EE569: Introduction to Digital Image Processing

ASSIGNMENT #1

Sonali B Sreedhar

USC ID: 1783668369

Email ID: sbsreedh@usc.edu

Problem 1: Image Demosaicing and Histogram Manipulation

a) Bilinear Demosaicing:

1. Abstract and Motivation:

Most modern digital cameras acquire images using a single image sensor overlaid with a CFA (Color Filter Array) i.e. the red, green and blue values are not sampled at the same position, So we need to perform interleaving using Bayes pattern, where we have to do interpolation to get the real color images. This is called Demosaicing, to get the RGB channels in all the pixels.
The idea behind this problem is to use the simplest demosaicing method based on bilinear interpolation.

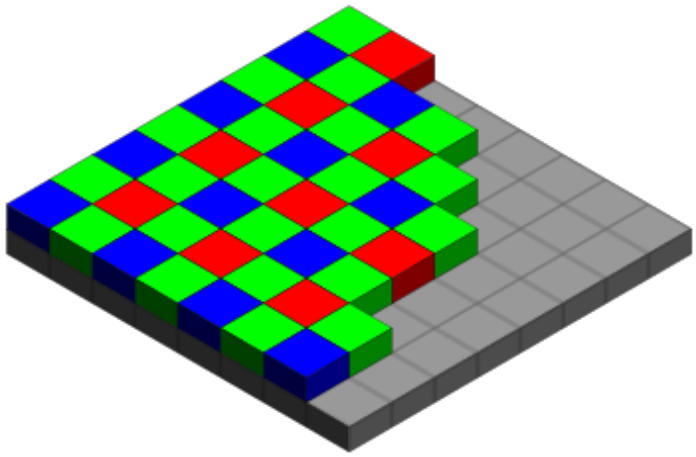


Figure 1: Bayer arrangement of color filters

2. Approach and Procedure:

Algorithm for Bilinear interpolation Demosaicing:

- Read Dog.raw image using fread() function.
- Define the three RGB channels mapped to 0,1,2 respectively.
- Use mirror reflecting on all sides of the image read, to add one boundary line to the input image.
- Bilinear interpolation is done by applying demosaicing by using the 3x3 convolution filter.
- Add the remaining two missing red/Green/Blue channel values to the respective pixel location by using the formula

$$\hat{R}_{3,3} = \frac{1}{2}(R_{3,2} + R_{3,4})$$

$$\hat{B}_{3,3} = \frac{1}{2}(B_{2,3} + B_{4,3})$$

$$\hat{B}_{3,4} = \frac{1}{4}(B_{2,3} + B_{2,5} + B_{4,3} + B_{4,5})$$

$$\hat{G}_{3,4} = \frac{1}{4}(G_{3,3} + G_{2,4} + G_{3,5} + G_{4,4})$$

- Use two sets of functions for odd and even values for green pixel location for odd and even values
- Crop the image to its original size removing the extended boundary.
- Write back the image computed into the data array on output.raw file using the fwrite() function.

3. Experimental Results:

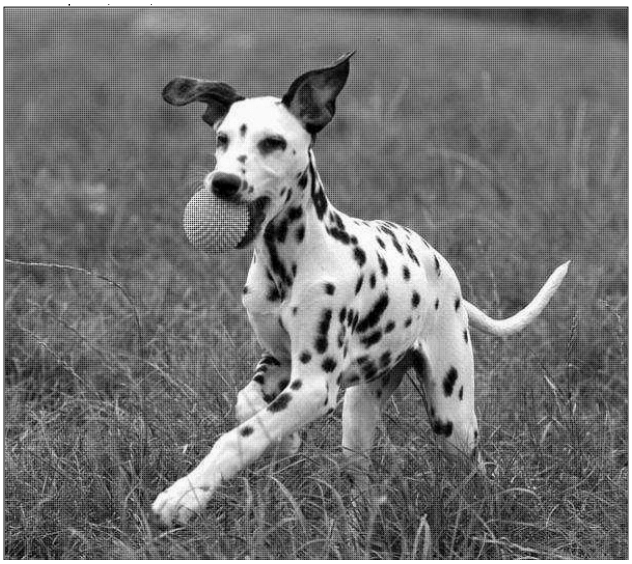


Figure 2: Input Dog.raw

Figure 3: Original image

Figure 4: Demosaiced image by Bilinear Interpolation

4. Discussion:

Yes, there are artifacts observed in the output image. We see a few aberrations in the output image due to the poor quality of the image.
The result after applying bilinear demosaicing to the intput image is shown above. Most of the artifacts appear at edges and areas of high frequency.
Few of the shortcomings on the output image are:
Zipper effect: This causes color changes at sharp edges like the border of the dog's body.
False color effect: This causes artificial/error colors like on the grass,
Aliasing when any pixel frequency goes above the Nyquist frequency. The images above show these artifacts present in the output image compared to the original image.
We might be able to improve the performance implementing non-linear methods and consider correlation among the RGB channels to avoid False color effect.

b) MALVAR-HE-CULTLER. (MHC) DEMOSAICING:

1. Abstract and Motivation:

MHC is an improvised linear interpolation technique for demosaicing algorithm.
This algorithm yields a higher quality demosaicing result by adding a second order cross channel correction term to the basic bilinear demosaicing result. (Source : Discussion 1)

2. Approach and Procedure:

The MHC algorithm is stated below.
To estimate a green component at a red pixel location, we have

$$\hat{G}(i,j) = \hat{G}^{bl}(i,j) + \quad _R(i,j) \tag{1}$$

where the 1st term at the right-hand-side (RHS) is the bilinear interpolation result given in (1) and the 2nd term is a correction term. For the 2nd term, alpha is a weight factor, and Δ_R is the discrete 5-point Laplacian of the red channel:

$$_R(i,j) = R(i,j) \quad \frac{1}{4}\big(R(i \quad 2,j) + R(i+2,j) + R(i,j \quad 2) + R(i,j+2)\big) \tag{2}$$

To estimate a red component at a green pixel location, we have

$$\hat{R}(i,j) = \hat{R}^{bl}(i,j) + \quad _G(i,j) \tag{3}$$

where Δ_G is a discrete 9-point Laplacian of the green channel.

