EE569 Digital Image Processing

HOMEWORK #1

Issued: 09/01/2010 Due: 09/02/2010

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #1
January 26, 2020

Problem 1: Image Demosaicing and Histogram Manipulation (50%)

(a) Bilinear Demosaicing (10%)

(b) Malvar-He-Cutler (MHC) Demosaicing (20%)

(c) Histogram Manipulation (50%)

1.1 Abstract and Motivation

The demosaicing operation plays a significant role in the field of digital image processing, which is aimed at reconstructing a full color image from incomplete color samples that are output from an image sensor overlaid with a color filter array (CFA). In other words, image demosaicing can help change lots of raw images into a viewable format.

Moreover, while conducting research on image processing, we usually utilize the histogram equalization with the purpose of increasing the global contrast of images. Pixel intensities can be distributed on the histogram better with help of this adjustment, making it possible to make areas of lower local contrast attain a higher contrast, which can improve the visual effect of images.

Therefore, in this report, I am about to use C++ to implement two image demosaicing algorithms: one is Bilinear Demosaicing, the other is Malvar-He-Cutler (MHC) Demosaicing. What's more, I will use the same programming language to realize two histogram equalization techniques: one is the transfer-function-based histogram equalization method, the other is the cumulative-probability-based histogram equalization method. After acquiring raw data files from Visual Studio 2019, I will use Matlab 2019b to show resulting images and plot required figures.

1.2 Approach and Procedures

(a) Bilinear Demosaicing

In this part, I will use C++ to implement the simplest method based on bilinear interpolation in order to translate a color filter array of primary colors into a color image containing the R, G, and B values at each pixel, meaning that the two missing color values at each pixel are approximated by using average values of its two or four adjacent pixels of the same color.

(b) Malvar-He-Cutler (MHC) Demosaicing

In order to improve the simple bilinear algorithm, I will add a 2nd-order cross-channel correlation term to basic bilinear demosaicing results in part (a).

To estimate the intensity value of the green channel at a red pixel location, we have

$$\hat{G}(i,j) = \hat{G}^{bl}(i,j) + \alpha\Delta_R(i,j)$$

where $\hat{G}^{bl}$ is the bilinear interpolation result in part (a) and the second term is an added correction term. For the second term, α is a weight factor, and Δ_R is the discrete 5-point Laplacian of the red channel.

To estimate the intensity value of the red channel at a green pixel location, we have

$$\hat{R}(i,j) = \hat{R}^{bl}(i,j) + \beta\Delta_G(i,j)$$

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

To estimate the intensity value of the red channel at a blue pixel location, we have

$$\hat{R}(i,j) = \hat{R}^{bl}(i,j) + \gamma\Delta_B(i,j)$$

where Δ_B is a discrete 9-point Laplacian of the blue channel.

The three weighting coefficients control how much correction is applied to calculate relevant values, and in my assignment, I set default values as:

$$\alpha = \frac{1}{2}, \beta = \frac{5}{8}, \gamma = \frac{3}{4}$$

The formulas above can be generalized to missing color components at each sensor location. While writing my C++ code, I will handle them accordingly.

(c) Method A: the transfer-function-based histogram equalization method

Step 1: Obtain the histogram: count numbers of pixels of each grayscale value (0~255) of the original image.
Step 2: Calculate the normalized probability histogram: divide the resulting histogram in Step 1 by the total number of pixels.
Step 3: Calculate the cumulative probability from the step 2's results
Step 4: Apply the mapping rule "x to CDF(x) * 255" and floor the decimal values obtained to the lower integer values.

(d) Method B: the cumulative-probability-based histogram equalization method

Step1: Suppose that there are 400*560=224,000 "balls" and 256 (gray scale 0-255) "buckets", calculate the number of pixels going into each bucket: 224,000 / 256 = 875 pixels/bucket.
Step2: Assign pixels in the original image to the corresponding bucket randomly.

First of all, I convert the raw data file into a one-dimensional array.

Next, I get three arrays representing three channels whose data come from that 1-D array and meanwhile record every element's location in that 1-D array.

Then, I sort three channels' data separately and assign them into different buckets, which means that every location in the original 1-D array will have a new intensity value.

Finally, I get the enhanced image because the original 1-D array is changed.

1.3 Experimental Results

The original "Dog" image and two resulting images by using Bilinear Demosaicing and MHC Demosaicing methods are shown in Fig.1.1, Fig.1.2, and Fig.1.3 respectively.

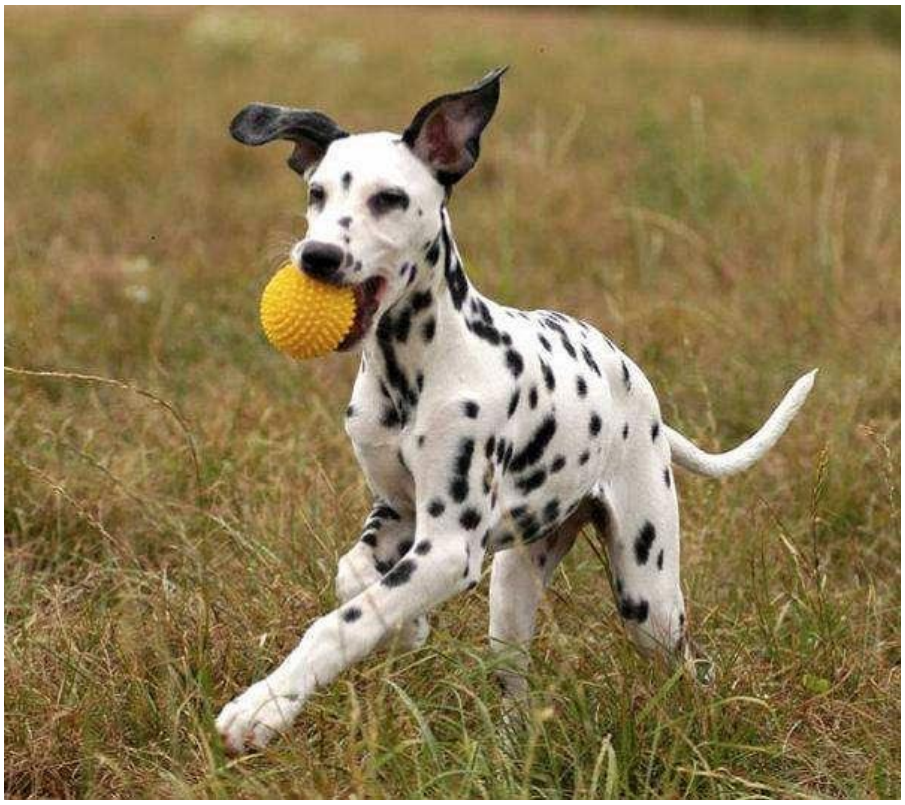
Figure. 1.1 The original “Dog” image

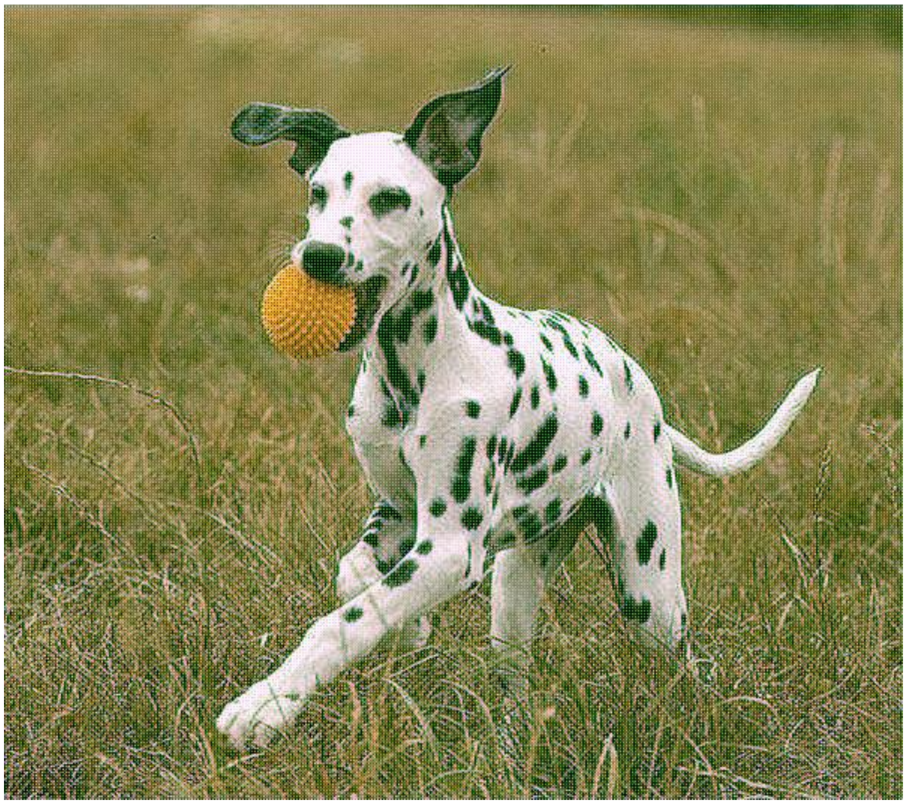
Figure. 1.2 The resulting image by using the Bilinear Demosaicing method

