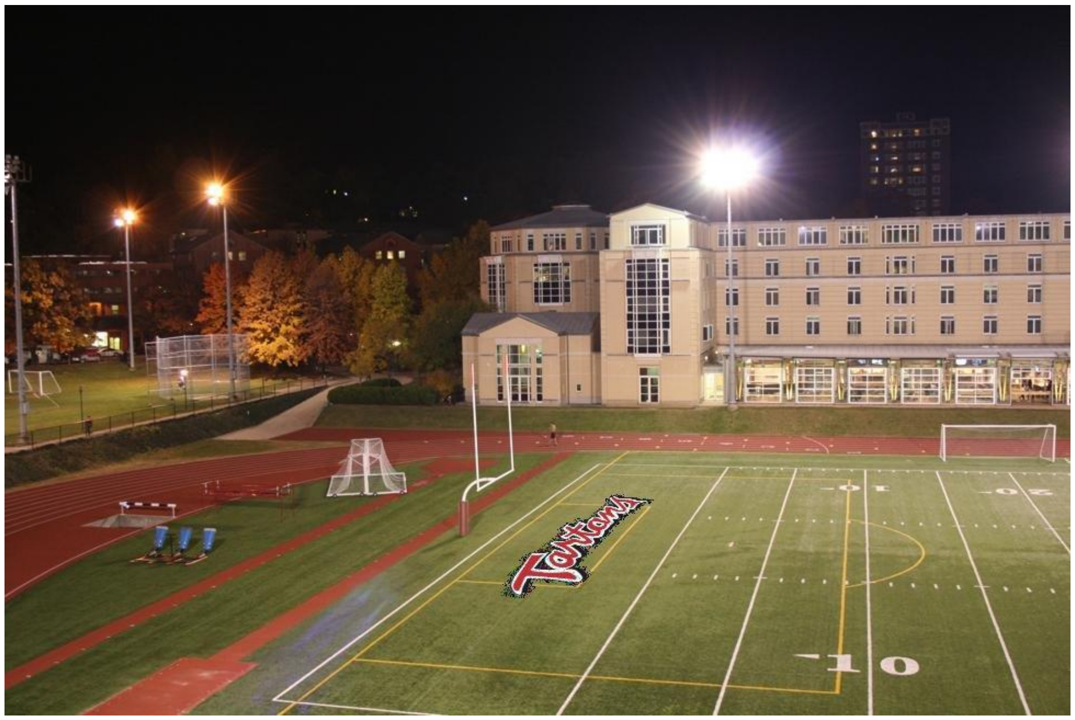



Figure 4: Puzzle Matching of Hillary and Trump Images

Problem 1)C Homographic Transformation

Shown below are the projected images of *Tartans* and *Trojans* Images



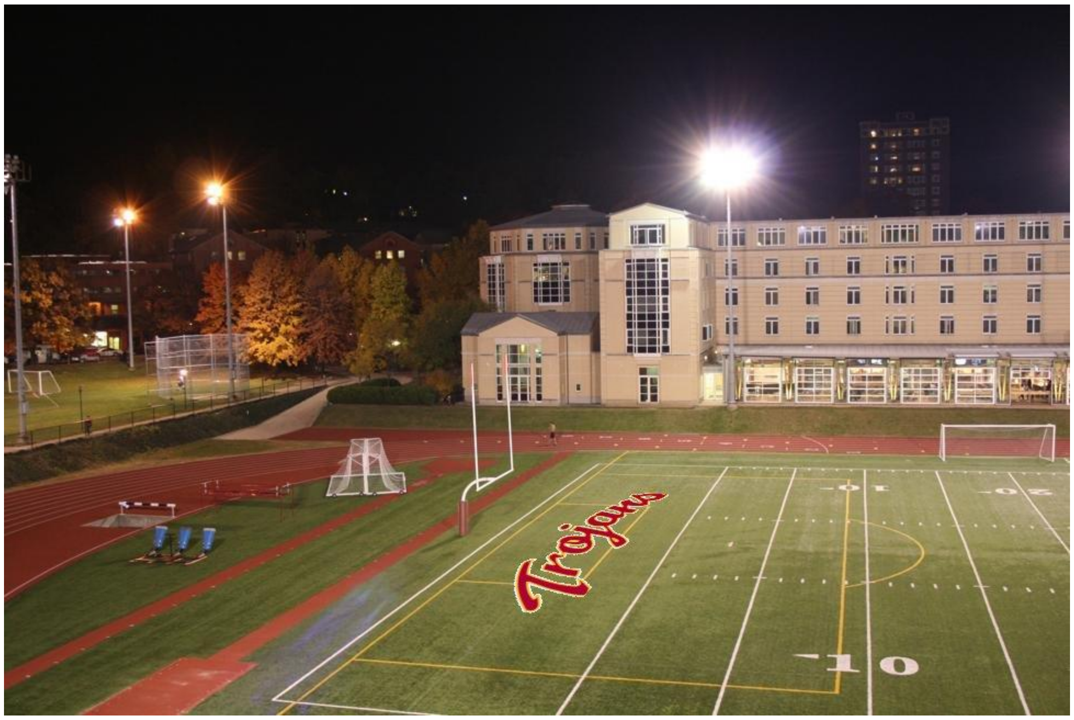


Figure 5: Homographic Projection of *Tartans and Trojans* Images onto *Field* image

1.4. Results and Discussion

Figure 2 shows the output on applying the spatial warping procedure on *Kitten_1* and *Kitten_2* images. It can be noticed that the warping distorts the original images to transform it into a diamond shape.

Figure 3 shows the results from the puzzle matching operation. It can be seen that both the puzzle images have been perfectly been placed in the missing space. The median filter blurs the boundary between the piece and the original image giving it a natural effect. The effect of the interpolation can be noticed in the *Trump* image as the region shows a significant blurring as compared to the rest of the image. The median filter was chosen instead of mean filter because intuitively the number of stray boundary pixels can be maximum of 2 and thus effectively be removed as it will not appear as the median inside a neighborhood of size 3 x 3.

Figure 4 shows the Homographic Transformation images where the *Trojan* and *Tartans* logos are overlaid on the field. It can be noticed that while the previous geometrical operations preserved the plane of the image, Homographic transformation help project the logo image onto a different plane. Further the white and the black backgrounds of the *Trojan* and *Tartans* images have not been projected giving the effect a natural tone.

Problem 2: Digital Halftoning

2.1. Abstract and Motivation

Digital Halftoning is a technique to represent an 8 bit grayscale or a 24 bit colour image in terms of only 1 bit (0 or 1) such that very little loss in human perception is observed. Many techniques such as Dithering and Error – Diffusion have been implemented to bypass the poor performance observed by naïve hard thresholding methods. Halftoning is extensively used in the print media to print images using just black and white ink colours. Obtaining a binary image using a hard threshold such as 128 often gives a poor representation of the variations in the image and fails to identify contours and features.

Dithering and Error diffusion algorithms simulate a continuous grayscale or colour transition in an image using only a single bit for every pixel [2]. In this problem, the 8-bit House image is converted to a 1 -bit Black and White image using Dithering Matrix approach and Error -Diffusion technique. The outputs of various methods have been compared and discussed in section 2.4.

2.2. Approach and Procedures

Dithering

The concept of dithering stems from the drawbacks of setting a hard threshold. Values closer to the threshold are abruptly truncated down to 0 or 255. In the ideal case, we would like to push approximately half of the values around the threshold above and below the threshold such that there is an even distribution of pixels. This is achieved by adding a randomized white noise [2] to the pixels in such a way that the quantization error is randomized and the relative colour patterns in the image are preserved.

The Dithering matrix method uses multiple thresholds at different pixel locations determined by the Bayer Index matrix. The formula for Bayer index matrix is as follows

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

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

Using the Bayer Index Matrix, a Threshold Matrix is obtained by the formula

$$T(x,y) = \frac{I(x,y) + 0.5}{N^2} * 255$$

Thus any pixel at F(i,j) is set to 1 if its value exceeds the Threshold, else it is set to 0.

The program accepts input as N which is the dimension of the Index matrix. It then computes the values of the Index Matrix through recursive function.

In every iteration, the function to generate the index matrix is called with parameters as the initial 2D index matrix and the N value which grows as a multiple of 2.

Every pixel is compared with a Threshold value and set to 1 or 0 to generate a binary dithered image.