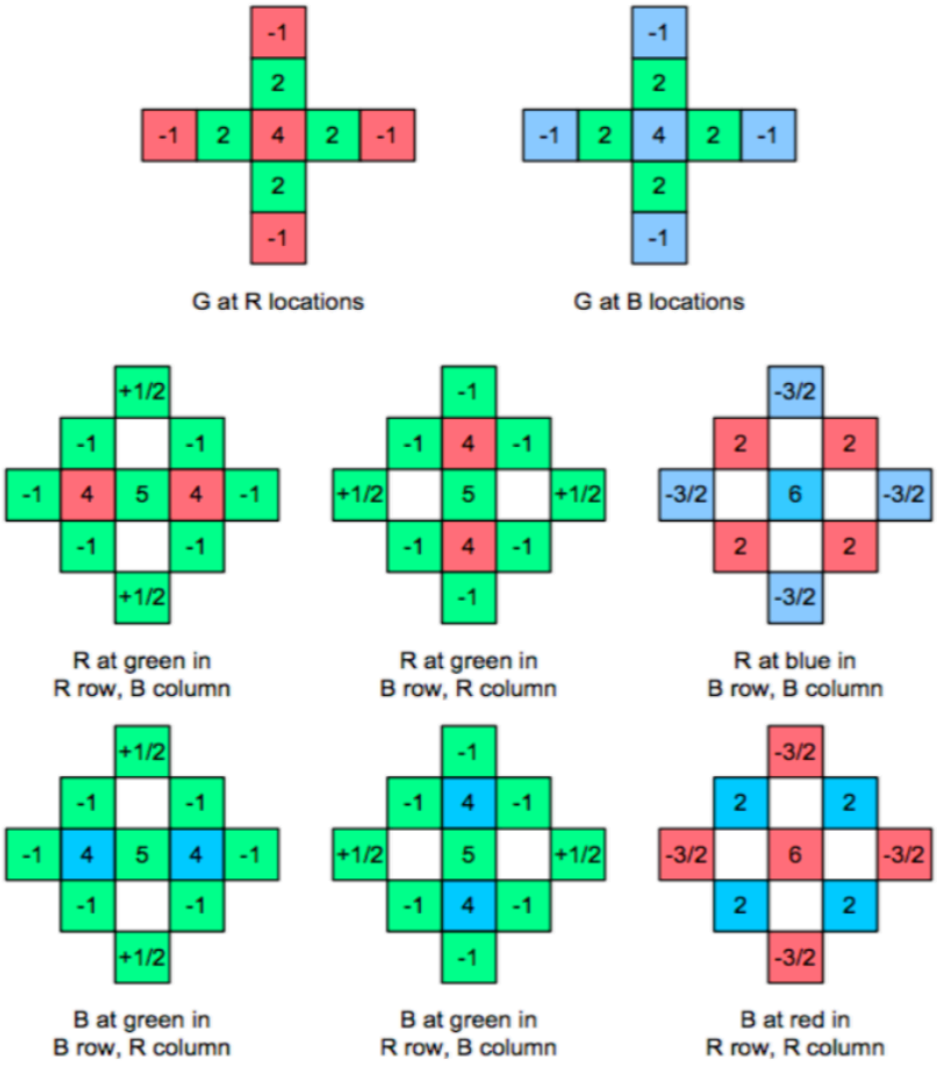
To estimate a red component at a blue pixel location,

$$\hat{R}(i,j) = \hat{R}^{bl}(i,j) + \quad _B(i,j) \tag{4}$$

where Δ_B is a discrete 9-point Laplacian of the blue channel. The weights , , control how much correction is applied, and their default values are:

$$= \frac{1}{2}, \quad = \frac{5}{8}, \quad = \frac{3}{4} \tag{5}$$

The procedure for MHC demosaicing is complicated. It can be implemented by convolution with a set of linear filters. Say, for example there are eight different filters for interpolating the different color components at different pixel locations.



This pattern is used to do the average on the surrounding pixels by using the formula above. So, we take the average on the neighbor surrounding pixels and use the weight. We calculate the weighted average of the center pixel by using the surrounding pixels in the corresponding rows and columns. Repeat this procedure for all the pixel locations to add the other two-color values at each pixel location. Before the above, we use boundary extension to deal with the boundary effect. We use the mirror reflecting algorithm considering the boundary line pixel values as the center and reflecting the other lines inside the image to the boundary on all the sides of the image. This is mainly done to avoid errors in the computation when a filter is being used.

Algorithm implemented in C++

- Read Dog.raw image which is a gray scale image using fread() function
- Define 3 color channels: channel 0 → Red, channel 1 → Green, channel 2 → Blue
- Add two boundary lines to the input image by boundary extension using mirror reflecting on all sides of the image border

- Apply demosaicing by using the 5x5 convolution filter for the bilinear interpolation
- Adding the remaining two missing green/red/blue channel values to the R/G/B pixel location by using the formula mentioned above by the combination of 8 different filters
- Using two sets of functions for green pixel location for odd and even values
- Crop the boundary extended image to the original image size after the computation is done
- Write the computed image data array on output.raw file using the fwrite() function

3. Experimental Results:

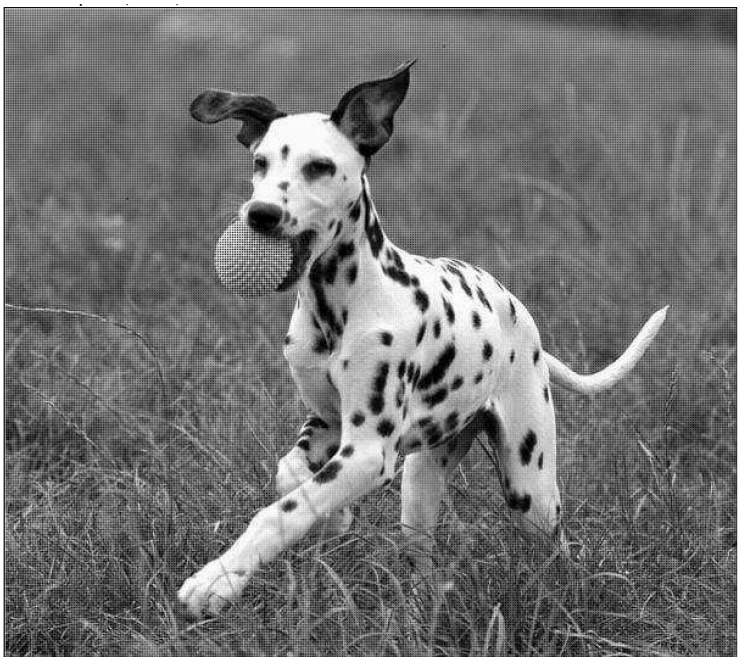


Figure 5: input image

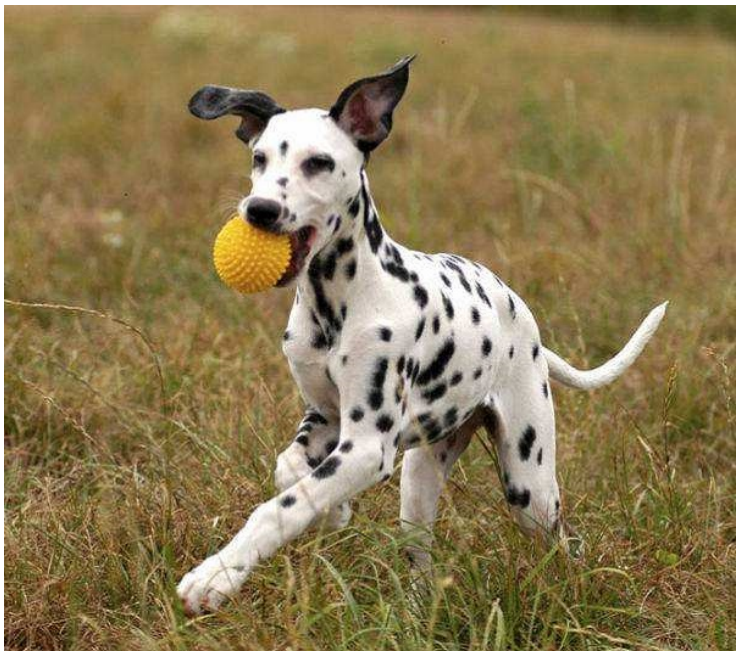


Figure 6: The original image

Figure 7 Demosaiced image by MHC

4. Discussion:

Figure 8: Bilinear Interpolation

Figure 9: MHC output

The result after applying MHC linear demosaicing algorithm to the dog image is shown above. We can see that the errors in the image resulted by the bilinear demosaicing method have almost disappeared when implementing the MHC demosaicing algorithm. We can compare the dog output image results by the bilinear and MHC demosaicing above.
For the comparison between the bilinear and MHC demosaicing: Bilinear demosaicing is easy to implement as it has lesser computations. It is good at previewing image which has simple, smooth or less various color. However, this method performs badly at the edges or details in the image and this causes various artifacts like Zipper effect, false-color effect, aliasing, blurring. In the figures above, we see fig 8 is more blur than fig 9. MHC linear demosaicing performs better at edges or details in the image compared to bilinear demosaicing. The output of the MHC Demosaicing is much sharper than that of Bilinear Interpolation.We can see this performance difference on various aspects in the image like the dog, grass and the ball. The image is better in colors and also less blurry. Since MHC uses second order correction term to the bilinear method, it helps to preserve the true color especially in the regions where colors change rapidly. Also, while applying MHC, since the correction terms use plus and minus coefficients, it will make some values go above 255 or below 0. Such values result into wrong color in the image. So to deal with it, we assign 255 to values above 255 and 0 to values below 0.