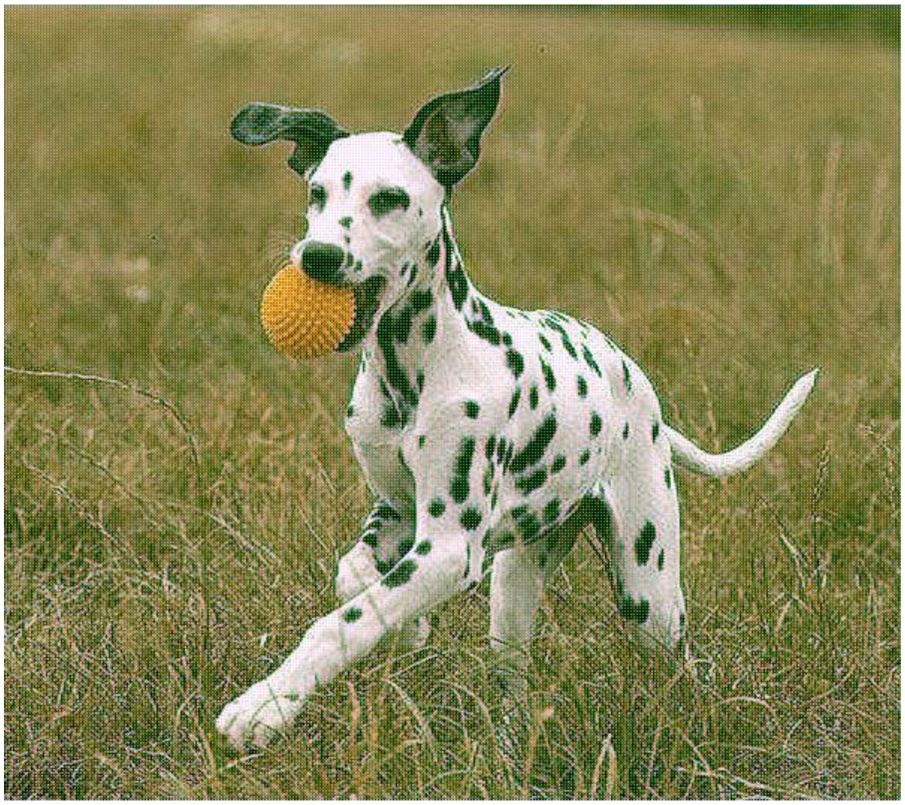

Figure. 1.3 The resulting image by using the MHC Demosaicing method

The histograms of the red, green and blue channels of the original “Toy” image are shown in Fig.1.4, Fig.1.5, and Fig.1.6 respectively.

The original “Toy” image is shown in Fig.1.7.

The enhanced image by applying the method A and its transfer functions for R, G, and B channels are shown in Fig.1.8, Fig.1.9, Fig.1.10, and Fig.1.11 respectively.

The enhanced image by applying the method B and its cumulative histograms for R, G, and B channels are shown in Fig.1.12, Fig.1.13, Fig.1.14, and Fig.1.15 respectively.

