**Name: Shreya Kate**

**USC ID: 2334973997**

**Email: shreyak@usc.edu**

**Submission Date: 02/16/2020**

**HOMEWORK #2**

**Issued: 01/29/2020** **Due: 02/16/2020**

**Problem 1:**

**(a) Sobel Edge Detector**

**For Gallery.raw:**

**Threshold = 127**

**(1)** I normalized the values of x- and y-gradient. The results can be seen in Fig. 1 and Fig 2:

**Results:**


Fig. 1: normalized x-gradient value for Gallery.raw


Fig. 2: normalized y-gradient value for Gallery.raw

**(2)** The normalized gradient magnitude map can be seen in Fig. 3:


Fig. 3: normalized gradient magnitude map for Gallery.raw

**(3)** The final edge map can be seen in Fig. 4:

The value of threshold is set to 127 for the Gallery.raw image. In terms of percentage this value is 50%.


Fig. 4: final edge map for Gallery.raw

**For Dogs.raw:**

**Threshold = 150**

**(1)** I normalized the values of x- and y-gradient. The results can be seen in Fig. 5 and Fig 6:

**Results:**

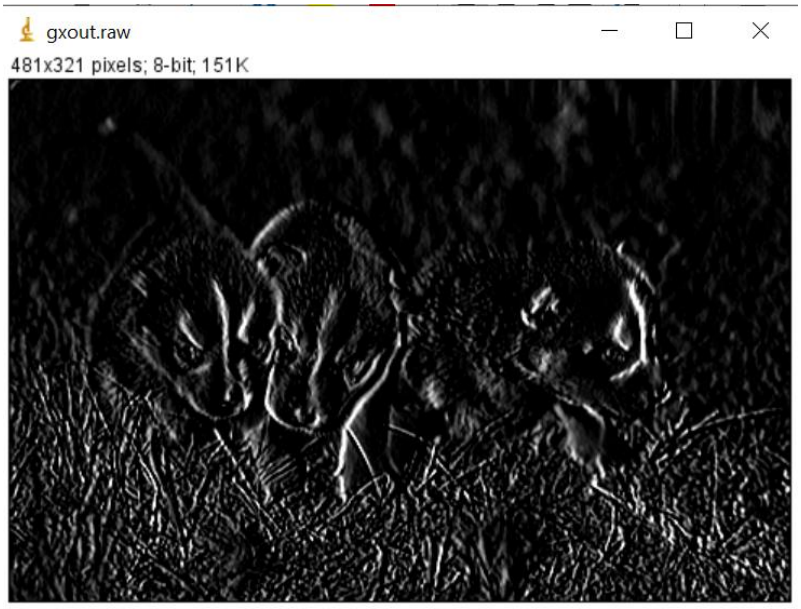



Fig. 1: normalized x-gradient value for Dogs.raw


Fig. 6: normalized y-gradient value for Dogs.raw

**(2)** The normalized gradient magnitude map can be seen in Fig. 7:


Fig. 7: normalized gradient magnitude map for Dogs.raw

**(3)** The final edge map can be seen in Fig. 8:

The value of threshold is set to 150 for the Dogs.raw image. In terms of percentage this value is 59%.

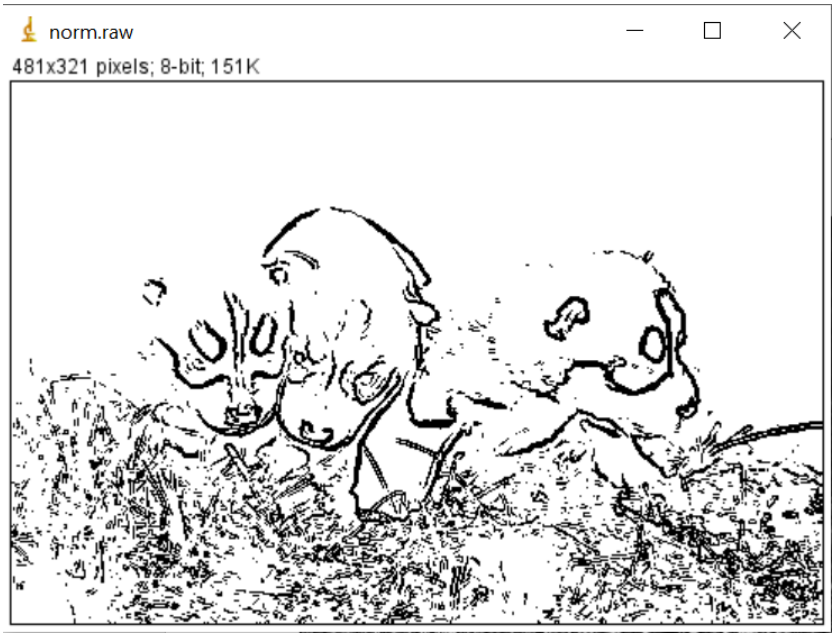



Fig. 8: final edge map for Dogs.raw

**(b) Canny Edge Detector**

**(1) Non maximum suppression**

Non-maximum suppression is used for thinning the edges after they have been detected. This is because the edge is still quite thick after it is extracted from the gradient. In NMS, the pixel values of the pixels before and after the pixel at which we are is considered and compared. In NMS the pixel at which there is the sharpest change in intensity (highest gradient value) is considered and all the other edge pixels are suppressed and set to 0.

The algorithm for NMS is as follows:

1. The gradient value of the pixels in the positive and negative gradient directions is compared.

2. If the gradient value is the highest compared to the other two values then it is preserved, otherwise it is suppressed.

By implementing this algorithm, the unnecessary edge pixels are suppressed which gives a thin, clear and precise edge.

**(2)**

After NMS, we get a better idea about the edges in the image. The image that we see after NMS, some edges are brighter and are stronger edges while some edges are fainter and are weaker edges. So, Canny edge detection uses hysteris thresholding in which we have 2 threshold values. One is called the High threshold and one is the Low threshold.

For edge detection, the values of intensity that are higher than the High threshold value are considered to be edges while the values of intensity lower than the Low threshold value are not considered to be edges. The pixels with intensity values between the High and Low thresholds are considered to be edges if they are connected to pixels that are edges. Otherwise they are suppressed.

**(3)**

**For Gallery.jpg**

**1. Threshold value = 0.2 (best)**

This is the best because only the necessary edges are detected. (Fig. 9)

Most of the important edges are detected with this threshold value.

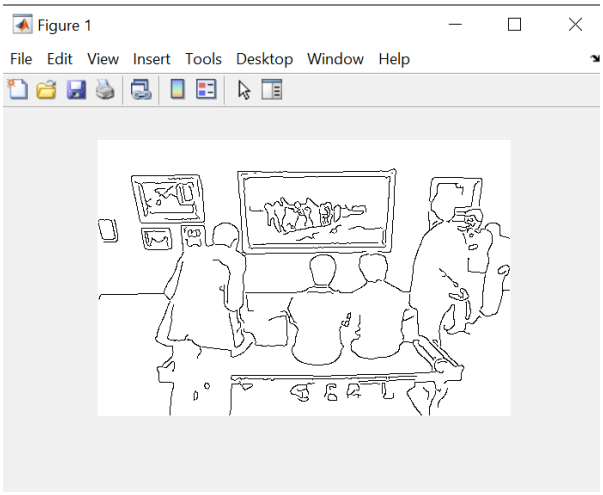



Fig. 9: Canny edge detection with threshold = 0.2

**2. Threshold value = 0.1**

A lot of unnecessary edges are detected in this image. (Fig. 10)

This happens when we decrease the threshold value below 0.2.

