EE569: HOMEWORK REPORT #1

Name: Boyang Xiao   USC ID: 3326-7302-74   Email: boyangxi@usc.edu

I. Problem 1: Image Demosaicing and Histogram Manipulation

1.1 Part a. Bilinear Demosaicing

1.1.1 Abstract and Motivation

As photographs and picture-capturing-techniques are more and more popular among nowadays communities, digital cameras and many other filming equipment have experienced a period of rapid development. When taking a picture with its optical lens system and CMOS component, a digital camera will most probably get a RAW picture with Bayer Pattern, which means that each pixel only records the intensity of one primary color, and that the pixels of red, green and blue are arranged alternately. However, in order to obtain a color picture that is easier for us to appreciate, the intensity of three primary colors (or channels) in each pixels should be obtained from the RAW picture. This processing technique is called Demosaicing.

1.1.2 Approach and Procedures

There are several fancy Demosaicing methods, such as Malvar-He-Culter Demosaicing, that can generate perfect results. In this project, the simple-but-useful Bilinear Interpolation Demosaicing will be introduced and implemented.

To obtain the intensity of a certain color in a certain pixel, Bilinear interpolation uses the intensity from the nearest neighboring pixels of the same color and conduct the linear averaging of these pixels. For example, to obtain the intensity of Green color in pixel *B2,3* from the Bayer pattern shown in Figure 1., we uses the nearest neighboring pixels of Green, that is, *G1,3, G2,2, G2,4* and *G3,3*. And then we do the linear averaging:

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

All the Green pixels have the same arrangement patterns, so we can do the same calculation to all the missing Green pixels. However, the Red and Blue pixels can have different arrangement patterns. For some of pixels, such as the Red intensity in pixel B2,3, we uses the Red intensity from the for neighboring corners, that is, R1,2, R3,2, R1,4 and R3,4:

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

For the intensity of Red channel in some of the other pixels, we uses the upper pixel and the lower pixel (or the left one and the right one) to calculate. For example, to calculate the Red intensity in pixel G2,2, we uses pixel R1,2 and R3,2 to conduct the averaging:

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

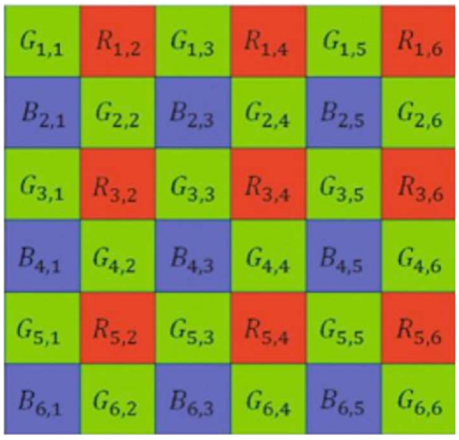

$G_{1,1}$ $R_{1,2}$ $G_{1,3}$ $R_{1,4}$ $G_{1,5}$ $R_{1,6}$
$B_{2,1}$ $G_{2,2}$ $B_{2,3}$ $G_{2,4}$ $B_{2,5}$ $G_{2,6}$
$G_{3,1}$ $R_{3,2}$ $G_{3,3}$ $R_{3,4}$ $G_{3,5}$ $R_{3,6}$
$B_{4,1}$ $G_{4,2}$ $B_{4,3}$ $G_{4,4}$ $B_{4,5}$ $G_{4,6}$
$G_{5,1}$ $R_{5,2}$ $G_{5,3}$ $R_{5,4}$ $G_{5,5}$ $R_{5,6}$
$B_{6,1}$ $G_{6,2}$ $B_{6,3}$ $G_{6,4}$ $B_{6,5}$ $G_{6,6}$

Figure 1. Bayer Pattern

And the Blue channels calculations are quite similar to those of Red channels. After conducting the Bilinear Interpolation described above, all the pixels will have three channels of intensity value, which stand for the three primary colors of Red, Green and Blue. And we can view a colored picture on the computer.

1.1.3 Experimental Results

The bilinear interpolation result is shown below in Figure 2. The gray scale image on the left is the original RAW image and the color image on the right is the generated image after Bilinear Interpolation processing from the original RAW image.