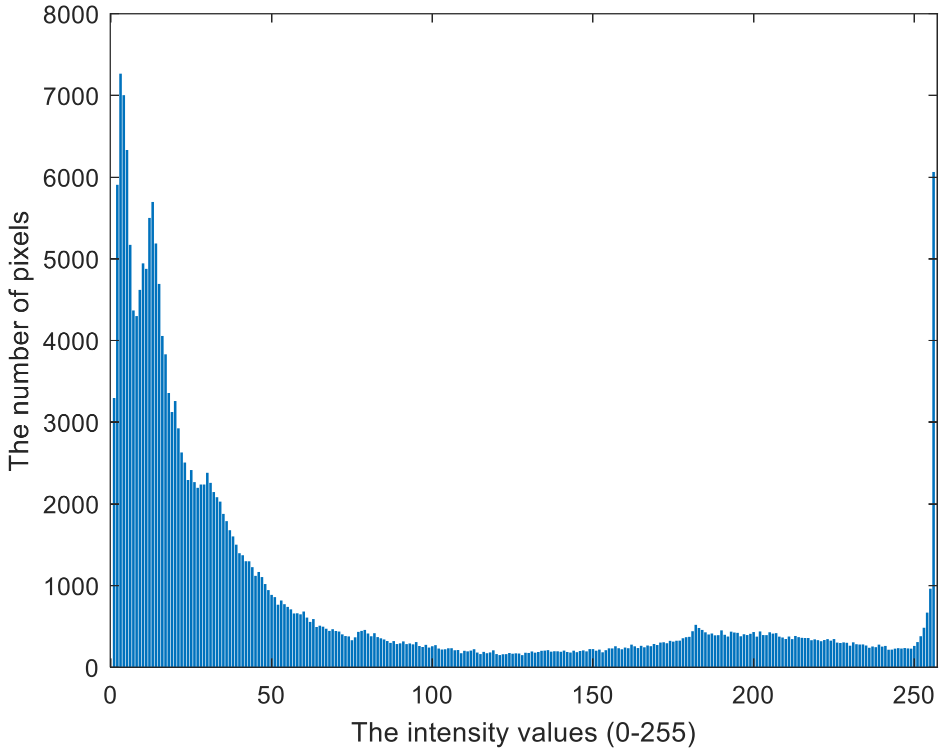

Figure. 1.4 The histogram of the red channel of the original image

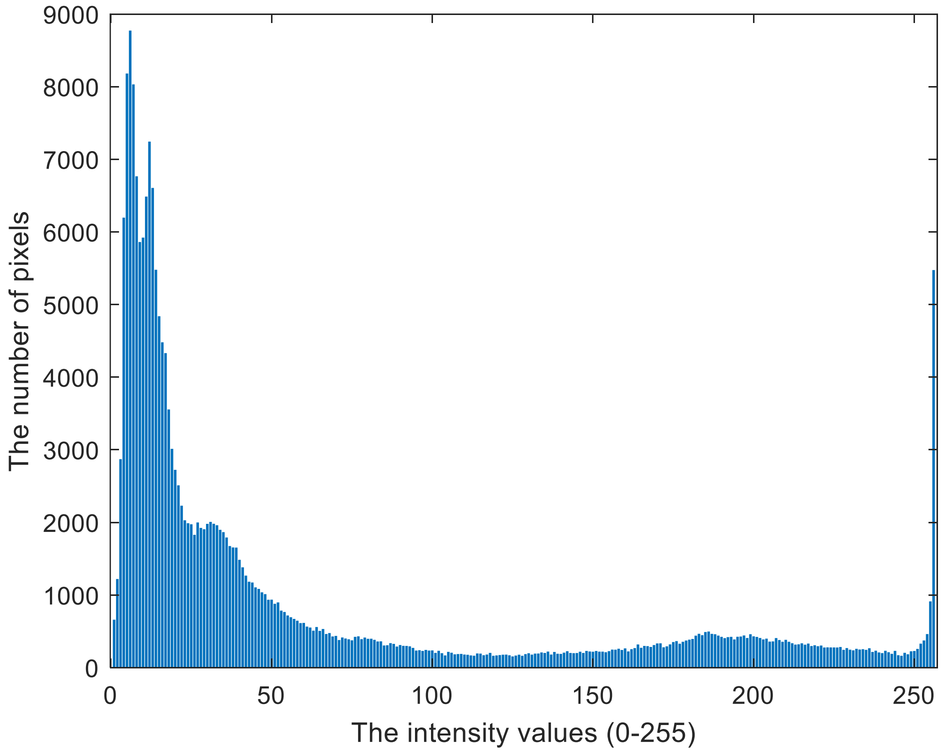

Figure. 1.5 The histogram of the green channel of the original image

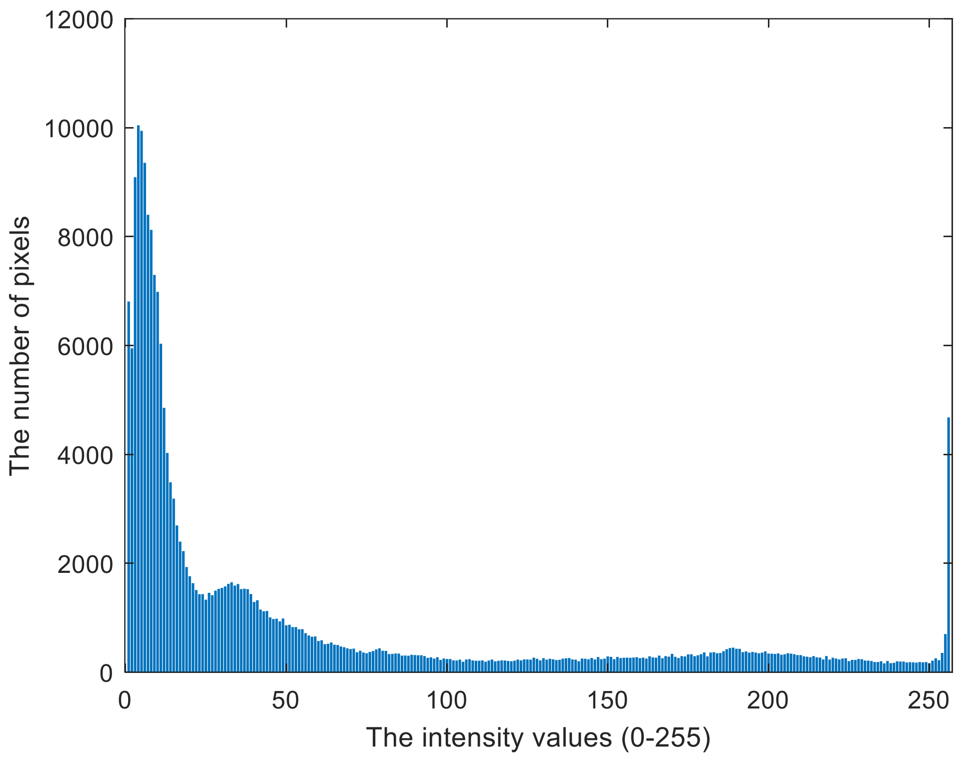

Figure. 1.6 The histogram of the blue channel of the original image

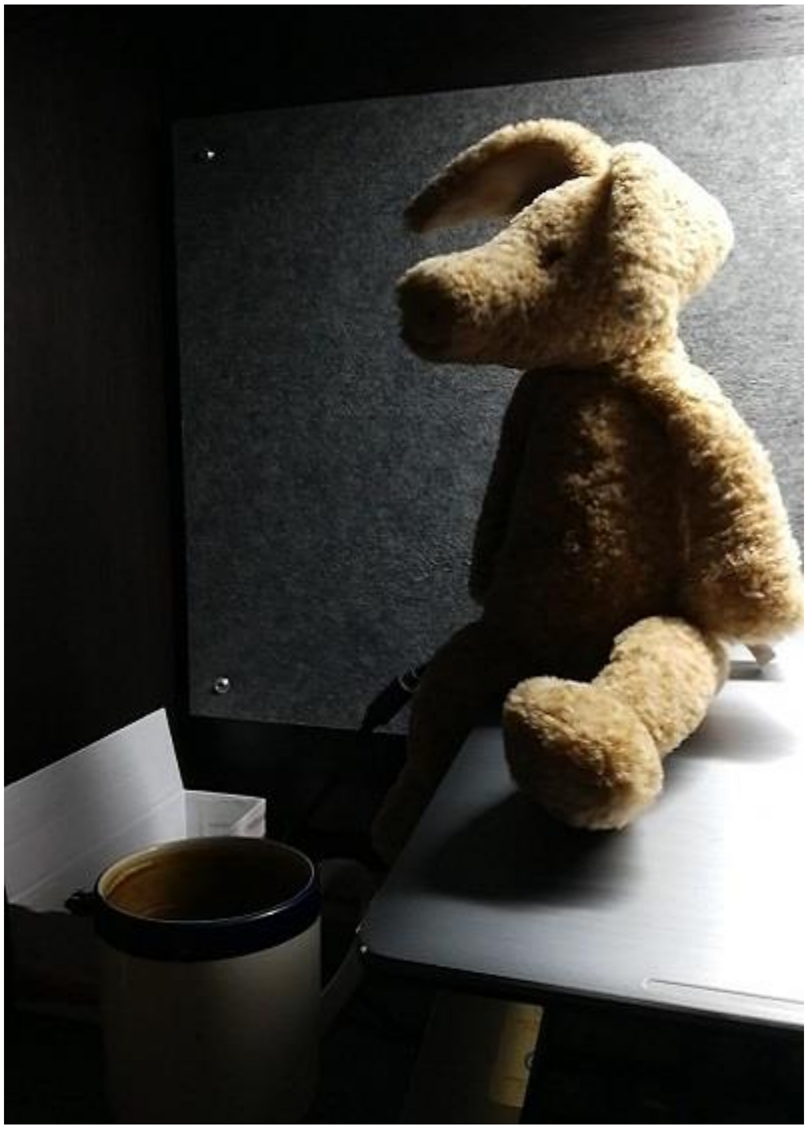
Figure. 1.7 The original “Toy” image

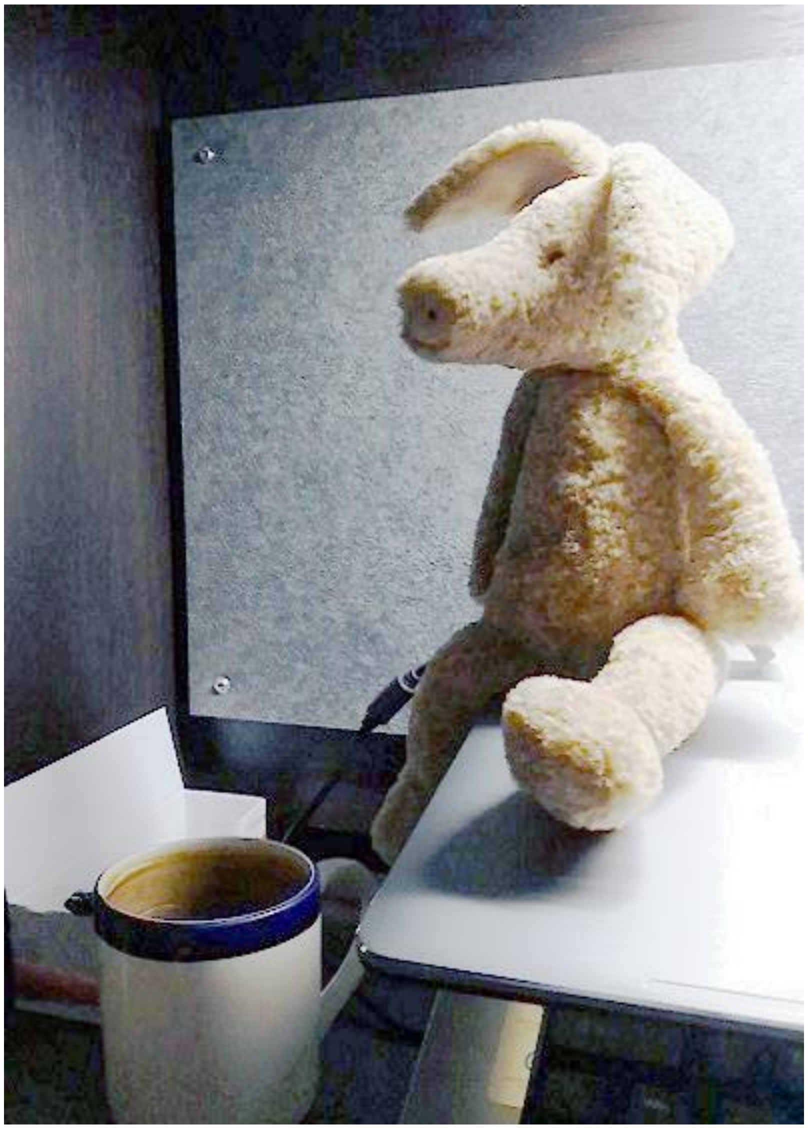
Figure. 1.8 The enhanced image by the Method A