4 Gray Intensity Level Dithering

In order to generate an image that has 4 intensity levels, The Threshold Matrix formula is generated at four different intensity levels, 0, 85, 170, 255

$$T(x,y) = \frac{I(x,y) + 0.5}{N^2} \quad * \mathit{Intensity\ Level}$$

Thus for every pixel location, the pixel level is compared with for 3 different Threshold Matrix location values, and the pixel values within one for 4 regions, i.e $0<p\leq 85$, $85 < p \leq 170$, $170 < p \leq 255$, $p > 255$. These are then truncated to generate the new dithered output.

Error – Diffusion Method

Error – Diffusion based Halftoning is a feedback procedure, where the quantization error is diffused or added to the pixels around the quantized pixels which are yet to be processed [3]. This way the pixels that may have been truncated to 0 might be changed to have a value slightly above the threshold due to the diffused error and thus be truncated to 1.

The Floyd- Steinberg error diffusion matrix determines the share of the quantization error that the neighboring pixel gets.

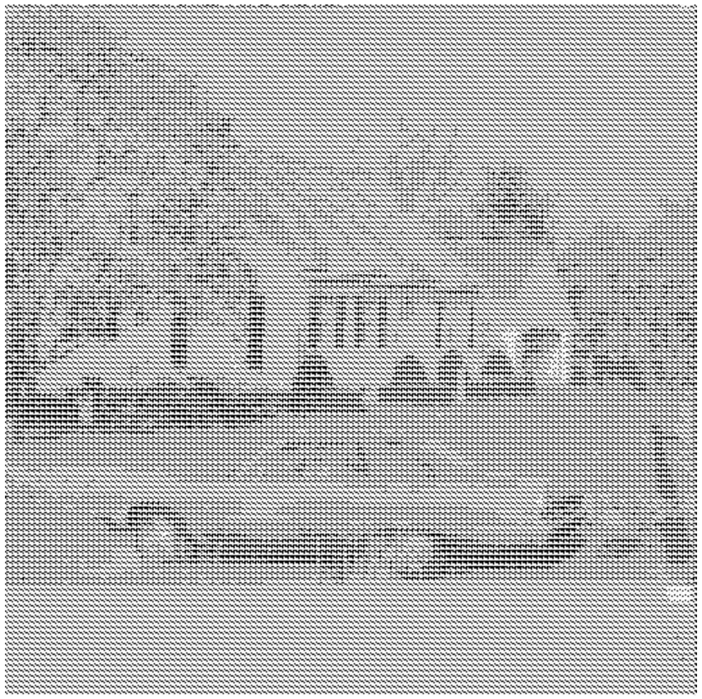
$$\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

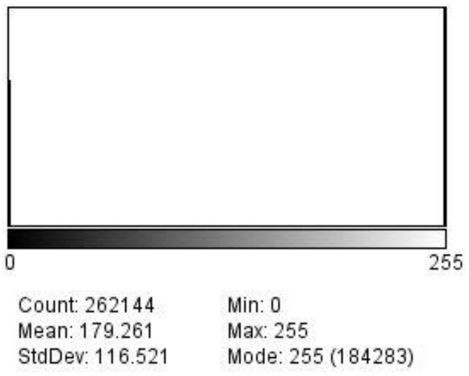
In order to avoid the buildup of residual error on the right edge of the image, **Serpentine** scanning is implemented where **odd rows** use the aforementioned matrix and **even rows** use the following formula

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

This procedure is repeated for different Diffusion matrices such as one proposed by Jarvis, Judice, and Ninke (JJN), and Stucki.

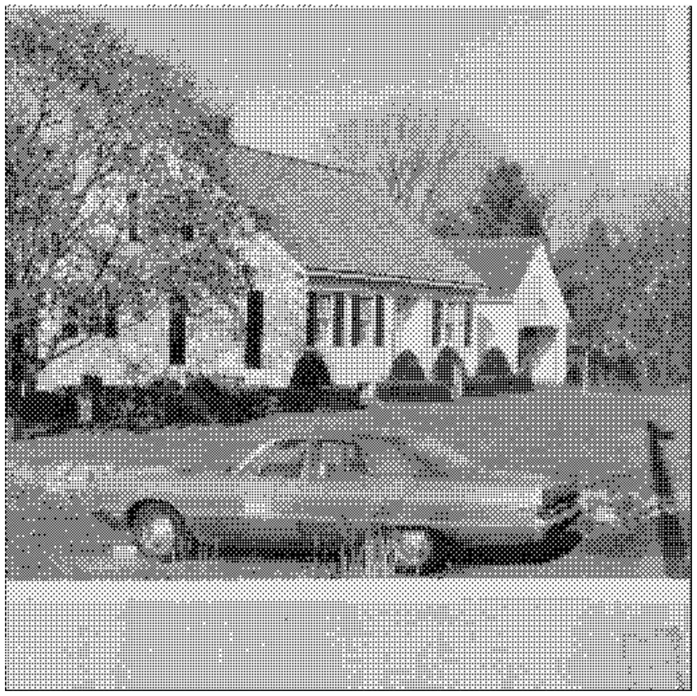
2.3. Experimental Results

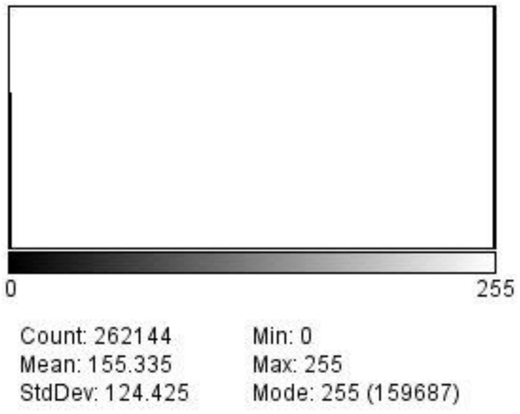




```
Count: 262144      Min: 0
Mean: 179.261      Max: 255
StdDev: 116.521    Mode: 255 (184283)
```

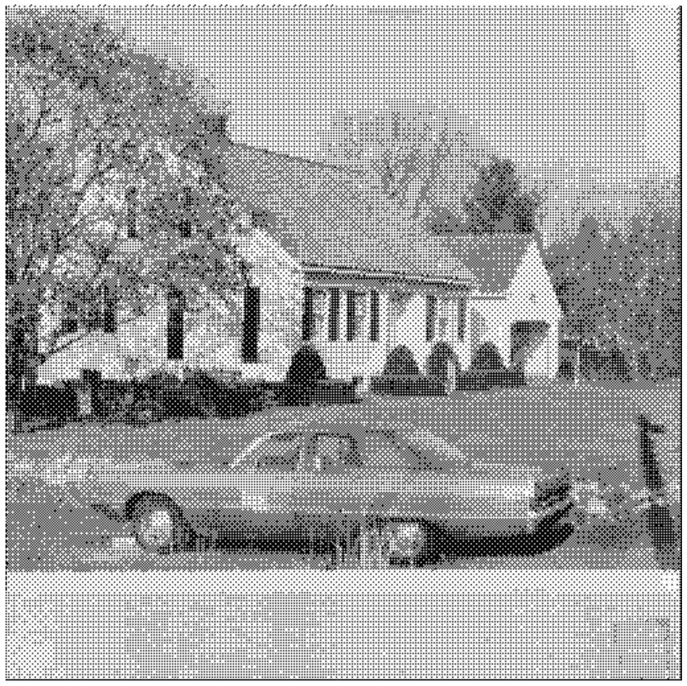
Figure 6: Dithered Image and its Histogram while using $I_2(i,j)$ Index Matrix

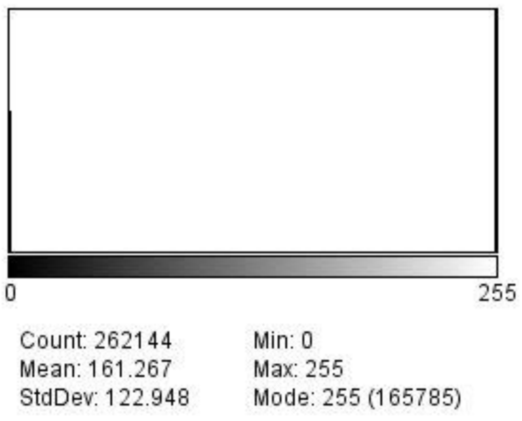




```
Count: 262144      Min: 0
Mean: 155.335      Max: 255
StdDev: 124.425    Mode: 255 (159687)
```

Figure 7: Dithered Image and its Histogram while using $I_4(i,j)$ Index Matrix





```
Count: 262144      Min: 0
Mean: 161.267      Max: 255
StdDev: 122.948    Mode: 255 (165785)
```