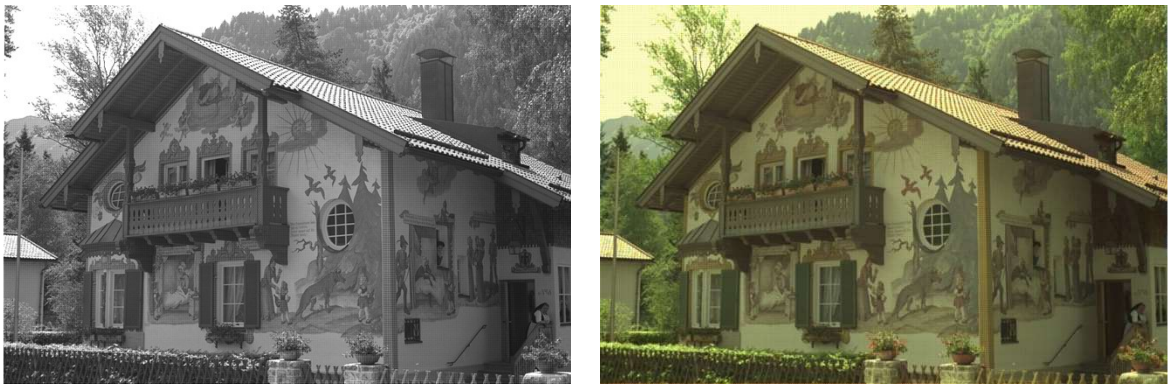


Figure 2. The original RAW image (left) and the colored image after Bilinear Interpolation (Right)

1.1.4 Discussion

Even though more than half of the intensity of colors are estimated from the existing intensity values, the overall color of the generated colored picture after Bilinear Interpolation accords a lot to the real-world color as we can see from our eyes.

However, some flaws can still be observed if the generated image is zoomed in. As shown in Figure 3., it is obvious that the pixels of some regions where the colors are complex are distorted and show some variegated color patterns (such as the tree leaves and the house roof). This kind of distortion happens because the pixels in these regions vary a lot from each other and the averaging results from the neighboring pixels can hardly estimate the real color in some certain pixel from these regions. Therefore, more dedicated algorithms are needed to handle this situation, such as some Demosaicing methods that has corrections to the bilinear interpolation results.

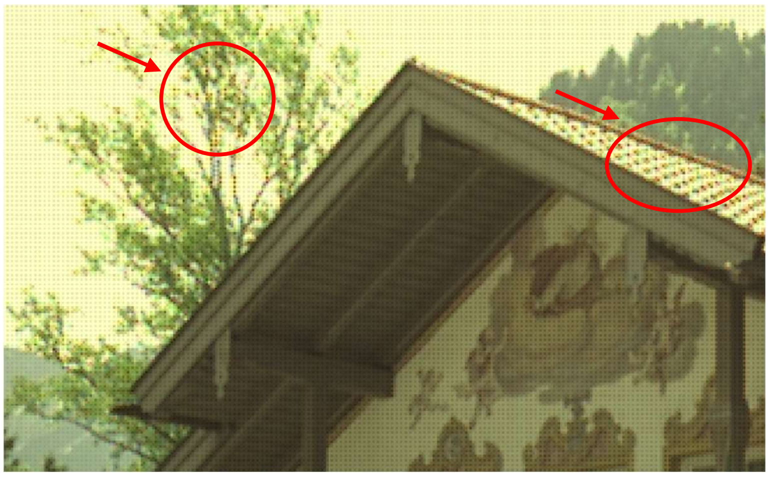


Figure 3. Zoomed details of the Demosaiced image. The red circles mark some color distortion in this image.

In addition, how to handle the image boundary can also be a tricky problem, since there is no pixel outside the boundary to estimate the pixel right on the boundary line. In my implementation, if a selected pixel coordinate exceeds the boundary of the image, the algorithm will otherwise select the pixel nearest it on the boundary. For example, to estimate the Green intensity on pixel R1,2, one of the selected pixel is G0,2, but G0,2 is obviously outside the image boundary. Therefore, the algorithm will select the intensity value in pixel R1,2 in place of G0,2. Although this is not quite a precise approach, the boundary pixels do not occupy a large proportion of the whole image, so the boundary processing effects can hardly be observed. Fancier algorithms or machine-learning-based methods can be applied if we need a better boundary of the image.