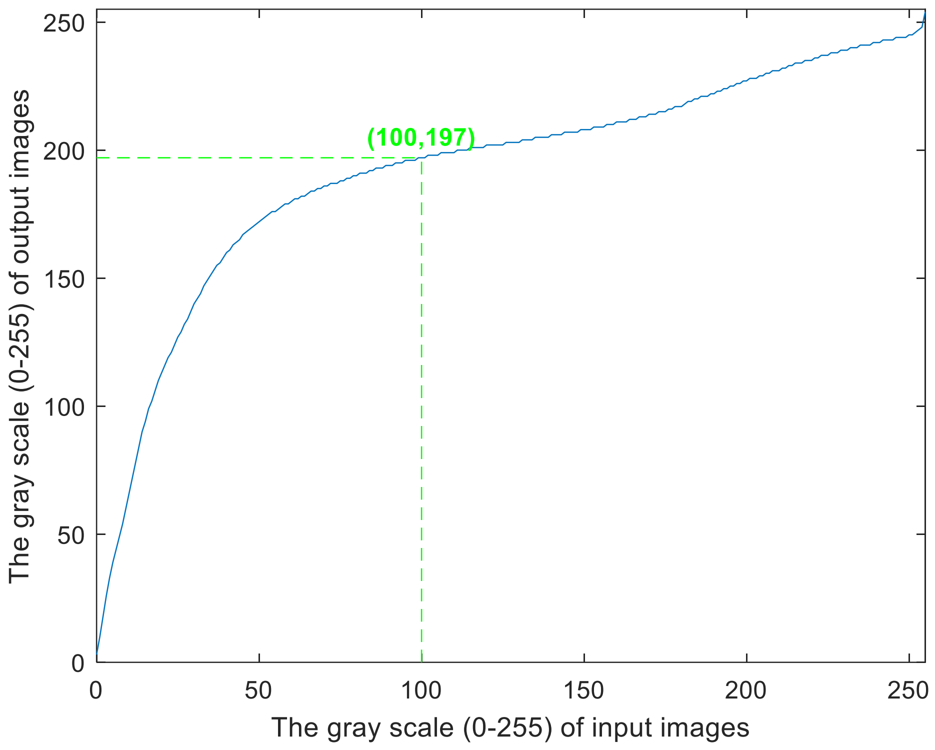

Figure. 1.9 The transfer function for the R channel from A

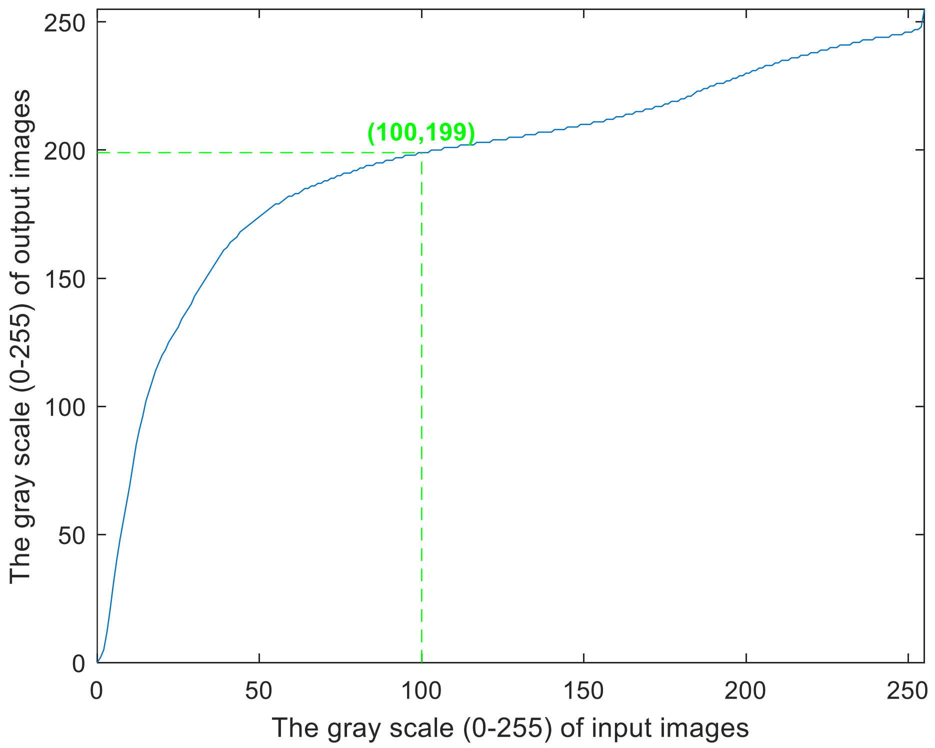

Figure. 1.10 The transfer function for the G channel from A

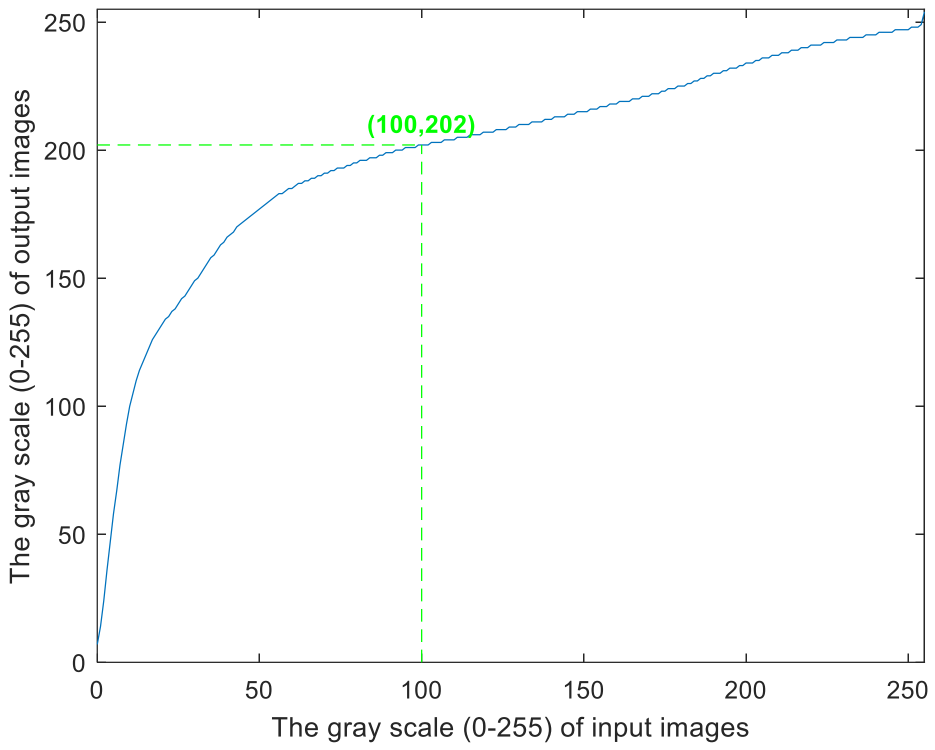

Figure. 1.11 The transfer function for the B channel from A

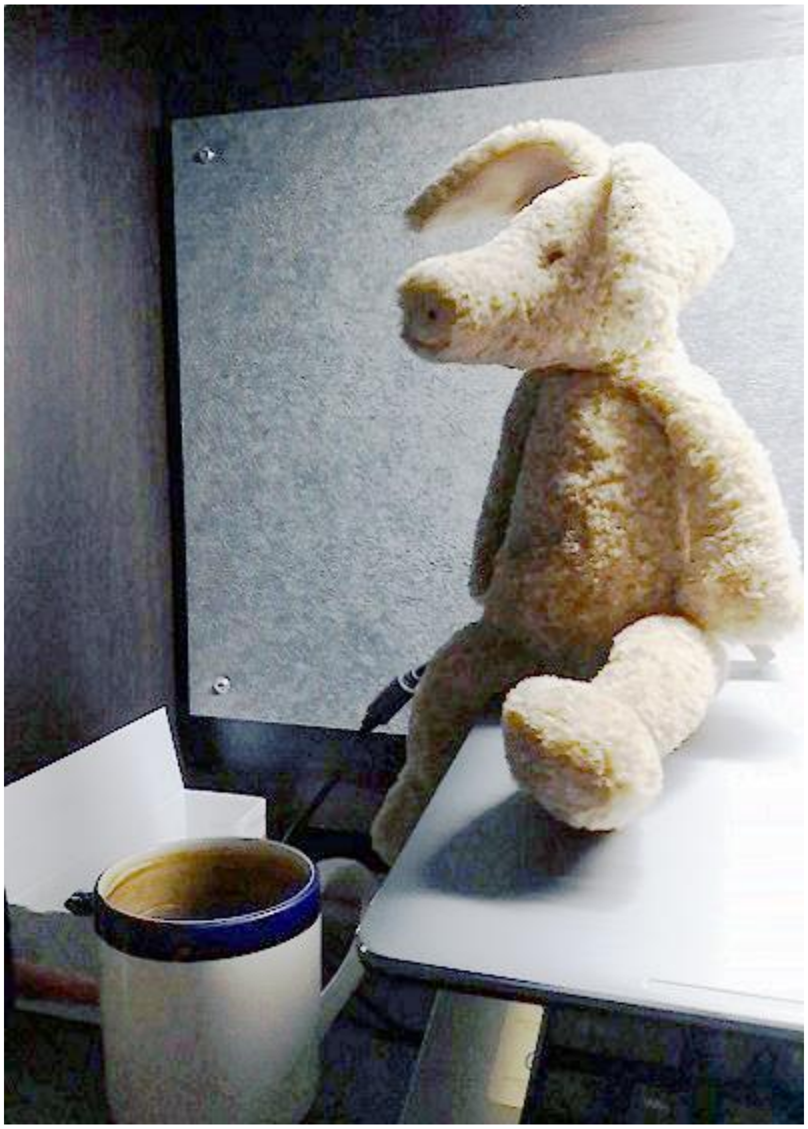
Figure. 1.12 The enhanced image by the Method B