Figure 8: Dithered Image and its Histogram while using $I_8(i,j)$ Index Matrix

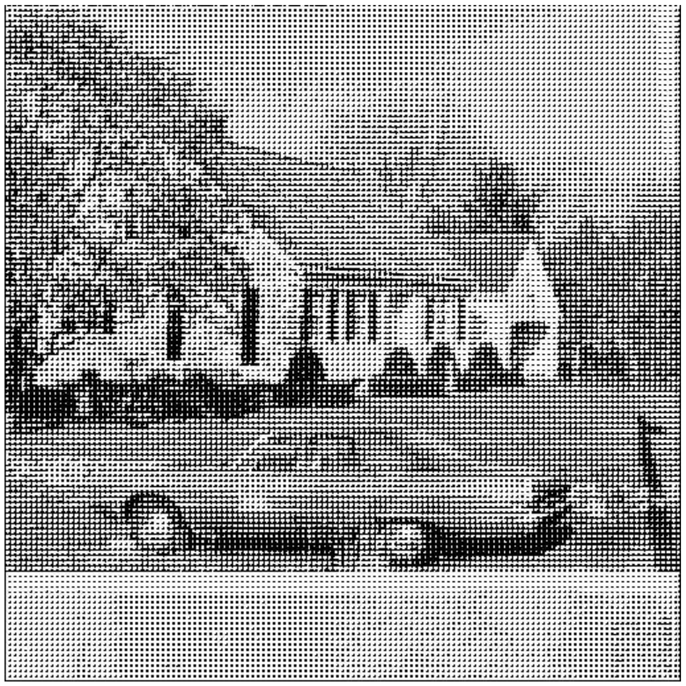
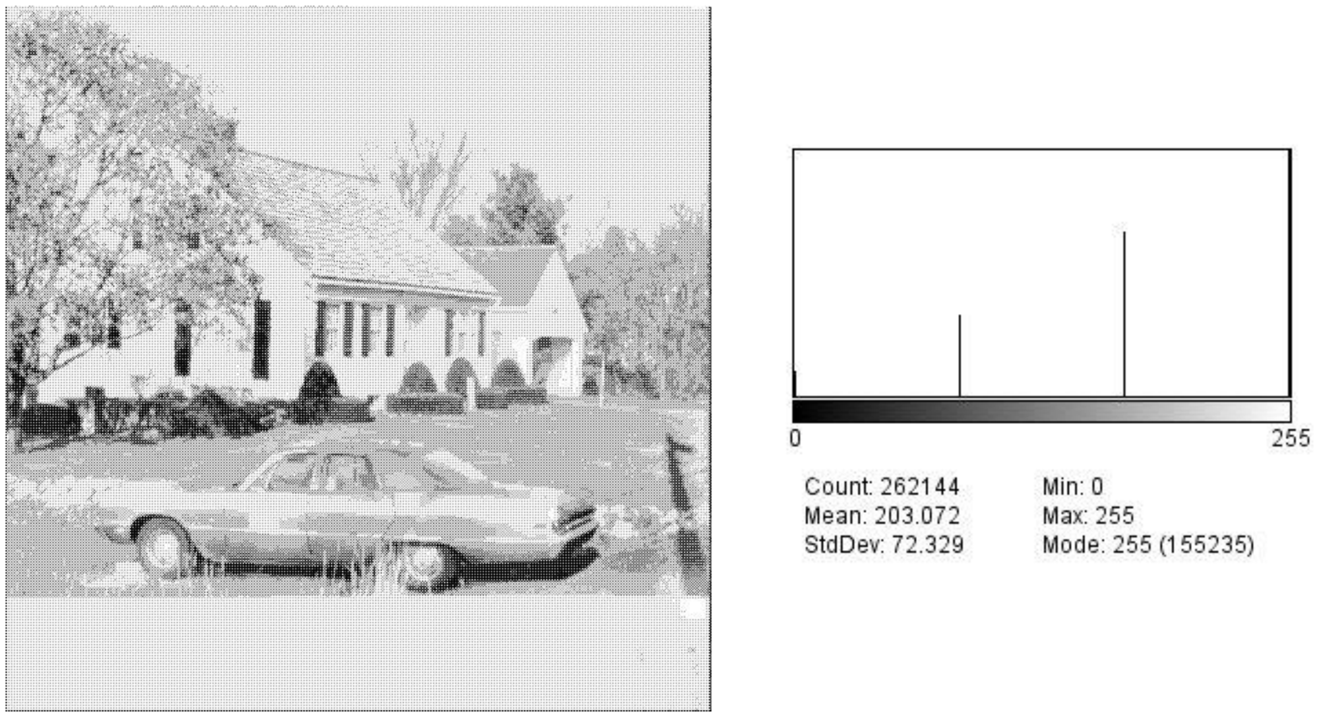


Figure 9: Dithered Image while using $A_4(i,j)$ Index Matrix



```
Count: 262144        Min: 0
Mean: 203.072        Max: 255
StdDev: 72.329       Mode: 255 (155235)
```

