EE569: HW#1

Student Name: Shi-Lin Chen
Student ID: 2991911997
Student Email: shilinch@usc.edu
Submission Date: Jan 28, 2020

In this report, I will have four different contents in each problem, which is 1. Motivation 2. Approach 3. Result and 4. Discussion. First, I will shortly talk about my motivation of each problem and what can it do in the **Motivation part**. The method how I implement or algorithm will be explained in **2. Approach**. Then show the result of my implementation at **3. Result**. In the end, I will put some my observations and answer the questions of the assignment and some details related to the Problem in **4. Discussion**.

Problem 1: Image Demosaicing and Histogram Manipulation

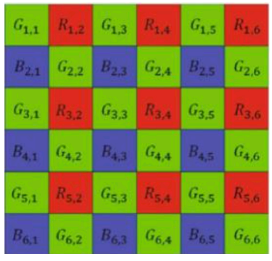
1.1 Motivation:

In this problem, we will implement **Image Demosaicing** and **Histogram Manipulation** with different method. The First one, Image Demosaicing could demosaic the image from gray scale to RGB by linear interpolation and MHC interpolation. Histogram Manipulation could adjust the Histogram of a image, which could enhance some details(like contrast) in the image and adjust the lightness. Though this problem, we could understand more about the method of image demosaicing and histogram.

1.2 Approach:

1.2.(a) Bilinear Demosaicing:

First, if we need to implement Bilinear Demosaicing, we need to figure out that digital camera sensors are usually arranged in form of CFA, called Bayer array. After understanding Bayer Array, we should understand bilinear interpolation. We will get the color component we wish to have by average the same color in neighborhood position.



<< Bayer pattern

Then we can start implementing Demosaicing.

Step 1: Read raw data and store them in cpp programming.

Step 2: We will do **mirror reflecting** to avoid Image boundary fail.

(These two steps are always the FIRST STEP to do in every programming, I will write this one time, but all programming includes these two steps)

Since bilinear interpolation will calculate neighborhood pixel values, and the boundary pixel can't calculate the pixel without value. For example: array Imagedata[-1][-1] doesn't have correct value when processing the interpolation on Imagedata[0][0].
Like below:

```
/**********************************************************
 partten: 1. GRG //all G1 values are at even row and even column
            BGB
            GRG

          2. GBG //all G2 values are at odd row and odd column
            RGR
            GBG

          3. RGR //all B values are at even row and odd column
            GBG
            RGR

          4. BGB //all R values are at odd row and even column
            GRG
            BGB
**********************************************************/
```

If original Image data is Width * Height, the processing one will be (Width+4) * (Height+4). After processing, we will crop the image to its original size.

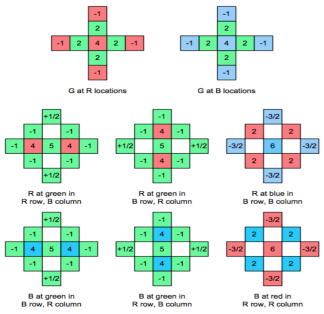
Step 3: Knowing Bayer pattern, we could categorize 4 different patterns. After knowing the pattern. we could implement bilinear interpolation to each pixel.
We could use double for-loop to run each pixel in the image. And then using "Switch-case" to recognize patterns if it is even odd row or column. And then calculate Red, Blue and Red value at each pattern. After computing one pixel, store RGB value at a new array, which is the image data I use to write a new raw data.

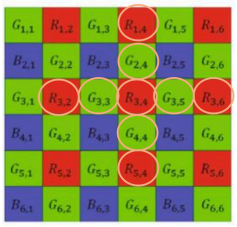
Step 4: Write into a new .raw file. (with size Width*Height*3(RGB))

Note: Since we do Image boundary in the beginning, the for-loop will start at row = 2, column = 2, which is the original pixel value we want to process.

1.2.(b.) Malvar-He-Cutler (MHC) Demosaicing:

This method improves linear interpolation demosaicing by add a 2^{nd}-order cross channel correction term to the basic bilinear demosaicing result. By these:





$$\hat{G}_{3,4}^{MHC} = \hat{G}_{3,4}^{bl} + \alpha\Delta_R(3,4)$$

$$\Delta_R(3,4) = R_{3,4} - \frac{1}{4}\left(R_{3,2} + R_{1,4} + R_{3,6} + R_{5,4}\right)$$

The steps are almost the same, except the calculating 2nd-order correction term.
The coefficient of correction can be obtain by the image on the left side!
Note: We still have different patterns to be recognized, I use if row and column is even or odd to do recognizing the pattern. (R_MHC is Red value that the original R + correction.)