c) Histogram and Manipulation for Contrast Enhancement:

1. Abstract and Motivation:

Histogram equalization technique deals with intensity values and contrast of an image. This is a technique improves the image contrast by changing the image/pixel intensities. Contrast adjustment is needed so as to make the image more viewable as it equalizes the dark and bright components, making the luminance distribution uniform, thus making the image more pleasant to the human eye. Histogram is the count of pixel values (0-255) in an image, it shows the frequency of each

pixel at different intensities or probability of pixels. Histogram of an image can be drawn plotting pixel intensities vs frequency or probability of pixel intensities.
There are two techniques used for histogram equalization:

a. Transfer function based and
b. Cumulative probability based.

The transfer function is the integration of the input histogram.
The cumulative probability is the discrete version of transfer function where it uses the probability density array partitioning that is partition the long string N x N into 256 segments of the same length.

The idea of this problem is to understand and implement Histogram equalization methods to enhance the image terms of contrast and overall visualization by the human eye. Since Image enhancement basically depends on visual evaluation which may vary from person to person, we don't have concrete methods to evaluate the performance of an Image Enhancement technique than visual appeal. The more soothing an Image is to the eye, better it is.

2. Approach and Procedure:

Method A: Transfer function-based Histogram Equalization:

(Source: https://www.youtube.com/watch?v=PD5d7EKYLcA)

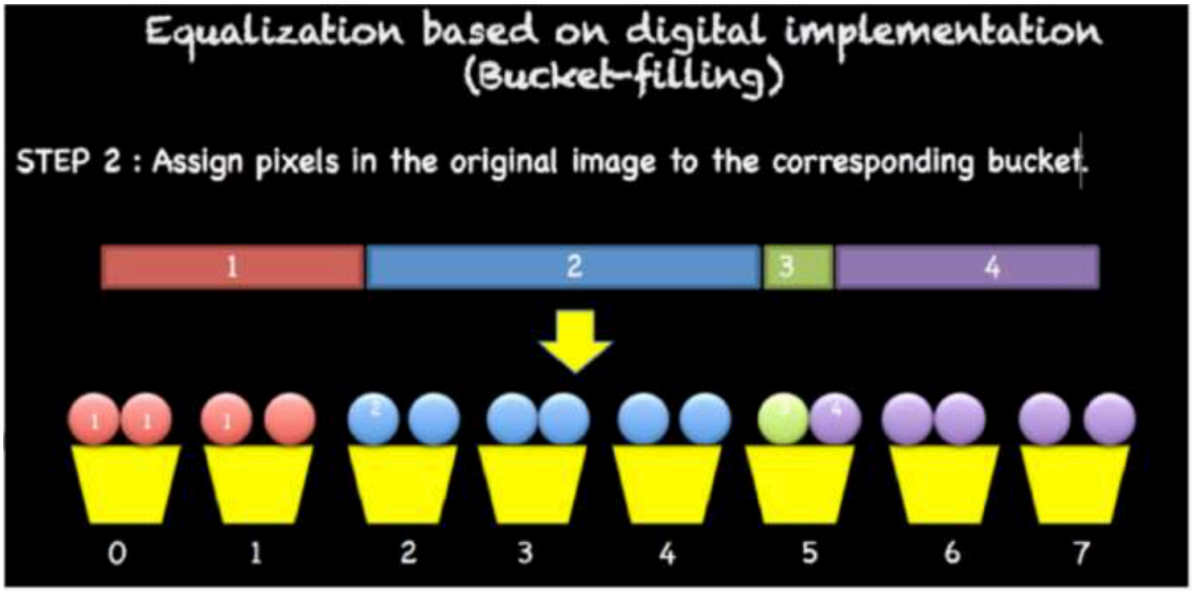
Algorithm implementation in C++:

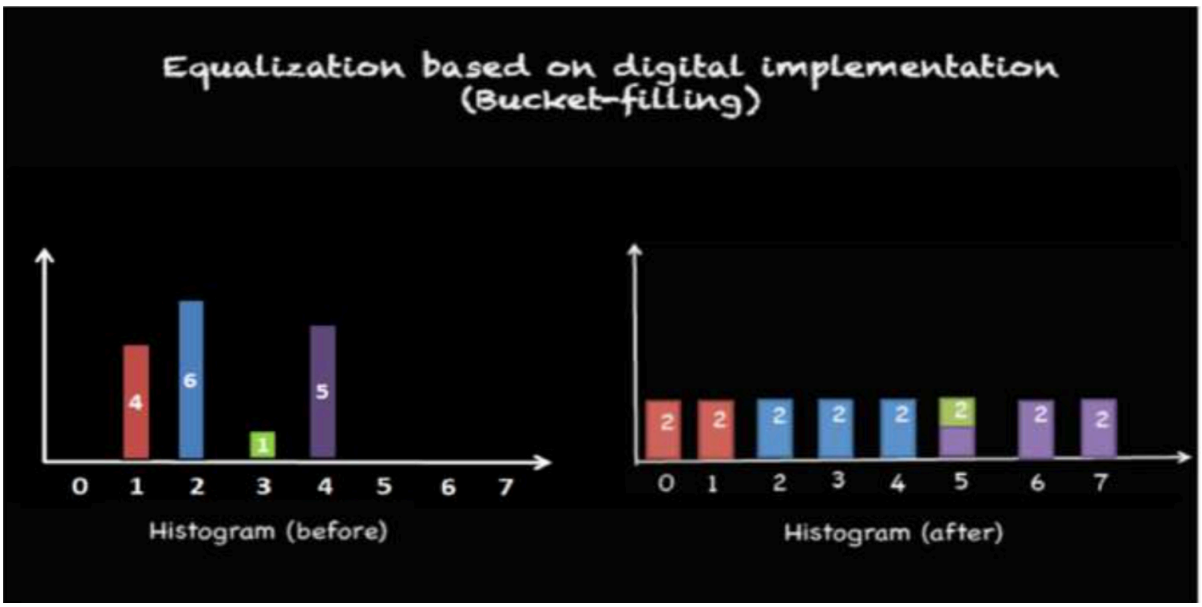
- Calculate the histogram data of the input image by counting the number of pixels corresponding to each intensity value from 0 to 255.
- Calculate the normal probability histogram by dividing the histogram by total number of pixels.
- Calculate the Cumulative Distribution Function (CDF).
- Create mapping matrix from CDF by multiplying levels and map the values to each pixel of the image.