Figure 10: Dithered Image having 4 Gray Intensity Levels

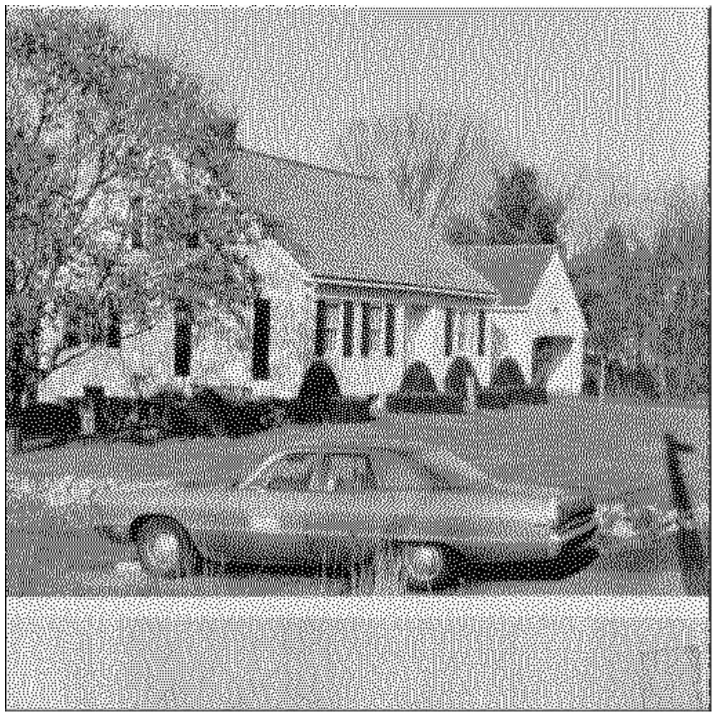


Figure 11: Floyd- Steinberg Error Diffusion

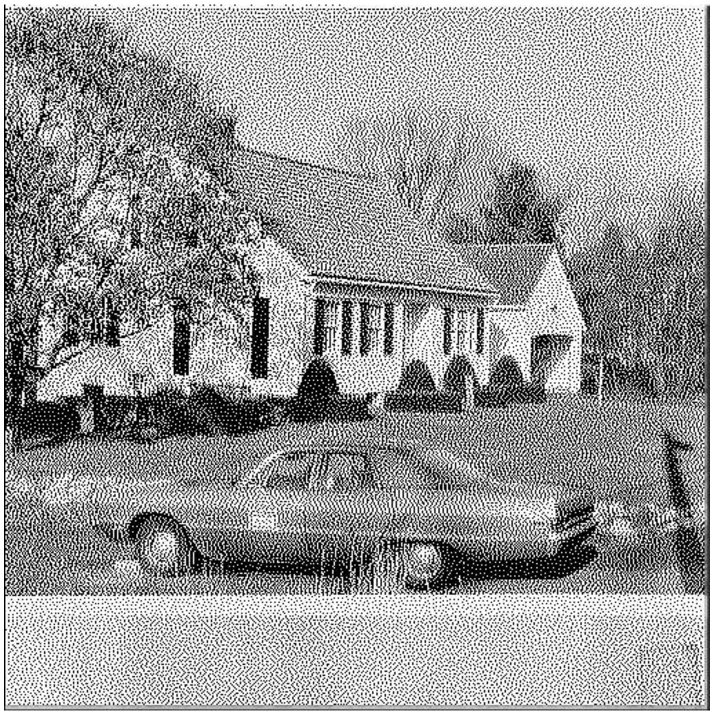


Figure 12: Jarvis, Judice, and Ninke (JJN) Error Diffusion

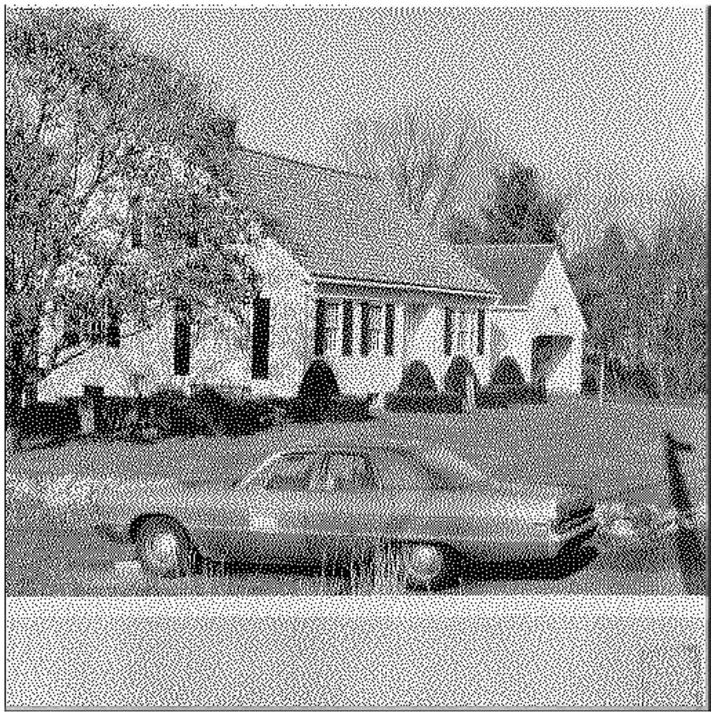


Figure 13: Stucki Error Diffusion

2.4. Results and Discussions

Figure 6 shows the dithered output while using a Bayer Index matrix of size N = 2. It can be seen that the image constructs the patterns of the House image but does not perform well for visual perception as the image appears bright with poor contrast. This might be because the 2 x 2 index matrix has very small values. Thus more pixels in the House image are truncated to 255 giving it a higher density of white pixels than dark pixels. This is evident in the histogram ***figure 6 b)*** of the Dithered image obtained using ImageJ

Figure 7 and 8 show the dithered output obtained when a Bayer Index Matrix of size 4 and 8 are chosen respectively. As both these matrices contribute to larger values in the Threshold Matrix, the concentration of light (255) values are reduced as compared to the Figure 6, giving a better contrasted and clearer image. This can be seen in the histogram of the two images where the number of pixels with value 255 are lower as compared to choosing a lower sized Bayer Index Matrix.

Further, from the histograms of ***figure 7 and 8***, it can be seen that the Bayer Index Matrix of size 8 increases the number of light pixels. The greater density of dark pixels in Figure 7 contribute to artificial features around the sky and around the trees, whereas the original Grayscale image does not contain these patterns. Thus the Bayer Index Matrix of size N = 8 performs better and would be the optimum choice for the Dithering process.

Figure 9 shows the Dithered output obtained when the Index Matrix $A_4(i, j)$ is chosen. As compared to the output from choosing $I_4(i , j)$ in Figure 7, the output in Figure 9 performs poorly, contributing to horizontal and vertical grid lines in the image. The reason for the horizontal and vertical grid lines is due to the lower values present along Rows 2 and 3. Thus pixels that align along rows 2 and 3 are more likely to be truncated to 255, contributing to grid like pattern of dark and light bands.

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

Figure 10 shows the Dithered output having 4 Gray intensity levels. This is obtained by having 4 Threshold matrices for each intensity level. Every pixel is then identified to lie with a range specified by the threshold matrix, and truncated to 1 of the four intensity level. Leveraging 4 intensity levels removes the dotted effect that was prevalent when Dithering to just 1 gray levels. The image appears clear and performs well in reconstructing the patterns in the original image.

Error Diffusion

Figure 11 shows the dithered output using Floyd Steinberg Error Diffusion matrix. The Error Diffusion method performs better than the Dithering Matrix based method. As explained in section 2.3. the error diffusion method diffuses the quantization error to the neighboring unprocessed pixels determined by the matrix. The first pixel is quantized to either 0 or 255 by hard thresholding at 127 value. The quantization error between the original pixel value and its value after thresholding is diffused to the neighborhood pixels.

In Floyd-Steinberg Error diffusion, the output is fine grained as the diffusion matrix is only 3 x 3 centered at every pixel.

Figure 12 shows the dithered output using Jarvis, Judice, and Ninke dithering. The JJN dithering has a 5 x 5 dithering matrix and thus more pixels around the neighborhood share a smaller share of the quantization error. In comparison to Figure 11, the JJN output produces a coarse image as the image is less fine grained.

Figure 13 shows the dithered output while using Stucki Dithering. Like the JJN dithering, the Stucki Dithering also produces a coarser image as it has a 5 x 5 error diffusion matrix. However, the image appears to be slightly more clear than the JJN error diffusion method.

Problem 3: Morphological Processing

3.1. Abstract and Motivation

Morphological operations such as Shrinking, Thinning and Skeletonizing is used to extract useful information from images. Operations such as Shrinking helps us identify the number of unique objects (say) in a Binary Image. Thinning and Skeletonizing also give useful information to identify and classify information.

In this problem, Morphological Operations are implemented on Rice image to identify and classify rice grains based on length and area. Further, Thinning and Skeletonizing operations have been implemented on Butterfly and Fly images along with Hole filling operation, to extract features such as the number of endpoints, T-junctions etc. that give us information to classify a Test shape such as the Probe image.

3.2. Approach and Procedures

a) Rice Grain Inspection

Binarization:

The first operation implemented on the Rice colour image is binarization. The following procedure was followed in order to obtain the Binary Mask for the Rice image

(1) A 3 x 3 Median filter was first applied to each channel (R, G, B) separately. This is performed to remove noisy stray pixels in the background that differ from the true background pixel value.
(2) The 3 x 3 Median filter to recursively applied three times to produce a cascaded effect.
(3) Thresholding based on Channel Mode.
  (a) In this stage, the mode of each channel was estimated. The mode of each channel will represent the background pixel value as it is the majority.
  (b) An error of 6 pixels was chosen by hit and try, to include pixels around the mode that might still represent the background.
  (c) For each channel, values in between mode – error ≥ pixel ≤ mode + error, were truncated to 0 and others were truncated to 255. Thus background is considered black and object as white.
(4) RGB to Grayscale conversion.

RGB to Grayscale conversion was performed using the following formula [4]
intensity = 0.2989*red + 0.5870*green + 0.1140*blue

A hard threshold of 100 was chosen to construct a binary image.

(5) A 5 x 5 Median Filter was applied to the binary image to remove stray white pixels.

Thus a perfect binary mask of the grains was obtained on which the Morphological Processing was performed.

Count the number of grains:

Shrinking:

The object shrinking operation was implemented as follows.

i) A header file named “Shrink.h” is created that stores the list of all stage 1 and stage 2, 3 x 3 binary masks in the form of a 2 dimensional vector. The masks are row-wise, where each row contains 9 integers obtained while scanning in raster scan approach.

For eg. The Matrix

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

Is stored as {0,0,1,0,1,0,0,0}

Thus 58 masks for Stage 1 Hit or Miss conditional Filter and 332 Unconditional Masks were obtained.

ii) Each row of the 2 dimensional vector was converted to its equivalent integer representation. Since each mask is unique, we obtain unique integer values for every row.

iii) Further, a Hashmap is maintained where Keys represent the integer conversion of every binary mask, and value is stored as 1.

iv) In the stage 1, every 3 x 3 block of the binary image is converted to its equivalent integer representation by passing it to binary2int function.

v) The value is used as key and is compared with the hashmap to check if the value has been set to 1. If the value is set to 1, it represents a match with a mask.
   (1) If true, then the value 1 is placed at the centre of the 3 x 3 window.
   (2) Else, value 0 is placed at the centre of the 3 x 4 window.

vi) In the Stage 2, the intermediary M matrix is used and a 3 x 3 window with centre 1 is converted to its equivalent integer representation by passing to binary2int function.
   (1) The integer value is used as key to check if the second stage hashmap is set to 1 for that particular integer value.
   (2) If there is a miss, the centre of the 3 x3 window is replaced with a 0 in the G matrix. Else, it remains unchanged.

vii) Steps iv) to vi) are repeated till there is no change in two iterations in the G matrix.

Once, shrinking is completed on the Rice_Binary image, every grain is reduced to a single pixel with value 255. To count the number of Rice_grains, the total number of white pixels in the Shrunk image is counted.

To compare the Size of the Rice Grains, Thinning operation is performed on each type of the Rice grain.

The coordinates are precomputed using ImageJ and the size of the individually cropped Rice images are fixed at 58 x 45 Binary images.

Thinning procedure is followed as follows.