1.2.(c.) Histogram Manipulation-Method A

There are two method to manipulate Histogram in this part, I will discuss one by one.
Method A: the transfer-function based histogram equalization method.

Step 1: obtain the histogram, count the frequency of pixel of each value (0~255) , and print them to a .txt file in the meantime. (for latter plotting histogram in matlab)

Step 2: Calculate the normalized probability histogram.

Step 3: Calculate the CDF.

Step 4: Create the mapping-table. (X to CDF(x) * 255)

By Calculating the CDF of the histogram, we could simply transfer our original pixel value(0~255) to new pixel value, since the CDF is from 0~1. For example, the original pixel value is 50, then the CDF of that point is 0.20. we could do just do 0.2 * 255 = 51, so the mapping of original data to new data will be 50 -> 51.
Note: We have to round down the value (25.5 -> 25), since CDF multiply 255 could usually be fraction.

Step 5: Write new Image data with new mapping table to a .raw file.

1.2.(c.) Histogram Manipulation-Method B

Method B: Cumulative-probability-base histogram equalization
In this method, I use "bucket filling" method to implement the equalization.

Step 1: Use a **unorder_map** to store RGB value. The **key is the intensity value** of the pixel(0~255), and the **value is their position**.(I use pair and stackdata type to store row and col) Therefore, the map would be like **(0~255) -> (row,col)** for every pixel.

Step 2: Put position data which bucket's size is more than 875 (Image size / 256)

into a queue. (from 0 to 255)

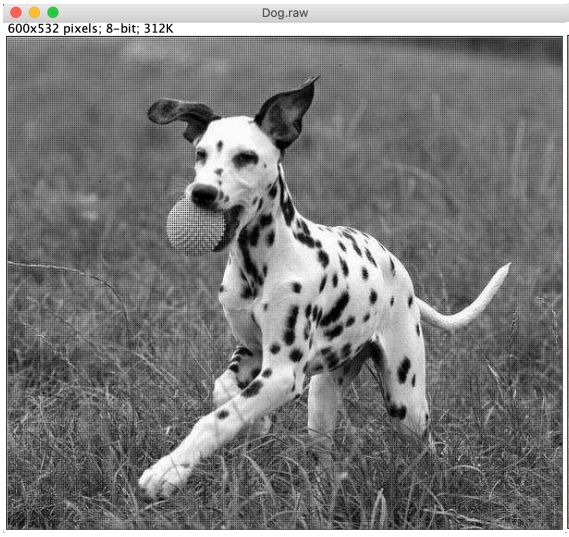
Step 3: pop the position data out form queue to those buckets which have pixel less than 875 until each bucket are filled with 875 position data. Now, we will have a map with 256 keys (0~255) related to value (875 pixel position).

Step 4: write the map into image and produce a new .raw file.