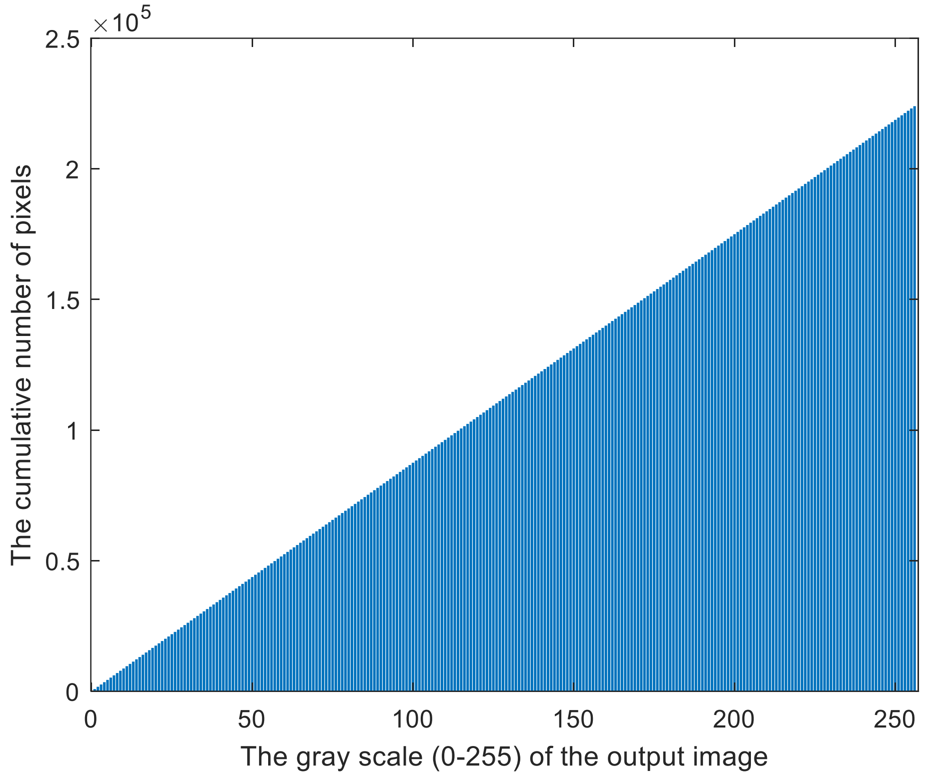

Figure. 1.13 The cumulative histogram for the R channel from B

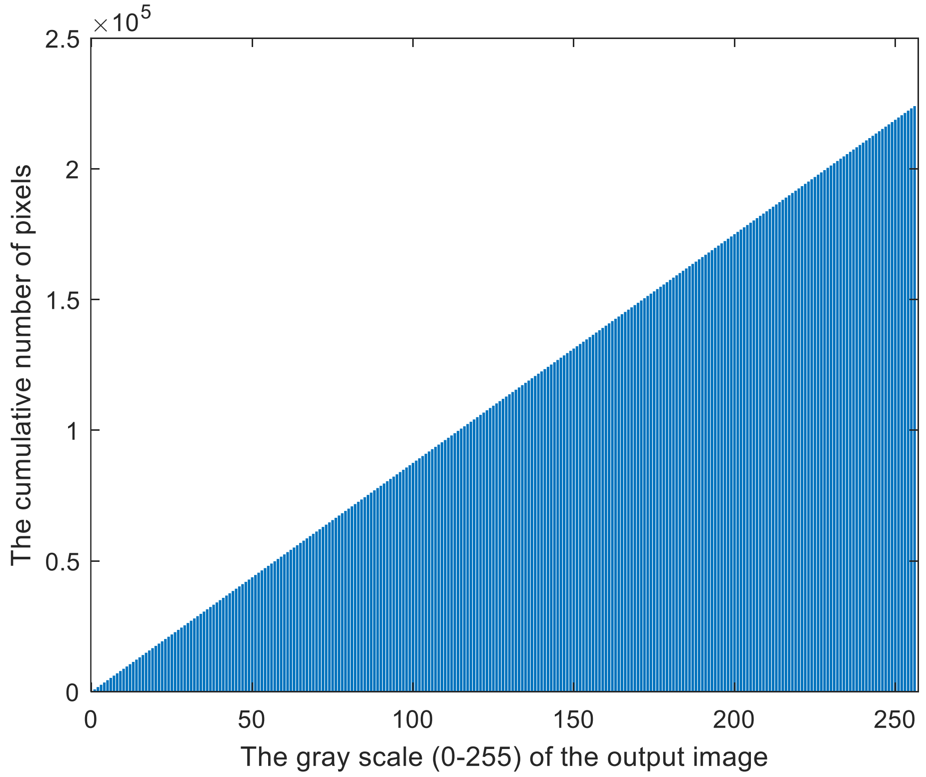

Figure. 1.14 The cumulative histogram for the G channel from B

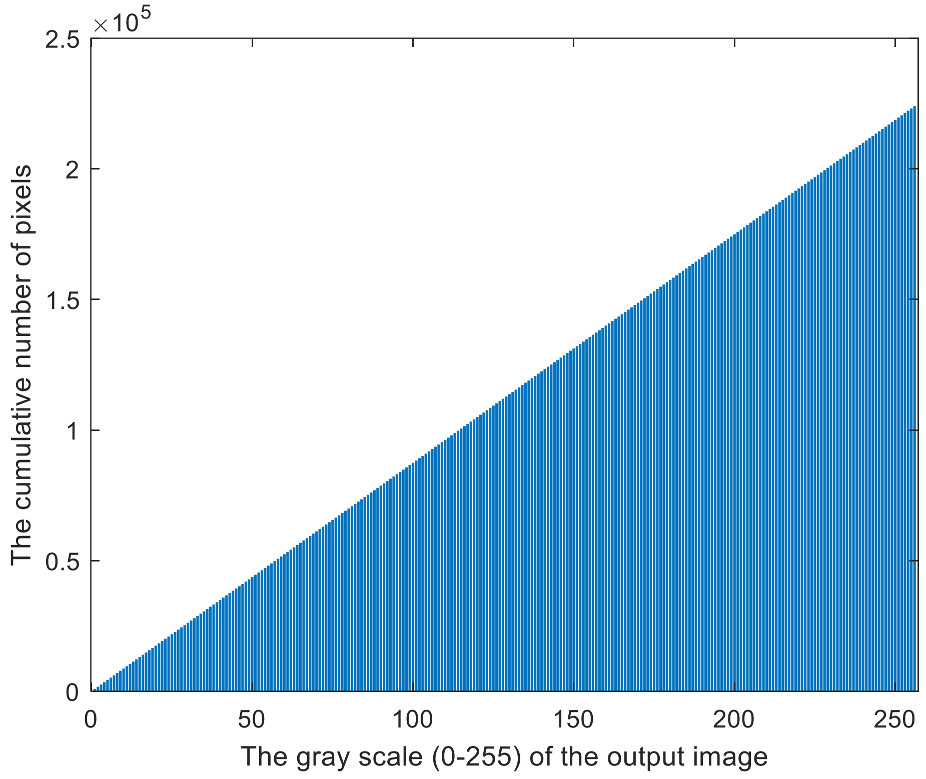

Figure. 1.15 The cumulative histogram for the B channel from B

1.4 Discussion

How to evaluate images' visual quality is always a huge challenge to people, even to image scholars themselves, because the process of that evaluation is so subjective that it's almost impossible for individuals to reach a consensus about whether certain images' visual quality is good enough. From the paper [1], the availability of mean opinion scores (MOS) allows the use of the designed database as a fundamental tool for assessing the effectiveness of visual quality, which has been collected by performing 985 subjective experiments with volunteers from 5 countries. It's not unapparent that in order to evaluate an image's quality objectively, we have to ask a number of people's opinions.

However, in this report, I will merely use my own judgement to evaluate the visual quality of resulting images. My discussion of experiment images of Problem 1 is stated as below.

First of all, compared with the ground truth color image Fig.1.1, my resulting image shown in Fig.1.2 is seemingly good at first glance. However, strictly speaking, there is a little difference about the hue between two images and after I zoom my resulting image in I will notice some artifacts, meaning that the bilinear demosaicing algorithm isn't robust enough. As far as I know, bilinear interpolation is performed using linear interpolation first in one direction, and then again in the other direction. So, the "linear" is a problem. While doing the bilinear implementation, we assume that the data in two directions can be linearized, which is just a thought of approximation that might cause some distortion. Moreover, my bilinear algorithm only considers most adjacent pixel values that will lose information from other pixels, which will also distort the real image. Therefore, in order to improve the demosaicing performance, I have to think about if the curves are a little bit convex or concave rather than purely linear. What's more, I am supposed to consider more pixels' influence in order to calculate one pixel's two other channels' intensity values.

What's more, compared with the resulting image shown in Fig.1.2 by using the bilinear demosaicing method, my resulting image shown in Fig.1.3 by using MHC method is seemingly the same at first glance. Frankly speaking, there is no clear difference between two resulting images after looking at them more carefully. Perhaps my MHC algorithm merely considers one more adjacent neighborhood pixels, which might still make real images lose some information. Or perhaps the image in my assignment doesn't contain lots of details and the MHC algorithm can be superior to the bilinear method when operating other images. According to the paper [2], MHC is visually sharper than bilinear, but not as sharp as Hamilton-Adams due to that Hamilton-Adams uses a nonlinear interpolation strategy. Hence there is a possibility that if I try the nonlinear interpolation strategy I can enhance the performance of image demosaicing.

Finally, after looking through the histograms of the three channels of the original "Toy" image shown in Fig.1.4, Fig.1.5, and Fig.1.6, I notice that the majority of pixels have lower intensity values, thus resulting in a apparently dark image. After I apply the method A (the transfer-function-based histogram equalization method), the resulting image is shown in Fig.1.8, which, in my opinion, improves the image's visual quality. From the plots of transfer functions for three channels shown in Fig.1.9, Fig.1.10, and Fig.1.11, it's not hard for us to notice that I have transformed pixels having lower gray scales into pixels having higher gray scales successfully, which can help increase the image's brightness. Moreover, after I apply the method B (the cumulative-probability-based histogram equalization method), the resulting image is shown in Fig.1.12, which, in my opinion, also improves the image's visual quality. From the plots of cumulative histograms for three channels shown in Fig.1.13, Fig.1.14, and Fig.1.15, it's not hard for us to notice that I have successfully assigned pixels in the original image to 256 (gray scale 0-255) buckets evenly, which can also help increase the image's brightness.

However, strictly speaking, there are some little patches where color is distorted in two resulting enhanced images. According to the TA's hint, I can use built-in function to convert the original RGB model into an HSI model. Then I can utilize the histogram equalization techniques to operate the intensity component of the HIS model, making images' details enhanced with more correct colors.

In a word, from my point of view, the resulting images of mine are desirable enough for people to view. But there are still some disadvantages that I ought to pay attention to, meaning that I should look for more advanced strategies in order to improve the visual quality of images.

Problem 2: Image Denoising (50%)

(a) Basic denoising methods (10%)

(b) Bilateral Filtering (10%)

(c) Non-Local Means (NLM) Filtering (10%)

(d) block matching and 3-D (BM3D) transform filter (10%)

(e) Mixed noise in color image (10%)

2.1. Abstract and Motivation

In the academic field of digital image processing, it’s inevitable that images will be introduced with disturbing noises and artifacts due to faulty acquisition techniques. For people conducting research on image processing, how to handle different types of noise is an unescapable challenge, meaning that individuals are supposed to reserve the details of images and removing random noises by all means because noisy image produces undesirable visual quality and lowers the visibility of low contrast objects. Hence, people ought to attach great significance to image denoising in order to enhance and recover fine details that are hidden in raw data.

Therefore, in this report, I am about to use C++ to implement two image denoising algorithms: one is basic denoising methods, the other is bilateral filtering. What’s more, I will use the open source codes in Matlab to realize Non-Local Means (NLM) Filtering and Block matching and 3-D (BM3D) transform filtering. After acquiring raw data files from Visual Studio 2019, I will use Matlab 2019b to show images and apply different parameters to analyze denoising effects. I will use the PSNR (peak-signal-to-noise-ratio) quality metric to evaluate the performance of my algorithms.

2.2. Approach and Procedures

(a) Basic denoising methods

In this part, I will use C++ to write a correlation algorithm to make the noisy image correlate a known uniform filter kernel and a known Gaussian filter kernel.