Thinning:

The same procedure that was followed for Shrinking is followed, except a header filer "Thinning.h" is included that stores 2, 2 Dimensional vectors for Stage 1 and Stage 2 binary masks.

After Thinning operation is performed for each type of rice grain, the number of white pixels gives the relative length of each rice grain.

Categorizing Rice Grains

Area and Length of each type of rice grain are chosen as descriptors for each type for categorization.

Area is computed by counting the number of white pixels in the cropped binary images of the grains.

The Rice grains are categorized by $\frac{Area\ of\ grain}{Length\ of\ grain}$. Grains with small length but large area due to the width will have high Area/Length Ratio. Grains that are narrow will have high length, but small area and thus the Area/Length Ratio will be small. std::sort() is used for sorting grains based on Length and Area.

b) MPEG 7 Dataset

Hole Filling:

For the given Butterfly, Fly and Probe Images, the holes are first filled with white pixels by implementing the Hole Filling algorithm.

i) The C++ program accepts a Binary images and switches the background and the object pixels. This is done by replacing 1s with 0s and 0s with 1s respectively.

ii) Now, the background and Holes are treated as objects, and the Shrinking procedure is performed.

iii) This the background and the holes are shrunk to a single white pixel.

iv) Disregarding the Shrunk, background pixel (which appears on the 5-pixel top edge) we proceed by applied the following formula.

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

Where B = $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ and X_0 = p where p is the location of the Shrunk hole.

i.e We check the overlap of every 3 x 3 window in the initial Shrunk binary image with B, and replace the centre of the 3 x 3 window with 1 if an overlap exists.

v) The intermediary matrix is then compared with the complement of the of A to check for intersection. (and operator). If true, 1 is placed that location, else 0

vi) This process is repeated recursively till $X_{k-1} = X_k$

vii) Once the iteration ends, the obtained matrix is added to the Original Image matrix to compute the union.

One the hole filling is completed, a 5 x 5 Median filter is applied to smoothen the contours and remove any holes that could not be removed by the Hole Filling Algorithm.

Once the Butterfly, Fly and Probe images are successfully filled, the thinning and skeletonize procedure is applied to both images the Thinned and Skeletonized images.

Skeletonizing:

For the Skeletonizing procedure, a header file named “Skeleton.h” is created which contains two 2 Dimensional arrays what contain the Stage 1 and Stage 2 binary masks. The Steps iv) to vi) are then followed which include determining the intermediary M Matrix that represent possible positions or pixels that need to be replaced.

Classification

A simple absolute distance based comparator is used to classify the Probe as Butterfly or Fly. Since the dataset is severely undersampled having only one sample for each class, the features obtained are coarse and applying advanced classifiers may not provide good results.

Features Selected:

The following features were chosen. **Number of T junctions, Number of End points, Number of four connected points**.

Number of T Junctions Computation

The number of T junctions represent the number of branches in the Thinning and Skeleton images and provides a useful feature to identify a particular shape. The algorithm to compute number of T junctions was implemented as follows

1. Check for pixel with value equal to 255
2. Count the number of white pixels in the 3 x 3 neighborhood centered around the pixel and check if Pixel Coordinates are set to false in the Hashmap.
3. If count is ≥ 3,
    a. Increment a variable to count number of branches
    b. Maintain a hashmap, such that all white pixel locations in the 3 x 3 Neighborhood are set to True Boolean type.
4. Repeat for every Pixel.

Number of End Points Computation

To compute the number of endpoints, the following algorithm was implemented.

1. Check for pixel with value equal to 255.
2. Count the number of white pixels in the 3 x 3 neighborhood centered around the pixel.
3. If count == 1, increment a variable that stores number of end points.
4. Repeat for every pixel.

Number of 4 Connected Pixels

A pixel at location (x,y) is said to be 4 connected if there exists 1 or more pixels in locations (x±1,y) and (x,y±1).

Thus for every white pixel, its 3 x 3 neighborhood was scanned for white pixels in the aforementioned locations and a variable to count the number 4 connected pixels was incremented.

Classifier:

For classification the **Ratio of Number of T junctions in Skeleton to Thinning image**, **Ratio of 4 connected pixels in Skeleton to Thinning image** and **Number of Endpoints in Thinning image** were used to make a decision of the Class of the Probe.

The three parameters were computed for each image, Butterfly, Fly and Probe image, and the **Absolute Distance** of the each of these parameters of the Probe image from Butterfly and Fly were computed. The Probe image is classified to the Class whose parameters were closest to a particular class.

3.3. Experimental Results

Shown below is the Binary Mask generated after following the procedure mentioned in Section 3.2.

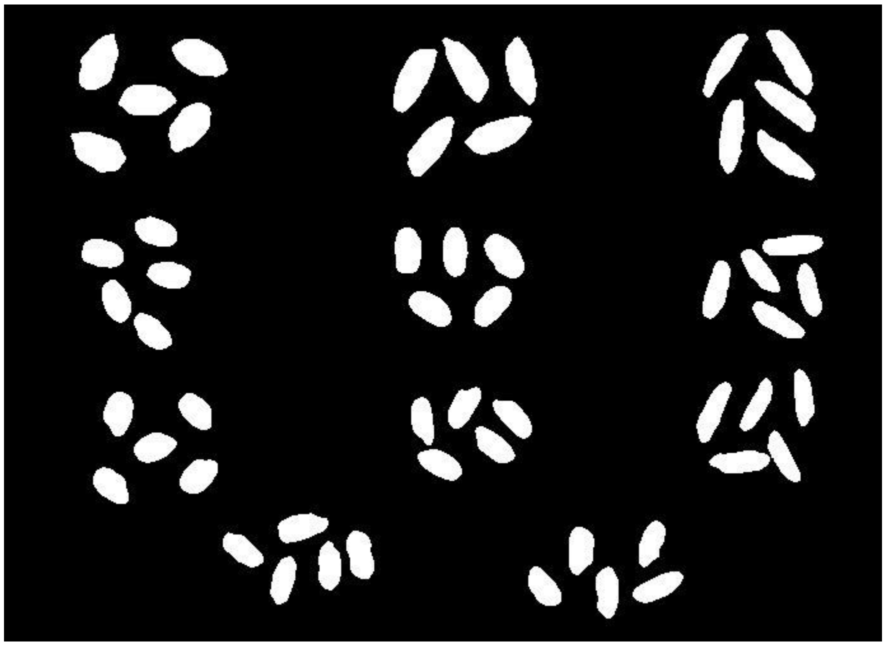
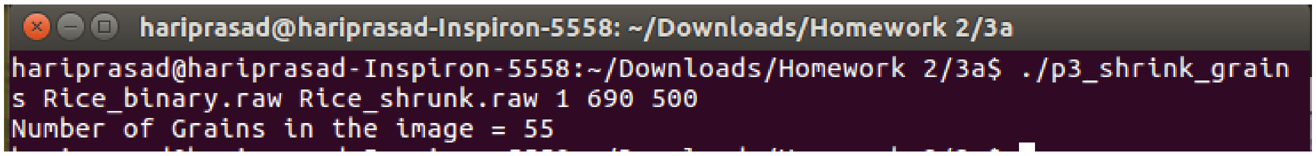


Figure 14: Binary Mask of *Rice Image*

Shown below is the Terminal window output to count the number of Grains in the Rice Image



hariprasad@hariprasad-Inspiron-5558: ~/Downloads/Homework 2/3a

```
hariprasad@hariprasad-Inspiron-5558:~/Downloads/Homework 2/3a$ ./p3_shrink_grain
s Rice_binary.raw Rice_shrunk.raw 1 690 500
Number of Grains in the image = 55
```

Figure 15: Terminal result to count the number of grains

Shown below is the output obtained after the Shrinking procedure is followed to count the number of Rice grains.