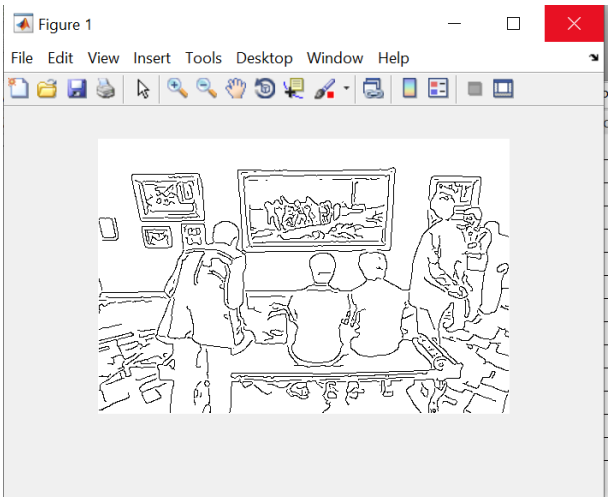



Fig. 10: Canny edge detection with threshold = 0.1

### 3. Threshold value = 0.5

Most of the important edges are lost as we increase the threshold value. (Fig. 11)

Fewer of the important edges are detected as we increase the threshold value beyond 0.2.

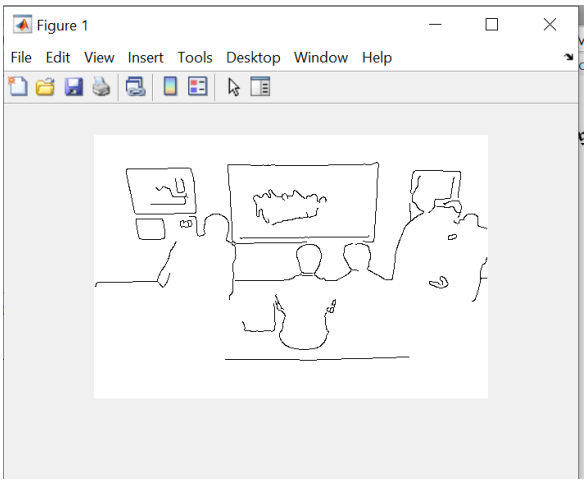



Fig. 11: Canny edge detection with threshold = 0.5

**For Dogs.jpg**

### 1. Threshold value = 0.34 (best)

This is the best balance between the edges of the dogs being detected and minimum edges of grass being detected.

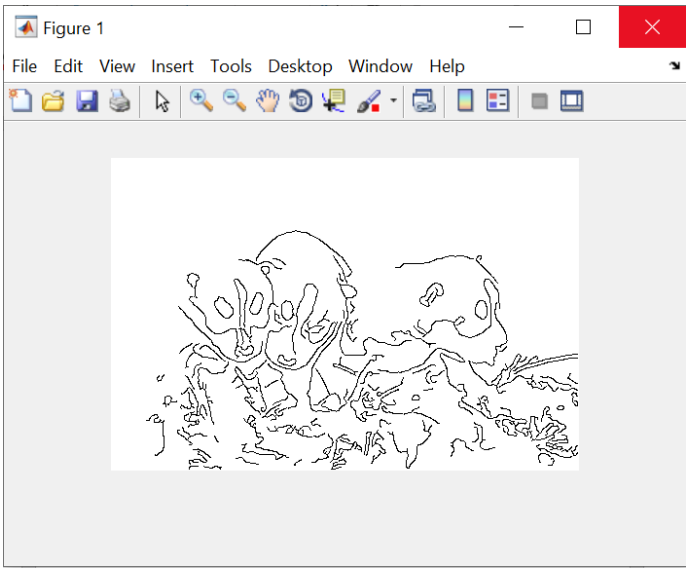



Fig. 12: Canny edge detection with threshold = 0.34

**2. Threshold = 0.5**

Th edges of the leftmost dog cannot be seen as we increase the threshold value beyond 0.34. (Fig. 13)

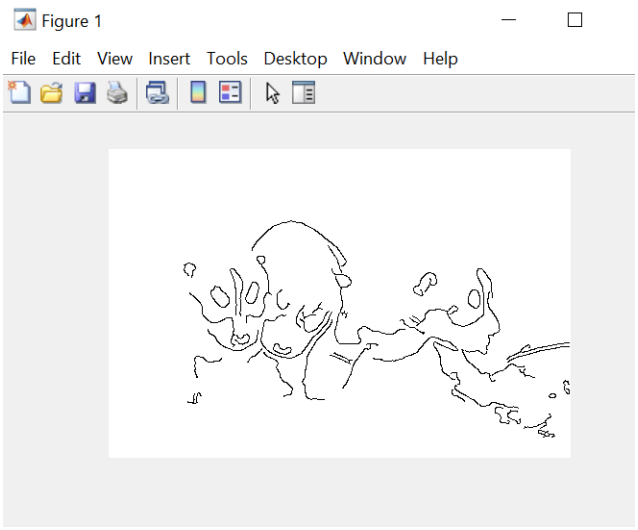



Fig. 13: Canny edge detection with threshold = 0.5

3. Threshold value = 0.2

With this threshold a lot of the unwanted edges of the grass can be detected and the edges of the dogs are lost among the edges of the grass. (Fig. 14)

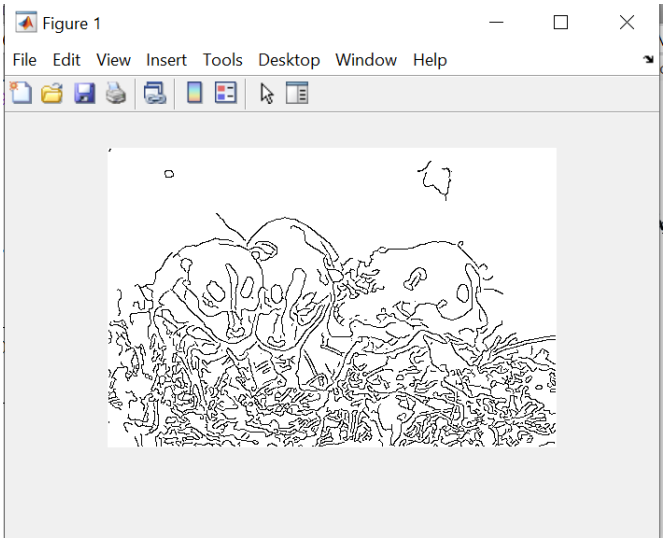
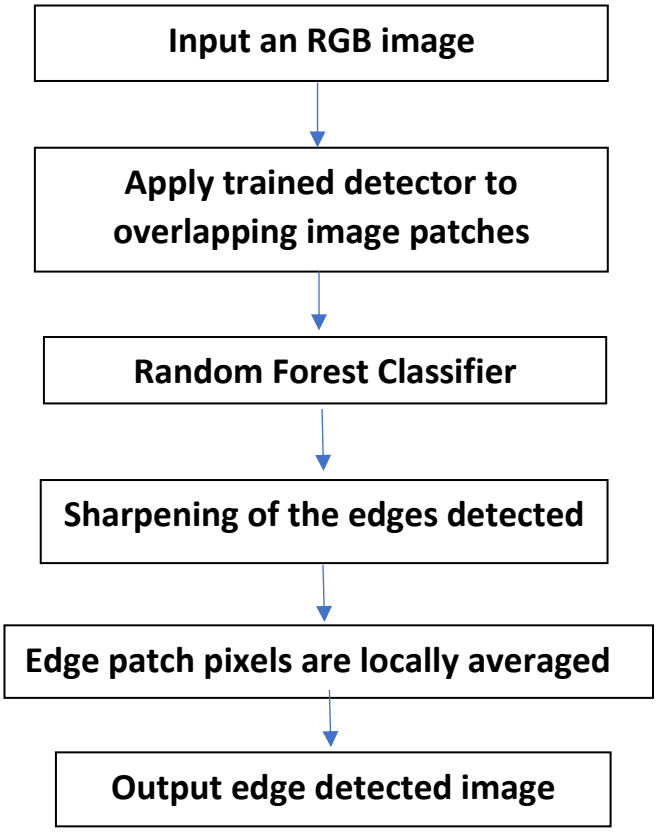