The uniform filter kernel is

$$\frac{1}{25}\times\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix},$$

and the Gaussian filter A’s kernel is

$$\frac{1}{273}\times\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 7 & 1\end{bmatrix}.$$

Next, in order to adjust parameters, denoted as the Gaussian filter B, I will implement the algorithm to realize the following Gaussian formulas:

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

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

where σ is the standard deviation of Gaussian distribution.

(b) Bilateral Filtering

Different from traditional uniform and Gaussian filters, bilateral filters replace the intensity of each pixel with a weighted average of intensity values from its nearby pixels, which can help preserve sharp edges better. People usually rely on the Gaussian distribution to estimate those weight coefficients. Crucially, the weights not only depend on Euclidean distance of pixels, but also on their radiometric differences.

In Visual Studio 2019, I will implement the algorithm to realize the following discrete bilateral filter:

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

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

where (k,l) is the neighboring pixel location within the window centered around (i,j), I is the image with noise, Y is the filtered image. σ_c and σ_s are the spread parameters.

(c) Non-Local Means (NLM) Filtering

From [3], the NLM method is based on a simple principle: replacing the intensity value of a pixel with an average of the intensity values of similar pixels. Due to a fact that the most similar pixels to a given pixel have no reason to be close at all, it's reasonable to scan a vast portion of the image in search of all the pixels really resembling the pixel people want to denoise.

In my report, I will use the online Matlab code from [4] to implement the Non-Local Means (NLM) filter for image denoising.

(d) block matching and 3-D (BM3D) transform filter

After reading the paper [5], I use my own words to explain the BM3D algorithm.

This algorithm is based on effective filtering in 3D transform domain by combining sliding-window transform processing with block-matching.

Step1: Given a single noisy image, process image blocks in a sliding manner and search for blocks exhibiting similarity to the currently processed one: by applying a 3D decorrelating unitary transform which produces a sparse representation of the true signal in 3D transform domain, induce high correlation along the dimension of the array to make matched blocks stacked together to form a 3D array.

Step2: Apply a shrinkage operator (e.g. hard thresholding or Wiener filtering) on the transform coefficients to attenuate noise efficiently.

Step3: Apply an inverse 3D transform of the filtered coefficients to reconstruct matched blocks, which have better denoising performance and effective detail preservation in the local estimates.
Step4: Get the final estimate by weighting average of all overlapping local block-estimates after processing all blocks.

In my report, I will use the online Matlab code from [6] to implement the block matching and 3-D (BM3D) transform filter for image denoising.

(e) Mixed noise in color image

Salt-and-pepper noise, also known as impulse noise, can be caused by sharp and sudden disturbances during the course of image processing. It presents itself as sparsely occurring white and black pixels, where white and black mean that the intensity value is 255 and 0 respectively. After knowing this fundamental knowledge, I can answer the question later.

2.3. Experimental Results

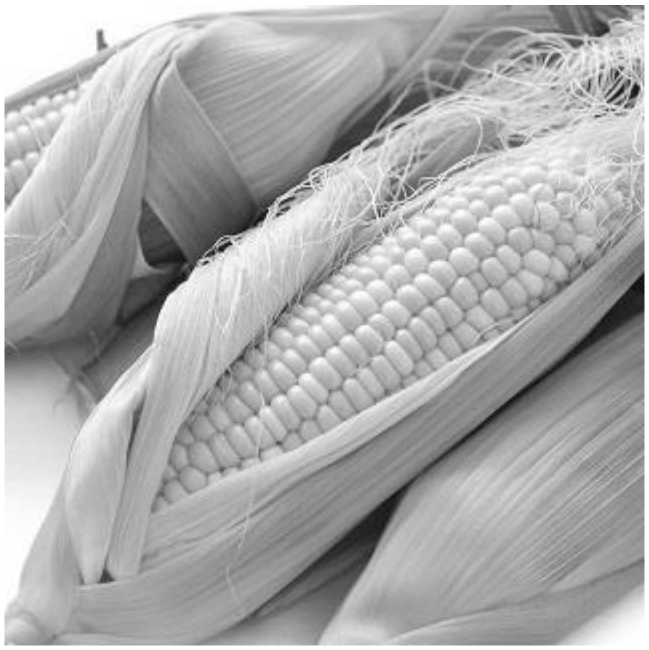

Figure. 2.1 The original “Corn” image

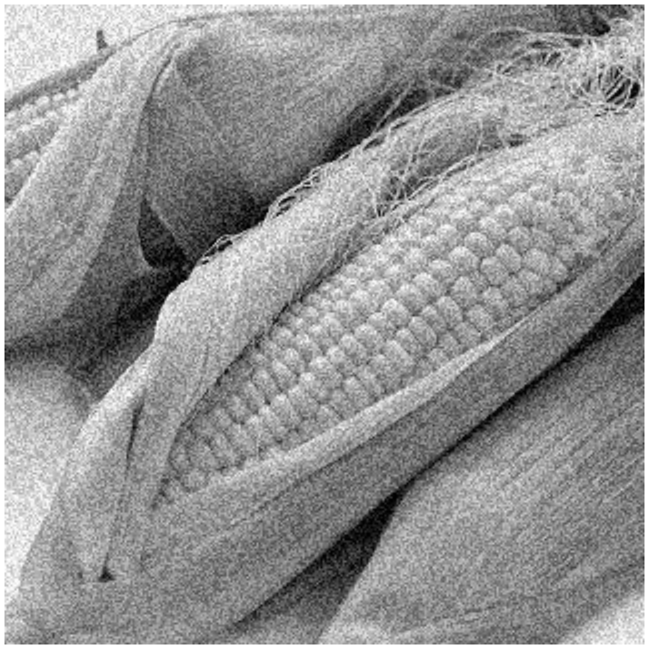
Figure. 2.2 The noisy “Corn” image
and its PSNR is 17.7062

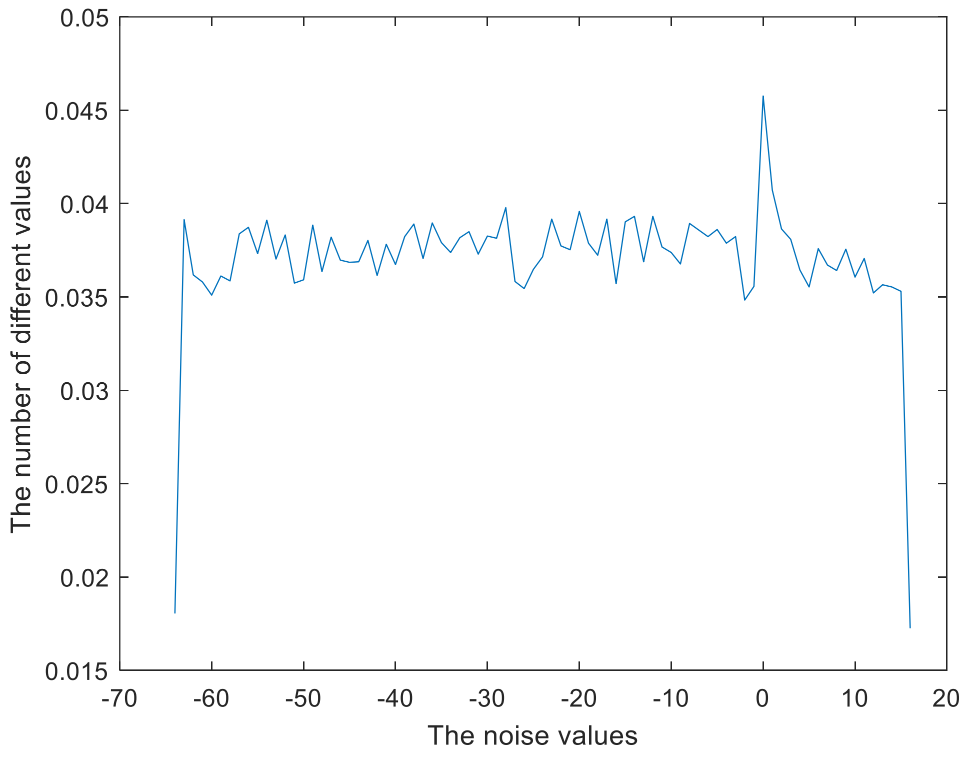

Figure. 2.3 The noise’s estimated probability density function

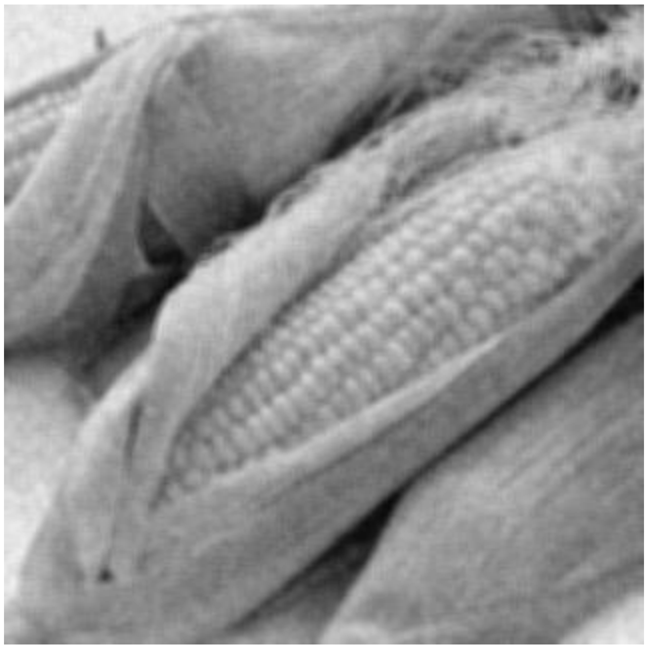
Figure. 2.4 The resulting image by using the uniform filter and its PSNR is 19.7778