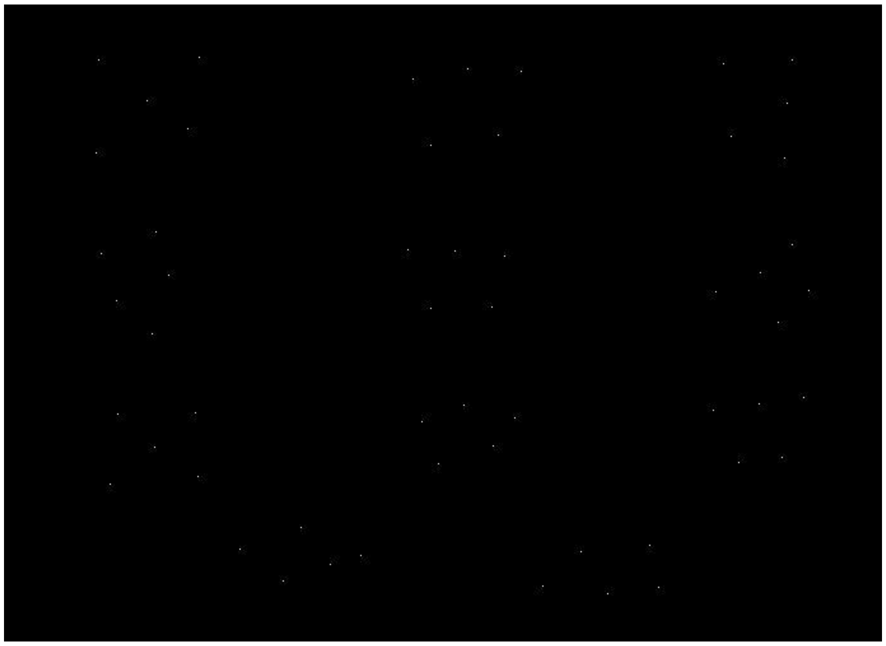


Figure 16: Shrinking each grain to a single pixel

Shown below are 58 x 45 sized images of grains of each type considered **row-wise**, and the corresponding output obtained after Thinning procedure.

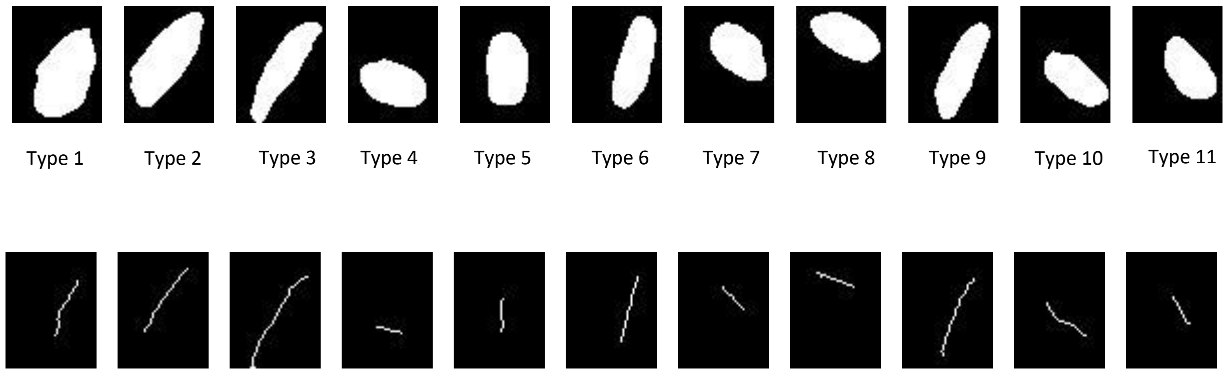
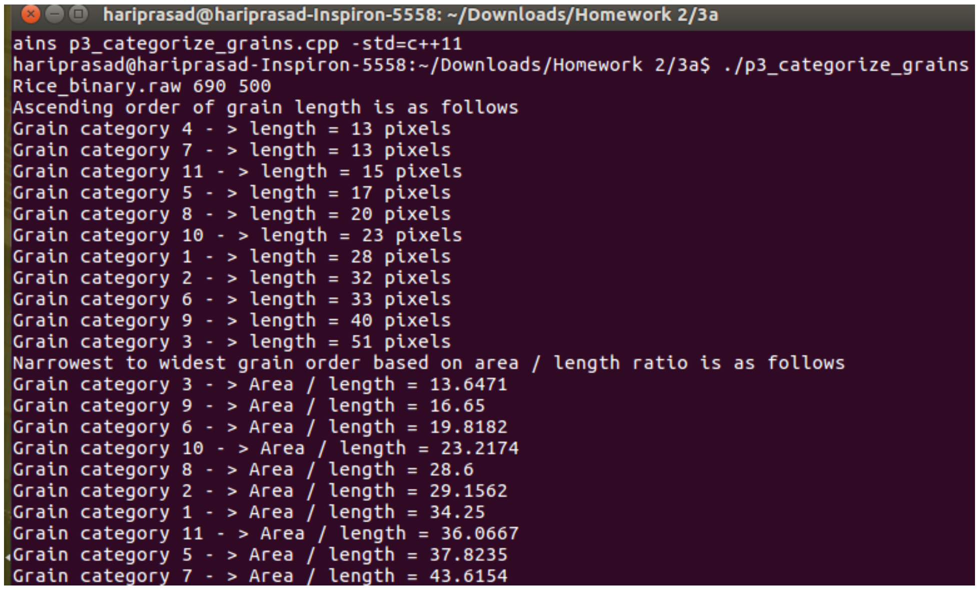


Figure 17: Cropped Images of Each Grain type taken row-wise and their corresponding thinning output



```
hariprasad@hariprasad-Inspiron-5558: ~/Downloads/Homework 2/3a
ains p3_categorize_grains.cpp -std=c++11
hariprasad@hariprasad-Inspiron-5558:~/Downloads/Homework 2/3a$ ./p3_categorize_grains
Rice_binary.raw 690 500
Ascending order of grain length is as follows
Grain category 4 - > length = 13 pixels
Grain category 7 - > length = 13 pixels
Grain category 11 - > length = 15 pixels
Grain category 5 - > length = 17 pixels
Grain category 8 - > length = 20 pixels
Grain category 10 - > length = 23 pixels
Grain category 1 - > length = 28 pixels
Grain category 2 - > length = 32 pixels
Grain category 6 - > length = 33 pixels
Grain category 9 - > length = 40 pixels
Grain category 3 - > length = 51 pixels
Narrowest to widest grain order based on area / length ratio is as follows
Grain category 3 - > Area / length = 13.6471
Grain category 9 - > Area / length = 16.65
Grain category 6 - > Area / length = 19.8182
Grain category 10 - > Area / length = 23.2174
Grain category 8 - > Area / length = 28.6
Grain category 2 - > Area / length = 29.1562
Grain category 1 - > Area / length = 34.25
Grain category 11 - > Area / length = 36.0667
Grain category 5 - > Area / length = 37.8235
Grain category 7 - > Area / length = 43.6154
```

Figure 18: Terminal output of lengths of each grain type and the Area/Length Ratio

Problem 3. b)

Shown below are the results obtained after applying the Hole filling operation on Butterfly, Fly and Probe images

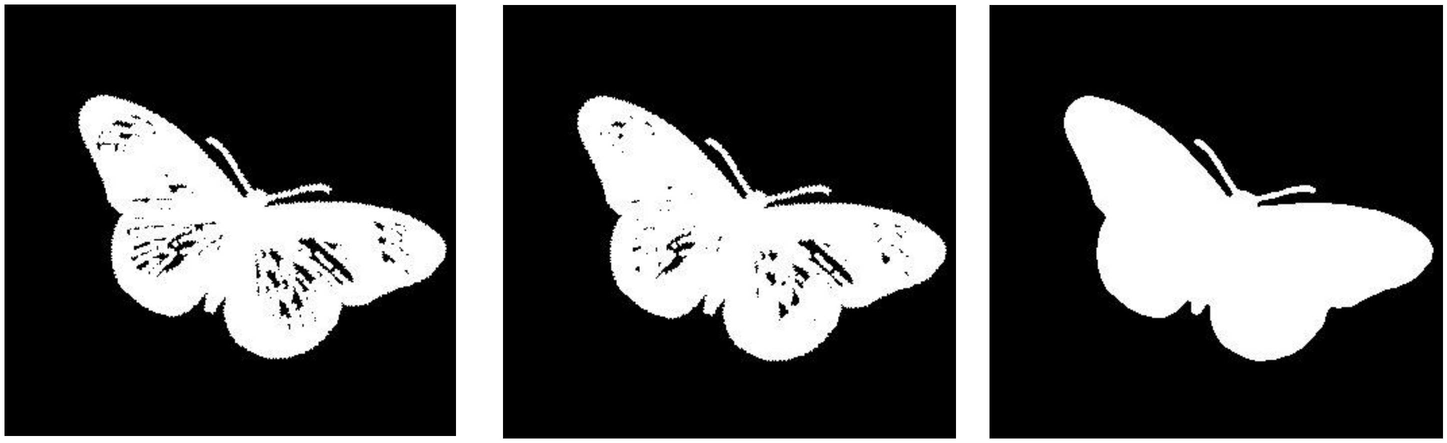


Figure 19 a: Output of Butterfly at different iterations of Hole Filling Procedure