Fig. 14: Canny edge detection with threshold = 0.2

**(c) Structured Edge**

**(1) SE detection algorithm flowchart**




The SE detection algorithm uses the fact that the portions of the image that have edges have similar patterns. That is, they have lines, parallel lines or T- and Y-junctions. The first step is the input of an RGB image. Next, a trained detector is applied to the image to search for overlapping image patches. Wherever similar edge patches are located by the trained detector, an edge is marked at that pixel value. The Random Forest Classifier is used to classify these detected edges to be edges or not. After this, the edges are diffused since the trained detector trains it over a lot of images to find similar image patches. So, to get rid of the diffusion the edges detected are sharpened and edge patch pixels are averaged to get a single value. The last step is for the output image to be displayed, where the edges are shown. In SE the background can be black, and the edges detected are white. At the end of the algorithm, we can interchange this so that we get a white background and black edges are detected on it.

**(2)** The Random Forest classifier is a supervised machine learning algorithm used for classification. The random forest has multiple trees which give classification outputs. The various decision trees will give different outputs. It will classify the edges as different types of edges in the image to finally conclude whether there is an edge or not. The decision tree in the random forest having the maximum number of votes is said to classify the edges correctly and is used for the edge detection. The decision trees are constructed based on various features or parameters such as brightness, texture variation etc. There can be different thresholds for different decision trees which will help us have a variety of different outputs to choose from.

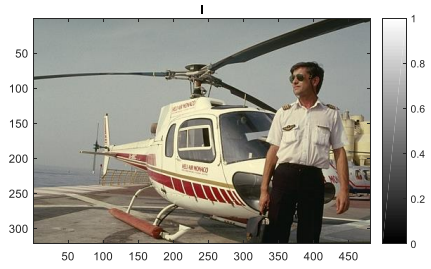
Random forest classifier help increase predictive power of the algorithm and they avoid the problem of overfitting as well. They choose the prediction of the decision tree that gets the most votes. This is done because we have training data and we know what the actual output should be, and we compare this with the output that a decision tree in the random forest is giving. Therefore, we can know which decision tree is giving the most accurate classification decision. The RF classifier is used in the SE detector and it gives decision son which image pixels are edges and which are not. Because of the random forests, the efficiency of SE detector is very high and the most important edges are detected and the others are discarded.

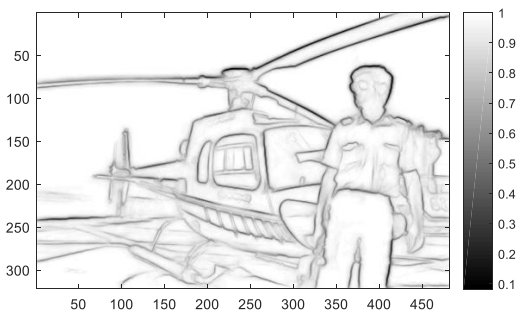
**(3)**

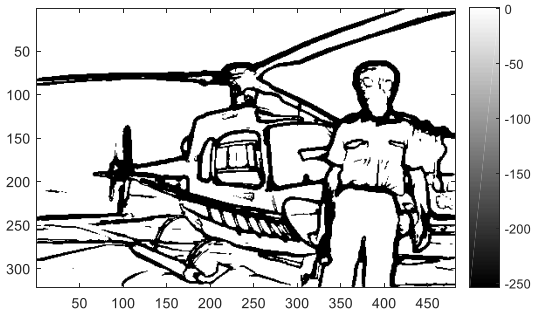
The GitHub link in [2] was used to implement the Structured Edge detector.

The output of the helicopter image can be seen below:

**Helicopter.raw**






**Parameter Selection:**

1. multiscale = 0 (I set this value for top accuracy as more edges of the leftmost dog were visible after setting this value to 0)

2. sharpen = 2 (I set this value because the edges were sharpest. For sharpen = 0 or 1, the edges were not as sharp)

3. nTreesEval = 5 (For this value, the eyes of the dog in the middle were getting detected the best)

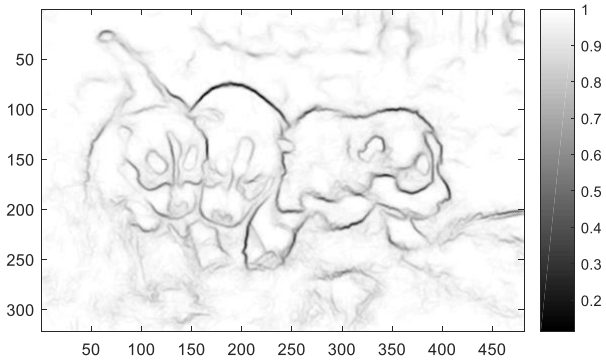
4. nThreads = 4 (This gave the best edge detection result in terms of evaluating the edges)

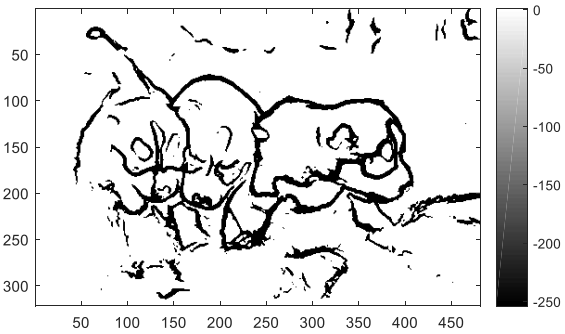
5. nms = 0 (I set non maximum suppression to false because edges were lost by doing non maximum suppression)

Threshold to binarize the probability edge map was taken to be 0.11 because the edges were the sharpest and most continuous for this threshold value.