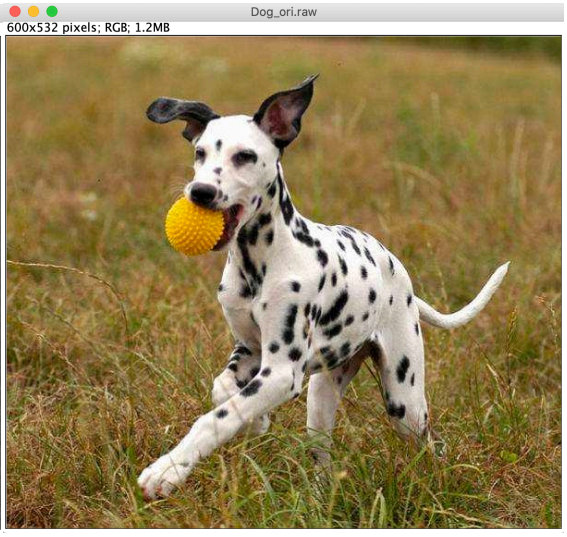
1.3 Result

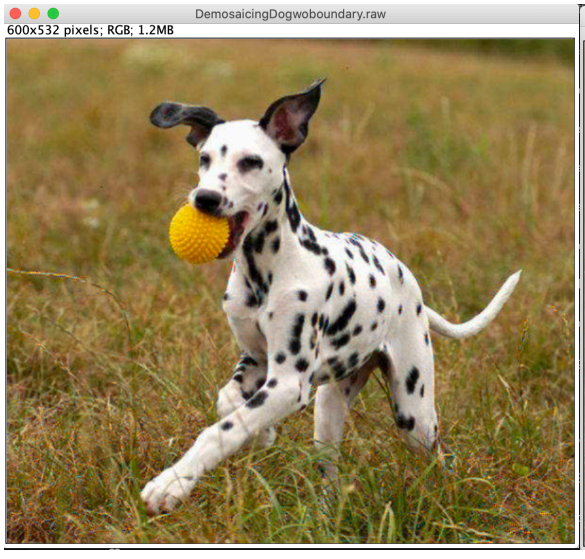
1.3.(a.) Bilinear Demosaicing & MHC Demosaicing