Method B: Cumulative Probability Based (Bucket Filing Method) Histogram Equalization

Algorithm implementation in C++:

The method B assigns equal count values for every pixel 0 to 255:(Source: EE569 – Discussion 2)





1. Read Toy.raw input images using fread() function and get height, width and bytesperpixel values
2. Run 3 nested for loops for height, width and bytesperpixel to access each pixel in the original image
3. Define the histograms of original image, equalized image and transfer function as unsigned char
4. Initialize 3 different 1D arrays to store values for the row index, column index and pixel value for each channel of RGB.
5. Sort these pixels in the order of 0 to 255
6. Store the corresponding row index and column index for each pixel for each RGB channel in the 1D arrays defined above
7. Change the corresponding pixel values at the given locations according to the bucket size such that each bucket will contain equal number of pixels
8. Replace the pixel valuesof all the three channels in the input image to new pixel values calculated by the bucket filling method and use the row and column index tracked in 1D arrays to put the new values of pixels in each channel.
9. Write the CDF based histogram equalized contrast enhanced image on output.raw file using the fwrite() function
10. Write the histograms to text file

3. Experimental Results:

Figure 10: Input image

Figure 11: Method A output

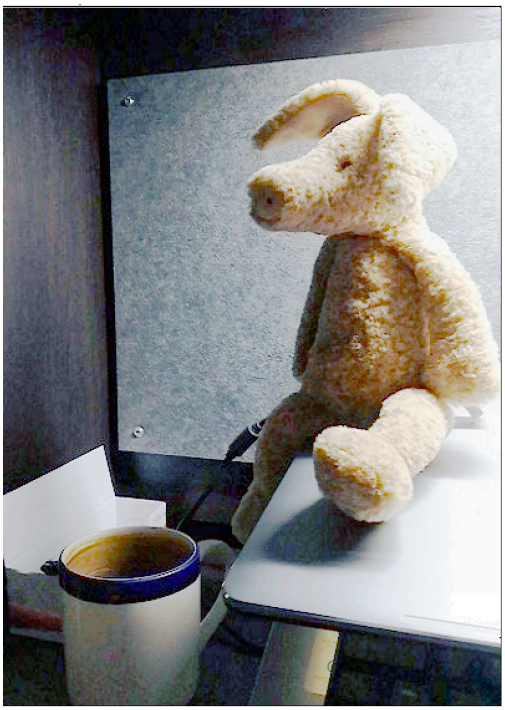
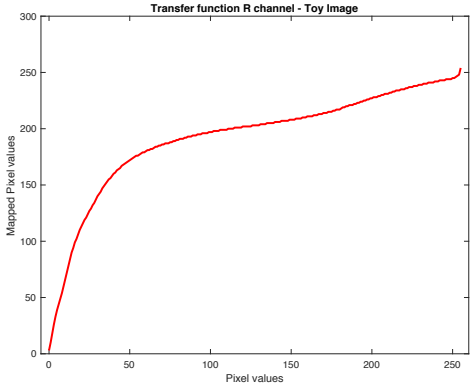
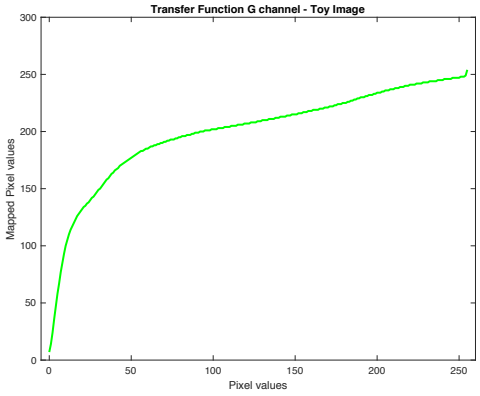
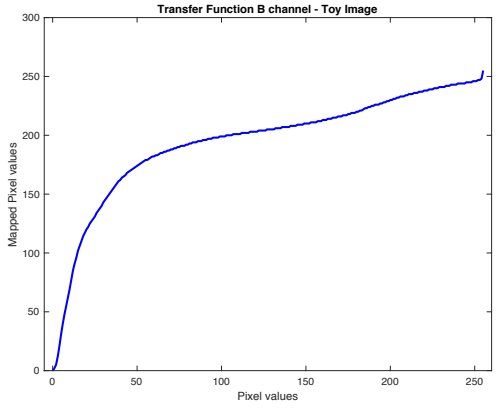
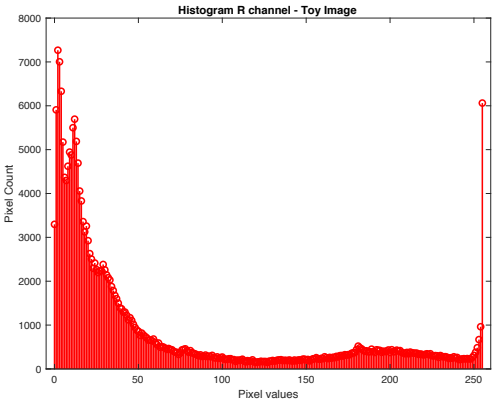


