EE569 Digital Image Processing

HOMEWORK # 2

Aagam Shah

USC ID-8791018480

USC Email – aagamman@usc.edu

Submission Date – February 16, 2020

Problem 1: Edge Detection

(a) Sobel Edge Detector:

1. ABSTRACT & MOTIVATION

As we know that the Human Vision perceives the edges and the overall appearance of an object for visual perception and object detection. Therefore, the Edge Detection is the crucial part for image processing, image analysis, pattern recognition and computer vision. Edge detection is nothing but the technique of finding the regions in an image where there is sharp change in an intensity or a sharp change in color. Sobel Edge Detection is one such technique used for edge detection, which is basically an approximation to a derivative of an image. Technically, Sobel Edge Detector is a discrete differentiation operator, computed using an approximation of the gradient of the image intensities. The computation of the Sobel Edge Detector is basically based on convolution of an image with an integer valued filter in the vertical as well as horizontal directions and a threshold value is chosen to make the value of the pixel intensities in only two levels i.e., either 0 or 255.

2. APPROACH & PROCEDURE

Basically, here in this technique we detect the edges of an image with the Sobel Edge Detector to find out the difference by placing the gradient matrix over each pixel of an image. Since, it's difficult to implement this technique through the continuous domain approach so we analytically approach the problem through discrete finite difference. The edges can be identified by sharp change in the intensities via first order gradient. The steps to perform the edge detection using Sobel Edge Detection Technique is as follows:

Step 1: Firstly, load the input RGB image and convert it into the grayscale image using the formula: 0.2989*R + 0.5870*G + 0.1140*B.

Step 2: Pad the boundaries of an input image according to the size of the Sobel Filter.
Step 3: Next, compute the finite difference in x-direction (Gx) and y-direction (Gy) at the center pixel of that particular location through the convolution using the gradient matrix in x-direction (Gx) and y-direction (Gy).

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Step 4: Then we calculate the Gradient Map of each pixel using the formula as given:

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

Step 5: Now we normalize the outputs of Gx and Gy to 0 – 255.
Step 6: Finally, the Cumulative Histogram of normalized Gradient magnitude map is used for setting the threshold to obtain the edge map with best visual performance whose pixel values are either 0 (Edge) or 255(background).

3. EXPERIMENTAL RESULTS

Fig. 1.1

Fig.1.2

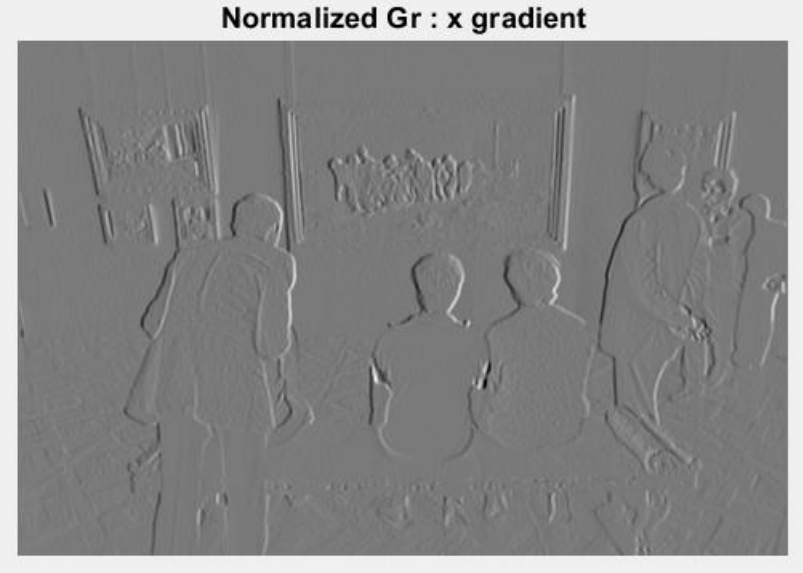


Fig.1.3 (X-gradient)

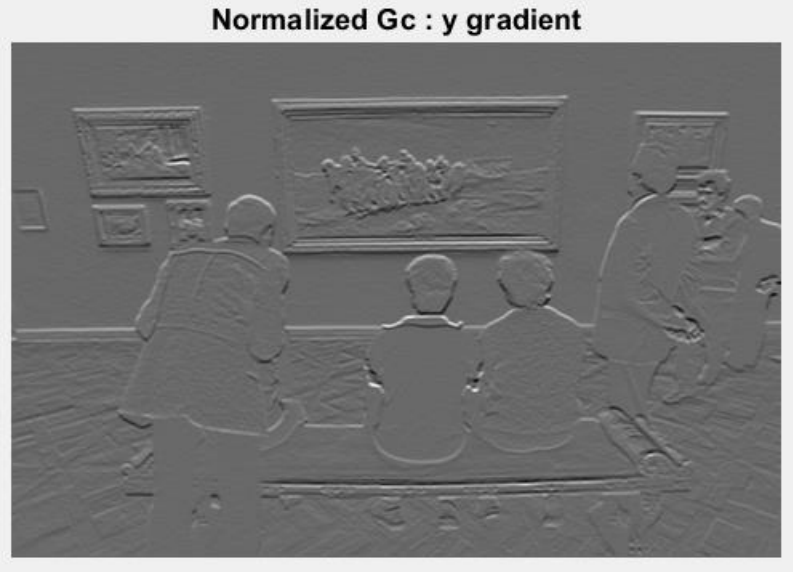


Fig.1.4 (Y-gradient)

Fig. 1.5 (Gradient Magnitude Map)

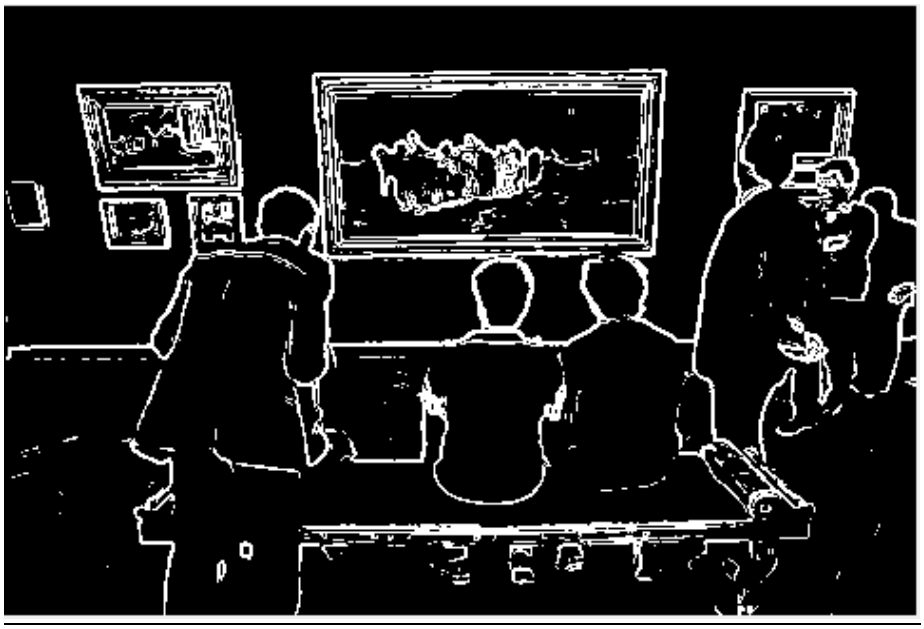


Fig.1.6 (Sobel Edge Detection)

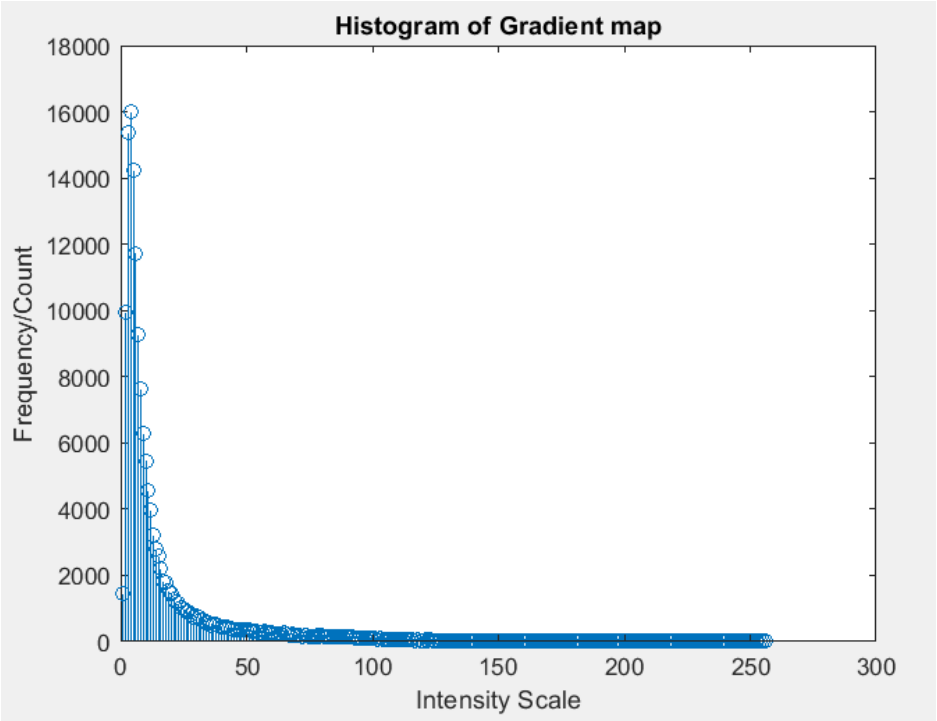


Fig. 1.7

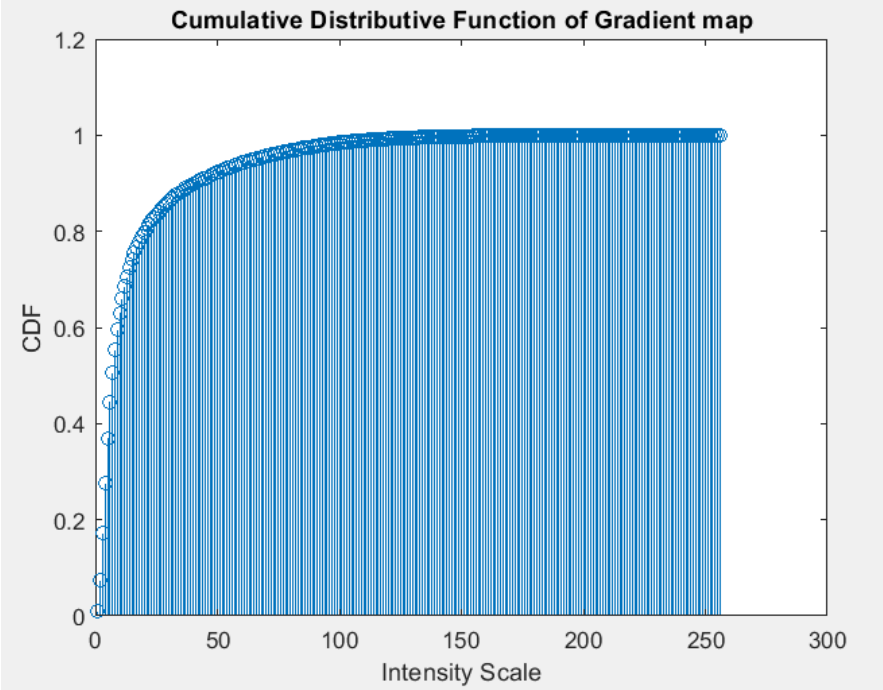

Cumulative Distributive Function of Gradient map
CDF
Intensity Scale
0
0.2
0.4
0.6
0.8
1
1.2
0
50
100
150
200
250
300

Fig. 1.8


RGB Image

Fig. 1.9

Fig. 1.10

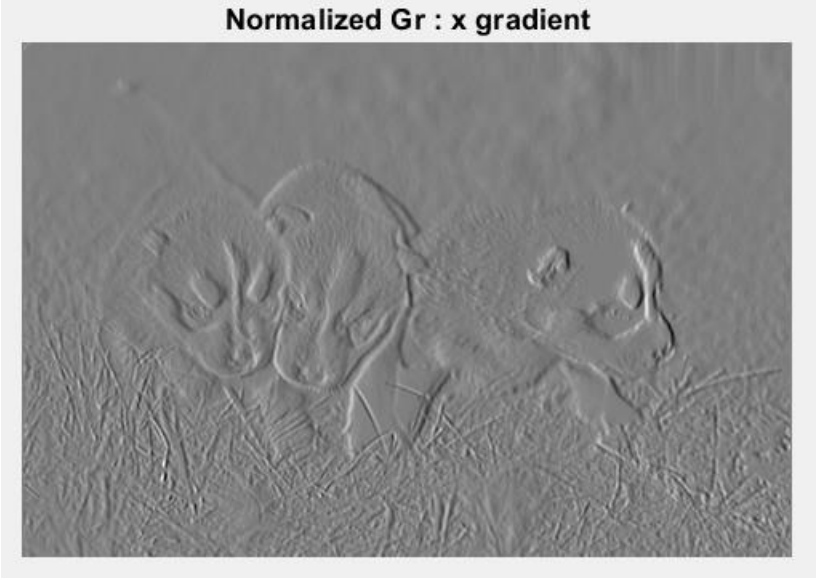


Fig. 1.11 (X-gradient)

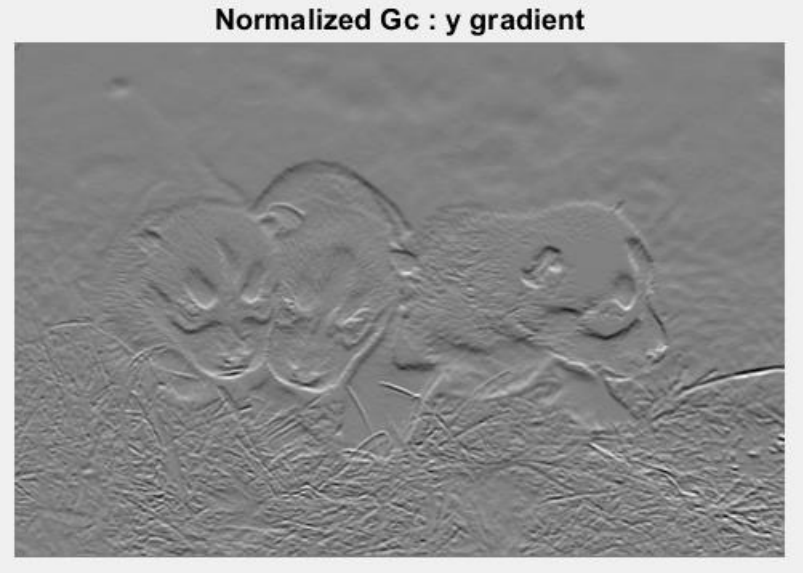


Fig.1.12 (Y-gradient)