Original gray-scale Image

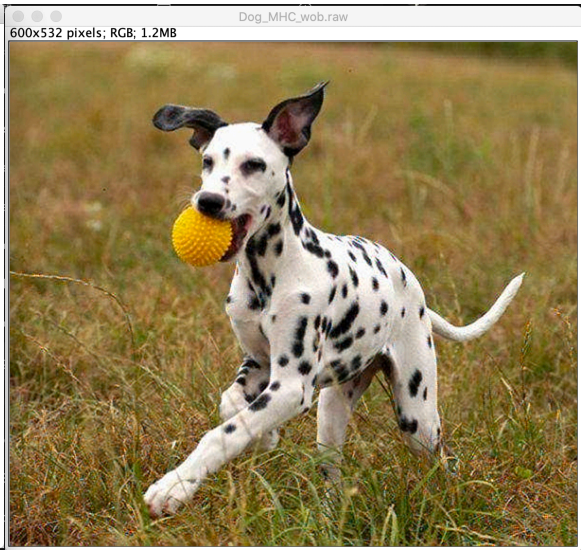


Original RGB Image

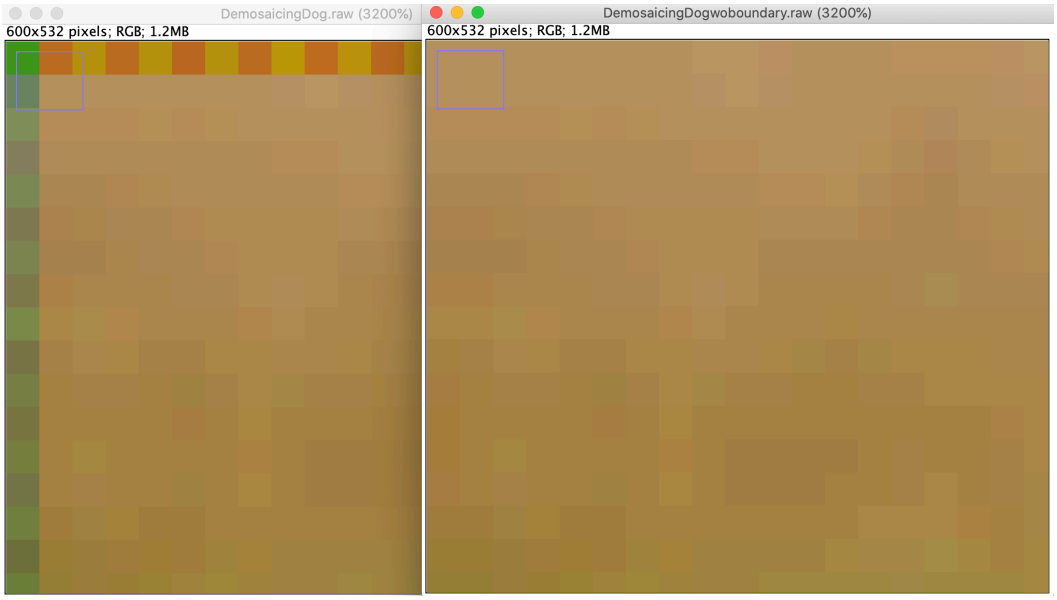




Bilinear Demosaicing



MHC Demosaicing



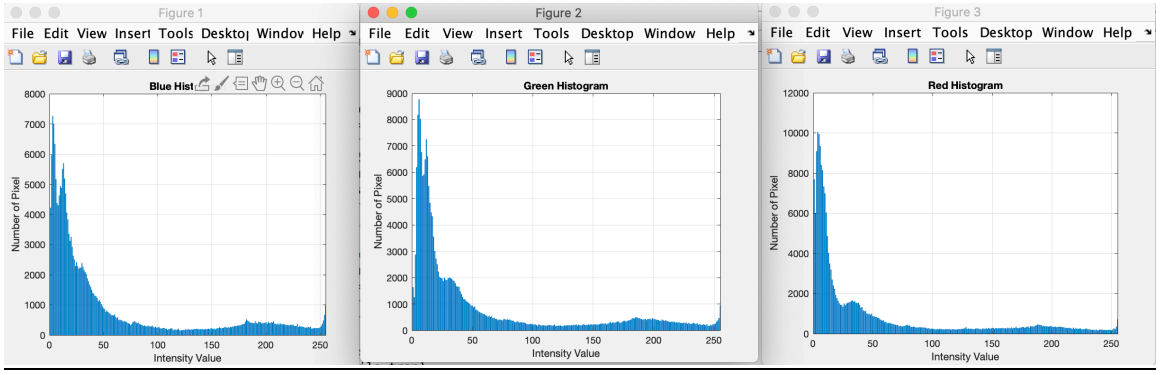
Note:

Left hand side is the image without mirror reflection (have strange pixels at 4 boundary).

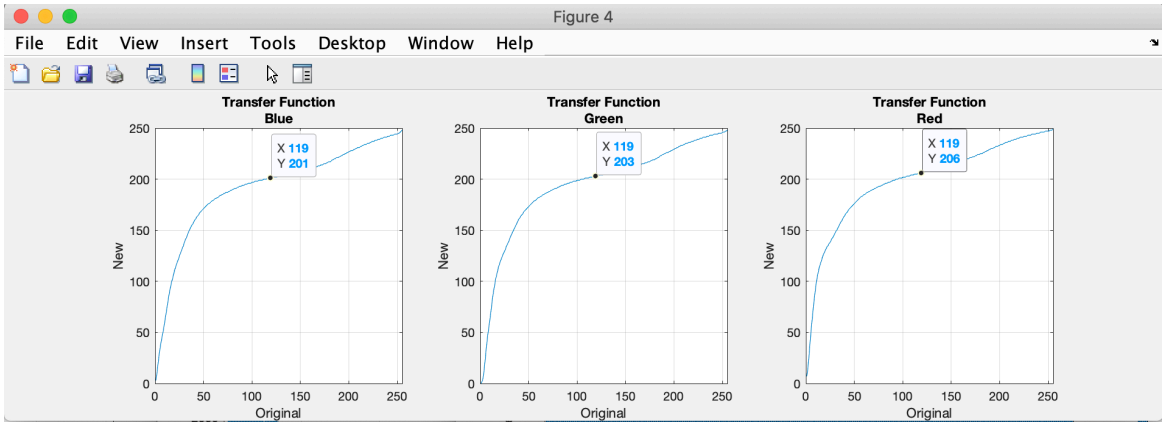
Right hand side is the image with mirror reflection on the image boundary

1.3.(c.) Histogram Manipulation

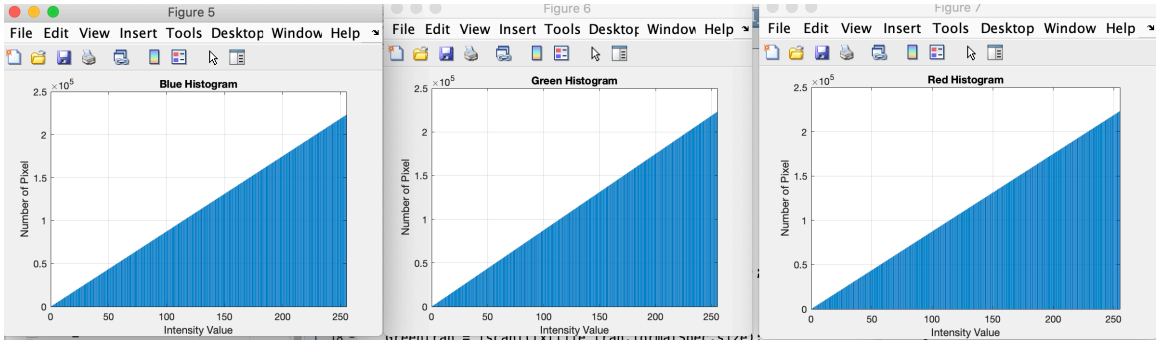
1. Histogram of 3 channels(B,G,R)