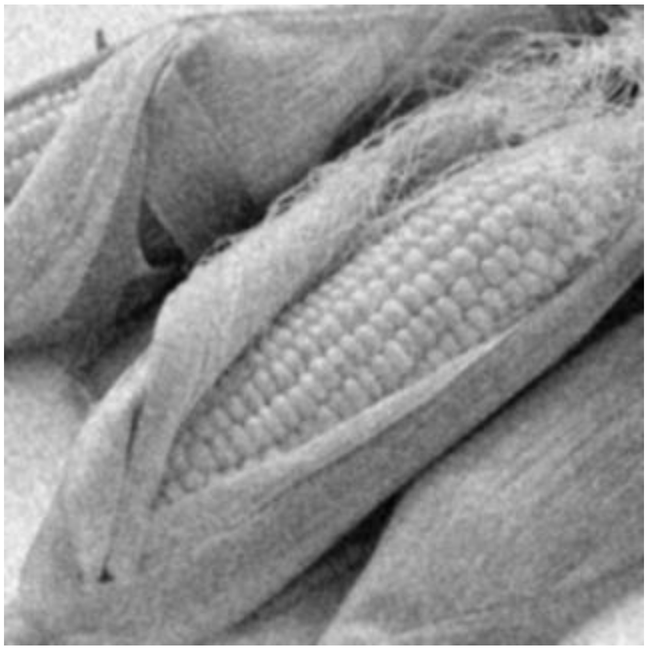
Figure. 2.5 The resulting image by using the Gaussian filter A and its PSNR is 21.0143

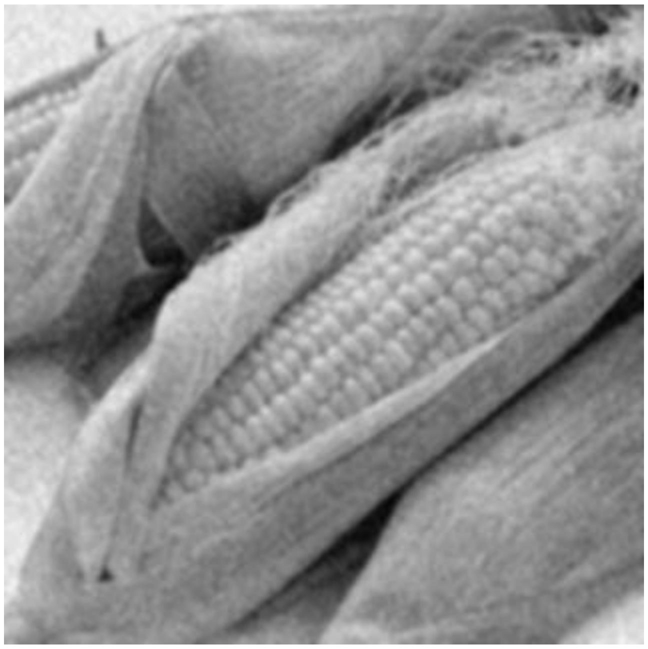

Figure. 2.6 The resulting image by using the Gaussian filter B when σ is 1.5 and its PSNR is 20.3244

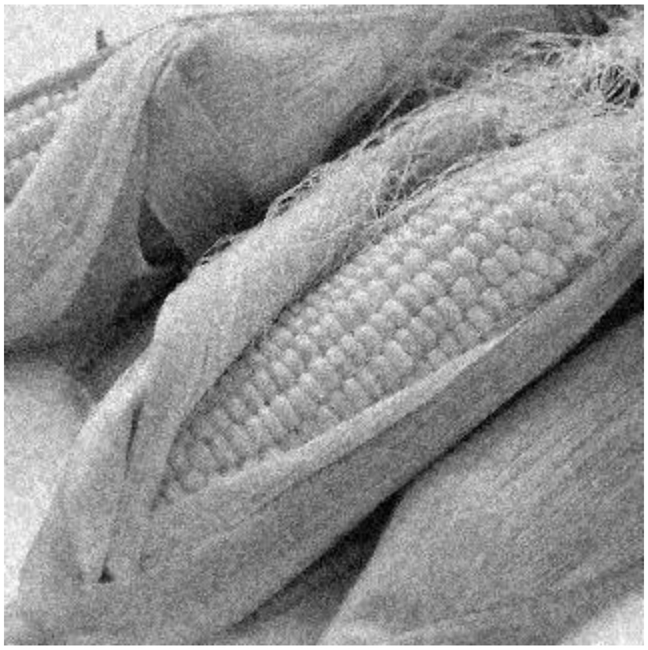

Figure. 2.7 The resulting image by using bilateral filtering when σ_c is 1.5 and σ_s is 3 and its PSNR is 25.2675

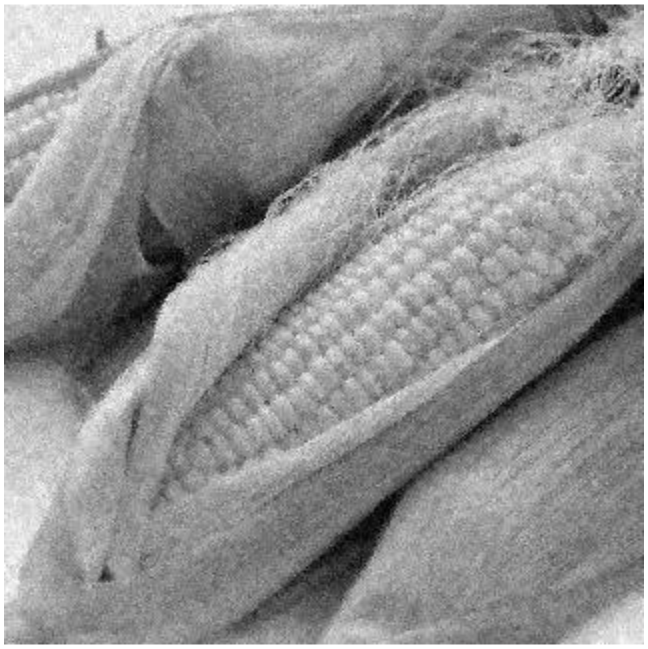
Figure. 2.8 The resulting image by using bilateral filtering when σ_c is 10 and σ_s is 3 and its PSNR is 22.4852

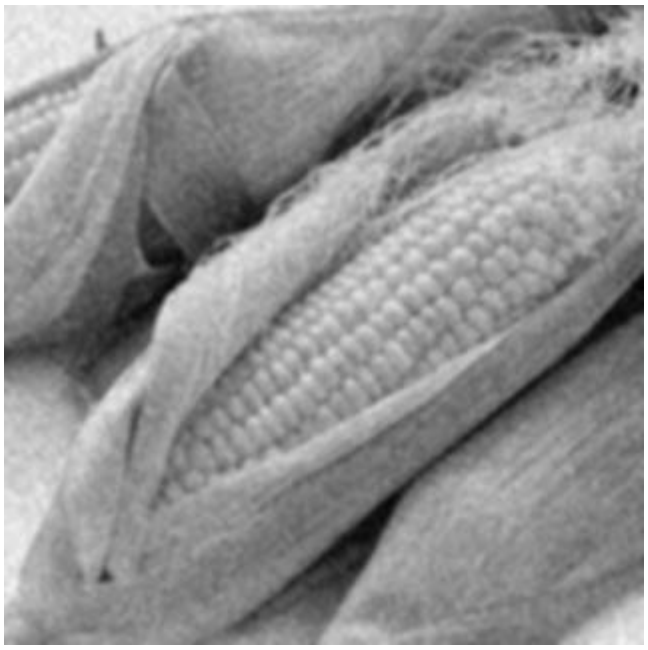
Figure. 2.9 The resulting image by using bilateral filtering when σ_c is 1.5 and σ_s is 300 and its PSNR is 20.3315

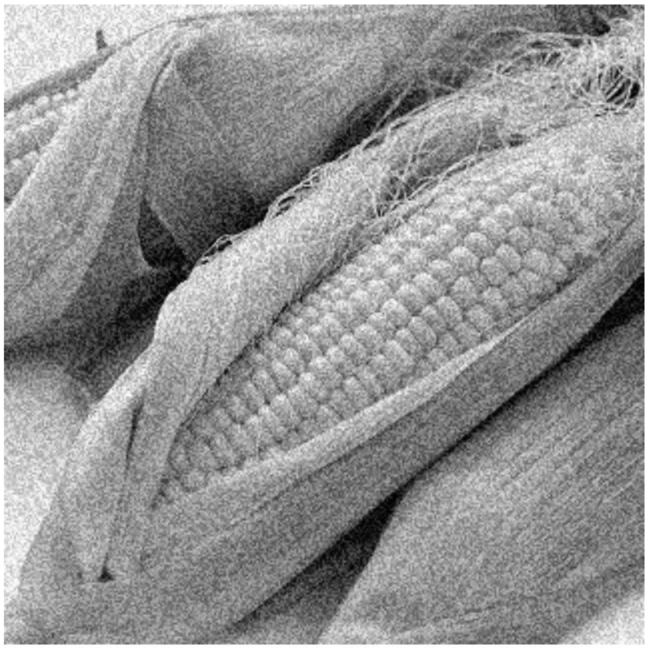

Figure. 2.10 The resulting image by using NLM filtering
when the search window is 7 by 7, the similarity window is 5 by 5, σ is 2, and the h is 10
and its PSNR is 58.5054