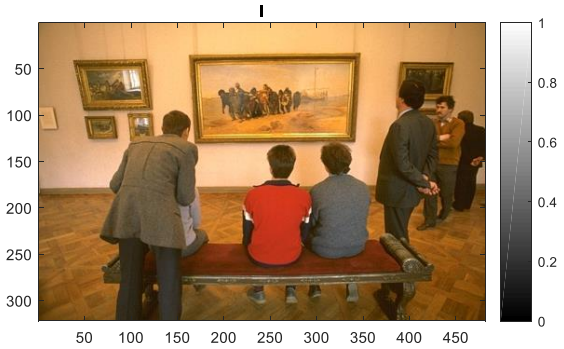
# Dogs.jpg

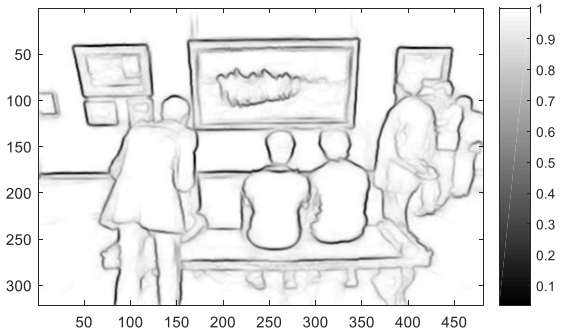

I
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.8
0.6
0.4
0.2
0

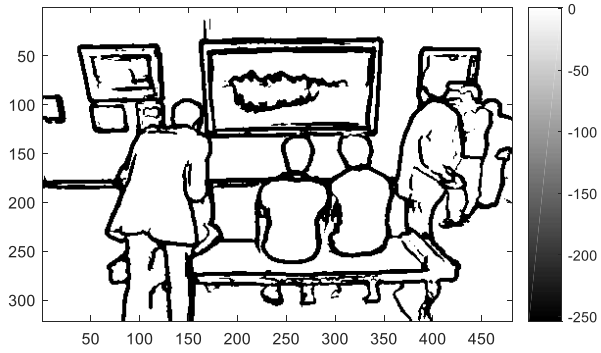

50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2


50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
0
-50
-100
-150
-200
-250

# Gallery.jpg


I
50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.8
0.6
0.4
0.2
0


50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


50
100
150
200
250
300
50 100 150 200 250 300 350 400 450
0
-50
-100
-150
-200
-250

**Canny vs SE**

In the Dogs.jpg image, the canny edge detector detects a lot of unnecessary edges such as the edges of the blades of grass. Whereas in SE detector it detects only the edges of the dogs. So the edge detection looks cleaner for SE detector.

In the Gallery.jpg image too, the canny edge detector detects unnecessary edges on the floor. The SE edge detector filters out these edges and shows only the edges of the humans and the paintings and the stool. In canny, a lot edges on the floor can be seen, which aren't meaningful edges.

**(d) Performance Evaluation**

**(1)**

**For Dogs.jpg**

| | Mean Precision | Mean Recall | F-measure |
|---|---|---|---|
| Sobel Edge Detector | 0.2180 | 0.8615 | 0.3479 |
| Canny Edge Detector | 0.2658 | 0.7698 | 0.3952 |
| SE | 0.7396 | 0.7374 | 0.7385 |

**Structured Edge**

>> evaluation_se

Elapsed time is 0.681969 seconds.

Elapsed time is 0.236306 seconds.

Mean precision is: 0.7396

Mean recall is: 0.7374

F-measure is: 0.7385

**Canny Edge**

>> evaluation_canny

Mean precision is: 0.2658

Mean recall is: 0.769

F-measure is: 0.3952

**Sobel Edge**

Mean precision is: 0.2180

Mean recall is: 0.8615

F-measure is: 0.3479

**For Gallery.jpg**

| | Mean Precision | Mean Recall | F-measure |
|---|---|---|---|
| Sobel Edge Detector | 0.3479 | 0.3106 | 0.4625 |
| Canny Edge Detector | 0.3105 | 0.9055 | 0.4625 |
| SE | 0.8573 | 0.9413 | 0.8974 |

**Structured Edge**

>> evaluation_se

Elapsed time is 0.795600 seconds.

Elapsed time is 0.293246 seconds.

Mean precision is: 0.8573

Mean recall is: 0.9413

F-measure is: 0.8974

**Canny Edge**

>> evaluation_canny

Mean precision is: 0.3105

Mean recall is: 0.9055

F-measure is: 0.4625

**Sobel Edge**

Mean precision is: 0.3106

Mean recall is: 0.9055

F-measure is: 0.4625

The Sobel and Canny edge detectors are able to produce high mean recall values but their mean precision is very low. That is why their F-measure value is small. The mean precision value of sobel is lower than that of canny, therefore it's F-measure is the lowest.

The SE detector strikes a balance between the mean precision and mean recall value and hence it is able to achieve a higher F-measure and is the best edge detector among these three edge detectors.

So from best to worst, SE is the best. Next is Canny and then Sobel.

**(2)** Gallery image is easier to get a higher F-measure as compared to Dogs. This is because of the intensity values in the image. The Gallery image has more starkly demarcated edges as compared to the Dogs image. In the Dogs image the edges of the grass need not be detected but they are detected because of the variations in intensity values at the edges of the grass blades. The grass also gets mixed up with the dogs and interferes in the Dogs' edges. So, the edges of the dogs cannot be seen clearly.

In the Gallery image, the background has lower intensity and is hence not detected as an edge and does not interfere with the main elements of the image.

**(3)** The precision and recall values lie between 0 and 1. If the precision is significantly higher than the recall (or vice versa) then the F measure will be low. For eg. If P = 0.8 and R = 0.15, then the F measure = 0.2526. Whereas if the precision and recall values are close to each other the F measure is higher. For eg. P = 0.5, R = 0.45, F = 0.47. This is because the F measure tends to be closer to the lower of precision and recall.

If sum of P and R is a constant, say 0.8, then F measure is maximum when P=R.

This can be shown with the help of a few examples.

1. P = 0.4,  R = 0.4, F = 0.4

2. P = 0.1,  R = 0.7, F = 0.175

3. P = 0.6,  R = 0.2, F = 0.3

As we can see, the value of F is max when P=R.

**Problem 2:**

**(a) Dithering**

1. Fixed thresholding

In this method, the pixel values of the input image were set to 0 if the pixel value was less than 128 and if the pixel value was greater than 128, then it was set to 255. The threshold was fixed to be 128.

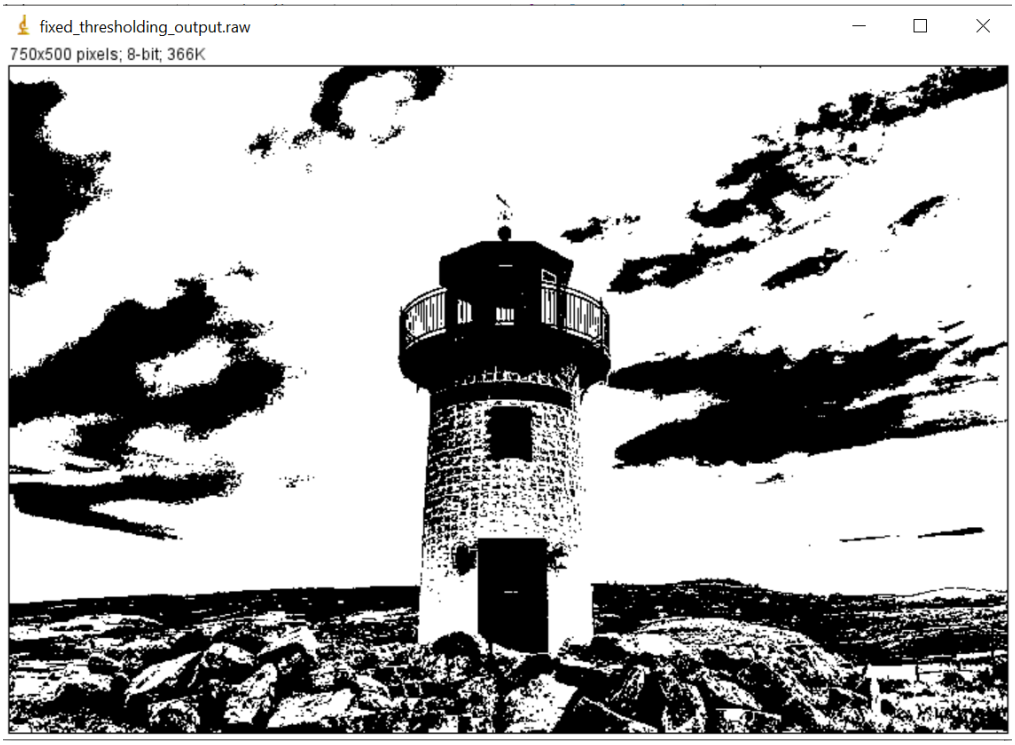
2. Random Thresholding

There are 750*500 = 375000 pixels in the image Lighthouse.raw. Therefore, 375000 random threshold values were created by using the ‘rand’ function in Matlab. The rand function generates uniformly distributed pseudorandom numbers. These values lie between 0 and 1. So I multiplied these random numbers by 255 to get the intensity values between 0 and 255.