2. Tansfer function for 3 channels (Method A)



3. Cumulative Histogram for 3 channels (Method B)



4. Implement result of method A and method B Histogram equalization to original Toy.raw

Original Method-A Method-B

1.4 Discussion:

1.(a-2) – Bilinear demosaicing

Yes, by the result bilinear interpolation performed, we can see artifacts across the edges and other high-frequency contents (like the grass at the downside of the image). The details (like edges) are being blurred in some way. The one way I can think of is to take correlation into account among RGB values, which is same as MHC doing!

1.(b-2) – MHC demosaicing

Compare to bilinear demosaicing, MHC has better performance, since MHC considers the correlation of RGB values at each pixel value calculating, which preserves more details than bilinear demosaicing does.

(In MHC result, grass at the downside of the image are obviously better)

1.(c-4) – Histogram manipulation

In method A, we use transfer function to obtain a new image, which we can see that the brightness of the image has increased. The detail at dark side are also more obvious.

In method B, there are some color distortions in the image (I might have some error implementing Method B. But, as my suspicion, the color distortion might be caused by the bucket filling method. Since bucket filling method could only implement each channel separately, the value of each pixel might have different level of enhance or attenuation. It means that the color components of each pixel might lost the original color proportion, therefore, the pixel color distorts.
Mu suggestion is, if we can consider three colors implementing together, and keep their proportion, for example, RGB value of one pixel could get 70% increasement in the same time (10,20,30) - > (17,34,51), the result might be improved or not losing the image's reality.

Problem 2: Image Denoising

2.1 Motivation:

In this problem, we will use different methods (Uniform filtering, gaussian filtering, bilateral filtering, NLM filtering and BM3D filtering) to implement denoising on a noisy image. We could learn different kinds of noise in image and what filter should we choose to denoise the image in order to have the best performance on different kinds of noise. We also get to learn PSNR (peak-signal-to-noise-ratio) to compare the performance of denoising.

2.2 Approach:

2.2.(a.) – Basic method filtering

In this problem, we need to apply two different filter to a noisy image to implement denoising. We can see that the process of denoising is average pixels value in a N*N area and calculate the average and put it in the center of the pixel, the approach of filtering in uniform function and gaussian function could be really similar, except the coefficient and weight of the neighborhood pixels value, so I will show two method below together.

Step 1: Build two filter, one with uniform function, another one with gaussian function. And my reference of choosing parameters are as below:

1	1	1
1	1	1
1	1	1

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

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

3*3 filter with Uniform function

- Low pass filter (Gaussian):

$\frac{1}{273}$

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

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

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

where σ is the standard deviation of Gaussian distribution.

5*5 filter with Gaussian function

Step 2: Do convolution to the Image data and the filter for all pixel. For example, we have a image data in nine pixels which are (1,1) (1,2) (1,3) (2,1) (2,2) (2,3) (3,1) (3,2) (3,3), we use these nice pixel values to do convolution with 3*3 filter, and we get one average value for the center of the 3*3 area, which is (2,2). The approach is the same in Gaussian filtering, and we do all these convolutions for all of the pixels in the image.

Step 3: Write data after convolution to a new .raw file and we get our result.

Note: We will also try different size of mask in uniform filtering (3*3, 5*5)

2.2.(b.) – Bilateral Filtering

In this problem, we should understand that Bilateral Filtering take two factors into account. The two factors are **distance weight** and **intensity weight**. We use the formula below to implement this method.

- Bilateral filter:

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

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

We can simply see that at the left hand side part of the w(i,j,k,l), is the distance factor, the closer two pixels are, they could influence the w function more. And at the right hand side part of the w(i,j,k,l) is the intensity factor, the value two pixels are, they could influence the w function more. Base on the method, we can start our programming.