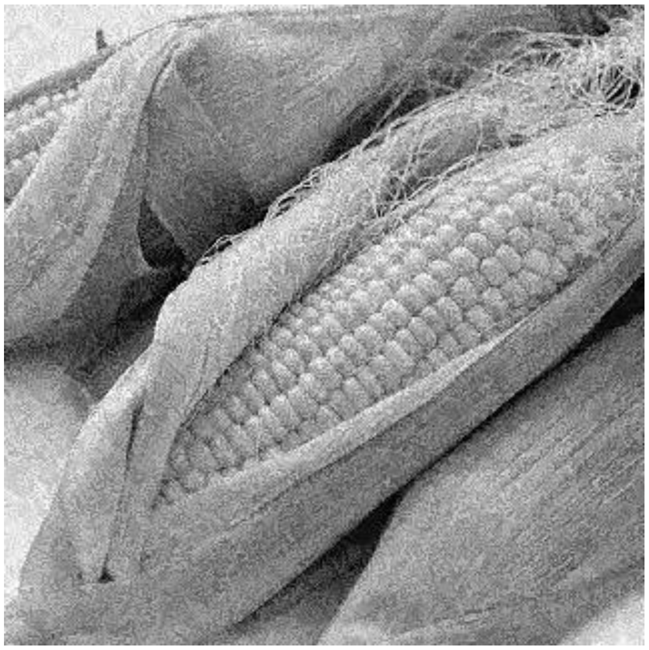

Figure. 2.11 The resulting image by using NLM filtering
when the search window is 7 by 7, the similarity window is 5 by 5, σ is 2, and the h is 15
and its PSNR is 33.8184

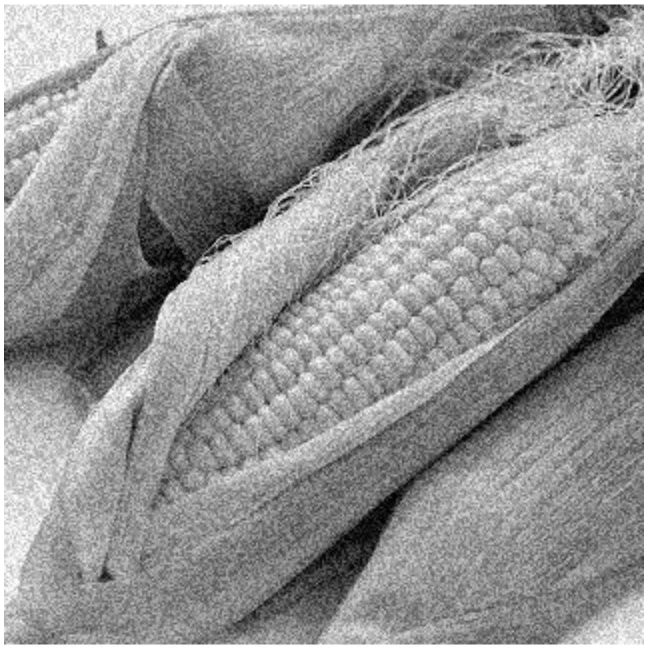

Figure. 2.12 The resulting image by using NLM filtering
when the search window is 7 by 7, the similarity window is 5 by 5, σ is 4, and the h is 10
and its PSNR is 58.1523

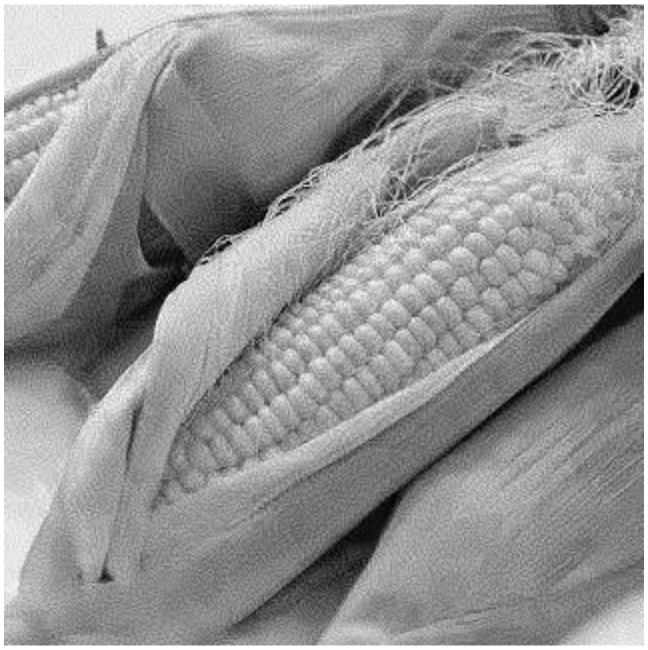

Figure. 2.13 The resulting image by using BM3D transform filtering
and its PSNR is 76.0307

2.4. Discussion

As stated before, in this report, I will merely use my own judgement to evaluate the visual quality of resulting images. My discussion of experiment images of Problem 2 is stated as below.

In part (a), according to the Fig.2.3, I can assume that the noise embedded in Fig.2.2 belongs to the uniform type. According to the formula, the bigger the PSNR is, the better the result of denoising is. The PSNR of the noisy image in Fig.2.2 is 17.7062. The resulting image filtered by the 5 by 5 uniform filter is shown in Fig.2.4 and its PSNR is 19.7778. The resulting image filtered by the 5 by 5 known Gaussian filter A is shown in Fig.2.5 and its PSNR is 21.0143. The resulting image filtered by the 5 by 5 unknown Gaussian filter B by letting $\sigma = 1.5$ is shown in Fig.2.6 and its PSNR is 20.3244. In my view, although noises can, to some extent, be removed by two kinds of filters. However, a great many fine edge details fail to be preserved due to the result of smoothing compared with the original image shown in Fig.2.1. By checking the values of PSNR, I can safely draw a conclusion that the Gaussian filter is better than the uniform filter when they have the same kernel size and both types of filters can help improve the visual quality of noisy images.

In part (b), σ_c and σ_s are smoothing parameters. As the range parameter σ_s increases, the bilateral filter gradually approaches Gaussian filters more closely, meaning that this parameter can adjust weighting coefficients of the influence of neighborhood pixels to one certain pixel. As the spatial parameter σ_c increases, the more features of images will get smoothed just like the Gaussian filter's sigma. Compare Fig.2.7 with the figure.2.8, once I increase the value of σ_c , more edge details of the image won't be preserved so that the PSNR in Fig.2.7 is higher than it in Fig.2.8. Compare Fig.2.6 with the figure.2.9, I increase the value of σ_s largely and the result of filtering is very similar to each other because the PSNR is almost the same. Hence when σ_s is very large, we can regard the bilateral filter as a Gaussian filter approximately. Also, we can see that the PSNR in Fig.2.7 is higher than those in Fig.2.8 and Fig.2.9, meaning that bilateral filtering does improve the performance of image denoising if we choose parameters appropriately.

In part (c), I use the Matlab code from [4] to implement the Non-Local Means (NLM) filtering. As shown in Fig.2.10, when the search window is 7 by 7, the similarity window is 5 by 5, σ is 2, and the h is 10, its PSNR will reach 58.5054. When I let the parameter h larger, the PSNR is smaller as shown in Fig.2.11, which is 33.8184. When I let the parameter σ larger, the PSNR is almost the same as shown in Fig.2.12, which is 58.1523. Hence, the parameter h plays a more important role in increasing the value of PSNR than the parameter σ. According to other individuals' experience, my choices of sizes of the search window and similarity window are appropriate enough.

In part (d), I have already explained the BM3D algorithm in part (d) of section 2.2. Now I use the Matlab code from [6] to implement the block matching and 3-D (BM3D) transform filtering. As shown in Fig.2.13, the PSNR, which is 76.0307, is tremendously bigger than PSNRs of other filtering results. The final image is much better from my view because I can see lots of fine details and texture of the corn in Fig.2.13.

In part (e), in my opinion, those noises must include the salt and black pepper types. Perhaps there are some uniform or Gaussian noises contained in the image. When removing the mixed noises in color images, I think we should perform filtering on individual channels separately for salt and pepper noises while we should convert the RGB model into the YUV model to handle other types of noise because transmission errors or compression artifacts can be usually more efficiently masked by the human perception when using the YUV model. As for the cascade order, firstly, I will use a median filter, which is generally less sensitive to outliers than traditional mean or Gaussian filters, to remove salt and black pepper noises. Then I will use BM3D transform filter to remove the remaining noises. We are bound to follow this order because the impulse noise are typical outliers, which must be removed first! If not, those annoying outliers will do some damage to our images when using other sorts of smoothing filters.

References

[1] Ponomarenko, Nikolay et al. “Image Database TID2013: Peculiarities, Results and Perspectives.” Signal Processing: Image Communication 30.C (2015): 57–77. Web.

[2] Pascal Getreuer. “Malvar-He-Cutler Linear Image Demosaicking.” Image Processing On Line 1 (2011): 83–89. Web.

[3] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. “Non-Local Means Denoising.” Image Processing On Line 1 (2011): 208–212. Web.

[4] Christian Desrosiers (2020). Simple Non Local Means (NLM) Filter (https://www.mathworks.com/matlabcentral/fileexchange/52018-simple-non-local-means-nlm-filter), MATLAB Central File Exchange. Retrieved January 27, 2020.

[5] Dabov, Kostadin, et al. "Image denoising with block-matching and 3D filtering." *Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning*. Vol. 6064. International Society for Optics and Photonics, 2006.

[6] [Online] Available: http://www.cs.tut.fi/~foi/GCF-BM3D/