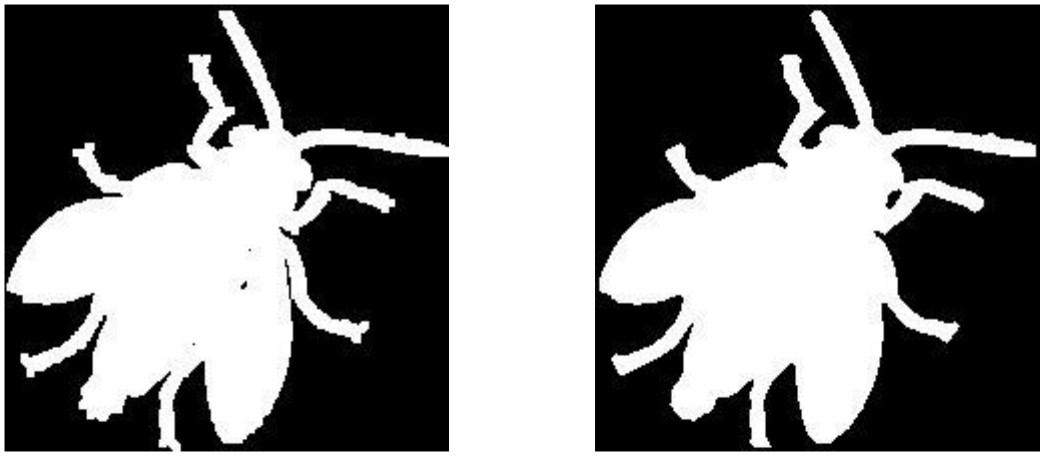


Figure 19 b: Output of Fly at different iterations of Hole Filling Procedure

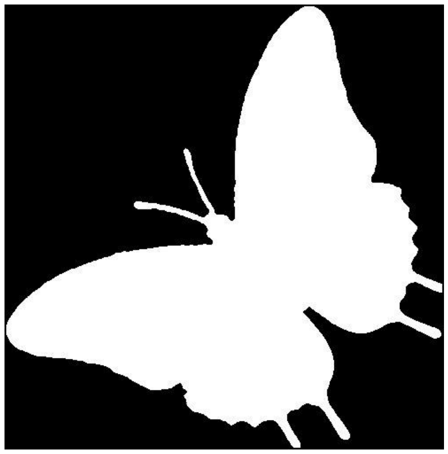


Figure 19 c: Hole Filling procedure completed on Probe Image

Shown below are the results obtained after the Thinning and Skeletonizing operations on Butterfly and Fly images respectively

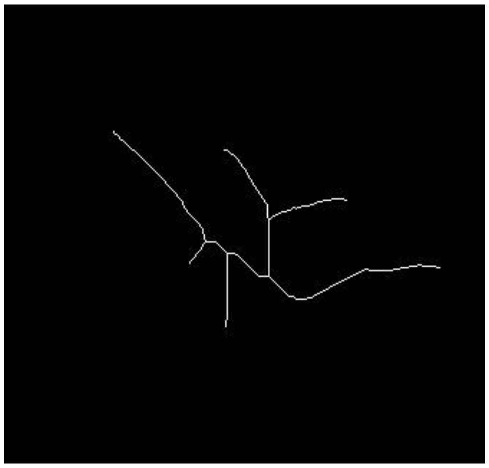


Figure 20 a: Thinning of Butterfly image

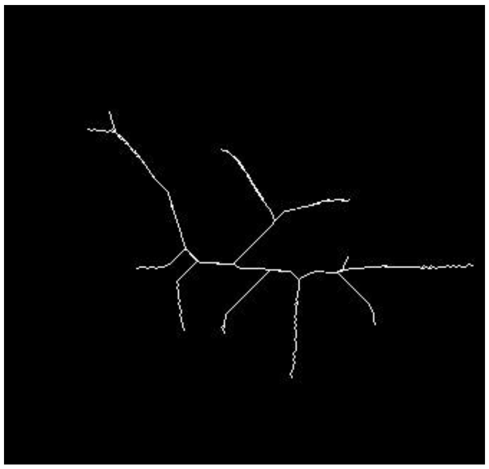


Figure 20 b: Skeleton of Butterfly image

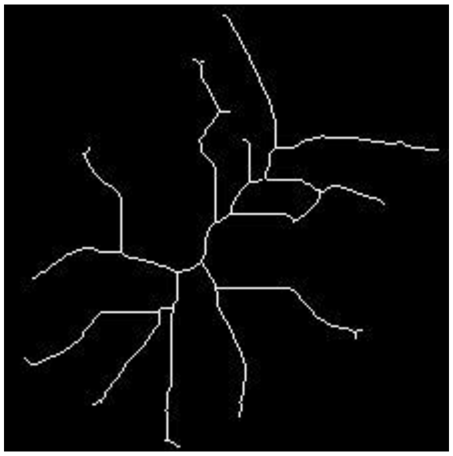


Figure 20 c: Thinning of Fly image

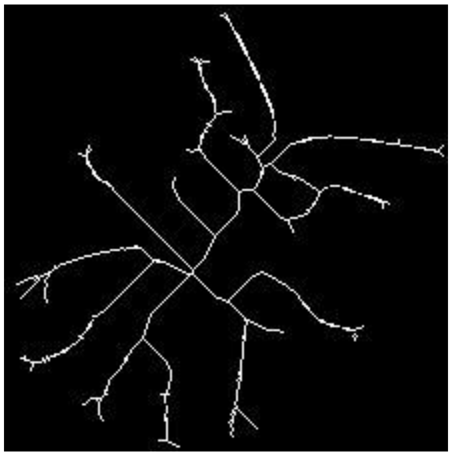


Figure 20 d: Skeleton of Fly image

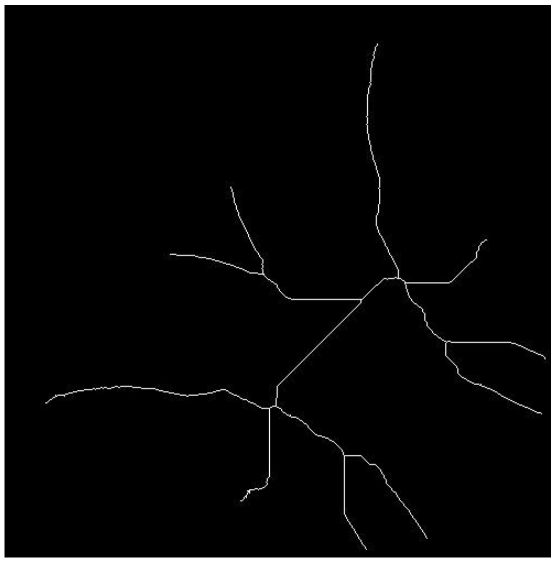


Figure 20 e: Thinning of Probe image

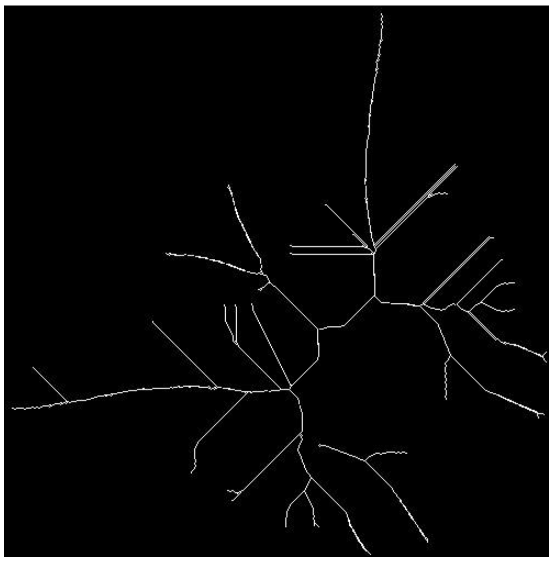
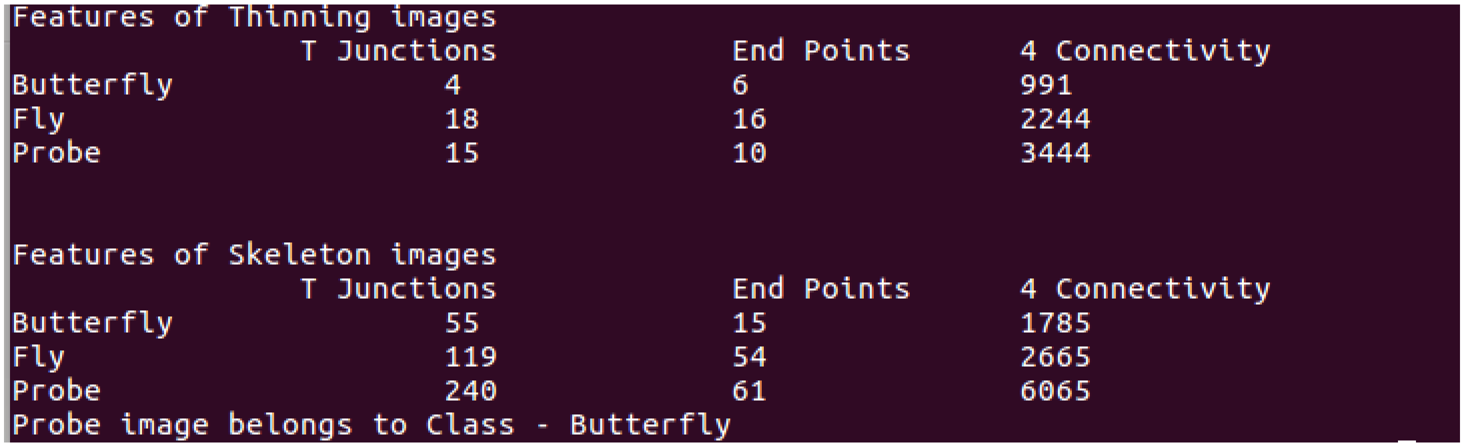