Fig. 1.13 (Gradient Magnitude Map)

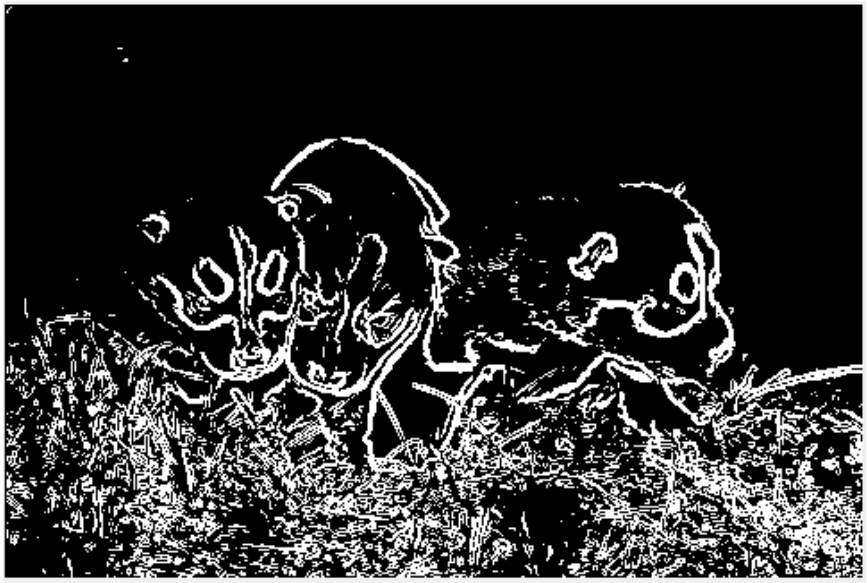


Fig. 1.14 (Sobel Edge Detection)

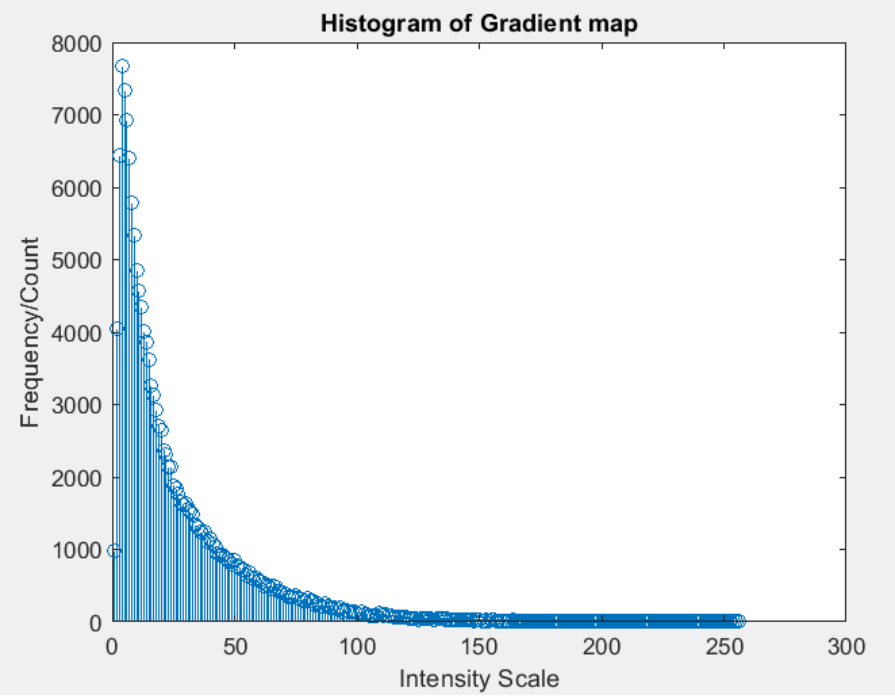


Fig. 1.15

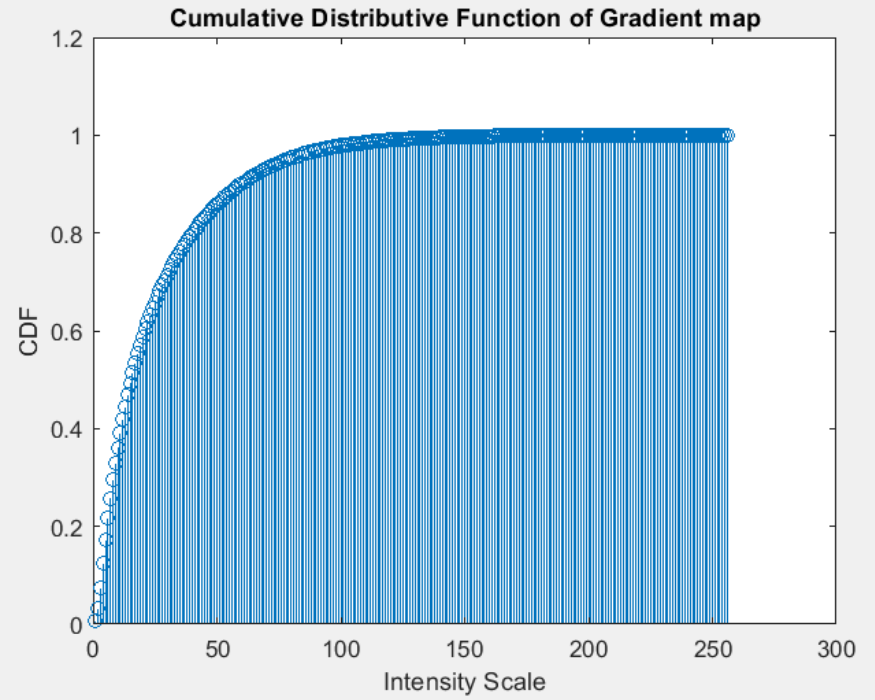


Fig.1.16

4. DISCUSSION

As we can see that the Sobel Edge Detector is successfully implemented which can be used for segmenting image. The Sobel Edge Detector is quite easy and simple to implement and execute as well. Hence the computation cost of this edge detection technique is also low. But on the other hand, the gradient computations obtained are relatively crude especially for the higher frequency variations of an image. But since, Sobel Edge Detection technique produces the same output every time its executed over an entire image, thus making this method a stable edge detection technique for image segmentation. So, Sobel Edge Detection reduces the unnecessary information from the image, but it preserves the structure of the image by extracting the important features such as lines, edges, corners and curves making it useful for applications such as segmentation, Object or Pattern Recognition, etc. It is also computationally less intensive but highly sensitive to noise. As Sobel is the first order gradient, so it is not good for edge detection as the threshold value will the output image drastically.

(b) Canny Edge Detector:

1. ABSTRACT & MOTIVATION

The Canny Edge Detector is an edge detection technique which is based on multi-stage algorithm to detect varieties of edges in an image. Canny edge detection is basically a technique used to extract the useful structural information of different objects present in

an image thereby reducing the amount of data to be processed. The Canny Edge Detector is basically a multi-stage edge detector. This edge detection technique is based on the derivative of Gaussian to calculate the intensity of the gradients of the pixels. Here, the Gaussian reduces the noise if present in the input image. The basic idea of the Canny Edge Detection technique is that the potential edges are thinned down by removing non maximum pixels of the gradient magnitude. Then lastly, the edge pixels are either retained or eliminated using the hysteresis thresholding on the gradient magnitude map of an image.

The Canny Edge Detection has 3 adjustable parameters i.e.,

i. The width of the Gaussian
ii. The lower threshold
iii. The higher threshold

So, noisier is the image the greater is the width of the Gaussian curve.

2. APPROACH & PROCEDURE

The flow of algorithm for Canny Edge Detection Technique is as follows:

Step 1: Firstly, in the preprocessing step the Gaussian filter is applied to smoothen the given input image in order to filter out the noise if present.

Step 2: Next we find the intensity gradients of an input image in the x-direction and y-direction, following the procedure analogous to the Sobel Edge Detection.

Step 3: Compute the Gradient Magnitude Map and Gradient Orientation given by:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Round off the orientation to one of the four possible angles (0, 45, 90, 135).

Step 4: **Non-Maximum Suppression**: It is basically an edge thinning technique. By this technique we can remove the pixels that are not considered to be the part of an edge. Technically, it suppresses all the gradient values to zero except for the local maxima.

Step 5: **Hysteresis Thresholding:** This is the final step for the Canny edge detection technique. As we know that the Canny Edge Detection technique uses two threshold values namely (upper and lower). These two threshold values divide the gradient map into 3 different regions of edges, non-edges and the ambiguous region. So basically, in this step:

i. If a pixel gradient value if greater than the upper threshold value, then that pixel is considered to be an edge.
ii. If a pixel gradient value is less than the lower threshold value, then that pixel is not accepted as an edge.

iii. If the pixel gradient value is between the two thresholds, then it will be accepted only if it is connected to the pixel which is above the upper threshold value.

PROCEDURE:

Step 1: Firstly, load the input RGB image and convert it into the grayscale image using the formula: 0.2989*R + 0.5870*G + 0.1140*B.
Step 2: Next, binarize the grayscale image with threshold value 128.
Step 3: Then we now pass the grayscale image and the threshold values through the inbuilt edge () function by selecting the method as 'canny'.
Step 4: Thus, the output image obtained through this function has black background and the objects in an image is recognized through white edges.

3. EXPERIMENTAL RESULTS

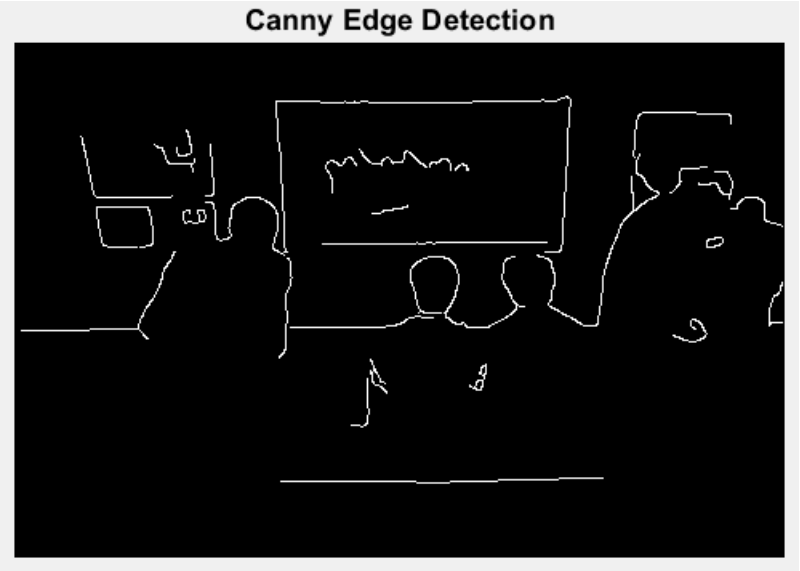


Fig. 1.17 (Low Threshold = 0.3 and High Threshold = 0.5)

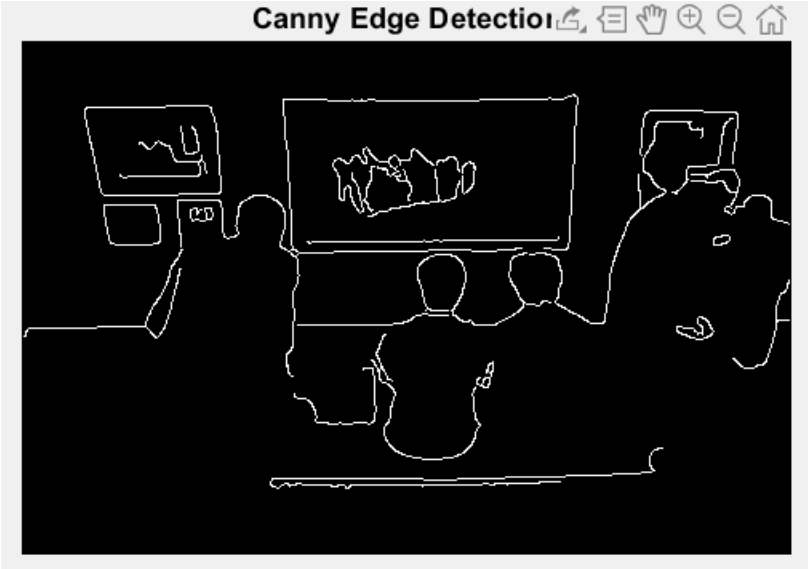


Fig. 1.18 (Low Threshold = 0.1 and High Threshold = 0.5)

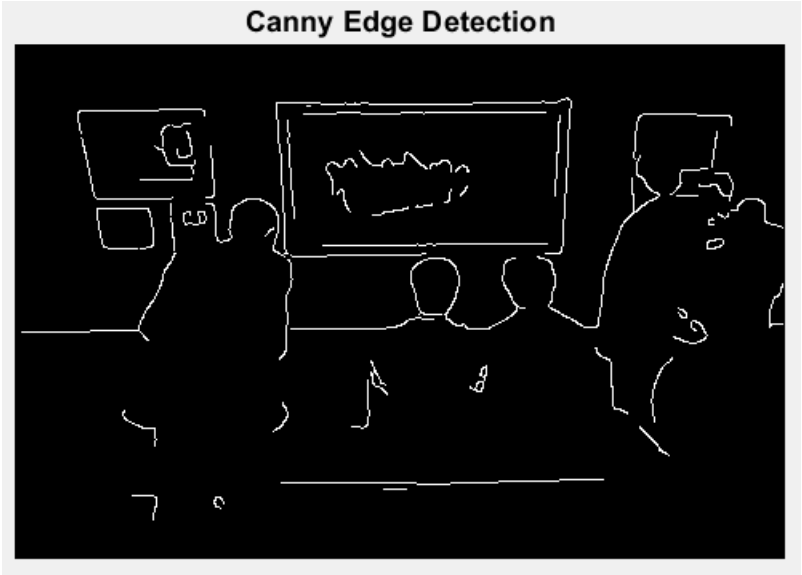


Fig. 1.19 (Low Threshold = 0.3 and High Threshold = 0.4)