3. Dithering Matrix

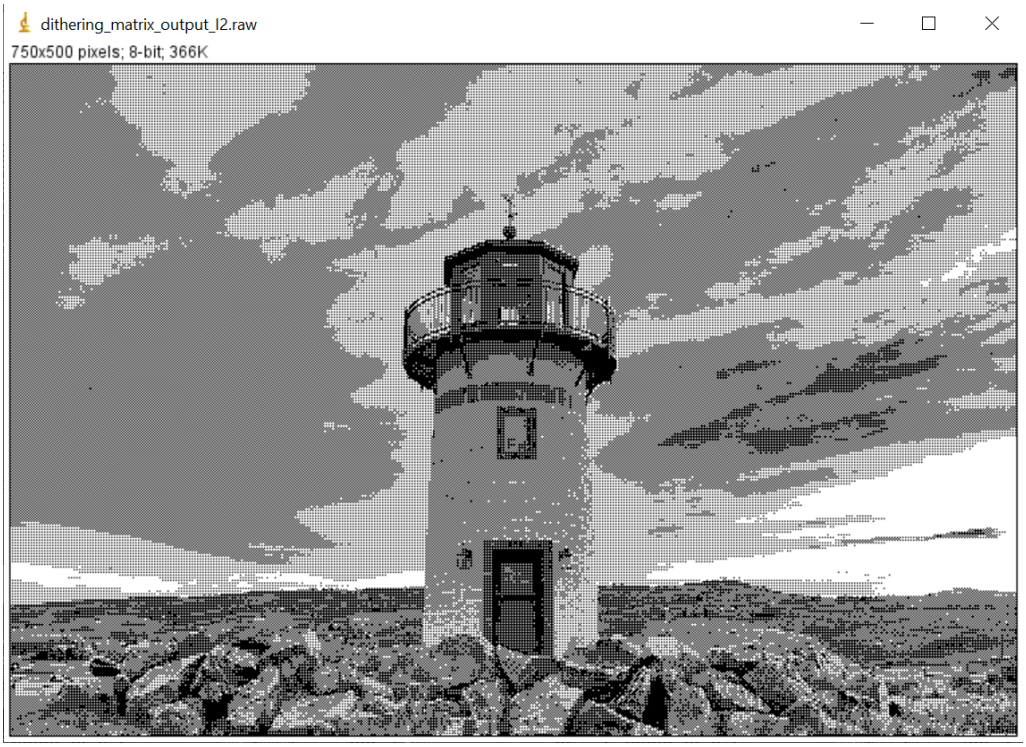
The dithering matrices I2, I8 and I32 were used to calculate the threshold matrix. After this the values were set to 0 if they were less than the threshold and they were set to 255 if they were more than the threshold.