Figure 20 f: Skeleton of Probe image

Shown below is the terminal screen snapshot of the features extracted from Butterfly, Fly and Probe images and the classifier output


```
Features of Thinning images
                T Junctions             End Points      4 Connectivity
Butterfly               4               6               991
Fly                     18              16              2244
Probe                   15              10              3444


Features of Skeleton images
                T Junctions             End Points      4 Connectivity
Butterfly               55              15              1785
Fly                     119             54              2665
Probe                   240             61              6065
Probe image belongs to Class - Butterfly
```

Figure 21: Terminal screen out describing the features and classifier output

3.4. Results and Discussion

Figure 14 shows the Binary Mask generated for the Rice Image. As discussed in the procedure, the 3 x 3 Median filter that is cascaded 3 times helps to remove stray pixels in the background and on the Rice grains. This makes the pixel values in the Background and Grains more uniform across all three channels.

The mode of each channel helps identify the background pixel value, is it has the highest frequency of occurrence. The threshold error of 6 pixels helps remove stray background pixel values around the mode that might not have been removed by the Median filter.

Finally, the 5 x 5 Median filter helps smoothen out the grains after Thresholding, and also helps remove errors contributed due to Thresholding. The Binary Mask thus looks segments each Rice grain efficiently.

Figure 15 and 16 shows the output obtained due to shrinking and counting of Rice grains. The algorithm perfectly identifies all 55 Rice Grains and 55 white pixels.

The concept behind the Shrinking, Thinning and Skeletonizing procedure can be understood by the flowchart.

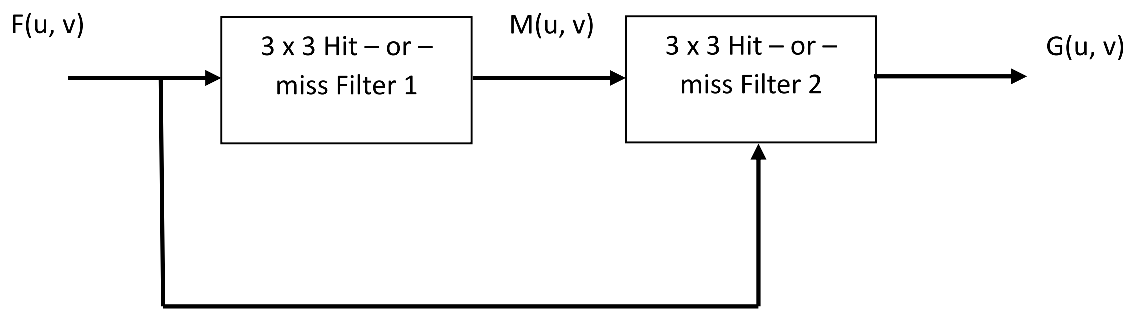

F(u, v)
3 x 3 Hit – or – miss Filter 1
M(u, v)
3 x 3 Hit – or – miss Filter 2
G(u, v)

Figure 22: Flowchart of Skinning, thinning and Skeletonizing Operation

We define our objective as to conditionally remove the boundary pixels based on a two stage process.

- In the first we pass the Binary image through and compare every 3x3 window with all sets of 3 x 3 binary masks defined for stage 1
- This produces an intermediary Image called M matrix, that conditionally places 1 to denote a candidate position whose pixel may be erased.
- The M Matrix is passed through a set of stage 2 binary filter masks that remove pixels based on Hit or Miss, a Miss would replace a pixel with 0 and a miss will retain the pixel.

This procedure is implemented as mentioned in section 3.2.

Thus the Shrinking, Thinning and Skeleton images are obtained for all images.

Figure 17 denotes the Rice Grains that were cropped from each category taken **row-wise**, to perform the thinning operation. Counting the number of white pixels in the thinned image thus gives a length measure of each type of rice grain.

Figure 18 denotes the terminal output that orders the 11 chosen rice grain based on the increasing order of their length and Area/length ratio. This gives us a measure to categorize the Grains as a degree of narrow-ness or by their lengths. Figure 23 and 24 arrange the grain types based on the chosen measures.

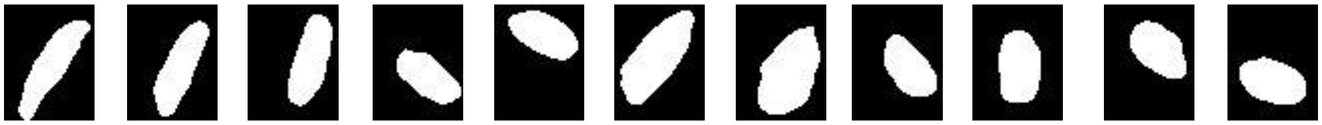


Figure 23: Grain Types Ordered in Decreasing order of Narrow-ness

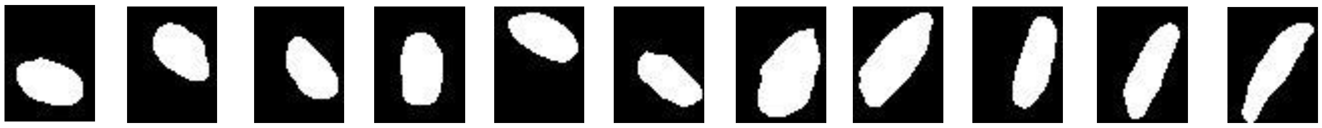


Figure 24: Grain Types Ordered in Increasing order of Length

Problem 3.b)

Figure 19 a, b and c capture the output of *Butterfly, Fly and Probe* images at different iterations during the Hole Filling procedure. The procedure followed in section 3.2. Helps fill all the holes to obtain a perfect binary image for each object. The 5 x 5 Median filter help further smoothen the contours.

Figure 20 a through f capture the Thinning and Skeletonizing outputs for *Butterfly, Fly and Probe* images. The skeleton images capture more details regarding the structure of the three images as compared to the Thinning images. Features such as the main body and antennae are well captured by both the thinning and skeleton operations.

For **Classification, the Ratio of Number of T junctions in Skeleton to Thinning image**, **Ratio of 4 connected pixels in Skeleton to Thinning image** and **Number of Endpoints in Thinning image** were computed and analyzed. The number of endpoints gave a good measure as a feature to classify the Probe image as Butterfly or as a Fly. This is primarily because the Fly image has a complex body structure with multiple limbs as legs and wings. This contributed to greater number of endpoints in the Fly image as compared to the those of the Probe and Butterfly image.

Comparing the Ratio of T junctions and 4 Connected pixels between the Skeleton and Thinning image gives a good descriptive feature because it takes into consideration the relationship between the Skeleton and Thinning of each unique shape and is **independent of Scaling effect**.

It was observed that the three parameters mentioned above for Probe image were closer to the parameters computed for the Butterfly image than the Fly image. Thus the Probe image was classified as a Butterfly.

Figure 21 shows the terminal screen output that displays the number of T junctions, Endpoints and 4 connected Pixels in Thinning and Skeleton images for each of the three images. The algorithm then classifies the Probe image as Butterfly based on these parameters.

References

[1] "Homographic Transformation" ~ http://www.corrmap.com/features/homography_transformation.php

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

[3] " Floyd- Steinberg Dithering" ~ https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering

[4] RGB to Grayscale ~ https://www.mathworks.com/matlabcentral/answers/99136-how-do-i-convert-my-rgb-image-to-grayscale-without-using-the-image-processing-toolbox