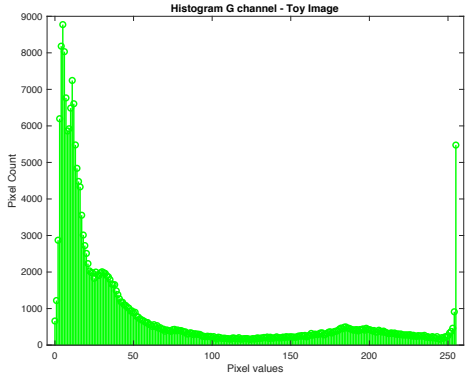
Figure 12: Method B output

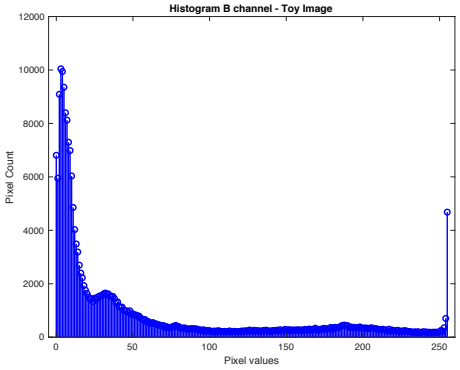

Transfer function R channel - Toy Image
Mapped Pixel values
Pixel values

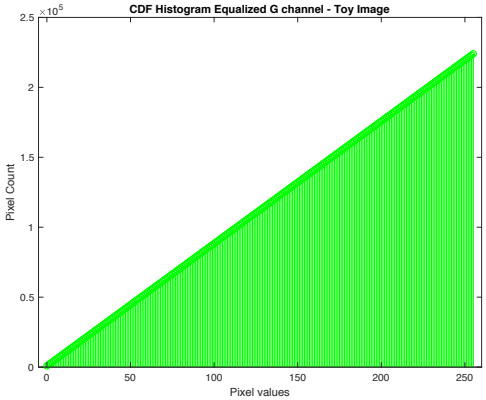

Transfer Function G channel - Toy Image
Mapped Pixel values
Pixel values

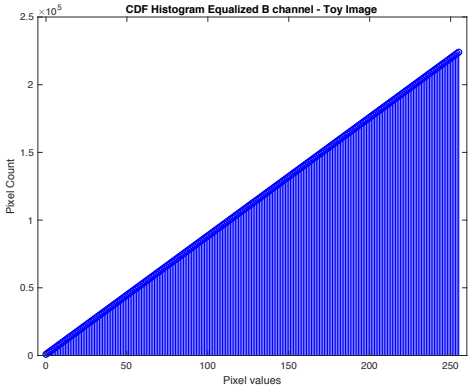

Transfer Function B channel - Toy Image
Mapped Pixel values
Pixel values

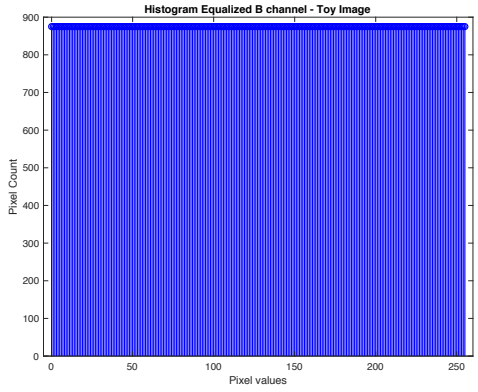

Histogram R channel - Toy Image
Pixel Count
Pixel values

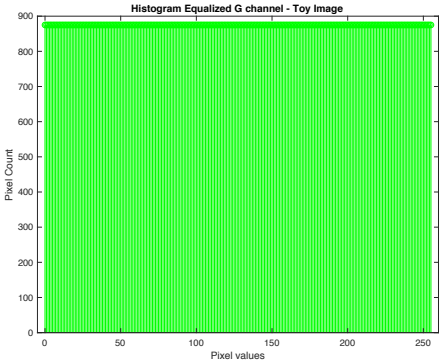

Histogram G channel - Toy Image
Pixel Count
Pixel values

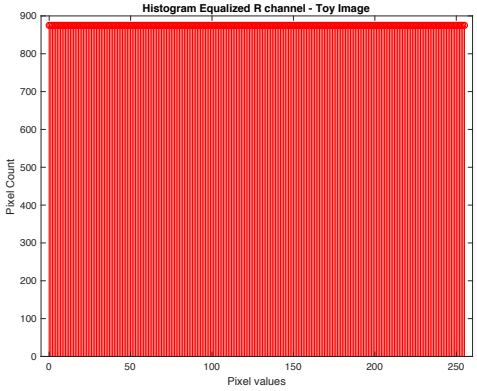

Histogram B channel - Toy Image
Pixel Count
Pixel values


CDF Histogram Equalized G channel - Toy Image
Pixel Count
Pixel values


CDF Histogram Equalized B channel - Toy Image
Pixel Count
Pixel values







4. Discussion:

- The histograms of original low contrast R, G and B channels are shown above.
- Method A was applied to the original image and the transfer functions for all three R, G and B channels are shown above
- Method B was applied to the original image and the cumulative histograms for all three R, G and B channels are shown above
- From the histograms of the enhanced image – The following can be observed:
  - Method A is like expanding the histogram. It's simple and enhances the contrast of the image. This method worked in the above image because the histogram was concentrated towards the left end. But if the pixel values were far apart from each other, then contrast enhancement using this method will fail. Also, the brightness of the image looks altered.
  - Method B perfectly equalizes the histogram. The number of pixels in every bin is equal. It has good performance than method A in contrast enhancement. But since every pixel has to be checked and binned, the process can become computationally expensive for larger images.
- Since both the methods have their own cons in enhancing the image, we can go with some other histogram equalization method, where it is computationally simple (like method A) and also equalizes the histogram for every bin (like method B).
- There are many other algorithms like Bi-Histogram Equalization, Par Sectioning, Odd Sectioning etc. We can choose the best method over Method A and Method B that meets the requirement.
-

Advantages in Method B output:

1. Better Contrast
2. Image more soothing to the eye
3. Sharper Edges
4. Lesser Distortion of Image
5. Linear Transfer Function for the entire range

Problem 2: Image Denoising

1. Abstract and Motivation:

In real-world any image we capture is affected by noise. There are many reasons why noises are added to the image. More often, the reason and the type of noise is unidentifiable. And images are exposed to different types of noise and denoising an image has always been a perfect research topic since denoising is an important concern in any Signal Processing Pipeline.

The mixed noise types that can be found in the image are:

- Salt Noise (Extra 255's in the image - white)
- Pepper Noise (Extra 0's in the image - black)
- Salt and Pepper Noise (Extra 0's and 255's in the image)
- Gaussian Noise (Noise added to the image follows Gaussian Distribution)

The type of filters that can be applied to the image to denoise are:

- Median Filter (3*3, 5*5 and 7*7) – Denoise Salt and Pepper Noise
- Low Pass Filter (3*3, 5*5 and 7*7) – Denoise Gaussian Noise
- Gaussian Filter ((3*3, 5*5 and 7*7) – Denoise Gaussian Noise

The idea of this section is to understand the type of distortions in images caused by noises of different type and try to denoise the images using different filters and algorithms.

The input image given has salt and pepper noise. There's no said thumb rule for deciding the filter to remove a particular kind of noise, or that certain filters only remove a particular kind of noise. Hence a lot of experimentation is needed to properly remove noise from an input image. Further, we can use a mix and match and cascading of filters to have end output image with least value of noise intensities.

Filters:

1. MEAN FILTER
In this filtering technique, the target pixel is replaced by the average of the pixels in its neighborhood pixel space depending on the kernel window size specified. This averaging process increases signal intensity and decreases noise intensity by making use of the uncorrelation between the image pixel intensities and noise pixel intensities. The output image of this filter is governed by the following set of equations:

$$Y(i,j) = \frac{\sum_{k,l} I(k,l)w(i,j,k,l)}{\sum_{k,l} \underset{w_1}{w(i,j,k,l)}}$$

$$w(i,j,k,l) = \frac{1}{w_1 \times w_2}$$

A typical 3*3 kernel for implementing Mean Filter is:

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

2. **GAUSSIAN FILTER**

In this filtering technique, the same concept as mean filter is followed, but the filter coefficients are obtained from the gaussian function:

$$w(i,j,k,l) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(k-i)^2 + (l-j)^2}{2\sigma^2}\right)$$

Example of 5*5 Gaussian Filter :

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

I obtained Kernel values for gaussian filter from this website:

http://dev.theomader.com/gaussian-kernel-calculator/

3. **BILATERAL FILTER**
   Since Linear filtering techniques like gaussian and mean filter cause blurring of edges, we use advanced filtering technique to overcome this. Bilateral Filter is such an option.
   The weights for the filter are defined by the following equation :

$$w(i,j,k,l) = exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_c^2} - \frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_s^2}\right)$$

   Where the tuning parameters for improving the performance of the filter are sigma_c and sigma_s.