**Original image (Lighthouse.raw)**

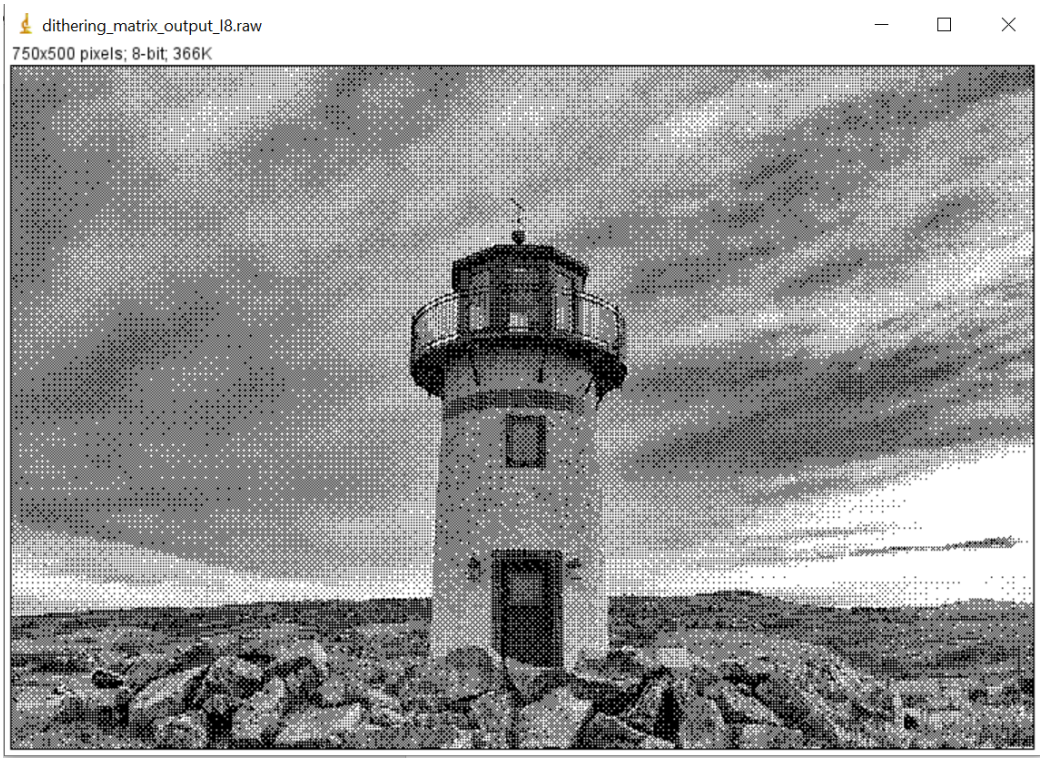



**Output after fixed thresholding**

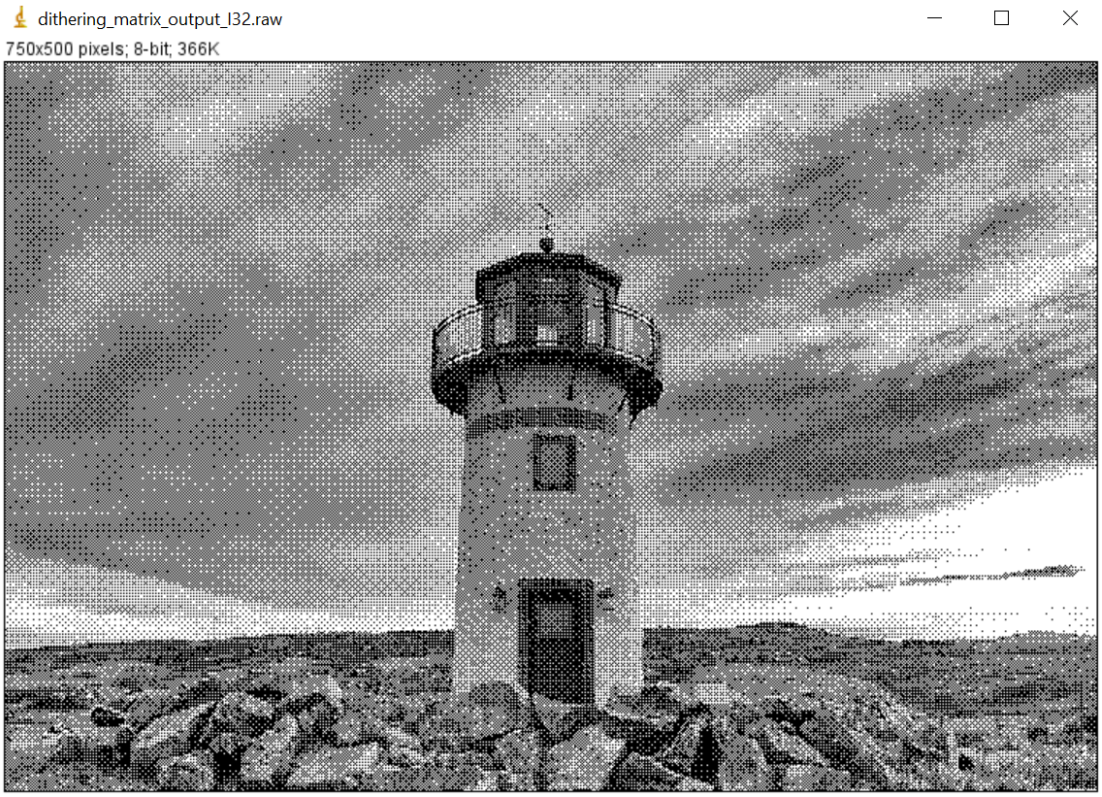

**Output after random thresholding**




**Dithering matrix I2**




**Dithering matrix I8**




**Dithering matrix I32**

**Comparison:**

Since the threshold is fixed in the first method, the pixel values located close to each other if being of the intensity values close to each other have intensity values of 0 or 255 and we can see a clear demarcation between white and black areas in the output image. The door and top part of the lighthouse has lower intensity, so it got set to 0 and became black. The darker part of clouds became completely black and the lighter parts became completely white.

In random thresholding, since the threshold keeps on changing for each pixel, some other shades apart from black and white can be seen, even though the image has pixels of intensities either 0 or 255. This is because the pixels are arranged in a way which gives an illusion of other shades.

The outputs of the dithering matrices show the shades more closely to the original image. These outputs also have only 2 intensity values i.e. 0 and 255. I could notice that as the dimension of the matrix increased from 2x2 to 8x8 to

32x32, the shades of gray were more clearly and accurately observed in the output images.

**(b) Error Diffusion**

In error diffusion I used 3 matrices, the first was the Floyd-Steinberg's error diffusion matrix, the second was the Jarvis, Judice, and Ninke (JJN) error diffusion matrix and the third was the Stucki error diffusion matrix. I used serpentine scanning in these which means that I started from the first row and scanned all the columns diffusing error to the neighboring pixels. Then for the second row I went from the last column to the first column and so on.

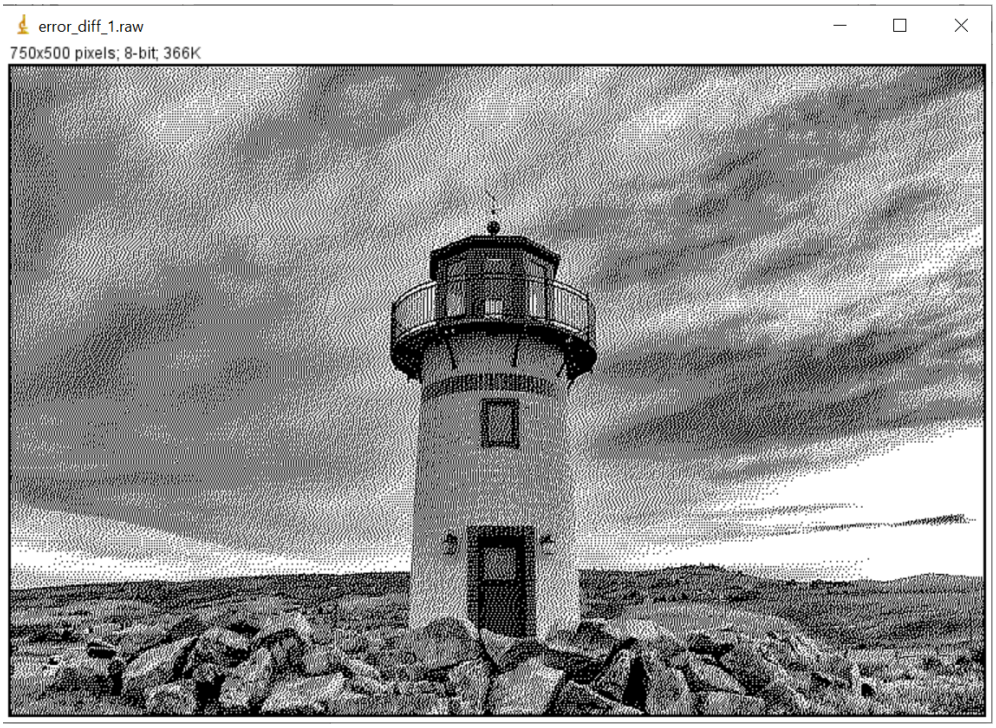
Therefore, for all the odd rows I scanned from left to right and for all the even columns, I scanned from right to left.

For suggesting my own method, I generated a new matrix of size 7x7:

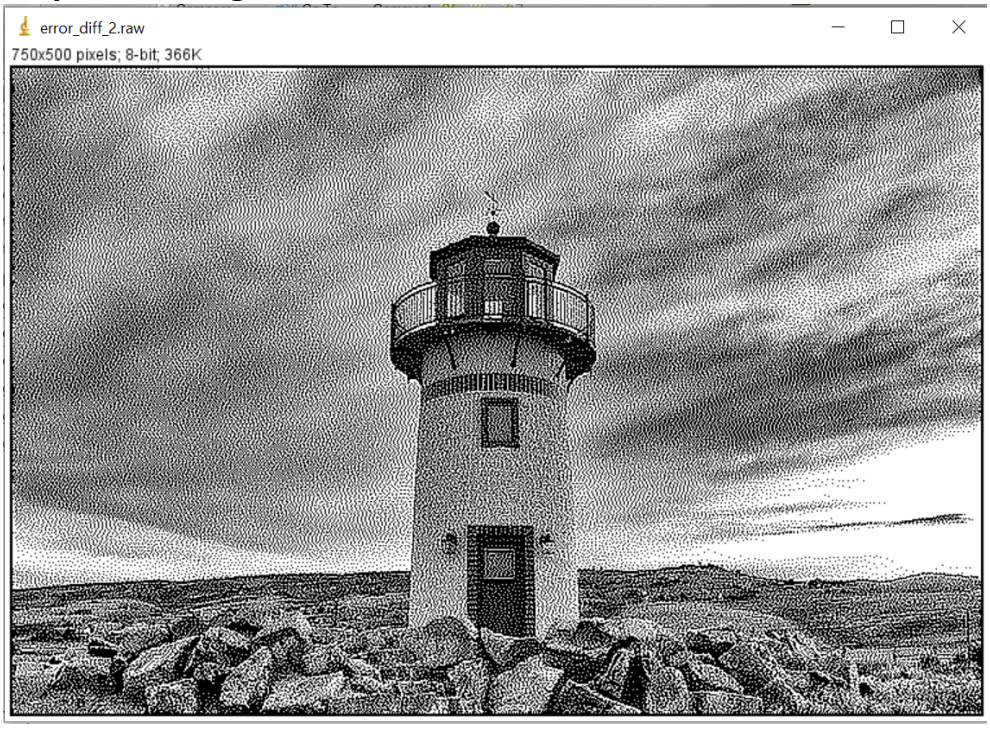
```
h4 = (1/144)* [0 0 0 0  0  0 0
               0 0 0 0  0  0 0
               0 0 0 0  0  0 0
               0 0 0 0  11 9 7
               5 7 9 11 9  7 5
               3 5 7 9  7  5 3
               1 3 5 7  5  3 1]
```