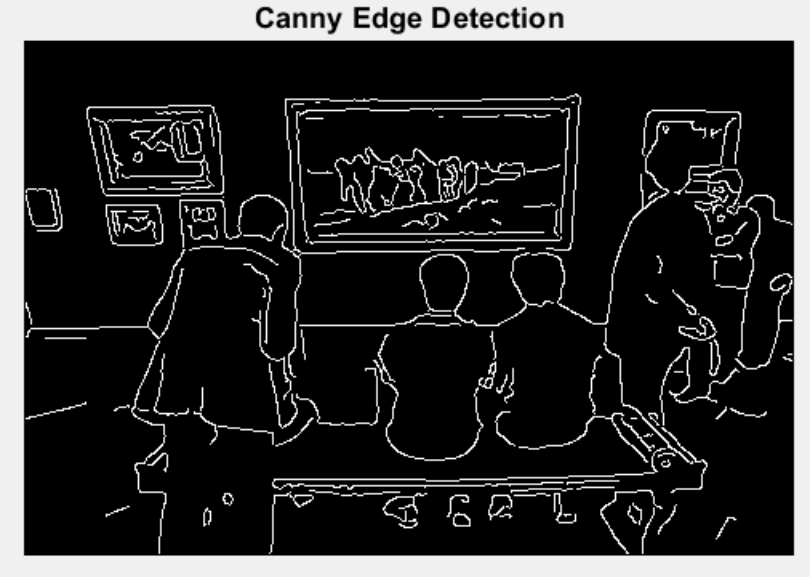


Fig. 1.20 (Low Threshold = 0.1 and High Threshold = 0.15)

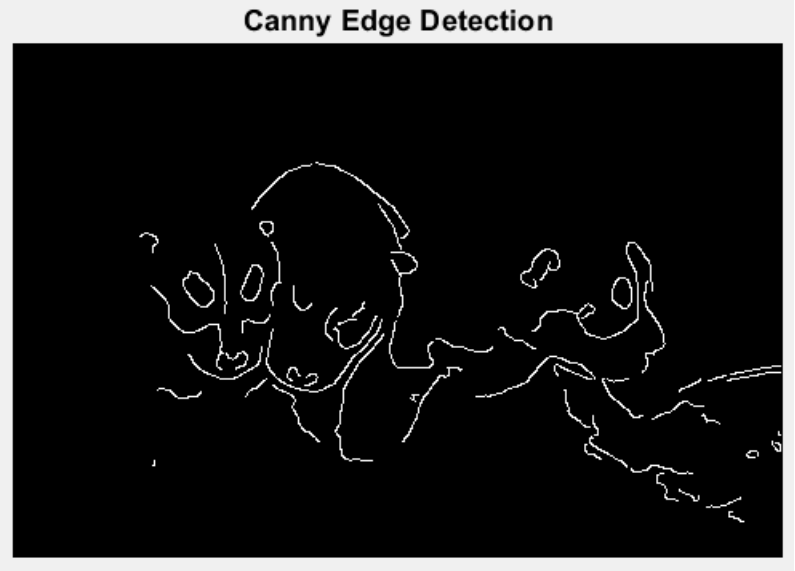


Fig. 1.21 (Low Threshold = 0.3 and High Threshold = 0.5)

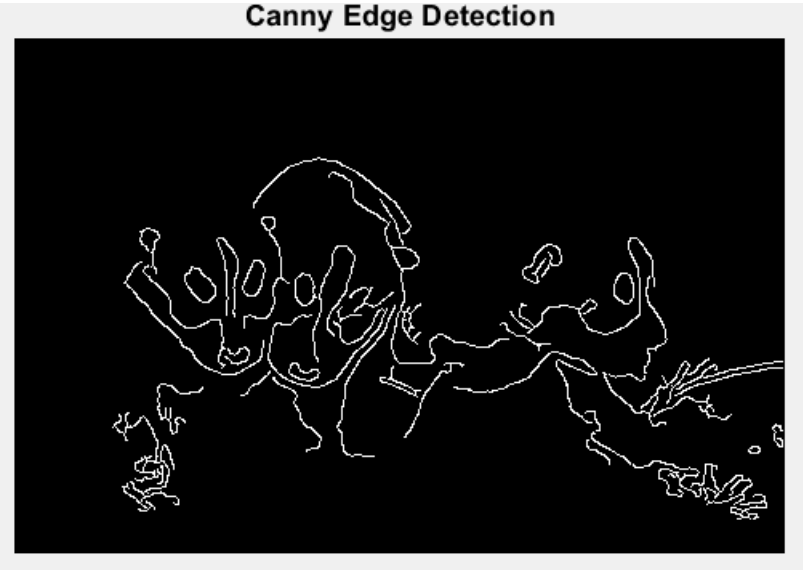


Fig. 1.22 (Low Threshold = 0.1 and High Threshold = 0.5)

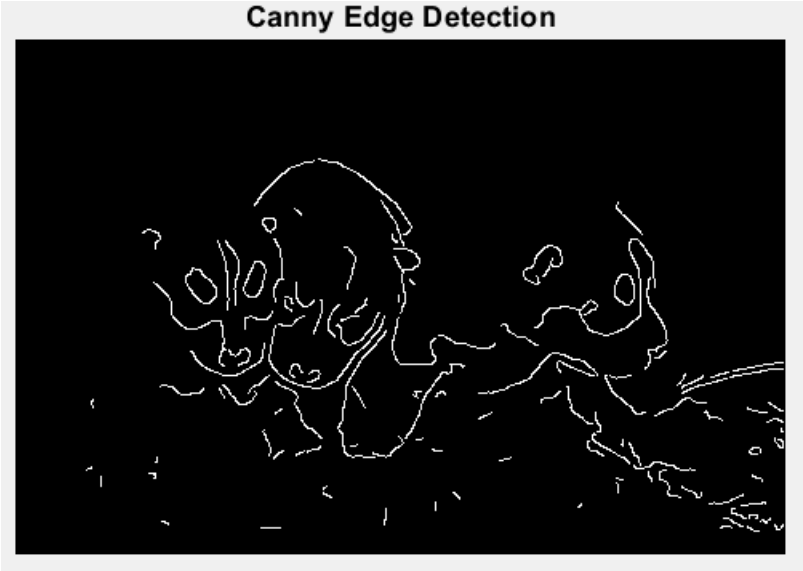


Fig. 1.23 (Low Threshold = 0.3 and High Threshold = 0.4)

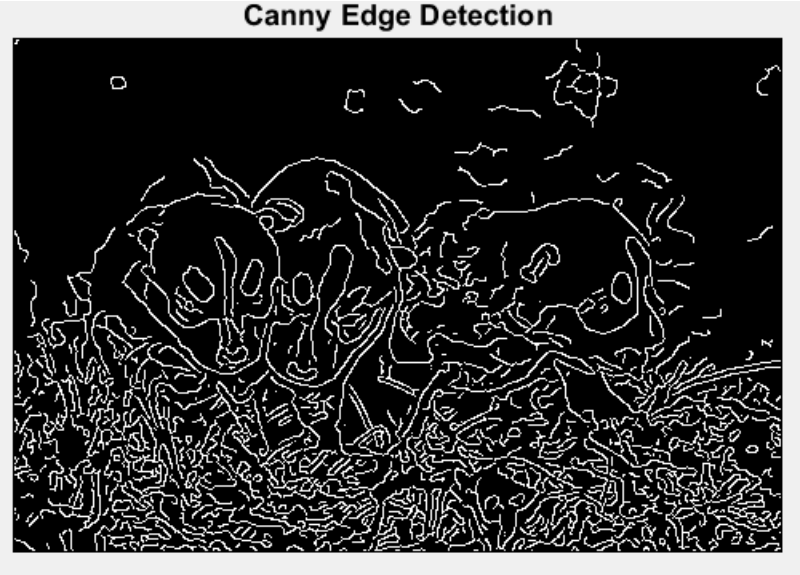


Fig. 1.24 (Low Threshold = 0.1 and High Threshold = 0.15)

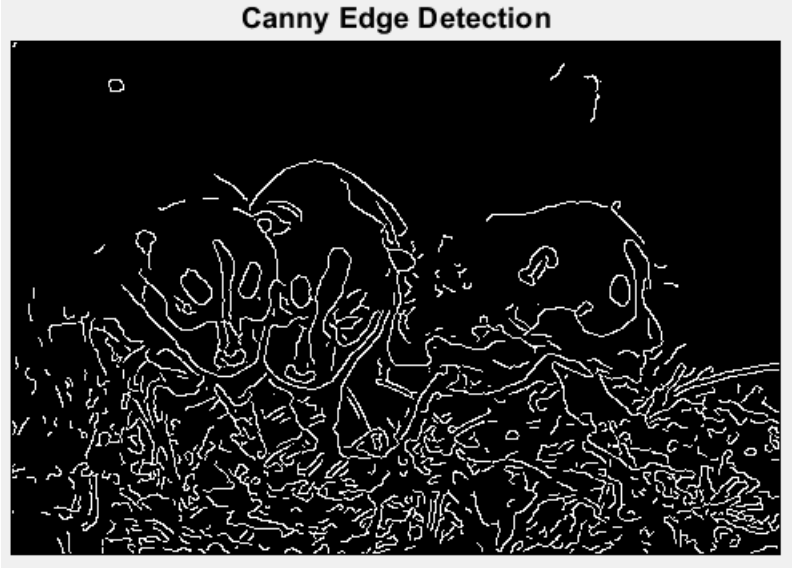


Fig. 1.25 (Low Threshold = 0.2 and High Threshold = 0.3)

4. DISCUSSION

Canny Edge Detection algorithm is basically used to extract the useful structural information of various objects present in an image. It is based on second order derivative. The criteria used for edge detection in this algorithm are:

i. Detection of the edges with low error rate means that the detection of edges should accurately identify as many as edges as they are present in an image.
ii. The edge point detected by the detector should precisely be located at the center of an edge.
iii. The particular edge in an image should be traced once, and the image noise should not create false edges.

Due to these three criteria of Canny Edge Detection and the ease of implementing this process, it is one the most popular algorithm for edge detection.

ANSWERS:

1. **Non-Maximum Suppression:** Non-maximum suppression is an edge thinning technique. It is applied to find the biggest edge. Basically, after applying the gradient magnitude operation, the edges obtained from the gradient value are quite blurred. So, by using non-maximum suppression all the gradient values to zero except the local maxima, which is nothing but the sharpest edge. The algorithm of this technique for each pixel in the gradient image is as follows:
    i. First, we compare the edge intensity of the current pixel with the edge intensity of the pixel in the positive and the negative gradient direction.
    ii. If the edge intensity of the current pixel is larger as compared to the other pixels in the mask along with the same direction, then the value of that current pixel is preserved or else it is suppressed.
2. So, after applying the non-maximum suppression the pixels after suppression gives more clear representation of the real edges of objects in an image. But still there are some edge pixels left out due to noise variation. Hence, it is essential to filter out those edge pixels having low gradient value and retain the edge pixel values with high gradient value. This can be achieved by carefully choosing the high and low threshold values. So basically, if an edge pixel gradient value is higher than the high threshold value, then it is categorized as strong edge pixel, otherwise if an edge pixel gradient value is smaller than the high threshold value but greater than the low threshold value then it will be categorized as the weak edge pixel, or if an edge pixel value is smaller than the low threshold value then it will be completely suppressed. These two-threshold values are determined by various empirical observations and carefully chosen to get the best output results.
3. As from the observations generated of edge maps by Canny Edge Detector and by trying different values of low and high thresholds, we observe that:
    - As we increase the upper threshold as well the lower threshold values the edges of the objects are not well defined.
    - If the lower threshold value is decreased and the upper threshold value is increased there is better contour detection as compared to the above case.
    - With increasing the lower threshold value and decreasing the upper threshold value more edge information of an image is retained.

- Finally, in the last case with decrease in lower threshold value and upper threshold value a very high amount of edge information and details are retained along-with unwanted information as well.

So it is necessary to choose the optimum lower and upper threshold value to extract the only required edge based information from the image.

(c) Structured Edge:

1. ABSTRACT & MOTIVATION

Structured edge is basically a data driven approach used for detecting the structure of each edge. It is basically the combination of the traditional edge detection and machine learning algorithm. To detect whether the pixel is an edge or not, we use Random Forest Classifier for that. Random Forest Classifier consists of many decision trees. These decision trees are then trained using the random sample of the data as well as feature set. The combination of 0's and 1's is used to identify the contour and then the segmentation mask predicts that whether the pixel is and edge or not.

2. APPROACH & PROCEDURE

The following algorithm is implemented to obtain the Structured Edge output:

Step 1: Parameters are initialized to train the decision tree using the Training Dataset through the Structured Edge Toolbox.
Step 2: Then the edgeTrain() function is used to train the Structured Edge Detector.
Step 3: Setting of various detection parameters such as sharpen, nms, multiscale, nTreesEval, etc. is done.
Step 4: Now we load the input image on which the edge detection have to be performed by running the edgesDetect() function and we obtain the probability edge map as an output.
Step 5: In order to obtain the Binary Edge Map, we binarize the probability edge map with certain threshold value.