Step 1: I produce a function window, with input of original pixel row and column(i,j) and its neighborhood pixels (k,l), sigma_c and sigma_s.

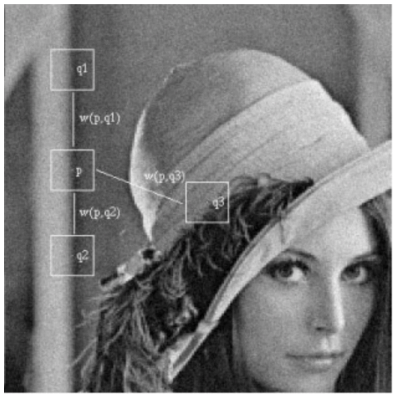
Step 2: write for-loop to calculate every new pixels Y(i,j) by the formula provided above.

Step 3: Write new data image to a .raw file.

Step 4: Try different parameters (adjust sigma_s and sigma_c).

2.2.(c.) – Non-Local Mean (NLM) Filtering

In this problem, we use NLM filtering to denoise the image. NLM is a method using several areas which has similar pixels contents (usually depends on the intensity of values of a pixel in N by N) to get average instead of the neighborhood average of a center pixel. Like the picture below. There are weights to be considered among these "similar areas" by distance. The area weights are more important when the distance of two areas is smaller. For example, in the picture below, p is the center pixel we want to process, and there are 3 different weights q1, q2, and q3. Q3 is the furthest area, therefore, the weight of q3 will influence p the less. And we should implement every pixel like this. The formula of this algorithm is:

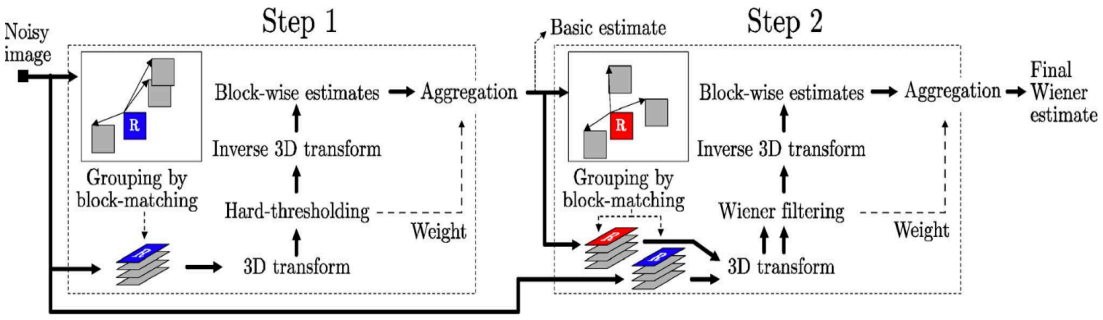


$$Y(i,j) = \frac{\sum_{k,l} I(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

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

2.2.(d.) – Block matching and BM3D transform filter

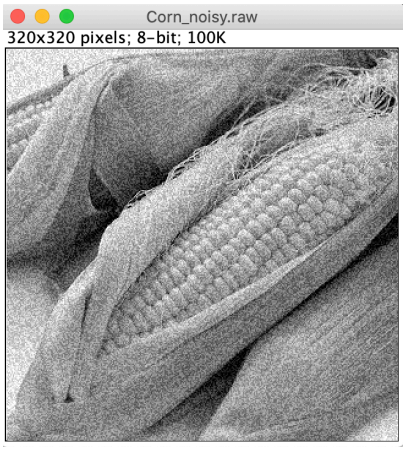
In this problem, we need to use 3-D Transform-Domain Collaborative Filter to denoise the image. There are two steps in the algorithm and they are very similar except the step in "Hard-thresholding" and "Wiener filtering".