The method I proposed is a 7x7 matrix and it diffuses error to more number of neighboring pixels and hence it gives a clearer image where the shades of the sky are smoothly seen.

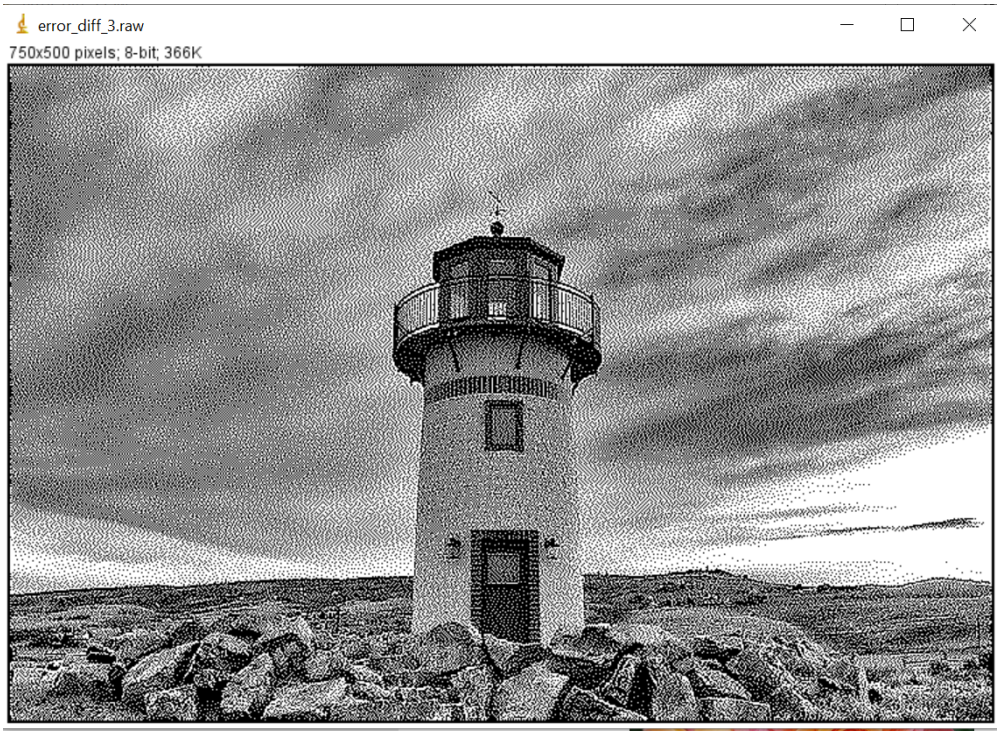
## Results:



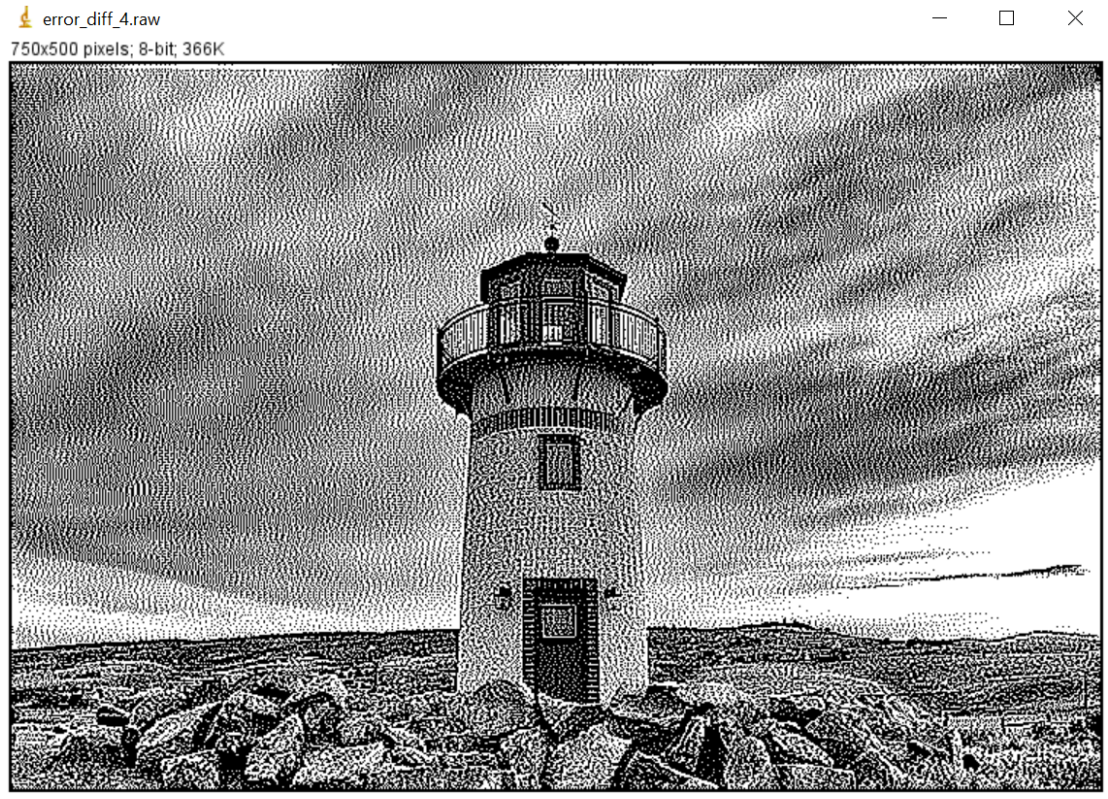

**Floyd-Steinberg's error diffusion**




**JJN's error diffusion**




**Stucki’s error diffusion**

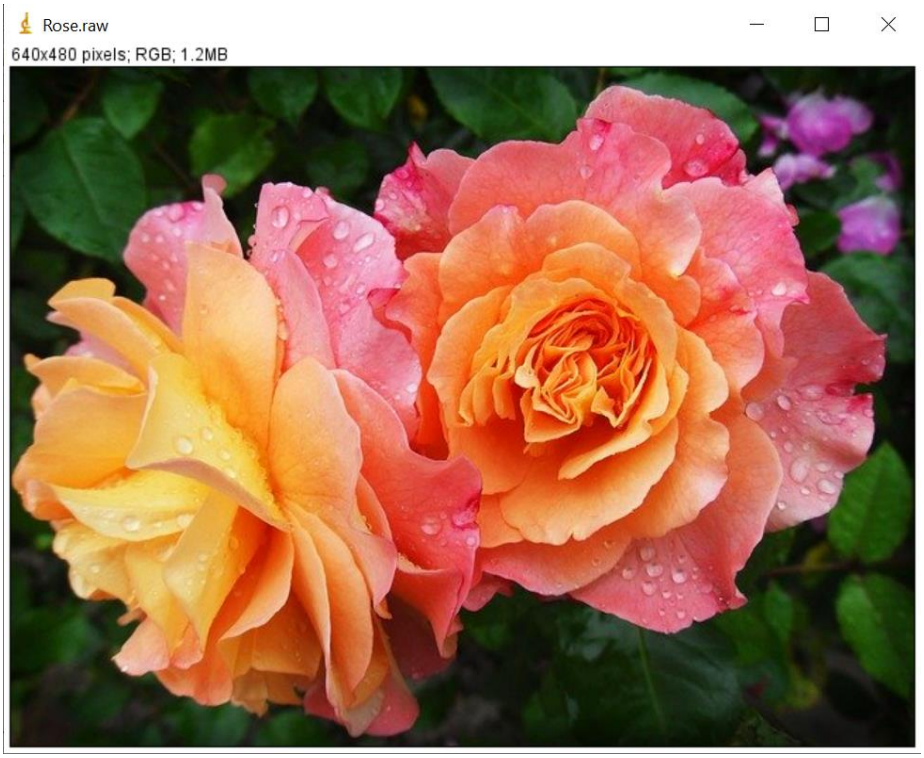



**suggested method (using a 7x7 error diffusion matrix)**

Comparing these results with that of part (a)'s dithering matrix, I find that I prefer the error diffusion method over the dithering matrix method. This is because the dithering matrix produces output images that are pixelated, and the pixels can be seen starkly.
Whereas, in the error diffusion method the images appear to be more uniform and closer to the original image.

The 3 error diffusion matrices give output images as shown in the results above. As the size of the matrix goes on increasing the output image becomes clearer and less pixelated. The output of Stucki and JJN look almost the same since the error diffusion matrix size is the same (5x5 matrix). The error is diffused in a different way since Stucki has even numbers and JJN has odd numbers. The Floyd-Steinberg method produces a more pixelated image since the matrix size is 3x3 and the error is diffused to fewer neighboring pixels.

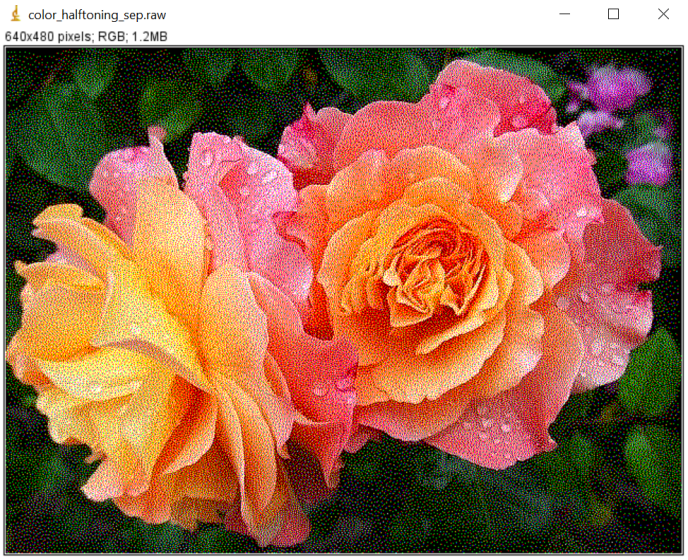
**(c) Color halftoning with error diffusion**



## 1. Separable error diffusion

In this, the RGB image was converted to CMY and the Floyd-Steinberg error diffusion matrix was applied to the CMY channels. Then it was converted back into RGB and the result was printed.

**Result:**




**Output of separable error diffusion**

The halftoned Rose image has dots all over the image which the original image doesn't have. The drawback is because of certain human color perception, the image is of lower quality as the separable error diffusion does not consider this human color perception. All 8 basic colors are used to print the output image of the separable error diffusion method. This causes problems related to brightness variation.

## 2. MBVQ-based error diffusion

The paper in [1] was used and the algorithm in the paper was followed to get the results. The getNearestVertex provided on DEN was also used to implement this.

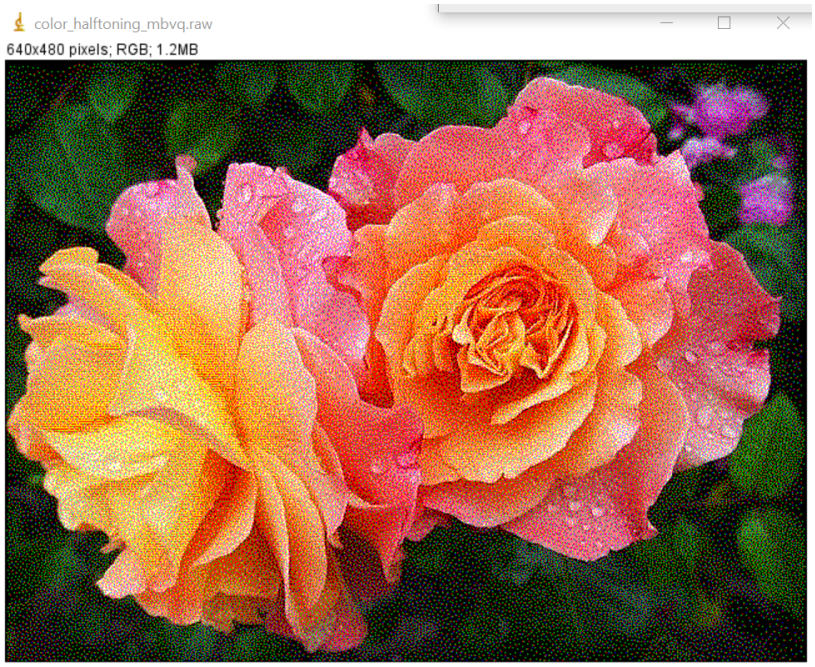
**(1)** The MBVQ- based method says that using 4 colors suffices for the halftoning process. We need a halftone set required by MBVC (Minimum Brightness Variation Criterion). Here we preserve the average color and reduce the average variation and find that the participating halftone colors are only 4. Therefore, we

have 4 MBVQs (4 quadruples) which are MYGC, RGMY, RGBM and CMGB. we can get every input color using either: RGBK, WCMY, MYGC, RGMY, RGBM, or CMGB, each of these has minimal brightness variation when we use it. For every RGB triplet we can compute the MBVQ by the location.
We compute the MBVQ od each pixel in the image. Then we get the nearest vertex to the RGB components. We then compute the quantization error and then diffuse this quantization error to the future pixels. We then get our output image.

This method overcomes the shortcomings of the separable error diffusion by giving a higher quality image. Since this method uses only 4 colors and finds the nearest vertex, the brightness variation is minimal, and the human eye can perceive these colors better. So, the image is of higher quality.

**(2)** On comparing the output of the separable error diffusion method with the MBVQ-based method, I noticed that the image produced by the MBVQ-based method is of much higher quality than that produced by the separable error diffusion. The dots that can be seen in the separable error diffusion are too high. The dots that can be seen in the MBVQ-based method are considerably lower making the quality of the image high.

**Result:**




**Output of MBVQ-based error diffusion**

## References:

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

[2] https://github.com/pdollar/edges