3. EXPERIMENTAL RESULTS

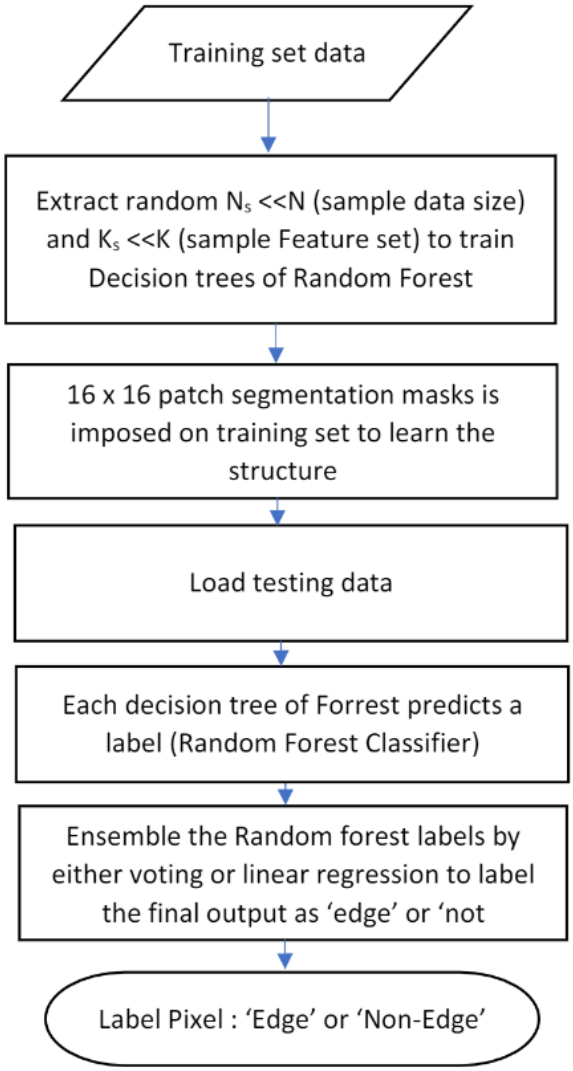
Fig. 1.26

Fig. 1.27

4. DISCUSSION

ANSWERS:

i. Structured Edge Detection algorithm can be explained using the flowchart as follows:



Initially, we load the training data set to train the decision trees of forest randomly in order to not to overfit the classifier. Then the required features are extracted to increase the throughput thereby making the classification unique. Each decision tree is similarly trained using the 16x16 segmentation mask. Set of all such decision trees is nothing but Random Forest Classifier. So as soon as the test data is passed through each decision tree, a label is assigned to each pixel. Random Forest Classifier the

classifies each pixel as either ‘edge’ or ‘not an edge’ through linear regression technique.

ii. Principle of RF Classifier:

Random Forest Classifier basically creates a set of decision trees from randomly selected data of training set. Then the aggregation of votes from different decision trees are used to decide the final class of the test object. So, basically in simple terms Random Forest Classifier constructs multiple decision trees and merges them together to get more accurate and stable output. Due to the search of the best random feature among the random subset of features, it results in a wide diversity which generally results in a better model.

a. A decision forest in an ensemble of T independent trees ft.
b. From the given sample x, the prediction results ft(x) from the set of the tress are combined into a single output using an ensemble model.
c. The choice of an ensemble model varies from one problem to another and it also depends on the gamma factor. The choice of parameters include regression averaging and classification based on majority voting.

Process of Decision Tree Construction:

As we know that the Random Forest Classifier is used for the Structured Edge Detection. The Random Forest Classifier basically consists of many decision trees and the results of these decision trees are combined into one final probability function.

The steps for the construction of decision tree is as follows:

Step 1: A sample x belongs to X is classified by the decision tree ft(x) by recursively branching out down the tree in either left or right direction until the leaf node of the branch is reached.

Step 2: If $h(x,\theta j) = 0$ then node j will send the ‘x’ to either left or right thereby terminating the branching out process at the leaf node.

Step 3: Although there are many features out of which any one can be chosen but the feature which divides the branch with the highest purity is chosen.

Step 4: Next the θ parameter of ‘x' data is compared against the standard threshold value. If it is greater than the threshold value, then the data ‘x’ is classified towards the left node or else towards the right node.

Step 5: This process of classification is carried on until the size of the training data set falls below the fixed threshold value or the maximum depth is achieved.

Step 6: The information gain of the classification is given by the formula:

$$I_j = H(S_j) - \sum_{k \in \{L,R\}} \frac{|S_{jk}|}{|S_{jl}|} H(S_j^k)$$

where $H(S_j) = -\sum_y p(1-\log(p))$ alternatively Gini Impurity is $H(S_j) = -\sum_y p(1-p)$

Step 7: The output prediction of the tree is stored at the leaf node with target label as y belonging to Y for some input x.

Step 8: Last but not the least, in this step the randomness of the system is exhibited by as each decision tree is trained by the random subset of data sets.

iii. The edge detection parameters set for edgesEval are multiscale, sharpen, nTreesEval, nms and the threshold value to convert probability edge map to binary edge map. The default parameters chosen are:
nTrees = 8
multiscale = 0 (This parameter can be increased for increasing the accuracy)
sharpen = 2 (In order to sharpen the edges)
nTreesEval = 4 (To control the time of the process)
nms = 0 (to perform the non-maximum suppression after detection of edges)

In general, if nms = 0 then it has thicker edges and if nms = 1 then it has thinner edges which has better performance for edge detection.

On comparison of Structured Edge (SE) Detection with Canny edge detection, SE performs better as compared to Canny because in Canny the edges get smoothen due to the Gaussian derivative. Also, as Canny has two levels of thresholds so it compromises on too many edges resulting in less fine output as compared to that of the SE detector. SE detection utilizes various parameters for edge detection while Canny only uses Gaussian derivative for edge detection. Hence, SE performs better in terms of contrast as compared to that of Canny.

(d) Performance Evaluation:

1. ABSTRACT & MOTIVATION

Performance evaluation is nothing, but the quantitative comparison of different edge maps obtained by different edge detectors. The edge detection is basically used to train the machine to generate the edges of objects. Therefore, we need an edge map of ground truth for evaluation of the quality of the machine generated edge map.

2. APPROACH & PROCEDURE

So, we need to find the error which can be done in the following way. The four classes are given as

1) **True Positive**: When edge pixels in the edge map coincides with edge pixels in the ground truth image. The algorithm clearly identifies these as edge pixels.

2) **True Negative:** When Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth image. The algorithm clearly identifies these as non-edge pixels.
3) **False positive:** When Edge pixels in the edge map correspond to the non-edge pixels in the ground truth image. The algorithm wrongly identifies these as fake edge pixels.
4) **False Negative:** When Non-edge pixels in the edge map correspond to the true edge pixels in the ground truth image. The algorithm skips these edge pixels.

Therefore, the pixels categorized in 1st and 2nd conditions are true while those categorized in 3rd and 4th are the error pixels. So, in total the performance of an edge detection algorithm is measured using the parameter known as F measure, which is nothing but the function of precision and recall. F-measure is given by the formulae given below:

$$\text{Precision}: P = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}$$

$$\text{Recall}: R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}$$

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

3. EXPERIMENTAL RESULTS

Calculation of Precision and Recall Value for *Gallery* and *Dogs* images is given in **Appendix** below. Please refer to it for the results.

4. DISCUSSION

The Precision and Recall are the two parameters which play an important role while evaluating the performance.
The F-measure is image independent.
The Precision can be made higher by reducing the threshold in obtaining the binary edge map. But this leads to the lower value of Recall.
So, more the F-measure the better is the edge detection output.

ANSWERS:

I. On comparing different edge detectors, I found that the evaluation of images when done in noisy environment then Canny, Laplacian of Gaussian, Robert, Prewitt, Sobel perform better, respectively.
II. As we know that the F-measure is image independent. But upon calculation of precision and recall values for both *Gallery* and *Dogs* image we find out that the Gallery image is easier to get high F-measure because in the Gallery image the edges

are more prominent and easy to identify so it takes lesser time as well as less complex computation as compared to the Dog image because the Dog image seems more noisier and complex computation due to the grass artifacts thereby making it complex to for computation of F-measure.

III. The rationale behind the F-measure definition is already explained in the discussion above. No, it is not possible to get a high F-measure if the precision is significantly higher than recall or vice versa because if the precision is made higher by decreasing the threshold in deriving the binary edge map then it will result in lower recall or vice versa thereby making-measure low.
If Precision + Recall = constant
Then F = (2*P*(constant-P))/constant
The above function will maximize when it is differentiated and then set to 0.
So, P = constant – P = R
Therefore, P=R.

Problem 2: Digital Half-toning

(a) Dithering:

1. ABSTRACT & MOTIVATION

Halftoning is nothing but the technique that simulates the continuous shades of gray by varying the size of the tiny black dots and varying spacing, thus generating a gradient-like effect. Digital Halftoning in representation of image with just two intensities, that is either black or white tones. This technique is basically used to create illusion of continuous tone for binary devices such as printer or HD display using the quantization technique. As we know that the human eye perceives the object by performing the spatial averaging. But the images generated by the binary devices are in the form of dots per inch (dpi) with varying size and spacing of dots and with only two tones that is either black or white thus, approximating the various gray levels, so for better perception of images generated by binary devices Digital Halftoning plays an important role to convert a gray scale image to halftoned image.
The three types of methods used to convert the gray scale images into half toned images are:

a. Fixed Thresholding
b. Random Thresholding
c. Dithering Halftone Matrix

2. APPROACH & PROCEDURE

Here I describe the basic approach and procedure for three techniques of digital halftoning to convert the gray scale image into halftoned image in detail.

EE569 Digital Image Processing

a. Fixed Thresholding:

In this approach of Fixed Thresholding, a threshold value T, is set to divide the 256 intensity levels into two different ranges. So basically, first we load the input grayscale image, then we chose one value T, as the threshold value to divide the 256 intensity gray levels into two different ranges. So, for each pixel value, if the value is greater than the threshold value then it is mapped to 255 and if the pixel value is less than the threshold value then it is mapped to 0. Mathematically, it is expressed as,

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

b. Random Thresholding:

In this approach of Random Thresholding, in order to break the monotones in the outputs from the fixed thresholding we use a random threshold value T, and it is chosen at random between 0-255 in order to divide the 256 gray intensity levels into two different ranges. So, at first, we load the input grayscale image, then for each pixel(i,j) a random number is generated in the range of 0~255, *rand(i,j)*. Next, we compare each pixel value with the *rand(i,j).* So, if the pixel value is greater than the random threshold value generated for each pixel then map it to 255, or else map it to 0. Mathematically, it is expressed as

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

The choice of random threshold value be from uniformly distributed random variables.

c. Dithering Halftone Matrix:

Dithering is another technique used for converting grayscale images into digital halftone image. In simple terms, Dithering can be thought of as thresholding of an input grayscale image with a dither matrix. Dithering parameters are specified by an index matrix. The matrix is convoluted repeatedly over the entire input image. The values of an index matrix indicate that how likely the dot will be turned on. Like for example, an index matrix is given by

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

Where 0 indicates the pixel that is the most likely to get turned on, while 3 is the least likely one to get turned on. Basically, this index matrix belongs to the special case of family of Dithering matrices known as Bayer index matrices.
The algorithm used for this technique is as follows:

Step 1: Initially we load the input grayscale image to convert it into halftoned image.

Step 2: The Bayer index matrices are generated recursively using the formula given below:

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

Here we create I_2, I_8 and I_{32} threshold matrices and apply them to halftone th given input image.

Step 3: Now we transform this index matrix into the threshold matrix T for an input grayscale image with the normalized pixel values by using the following formula:

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

Where N^2 denotes the total number of pixels present in the threshold matrix & (x, y) denotes the location of the pixel in the matrix.

Step 4: As the input image is usually larger than the size of the threshold matrix, the threshold matrix is recursively repeated over the entire image using the formula given below:

$$G(i,j) = \begin{cases} 0 & if\ F(i,j) \le T(i\ mod\ N, j\ mod\ N) \\ 255 & otherwise \end{cases}$$

Step 5: So if the given pixel value is greater than the threshold value then it is map to 255 or else if the given pixel value is less than the threshold value then it is map to 0.

3. EXPERIMENTAL RESULTS

Fig. 2.1

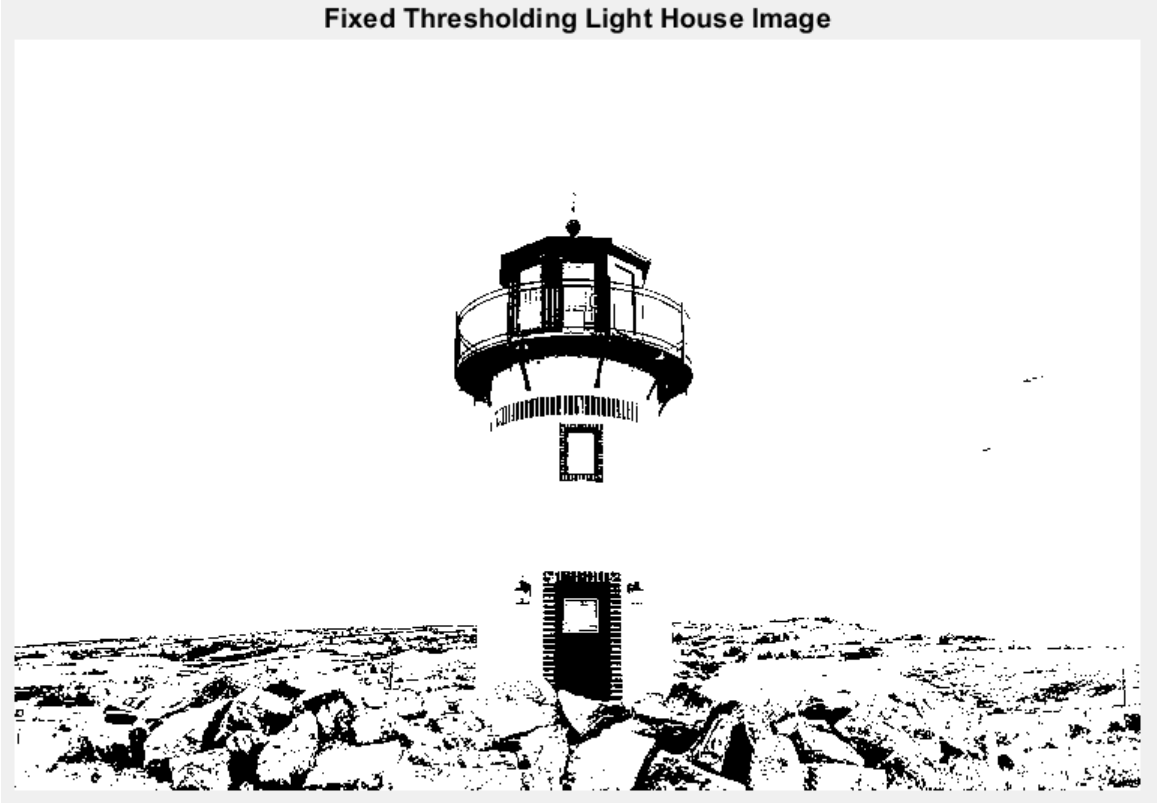


Fig. 2.2 (Threshold=70)

Fig. 2.3 (Threshold=128)

Fig. 2.4 (Threshold = 160)

Fig. 2.5

	1	2
1	1	2
2	3	0

Fig.2.6 (I2 matrix)