4. **NON-LOCAL MEAN FILTER:**
   This is another non-linear advanced filtering technique. This filtering technique makes use of the fact that if two pixels have a similar neighborhood, they have higher probability of being similar. The filter is governed by the following equations:

$$Y(i,j) = \frac{\sum_{k=1}^{N'}\sum_{l=1}^{M'} I(k,l)f(i,j,k,l)}{\sum_{k=1}^{N'}\sum_{l=1}^{M'} f(i,j,k,l)}$$

$$f(i,j,k,l) = \exp\left(-\frac{\left\|I(N_{i,j}) - I(N_{k,l})\right\|_{2,a}^2}{h^2}\right)$$

$$\left\|I(N_{i,j}) - I(N_{k,l})\right\|_{2,a}^2 = \sum_{n_1,n_2 \in N} G_a(n_1,n_2)(I(i-n_1,j-n_2) - I(k-n_1,l-n_2))^2$$

   and

$$G_a(n_1,n_2) = \frac{1}{\sqrt{2\pi}a}\exp\left(-\frac{n_1^2+n_2^2}{2a^2}\right)$$

5. **PSNR(Peak Signal to Noise Ratio)**
   This quality metric is used to assess the performance of the denoised output image. The formula used are:

$$\text{PSNR (dB)} = 10\log_{10}\left(\frac{\text{Max}^2}{\text{MSE}}\right)$$

$$\text{where MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(Y(i,j) - X(i,j))^2$$

X : Original Noise-free Image of size $N \times M$

Y : Filterd Image of size $N \times M$

Max: Maximum possible pixel intensity = 255

(Source: Lecture notes)

6. BLOCK MATCHING AND 3-D TRANSFORM FILTER

A sparse image representation is image representation using optimal features and information. Block matching is novel denoising algorithm based on enhanced sparse representation in transform-domain. The enhancement of the sparsity is achieved by grouping similar 2D image fragments (e.g. blocks) into 3D data arrays which we call "groups" [8].

Collaborative filtering is special type of algorithm which is used while working in 3D groups. Input noisy image is divided in various 2D blocks and similar blocks are stacked together forming one 3D groups. We get hard threshold transform coefficients after applying transform. Now converting groups again back to separate 2D blocks gives a basic estimate of an image. By reducing noise up to great extent, collaborative filtering reveals very fine details shows by stacked 2D groups. The filtered blocks are returned to the original position. Just like PCA, overlapping of blocks occurs in this algorithm and aggregation of block output is taken to produce better estimate. Even better improvement is obtained using special Weiner filter.

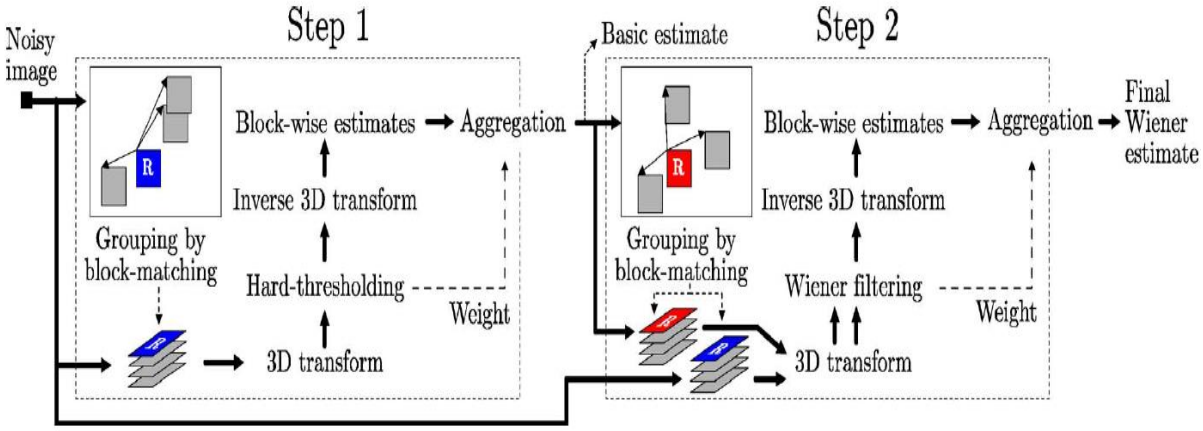


Figure 13: BM3D Algorithm

1. Approach and Procedure:
   We apply the Mean filter, Gaussian filter, Bilateral filter and NLM filter to the Corn_noisy.raw image and get the denoised output for various window sizes (NxN) and also calculate the PSNR for each of the N and compare the performance of the filters. The output image for each of these parameters is shown below.
   The first step is to choose a 2D window size for capturing the neighboring pixels. the filter traverses through the image and replaces the center pixel with new pixel value created using the filter.

   Boundary Extension of Images: For implementing the filter windows on the edge pixels of the image, we need to pad dummy pixels on the boundaries of the image depending on the size of the filter window selected. There are four possible methods of boundary extension.

   1. Zero padding
   2. Pixel Replication
   3. Reflection
   4. Linear Extrapolation

Here I have used Reflection on the image borders.

Algorithm for Implementing Denoising :

- **For Gaussian and Linear Filters :**
    1. Define Linear and Gaussian Filter Kernels
    2. Augment the image i.e implement boundary extension.
    3. Define a convolution function that convolves each input pixel with the corresponding filter. The final output pixel is the weighted average of the filter kernel convolution with the pixel.
    4. Iterate through the entire image pixel, convolve it with the corresponding filter kernel and store the new pixel in another output image data created.
- **For Bilateral Filter:**
    1. Define Bilateral Filtering Function
    2. Augment the image i.e implement boundary extension.
    3. Iterate through the entire image pixel, convolve it with the corresponding bilateral filtering function and store the new pixel in another output image data created.
- **For NLM Filter**
    1. Define NLM filter Function as per the equation.
    2. Augment the image i.e implement boundary extension.
    3. Iterate through the entire image pixel, convolve it with the corresponding NLM filter kernel and store the new pixel in another output image data created

2. Experimental Result:

LINEAR FILTER OUTPUTS:

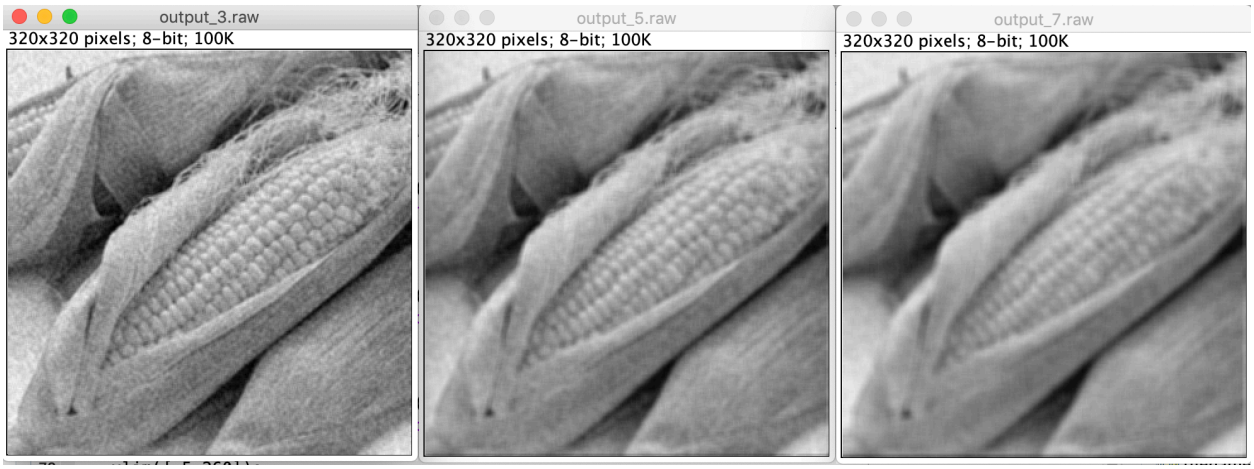


Figure 14: Output of Linear filter with N=3,5,7 from left to right

Table 1: PSNR values of Linear Filter

Filter Size	PSNR
N=3	20.564
N=5	19.7328
N=7	19.295

GAUSSIAN FILTER OUTPUT IMAGES:

- **Gaussian filter output images for Sigma=1:**

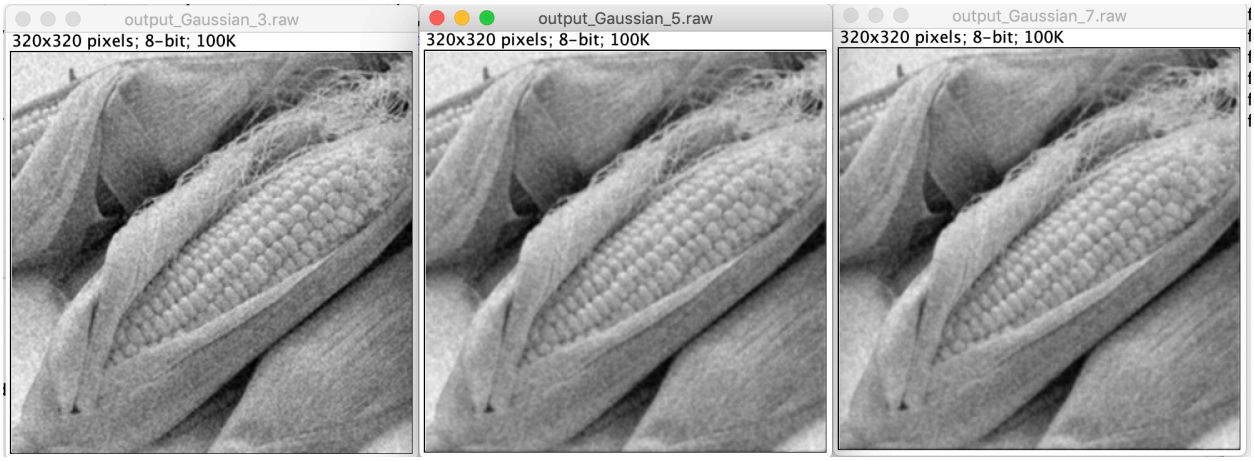


Figure 15: Output of Gaussian filter with N=3,5,7 with Sigma value 1

Table 2: PSNR values of Gaussian Filter with SIgma =1

Filter Size	PSNR
N=3	21.4399
N=5	20.9624
N=7	20.9015

- **Gaussian filter output images for Sigma=2:**

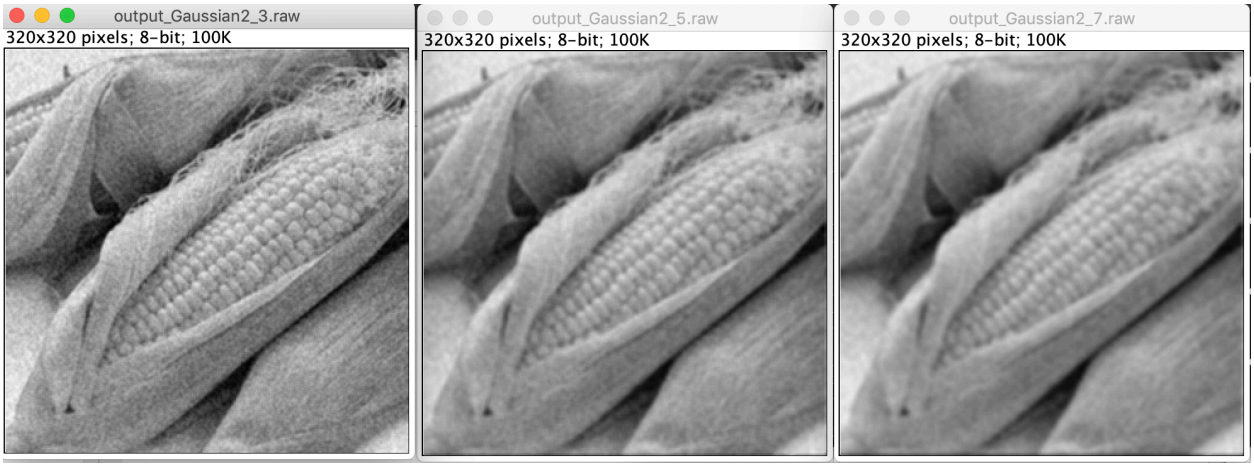


Figure 16: Output of Gaussian filter with N=3,5,7 with Sigma value =2

Table 3: PSNR values of Gaussian Filter with SIgma =2

Filter Size	PSNR
N=3	20.7651
N=5	20.0195
N=7	19.7269

BILATERAL FILTER OUTPUTS:
Specifications:
Sigma_s=75
Sigma_c=10

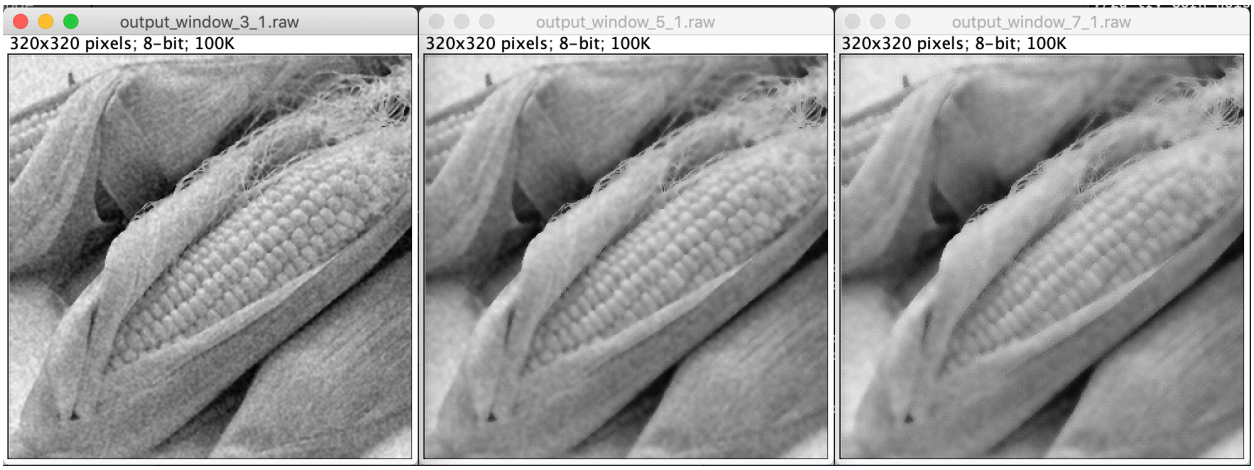


Figure 17 Output for Bilateral filter for N=3,5,7

Specifications:
Sigma_s=750
Sigma_c=100

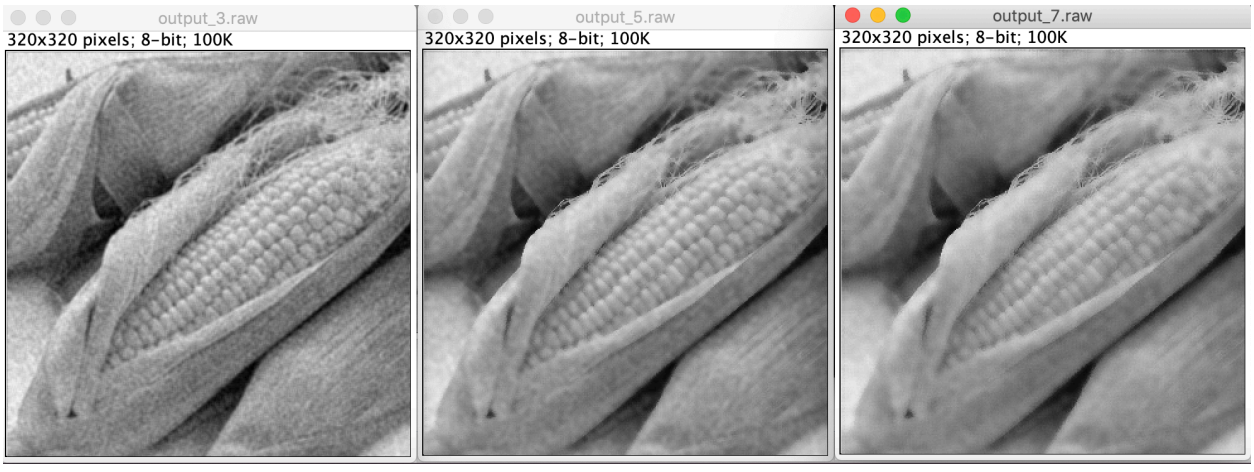


Table 4: PSNR values for diiferent parameters of Bilateral Filter

Filter Size	PSNR for σ_c=10, σs=75	PSNR for σ_c=100, σs=750
N=3	20.7651	18.9729
N=5	20.0195	17.8846
N=7	19.7269	17.1859