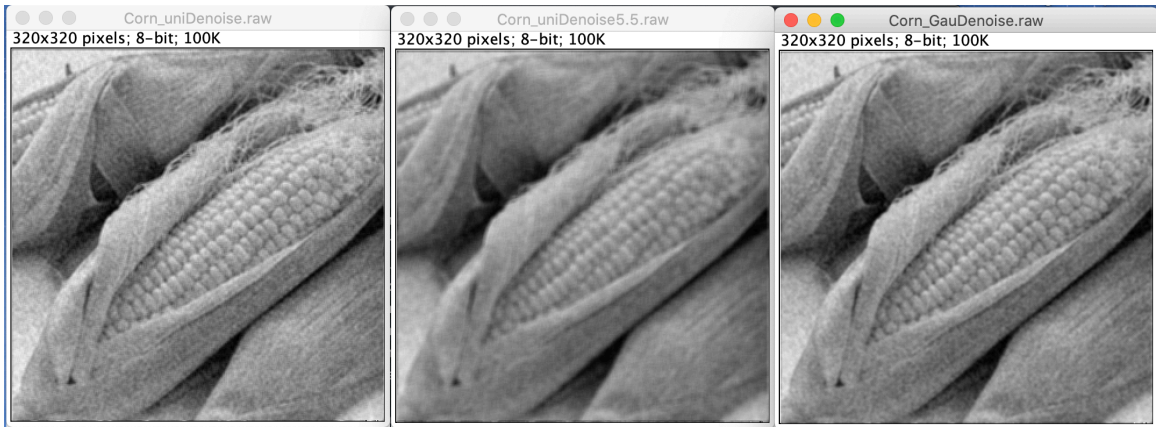
First, we need to group the image area by block-matching. We have to find blocks that are similar to the reference one(block-matching) and then stack them together to form a 3-D array. Then we need to process all reference blocks and use hard-thresholding method to threshold the data at 3th dimension of the array. The we set the values below the threshold to zero, and we will calculate the weight of other values above threshold. After calculation the weights, we can use inverse 3D transform to transform our 3-D array to 2-D array and then map into the image. Then we use non-zero values at 3-dimension to compute weight and the image data, and we will get the image data filtering most of noise.

2.3 Result

2.3.(a.) – Basic Denoising Method



Original Noisy Image



3*3 uniform filtering 5*5 uniform filtering 5*5 gaussian filtering

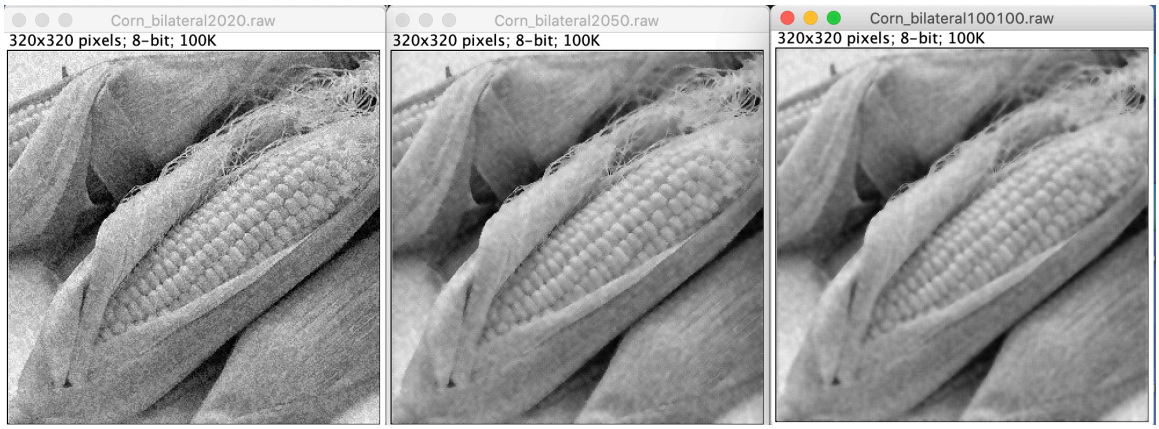
The PSNR of corn Image (by 3*3 uniform function filter) is: 18.2409

The PSNR of corn Image (by 5*5 uniform function filter) is: 16.2397

The PSNR of corn Image (by 5*5 Gaussian function filter) is 16.3518

2.3.(b.) – Bilateral Filtering

Note: In this implementation, I choose different value of parameters of sigma_c and sigma_s and their result is below:



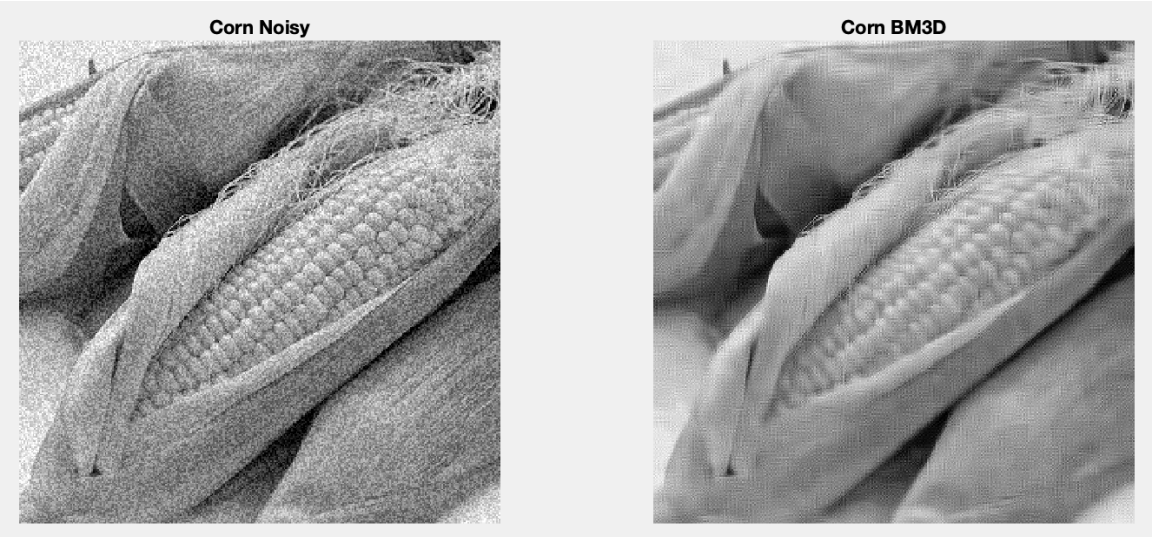
Sigma_c = 20 , sigma_s = 20 c = 20, s = 50 c = 100, s = 100

PSNR = 18.6363 PSNR = 19.5654 PSNR = 19.3431

2.3(d.) – BM3D

Source code reference:

http://www.cs.tut.fi/~foi/GCF-BM3D/



PSNR = 18.8066

2.4 Discussion

Denoising image is somehow averaging the pixel value at each points. There are some factor that will influence the performance, one is distance, another one is the similarity of intensity, which will be discussed below by answering the questions in assignment.

2.4.(a.1)

The noise in Corn_noisy.raw is gaussian noise.

And we can see the performance of Gaussian function filter is a little bit better than Uniform function filter, since it takes distance weight (the coefficient of filter) into account when denoising. The PSNR is also better, when both filter mask is 5*5!

2.4.(b.2)

Sigma_c is **spatial parameter** and it can control the **weight of distance** of pixels.
Sigma_s is **range parameter** and it can control the **weight of intensity value** of pixels.
Since the weights are multiplied, which implied that as soon as one of the weights is close to 0, no smooth occurs! When we increase **sigma_s**, the Bilateral Filter becomes closer to Gaussian blur, because the rage Gaussian is flatter. When increasing the spatial. When we increase sigma_c, the Bilateral Filter will smooth larger features. From the implement result, we can see, if we increase the parameters, we will get a smoother noise image, which means the noise will be more blurred.

2.4(b.3)

The Bilateral Filter seems to perform better than linear filter. In linear filter, everything will be blurred, including the details and edges. But when we use Bilateral Filtering denoise the image, we could preserve more edges. If we look at the image above at

Bilateral Filter C = 20, S=50, and compare it to linear filter, we can see that the edges (several wires in the middle of corn) at Bilateral Filtering image are more clear than the edges at linear filter.

2.4(d.1)

The algorithm has already been explained in 2.2(.d) approach part.

2.4(e.1)

I think it is mix noises with gaussian noise and pepper-salt noise.

2.4(e.2)

Yes!

2.4(e.3)

I think we should perform filtering step by step and deal with both noises separately. I will denoise pepper and salt noise first, since its noise value are either 0 or 255, which can be simply took out. Then we should put the average pixel value at neighborhood pixels in the pixel we took out. After filtering pepper and salt noise, we could use linear or non-linear filter which I explain above to deal with last gaussian noise!