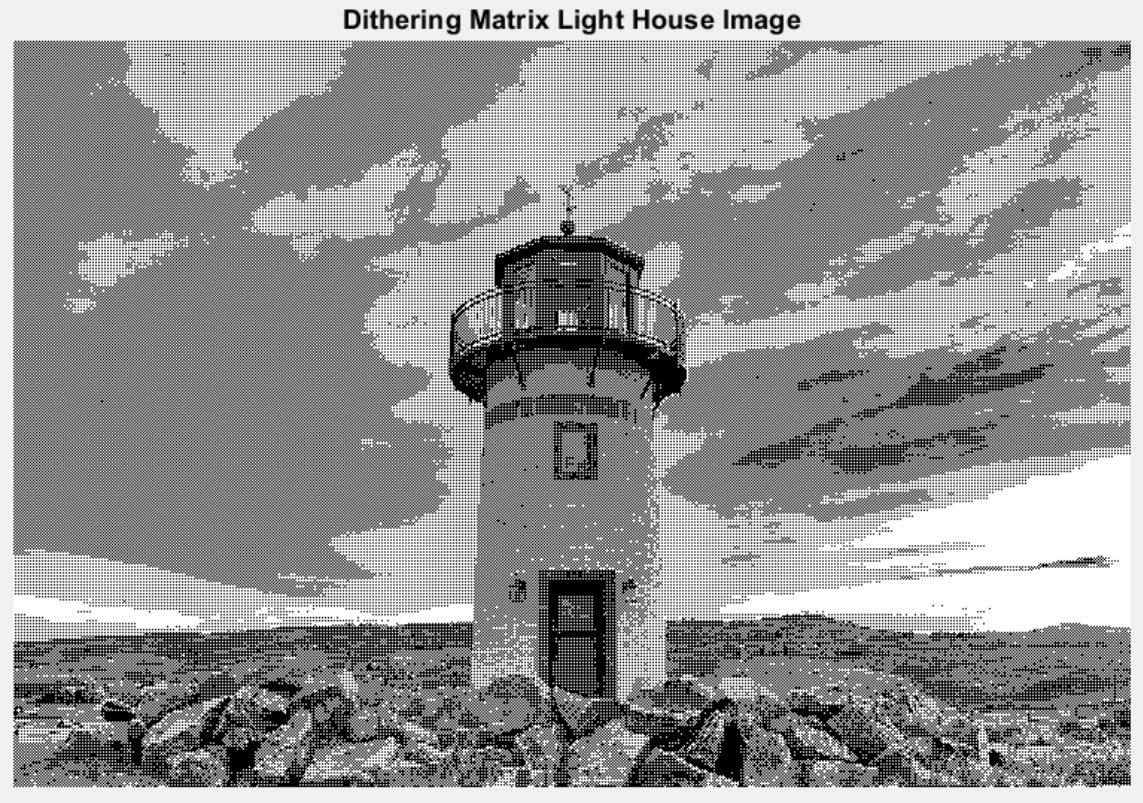


Fig. 2.7

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

Fig. 2.8 (I8 matrix)

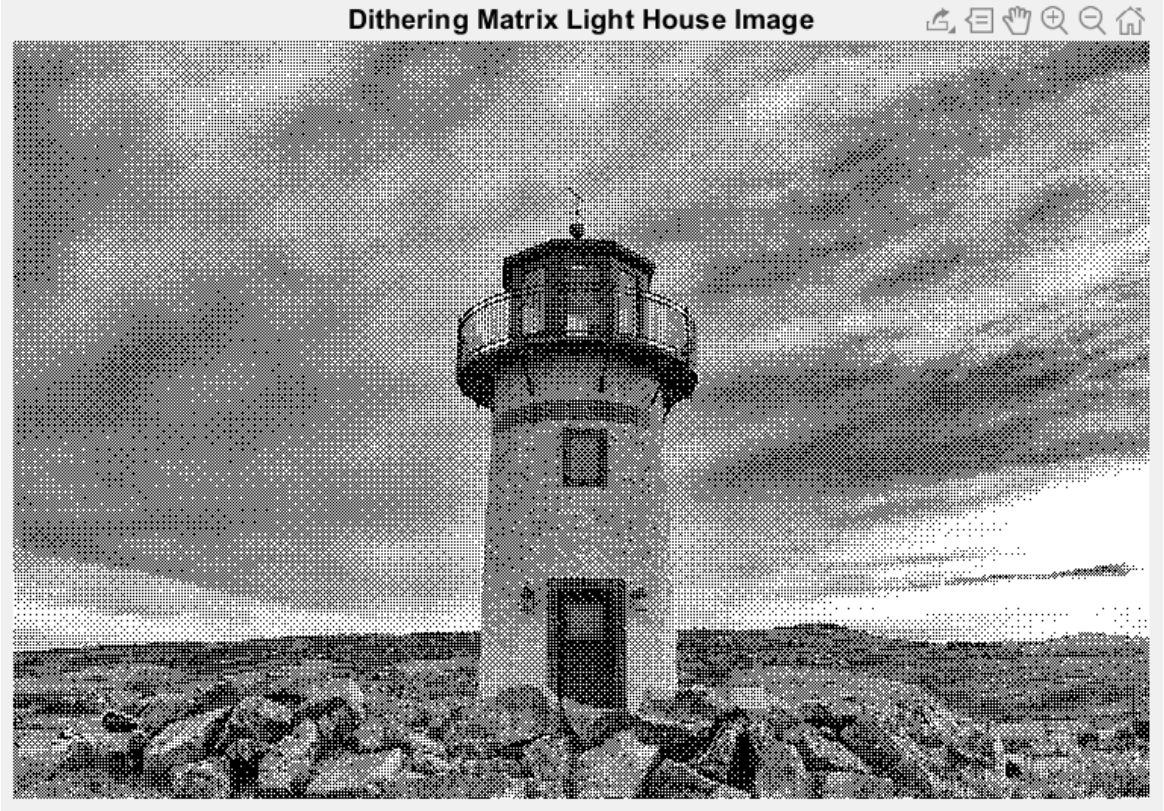


Fig. 2.9

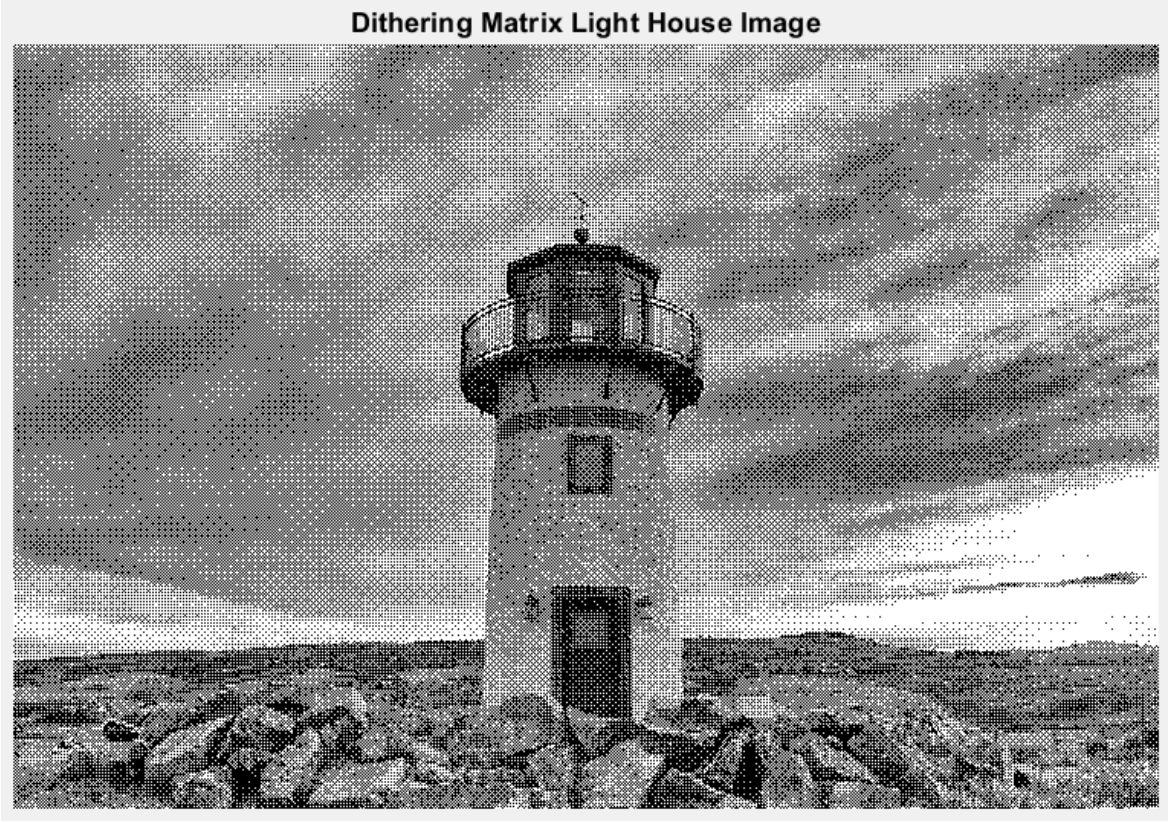


Fig. 2.10

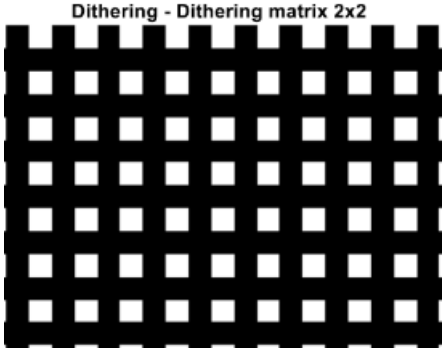
4. DISCUSSION

- As we can see that on comparing the results among the above three methods namely, the Fixed Thresholding, Random Thresholding and the Dithering Matrix (I2, I8, I32) the best output is given by the Dithering matrix I32 as it preserves the fine details and textures of an image as in the edges, contrast, even toning of patches and boundaries, but the index matrix size should not be too large otherwise it spreads out the fine details of an image.
- While in Fixed thresholding the image contains a greater number of white pixels if the threshold is lower than 50% and it contains more amount of black pixel if the threshold is more than the 50%.
- In random thresholding the image looks more noisier due to random noise present in it and it is difficult the identify the texture of image and the boundary details of objects in it. Thus, these artifacts are resolved by the Dithering matrix based halftoning.

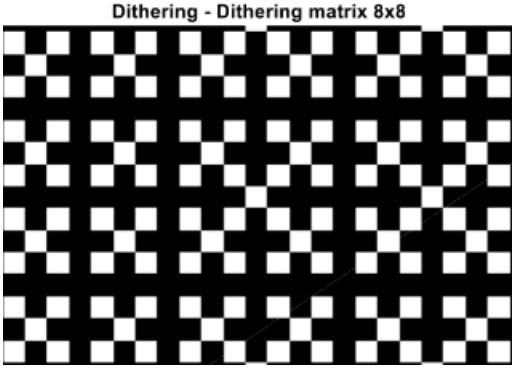
Artifacts or pattern observed:

Dithering - Random Thresholding

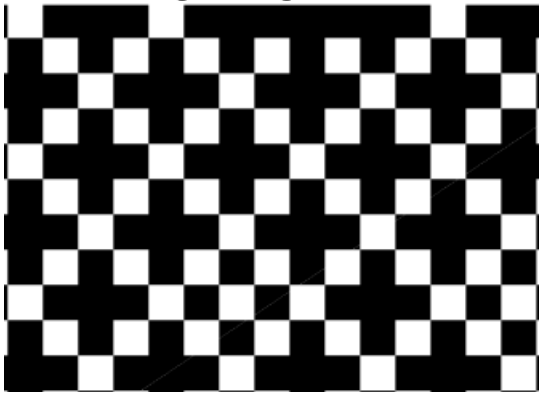
Dithering - Dithering matrix 2x2



Dithering - Dithering matrix 8x8



Dithering - Dithering matrix 32x32



(b) Error Diffusion:

1. ABSTRACT & MOTIVATION

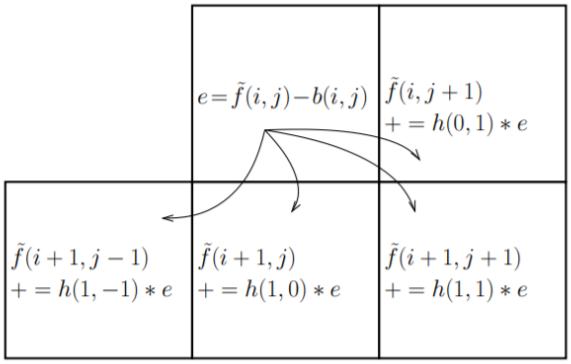
Error diffusion is nothing, but another kind of technique used to produce the digital halftoned image from the input grayscale image. It is also known as spatial dithering. The basic idea of Error Diffusion is that, in this method the error diffusion traverses sequentially each pixel of an input image and each pixel of an input image is compared with a certain threshold value. If the given pixel value is greater than the threshold value, then it is mapped to 255 or else it is mapped to 0. The error difference obtained between the input image pixel and the output value is dispersed to the nearby neighboring pixels.
Error diffusion is basically a neighborhood process because it not only operates on the input image pixel but also on its neighbors. Usually, the neighborhood operations yield the higher quality results as compared to that of the point operations. There are 3 different methods used to perform error diffusion with various types of scanning order. But her we will follow the Serpentine Scanning approach.

2. APPROACH & PROCEDURE

The Error Diffusion Algorithm using Serpentine scanning using 3 different approaches is as follows:
Step 1: At first, we load an input grayscale image. Then let F(i,j) be the pixel intensity value at i^{th} row and j^{th} column of an input grayscale image and let G(i,j) be the halftoned output image.

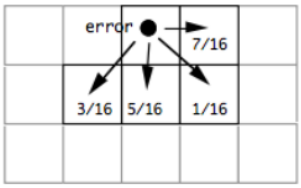
Step 2: Next, each (i,j) pixel value is compared with the input threshold value to binarize the pixel to either to 0 level or 255 level. That means is the pixel value at (i,j) is less than the set threshold value then it is mapped down to 0 or else if it is greater than the set threshold value then it is mapped to 255.

Step 3: Then the quantization error is computed by evaluating the difference between the input grayscale image pixel intensity value and the new binarized pixel intensity value. This error is then diffused to the neighboring pixels to the right as well as to the bottom of the current pixel with different weights depending on the type of Error Diffusion technique implemented. The pictorial representation of the error diffusing scheme is as follows:



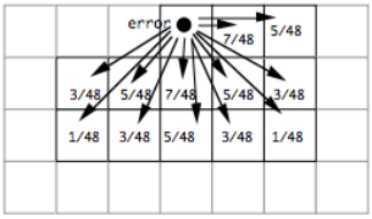
- **Floyd-Steinberg**

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



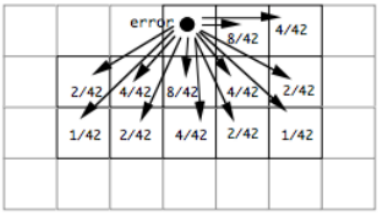
- **JJN**

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



- **Stucki**

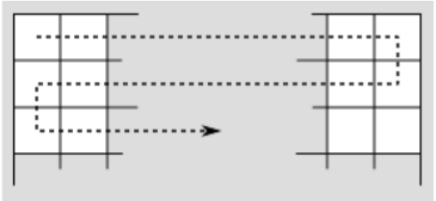
$$\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$



Step 4: Depending upon the scanning order, the error diffusion matrix is mirrored accordingly.