NLM FILTER OUTPUT:

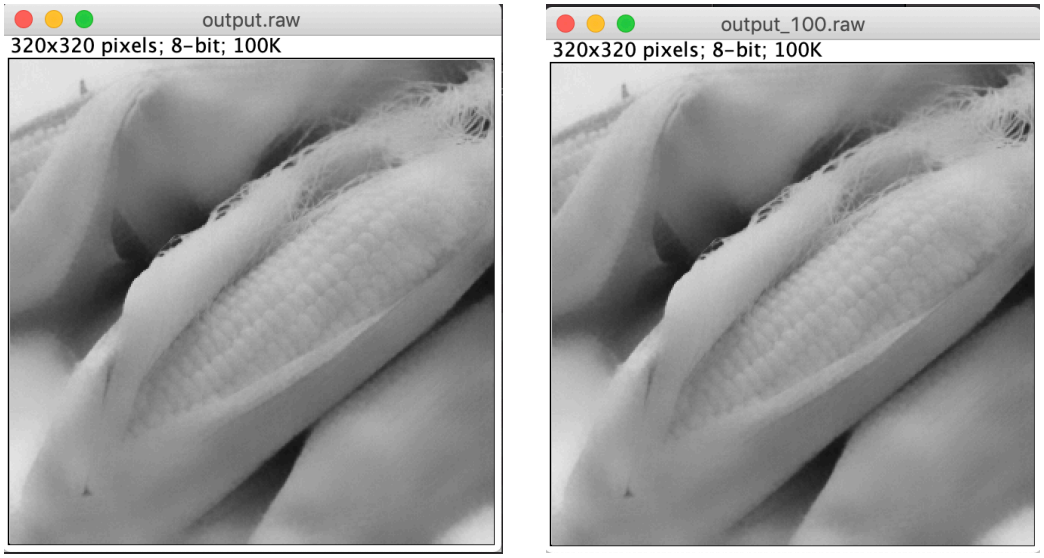


Figure 18: NLM Filter output with N=3, h=10(left) and h=100(right)

BM3D FILTER OUTPUT:

PSNR 19.128

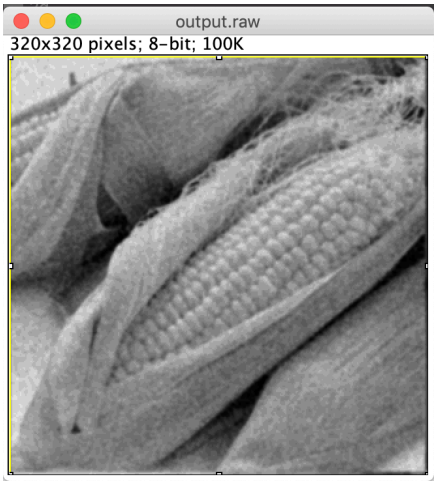


Figure 19: BM3D Filter output with N=3

- **Discussion:**

We know that, higher the PSNR, better the Denoising. From the above results, we can see that, the filtering is best for Filter Size :

Further, window size plays an important role in denoising. On increasing the window size, we get better denoised output. But, it also comes at the cost of more deblurring in the image. The linear filter of window size NxN is applied where N can take odd values – 3,5,7,9.., N if small isn't so powerful but if N is large, it causes blur.

The results showing PSNR values for different window sizes for different types of filters and the output denoised images is shown above.
The Gaussian filter PSNR decreases rapidly as the window size is increased gradually whereas for the Bilateral filter the PSNR drops by a lesser value as the window size increases.

Higher PSNR indicated better denoising, but it doesn't take into consideration burring effect. So, we might have an image which is more blurred but it also might have high PSNR. Further, window size affects sharpness of Image. We get more blurring of image as we increase the window size .

In Bilateral filtering, the PSNR increases as we increase sigma_s and sigma_c values, Hence, we get better denoising images as we increase sigma_s, sigma_c. The Bilateral filter performs better than Linear and Gaussian filters , we can be easily seen through oputput images as well as through high PSNR values of Bilateral filter compared to other.

Further the Non-Local Mean (NLM) filter was implemented which gives much better results. The denoising effect increases with the increase in filter parameter. However, it is computationally expensive.

BM3D gives very shape edges and very less distortion is produced. It is collaborative algorithm where from input image 2D blocks of similar patterns are grouped together, filter is applied to this group and Weiner estimate gives output 2D blocks which are then placed in original image.

BM3D is both spatial domain as well as frequency domain filter. It involves partition of input image into similar blocks in spatial domain and Weiner filter is applied and estimated in frequency domain to remove the noise.

Generally, higher PSNR indicates better denoising but often undermines the blurring effect. So, an image which is blurred output might have high PSNR. Window size alters sharpness of image as it generally loses it high frequency content. Comparing performance of BM3D and PLPCA, BM3D produces better output in both sense. One being PSNR is greater than PLPCA and image has very sharp edges and there is no distortion.

Hence BM3D might be considered as better algorithm to denoise due to novel collaborative algorithm but there might be certain cases where PLPCA works better than BM3D.

Research is still going on this noise removal area to increase the PSNR value. We can try other filters like max-min filter, mid-point filter, alpha-trimmed mean filter. There are also many other techniques developed to remove noise like Non-Local Means, Wavelet transforms, block-matching algorithms and other statistical methods involving machine learning.

2e. Mixed Noises in Color Images:

1) The color image has Salt and Pepper noise as it has white and black dots in it. And the noise is uniformly distributed. Hence it has uniform noise.
2) Filtering should be performed on individual channels separately for both noise types. This is because all channels have mixed noises present and applying filters to the whole would not result in a clean image as compared to filters applied to each channel individually. Hence, denoising is done by applying filters to individual channels for getting better results soothing to the human eye
3) As the nature of noise is mixed noise, we need to use both linear and non-linear filters to remove uniform and impulse noise. We need to cascade two kind of filters so as to remove both types of noise.

References:

1. https://en.wikipedia.org/wiki/Demosaicing
2. http://www.ipol.im/pub/art/2011/g_mhcd/?utm_source=doi
3. http://dev.theomader.com/gaussian-kernel-calculator/
4. https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf
5. http://www.cs.tut.fi/~foi/GCF-BM3D/
6. http://www.librow.com/articles/article-1
7. http://www.cplusplus.com/reference/
8. https://stackoverflow.com/questions/6533570/implementation-of-non-local-means-noise-reduction-algorithm-in-image-processing