Step 5: Here in this approach I have adopted the Serpentine Scanning method.

- **Serpentine parsing**



In Serpentine Scanning approach for every odd i^{th} row, the scanning is performed in the fashion from left to right (j=1:1:column) by convoluting the error diffusion mask with the underlying pixels of the input grayscale image. The logic for 4 boundary conditions and the 4 corners of an input image are taken care of while convolving with the error diffusion matrix by using alternatingly if-else loop statements both for rows as well as columns.

Step 6: Now, for every even i^{th} row, the scanning is performed in the fashion from right to left (j=column:-1:1) by convoluting the error diffusion mask with the underlying pixels of the input grayscale image. The logic for 4 boundary conditions and the 4 corners of an input image are taken care of while convolving with the error diffusion matrix by using alternatingly if-else loop statements both for rows as well as columns.

Step 7: Thus, now after performing the above steps the G(i,j) is the final resultant halftoned image obtained by error diffusion technique with 3 different types of methods.

3. EXPERIMENTAL RESULTS

Fig. 2.11

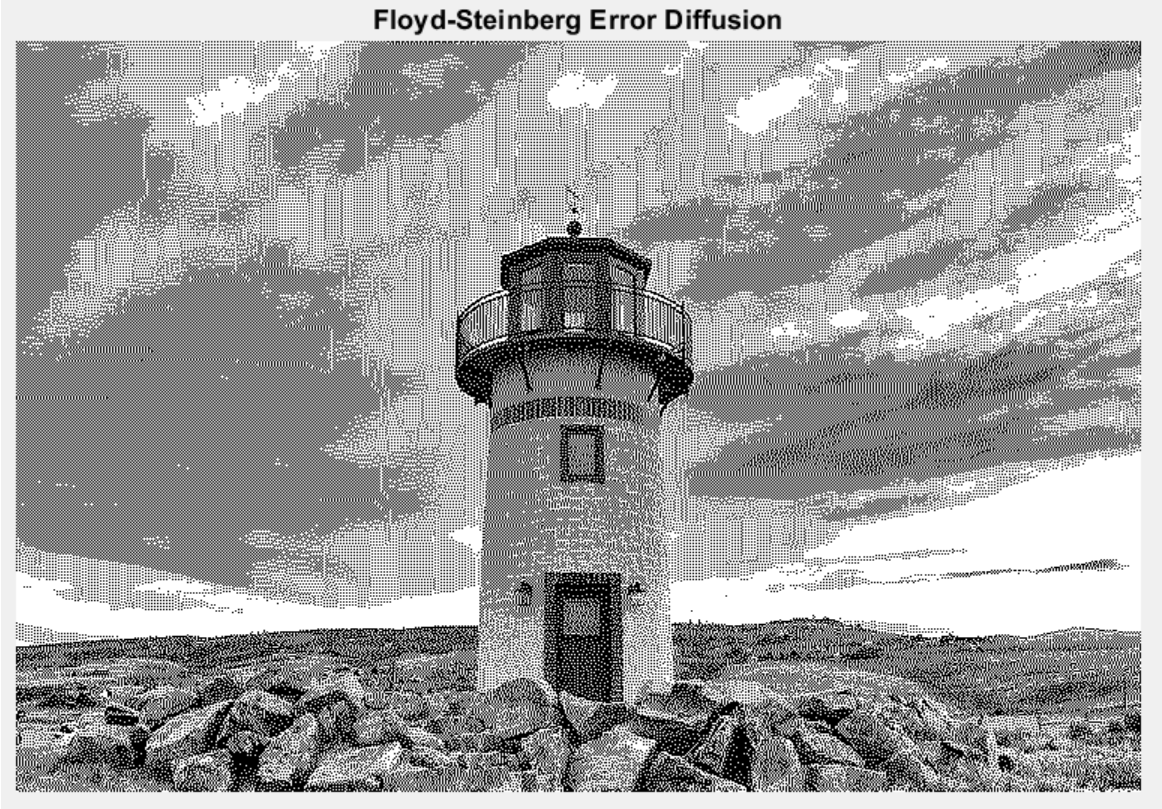


Fig. 2.12

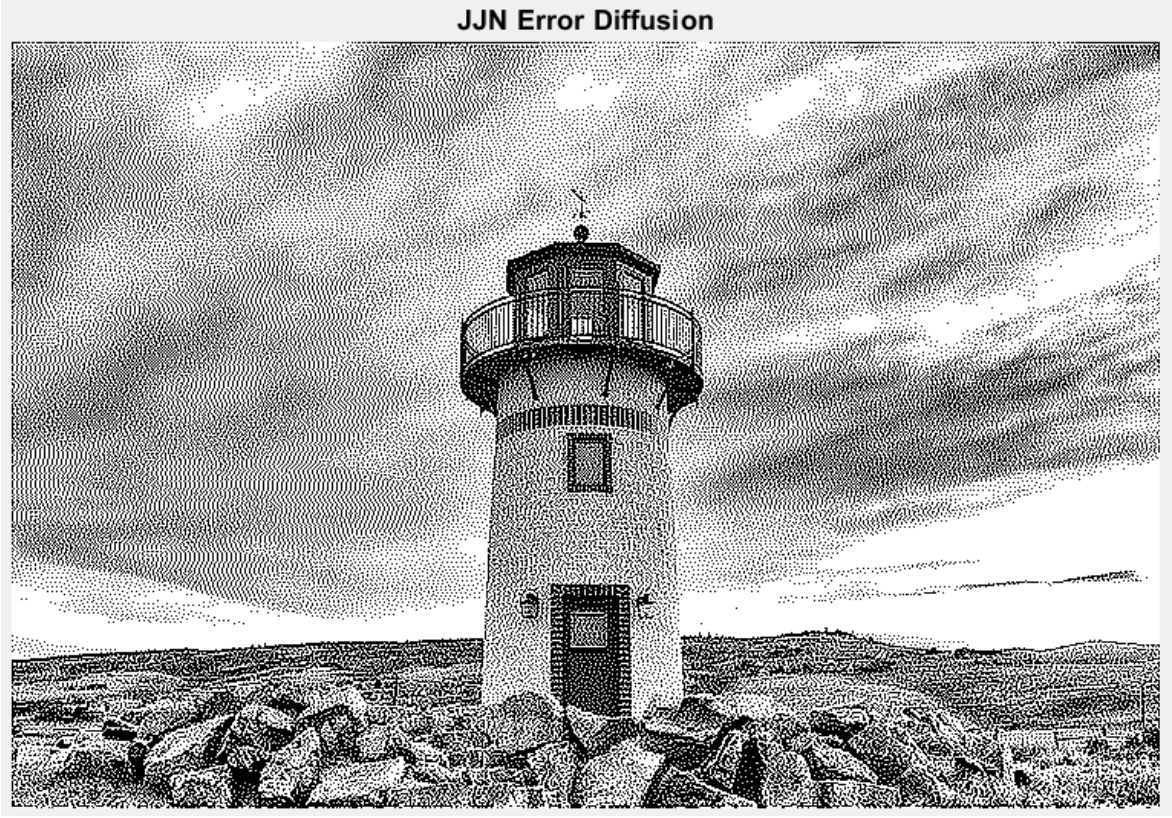


Fig. 2.13

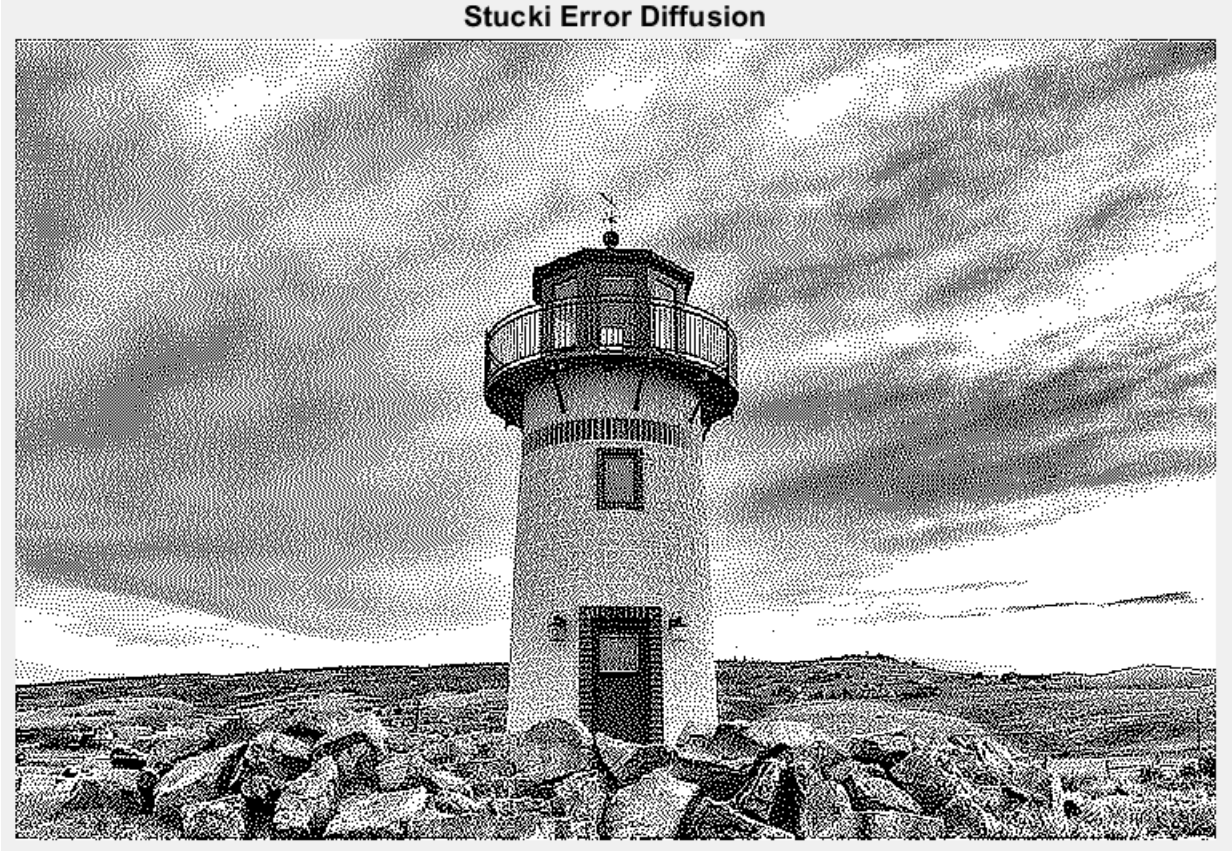


Fig. 2.14

4. DISCUSSION

If we compare the results of Error Diffusion with the Dithering techniques, then the error diffusion does not generate those artifacts as introduced in the fixed thresholding technique. Since in error diffusion technique the errors are diffused to the neighboring pixel locations, thereby making this technique computationally intensive.

Error Diffusion method is better than the Dithering. The output image generated by the error diffusion method is cleaner and more enhanced with fine details in it. As in this technique the error generated at a particular pixel value is diffused among its neighbors thereby it preserves the sharp edges, textures and even toning of grayscale. More quantization error is place along the diagonals in the serpentine scanning. The output generated by Floyd-Steinberg has some artifacts and uneven patches of gray tones which may be due to 3x3 diffusion matrix. As we increase the size of the diffusion matrix and the weights of the diffusion matrix being varied accordingly we achieve better results in terms of edge sharpness preservation, textures, contrast, even toning and fine details in an image as we can see from the output generated by JJN and Stucki Error Diffusion methods. Also, the complexity of JJN and Stucki algorithm is higher than that of Floyd-Steinberg algorithm.

(c) Color Halftoning with Error Diffusion:

1. ABSTRACT & MOTIVATION

Color Halftoning using Error Diffusion is similar to the Error Diffusion technique used for the grayscale image but instead here the same process is carried out on the three channels that is Red, Green and Blue or (Cyan, Magenta, Yellow, Black). Color Halftoning using Error Diffusion reduces the quantization levels. The same algorithm is applied to each channel of a color image to achieve the color effect on the color printers. The motivation for this process is that human vision perceives small error differences of lightness in small local areas rather than the similar differences in same areas. This error difference is then diffused to the neighboring pixels to minimize the quantization error and yield the higher quality results compared to the point operations. The two different approaches adopted here are:

- Separable Error Diffusion
- Minimum Brightness Variation Quadruples (MBVQ)

2. APPROACH & PROCEDURE

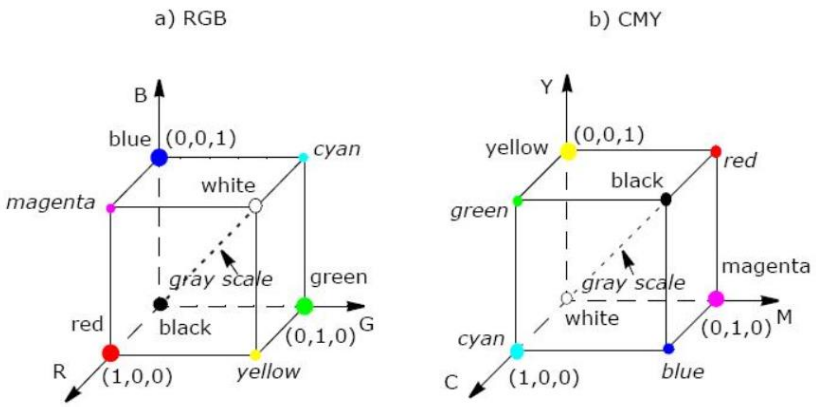
The algorithm for both the methods can be described as follows:

- **Separable Error Diffusion:**

  In this approach of color halftoning we separate the given input image into CMY channels and apply the Floyd-Steinberg Error Diffusion algorithm to quantize each channel separately. The flow of the algorithm is discussed as follows:

  Step 1: First we load the RGB image and separate the image into three color planes namely Red, Green and Blue.

  Step 2: After getting the RGB channels of an original image they are transformed into the CMY channel using the following transformation:



$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Step 3: Now following the same Error Diffusion process as we did for the Floyd-Steinberg algorithm. Here, we have input image as F(i,j) and the halftoned output as G(i,j) . So every time G(i,j) gets updated as the error between the original input image F(i,j) and the G(i,j) is calculated and diffused to the neighboring pixels. Here we employ the Floyd-Steinberg Error Diffusion method along-with Serpentine Scanning Order.

Step 4: Let F(i,j) be the pixel intensity value at i^{th} row and j^{th} column of C channel of CMY model and let G(i,j) be the halftoned output image.

Step 5: Next, each (i,j) pixel value is compared with the input threshold value to binarize the pixel to either to 0 level or 255 level. That means is the pixel value at (i,j) is less than the set threshold value then it is mapped down to 0 or else if it is greater than the set threshold value then it is mapped to 255.

Step 6: Then the quantization error is computed by evaluating the difference between the input image pixel intensity value and the new binarized pixel intensity value for C channel of CMY model. This error is then diffused to the neighboring pixels to the right as well as to the bottom of the current pixel with different weights as there for Floyd-Steinberg Error Diffusion Technique.

Step 7: Depending upon the scanning order, the error diffusion matrix is mirrored accordingly. Here, I am implementing it using the Serpentine Scanning Parsing.

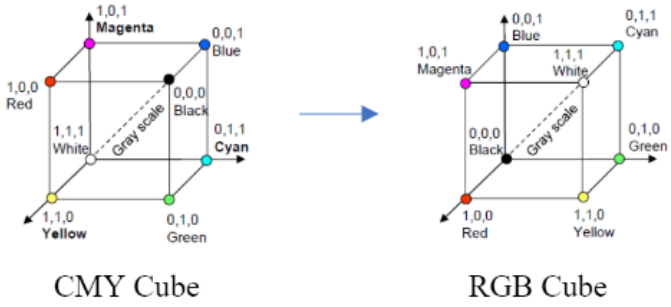
Step 8: In Serpentine Scanning approach for every odd i^{th} row, the scanning is performed in the fashion from left to right (j=1:1:column) by convoluting the error diffusion mask with the underlying pixels of an input image. The logic for 4 boundary conditions and the 4 corners of an input image are taken care of while convolving with

the error diffusion matrix by using alternatingly if-else loop statements both for rows as well as columns.

Step 9: Now, for every even i^{th} row, the scanning is performed in the fashion from right to left (j=column:-1:1) by convoluting the error diffusion mask with the underlying pixels of an input image. The logic for 4 boundary conditions and the 4 corners of an input image are taken care of while convolving with the error diffusion matrix by using alternatingly if-else loop statements both for rows as well as columns.

Step 10: Now G(i,j) is the color halftoned image obtained by Error Diffusion using Floyd-Steinberg algorithm for channel C of the CMY model. The same procedure is repeated for the M as well as Y channel of the CMY model.
Step 11: Now lastly after obtaining the Error Diffusion image for C, M and Y channel of the CMY model, these channels are again transformed back to R, G and B channel respectively. The transformation can be pictorially represented as:

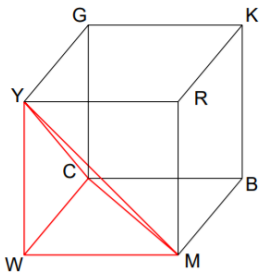
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$



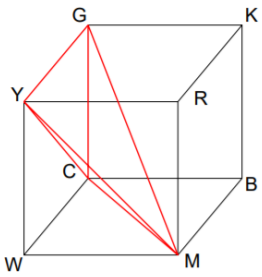
CMY Cube RGB Cube

- **Minimum Brightness Variation Quadruples (MBVQ):**

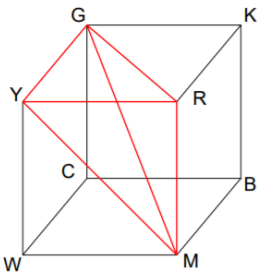
As we know that the Human Visual System is more sensitive to the changes in brightness as compared to that of the chrominance which average at much lower frequencies. Thus, the MBVC criteria states that: "To reduce halftone noise, select from within all halftones sets by which the desired color may be rendered, the one whose brightness variation is minimal". MBVQ based Error Diffusion quantizes the error but preserves the brightness. MBVQ is nothing but basically the transformation of the halftone pattern to preserve the average color, reduce the brightness variation and the number of participating halftone colors to minimum. Thus, then the resulting halftone quadruple is the required halftone set.
The basic approach for this method proceeds with separation of RGB color space into minimum brightness variation quadrants (MBVQ). Then render each input color using one of the six complementary quadruples: RGBK, WCMY, MYGC, RGMY, RGBM, or CMGB each with minimal brightness variation.



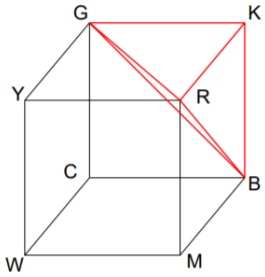
CMYW



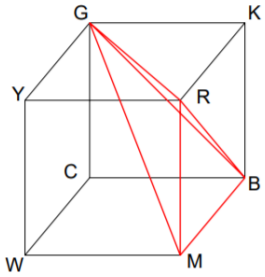
MYGC



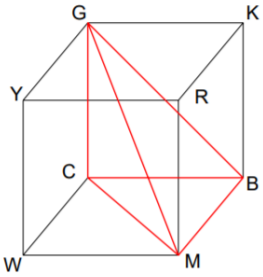
RGMY



KRGB



RGBM



CMGB

The algorithm used for implementing MBVQ based Error Diffusion is as follows:

Step 1: Firstly, we load an input image and for each pixel (i,j), MBVQ(RGB(i,j)) is determined.
Step 2: The quantization error of all the pixel is initialized to zero.
Step 3: Now following the same Error Diffusion process as we did for the Floyd-Steinberg algorithm. Here, we have input image as RGB of F(i,j) and the halftoned output as H(i,j) which is Vector based on MBVQ Quantization. So every time H(i,j) gets updated as the error between the original input image RGB of F(i,j) and the H(i,j) is calculated and diffused to the neighboring pixels. Here we employ the Floyd-Steinberg Error Diffusion method along-with Serpentine Scanning Order.
Step 4: Next, this is the most crucial step of this technique. In this step we find the MBVQ tetrahedron which is closet to RGB(i,j) thereby limiting the pixel to one quadrant from RGBK, WCMY, MYGC, RGMY, RGBM, or CMGB by the following the given algorithm:

```
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;
}
```

Step 5: Next we determine the closet vertex, V, of the MBVQ tetrahedron which is closet to the RGB of F(i,j) thereby limiting the pixel to anyone of the 8 colors namely, 'White', 'Black', 'Red', 'Green', 'Blue', 'Yellow', 'Cyan', 'Magenta' with the help of the getNearestVertex() function.

Step 6: In this step we compute the quantization error RGB of F(i,j)-H(i,j).

Step 7: Depending upon the scanning order, the error diffusion matrix is mirrored accordingly. Here, I am implementing it using the Serpentine Scanning Parsing.

Step 8: In Serpentine Scanning approach for every odd i^{th} row, the scanning is performed in the fashion from left to right (j=1:1: column) by convoluting the error diffusion mask with the underlying pixels of an input image. The logic for 4 boundary conditions and the 4 corners of an input image are taken care of while convolving with the error diffusion matrix by using alternatingly if-else loop statements both for rows as well as columns.

Step 9: Now, for every even i^{th} row, the scanning is performed in the fashion from right to left (j=column: -1:1) by convoluting the error diffusion mask with the underlying pixels of an input image. The logic for 4 boundary conditions and the 4 corners of an input image are taken care of while convolving with the error diffusion matrix by using alternatingly if-else loop statements both for rows as well as columns.

Step 10: Hence, we obtain the H(i,j) as the final halftoned image using the MBVQ Error Diffusion technique.

3. EXPERIMENTAL RESULTS

Fig. 2.15

Fig. 2.16

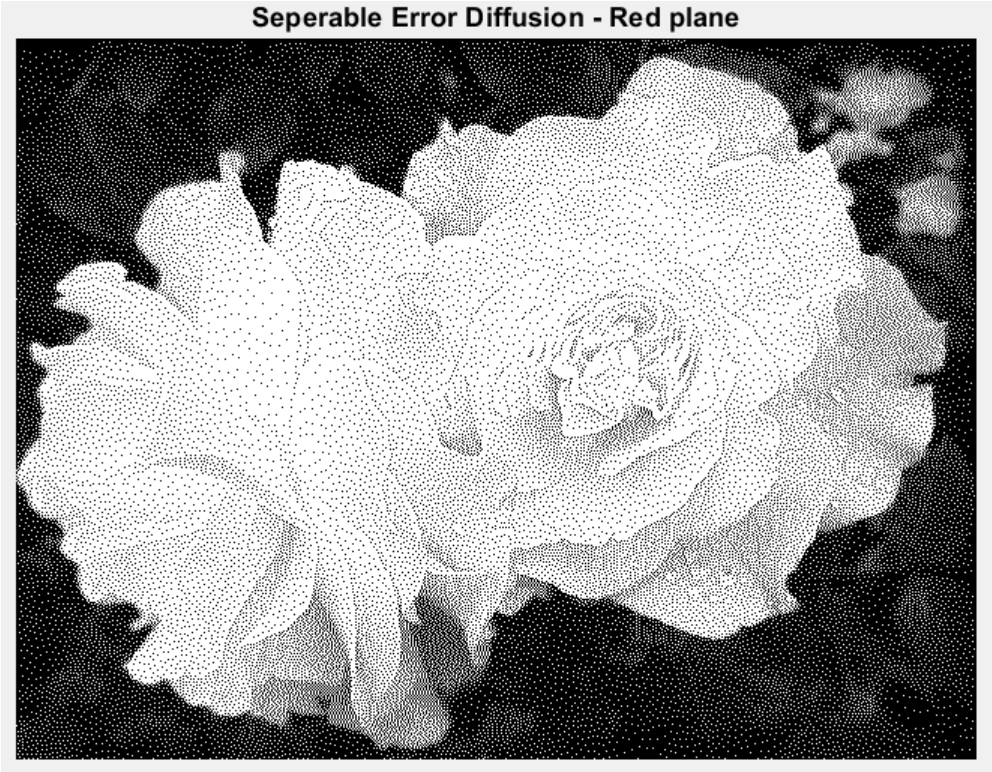


Fig. 2.17

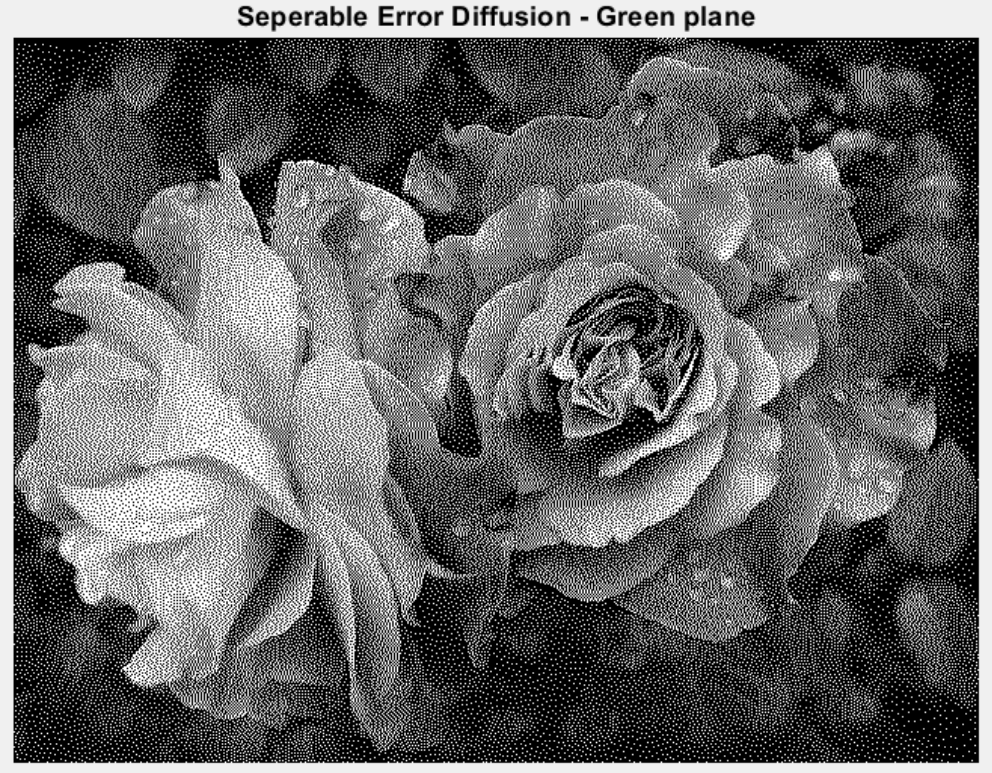


Fig. 2.18

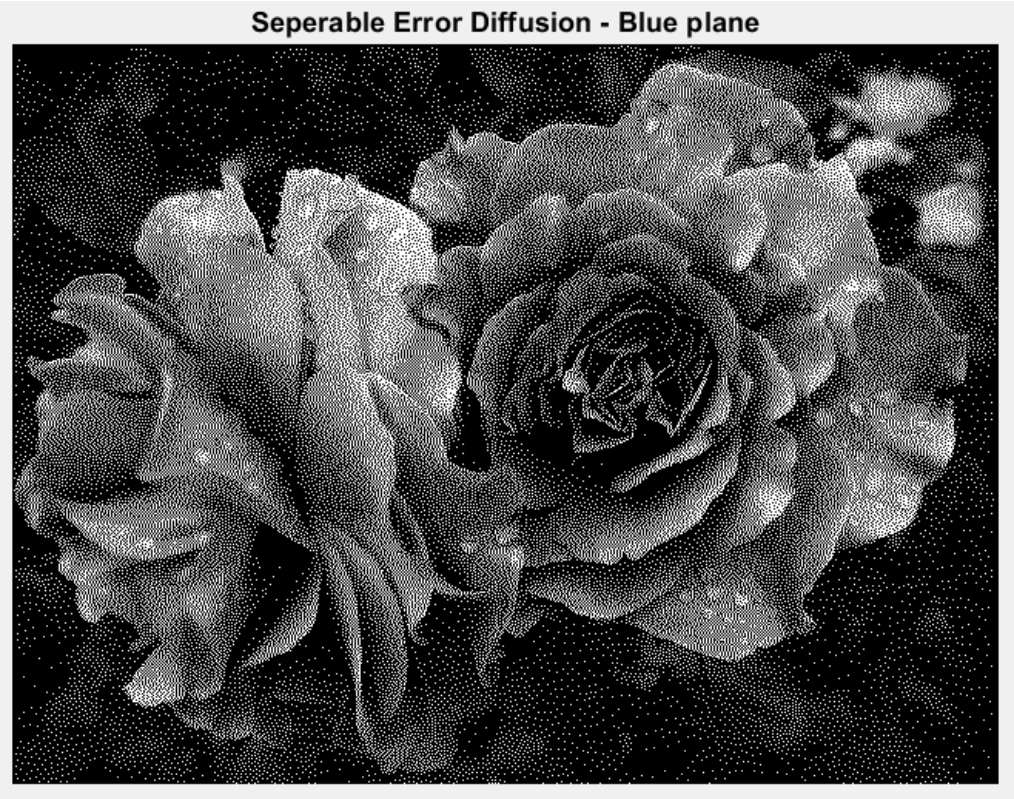


Fig. 2.19

Fig. 2.20

Fig. 2.21

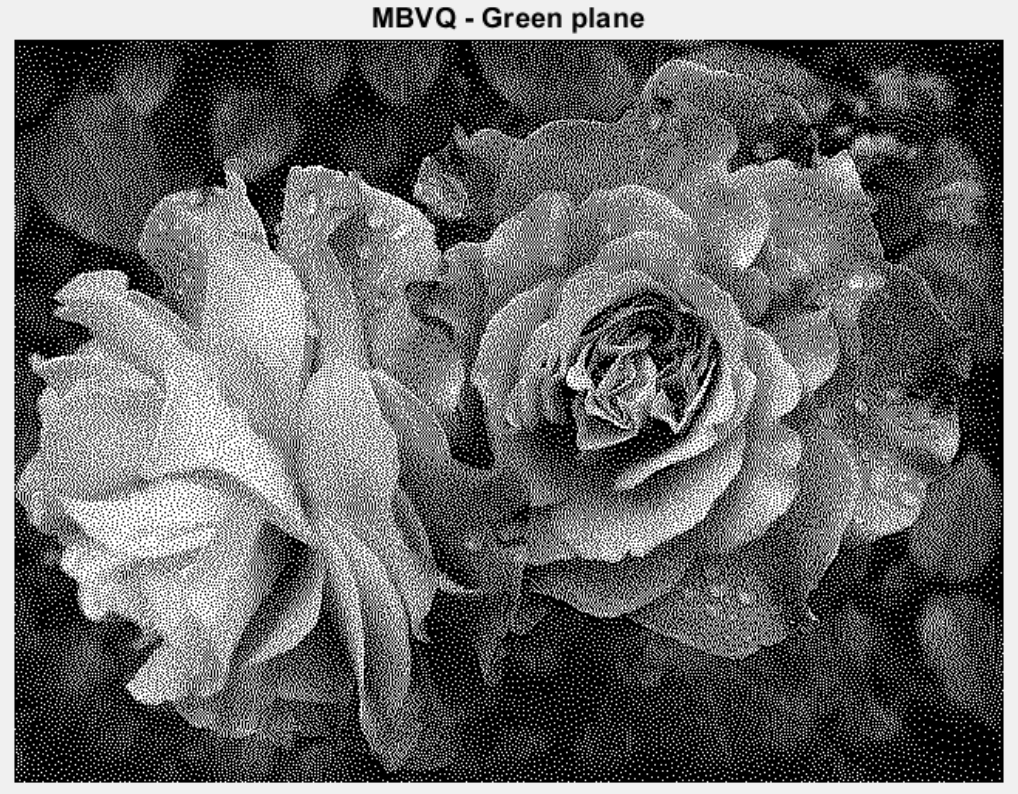


Fig. 2.22

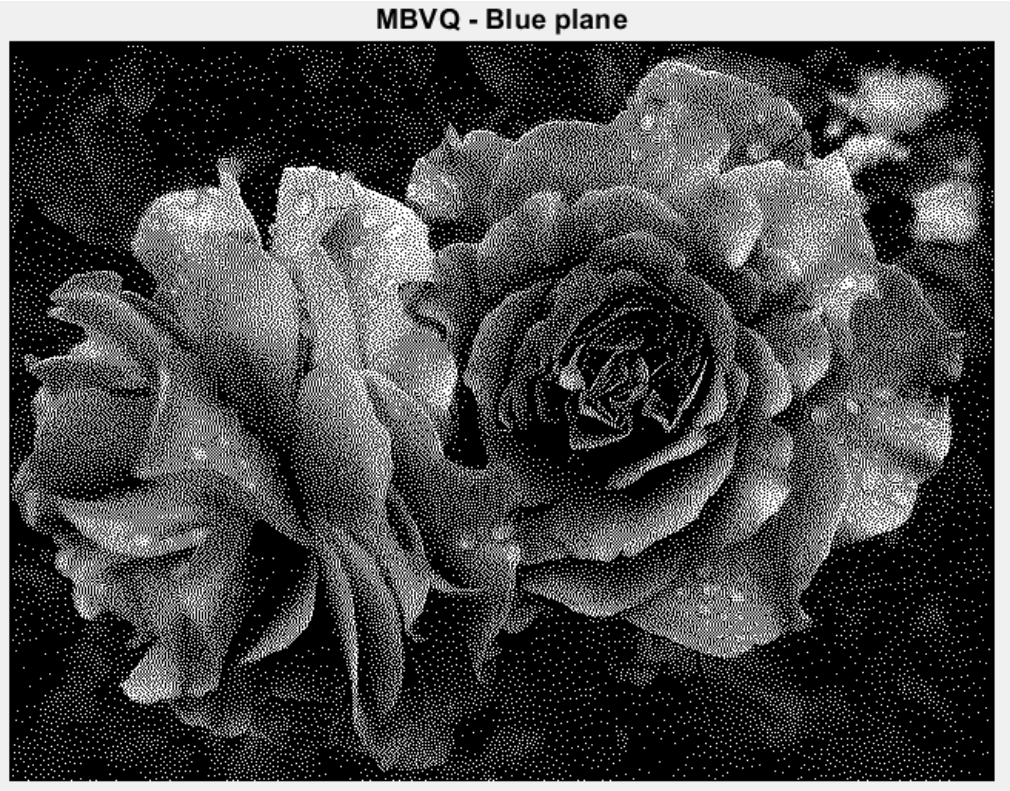


Fig. 2.23

Fig. 2.24

4. DISCUSSION

In this problem if we compare the two methods of color halftoning the MBVQ yields the better results as compared to separable error diffusion method. Although separable error diffusion methods give good estimation of channels and correlation of luminance of three channels. But both the methods have their own shortcomings which are discussed as below:

Separable Error Diffusion:

1) Computationally it is very intensive.
2) The Green pixels are higher as compared to other.
3) Poor image quality as it has more artifacts.
4) Does not display an output image with optimum brightness.
5) Noise present in the output image is high.

MBVQ:

1) As compared to Separable Error Diffusion method the output image generated is of higher quality as artifacts are minimized by MBVQ technique.
2) There are a smaller number of Green dots present with minimum brightness variation.
3) MBVQ minimizes the variation in brightness of the pixels.
4) Halftone noise is also reduced as compared to that in case of Separable Error Diffusion method.
5) Spatial domain correlation is implemented to determine MBVQ quadruple as all the red, green and blue pixel intensities either being at same location or different location are considered together.
6) MBVQ preserves color and brightness of an image.

Appendix:

Gallery

1) Recall

7.095724e-01

7.081011e-01

7.433234e-01

7.952651e-01

8.580409e-01

8.911677e-01

9.176067e-01

9.296001e-01

9.390075e-01

9.422177e-01

9.397209e-01

9.364216e-01

9.284854e-01

9.232244e-01

9.143074e-01

9.057470e-01

8.913906e-01

8.790851e-01

8.649516e-01

8.452450e-01

8.304427e-01

8.139462e-01

7.904053e-01

7.735521e-01

7.565652e-01

7.416737e-01

7.258906e-01

7.090374e-01

6.877703e-01

6.664140e-01

6.503634e-01

6.323956e-01

6.138036e-01

5.975300e-01

5.841099e-01

5.682821e-01

5.572250e-01

5.442507e-01

5.340408e-01

5.225824e-01

5.119265e-01

4.993981e-01

4.910607e-01

4.794240e-01

4.645771e-01

4.503545e-01

4.412591e-01

4.321191e-01

4.179856e-01

4.077311e-01

3.955593e-01

3.818717e-01

3.741585e-01

3.603816e-01

3.516875e-01

3.439297e-01

3.334522e-01

3.235543e-01

3.149494e-01

3.042935e-01

2.935931e-01

2.844086e-01

2.704534e-01

2.542244e-01

2.368808e-01

2.206072e-01

2.004548e-01

1.874805e-01

1.718311e-01

1.607294e-01

1.491819e-01

1.403540e-01

1.302778e-01

1.201569e-01

1.114183e-01

1.034375e-01

9.394088e-02

8.315128e-02

7.262919e-02

6.192875e-02

5.399260e-02

4.819653e-02

4.070623e-02

3.459807e-02

2.782112e-02

2.269383e-02

1.810156e-02

1.399973e-02

1.119087e-02

7.802399e-03

6.241919e-03

5.127291e-03

4.458514e-03

2.229257e-03

6.687770e-04

4.458514e-04

0

0

0

2) Precision

4.884125e-01

5.809513e-01

6.609575e-01

7.078791e-01

7.513935e-01

7.785150e-01

7.954799e-01

8.090438e-01

8.285959e-01

8.367259e-01

8.566155e-01

8.704742e-01

8.849294e-01

8.924697e-01

9.054342e-01

9.155167e-01

9.197999e-01

9.240529e-01

9.277087e-01

9.316366e-01

9.363239e-01

9.436703e-01

9.487860e-01

9.524411e-01

9.543784e-01

9.560098e-01

9.600548e-01

9.629021e-01

9.650570e-01

9.643395e-01

9.623808e-01

9.645900e-01

9.677686e-01

9.693617e-01

9.696793e-01

9.699700e-01

9.704615e-01

9.708492e-01

9.733853e-01

9.753580e-01

9.770940e-01

9.768502e-01

9.777778e-01

9.800664e-01

9.805344e-01

9.809750e-01

9.820993e-01

9.841402e-01

9.851787e-01

9.856180e-01

9.851715e-01

9.864800e-01

9.880597e-01

9.875195e-01

9.887279e-01

9.905921e-01

9.913793e-01

9.910661e-01

9.907350e-01

9.916560e-01

9.926421e-01

9.923823e-01

9.919883e-01

9.914463e-01

9.924937e-01

9.919355e-01

9.911067e-01

9.925453e-01

9.917647e-01

9.911616e-01

9.917582e-01

9.926579e-01

9.936306e-01

9.982699e-01

1

1

1

1

1

1

EE569 Digital Image Processing

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

NaN

NaN

NaN

DOGS

1) **Recall**
   7.618878e-01
   8.184161e-01
   8.285486e-01
   8.313928e-01
   8.233046e-01
   8.081059e-01
   8.025064e-01
   7.894409e-01
   7.701538e-01

7.512221e-01
7.372678e-01
7.161141e-01
6.960270e-01
6.648298e-01
6.449205e-01
6.246556e-01
5.982579e-01
5.778153e-01
5.427073e-01
5.220869e-01
4.984446e-01
4.797796e-01
4.581815e-01
4.346280e-01
4.221847e-01
4.010310e-01
3.775664e-01
3.597014e-01
3.465470e-01
3.293929e-01
3.170385e-01
3.050396e-01
2.938405e-01
2.846858e-01
2.725091e-01
2.564217e-01
2.436228e-01
2.270020e-01
2.170474e-01
2.061150e-01
1.950938e-01
1.864723e-01
1.774065e-01
1.685184e-01
1.598969e-01
1.547418e-01
1.505644e-01
1.432762e-01
1.368767e-01
1.320771e-01
1.267443e-01
1.213225e-01
1.172340e-01

1.115456e-01
1.052351e-01
1.008799e-01
9.394720e-02
8.950316e-02
8.265932e-02
7.839303e-02
7.448227e-02
7.146031e-02
6.559417e-02
6.097236e-02
5.732824e-02
5.341747e-02
4.852902e-02
4.310728e-02
3.946316e-02
3.670785e-02
3.350813e-02
2.755311e-02
2.533108e-02
2.355346e-02
2.133144e-02
1.866501e-02
1.688739e-02
1.555417e-02
1.155453e-02
8.443694e-03
5.777264e-03
3.999644e-03
1.777620e-03
1.777620e-03
1.333215e-03
8.888099e-04
8.888099e-04
8.888099e-04
0
0
0
0
0
0
0
0
0

0
0

2) **Precision:**
2.858216e-01
3.541493e-01
4.236018e-01
4.713264e-01
5.264365e-01
5.752559e-01
6.069136e-01
6.401143e-01
6.648544e-01
6.922750e-01
7.292637e-01
7.602443e-01
7.791176e-01
8.061968e-01
8.290344e-01
8.408366e-01
8.536027e-01
8.584753e-01
8.632138e-01
8.696268e-01
8.733536e-01
8.826393e-01
8.944928e-01
8.983791e-01
9.031839e-01
9.032481e-01
9.086142e-01
9.090184e-01
9.164578e-01
9.202128e-01
9.240741e-01
9.338521e-01
9.334006e-01
9.379600e-01
9.397321e-01
9.369798e-01
9.374218e-01
9.379217e-01
9.427754e-01

9.400922e-01
9.393443e-01
9.506173e-01
9.550562e-01
9.537223e-01
9.590517e-01
9.618834e-01
9.857482e-01
9.923469e-01
9.919786e-01
9.972067e-01
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

1
1
1
1
1
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN

References:

1. https://www.imageeprocessing.com/2011/12/sobel-edge-detection.html
2. https://en.wikipedia.org/wiki/Ordered_dithering
3. https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900
4. https://www.cs.cmu.edu/afs/cs/academic/class/15462-s09/www/lec/24/lec24.pdf
5. https://en.wikipedia.org/wiki/Sobel_operator
6. https://en.wikipedia.org/wiki/Canny_edge_detector
7. https://aishack.in/tutorials/canny-edge-detector/
8. https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html
9. https://en.wikipedia.org/wiki/Halftone#Digital_halftoning
10. http://www.ece.ubc.ca/~irenek/techpaps/introip/manual04.html
11. https://scc.ustc.edu.cn/zlsc/sugon/intel/ipp/ipp_manual/IPPI/ippi_ch6/ch6_color_models.htm
12. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.927&rep=rep1&type=pdf

EE569 Digital Image Processing