1.1.5 Other non-programming questions

Q: Compare your demosaiced image with the color image House_ori which is obtained from a more advanced demosaicing algorithm. Do you observe any artifacts? If yes, explain the cause of the artifacts and provide your ideas to improve the demosaicing performance.

A: The comparation of two images is shown in Figure 4. It is obvious that the House_ori.raw image is brighter in color and comforts more to the real-world color, while the image generated by Bilinear Interpolation has more greenish color, which makes it less real or less natural. The reason that causes this can be that the green pixels are twice as many as red pixels and blue pixels. Therefore, the green pixels are estimated more precisely while the red and blue ones are not that much. Some estimated pixels of red and blue might have less intensity than they really have, so the overall image can be more "greenish".

To improve the algorithm, some correction can be added to the bilinear interpolation terms. Just as MHC algorithm does, it adds a discrete Laplacian term from several neighboring pixels that have the same color as the estimated pixel position does. (For example, if we want to estimate the Green color intensity of a Red pixel, we can add a correction term from the neighboring red pixels to the bilinear interpolation term to correct the result.) Also, we can apply different weight values to the correction terms in different channels. If we think the Green is "over-corrected", we can make the weight value of correction in Green channel lower. In this way, the Bilinear Interpolation can be improved and get a better result.

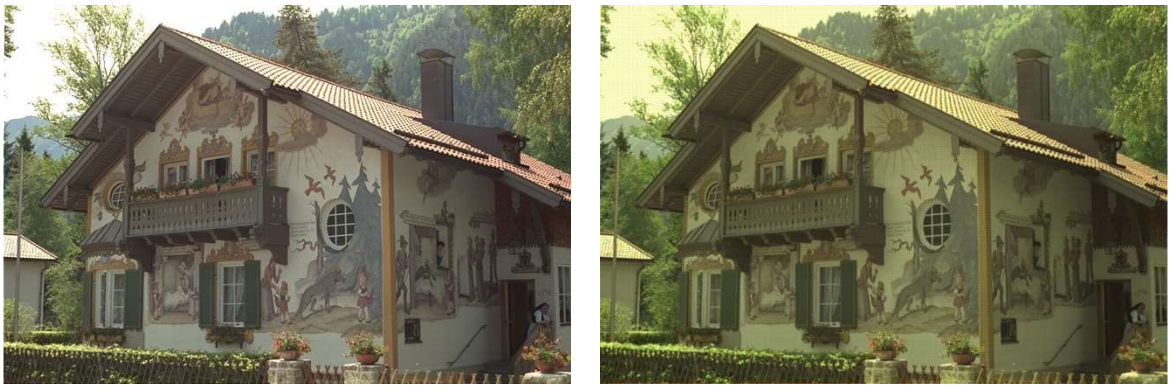


Figure 4. House_ori.raw (left) the image generated by my implementation

1.2 Part b. Histogram Manipulation

1.2.1 Abstract and Motivation

Image contrast means the difference in luminance or color that makes the contents in the image distinguishable. When a image is of low contrast, it means that the gray scale intensity values of most of the pixels are within a very narrow range, which is hard for people to observe. Otherwise, high image contrast means that the gray scale intensity values of most of the pixels are distributed uniformly in the range of 0 to 255. The contrast enhancement techniques are designed to convert images of low-contrast to images of high-contrast. In this part, some methods of contrast enhancement will be introduced and implemented.

1.2.2 Approach and Procedures

In this part, the histogram manipulation methods are used to enhance the contrast. There are two approaches: A: the transfer-function-based histogram equalization & B: the cumulative-probability-based histogram equalization.

Method A: the transfer-function-based histogram equalization method uses the cumulative probability distribution (CDF) from the original histogram and generates the mapping from the original CDF to a uniform CDF to get a manipulated histogram.

Firstly, after we acquire the histogram of the image, we can calculate the probability density distribution (PDF) of the numbers of pixels in each gray scale intensity values. And we can calculate the CDF from this PDF, adding the probability density values cumulatively one by one. Then, we can mapping the x-value of the point on this CDF function to the x-value of the point on a uniformly distributed CDF function which has the same y-value, as shown in Figure 5. Below.

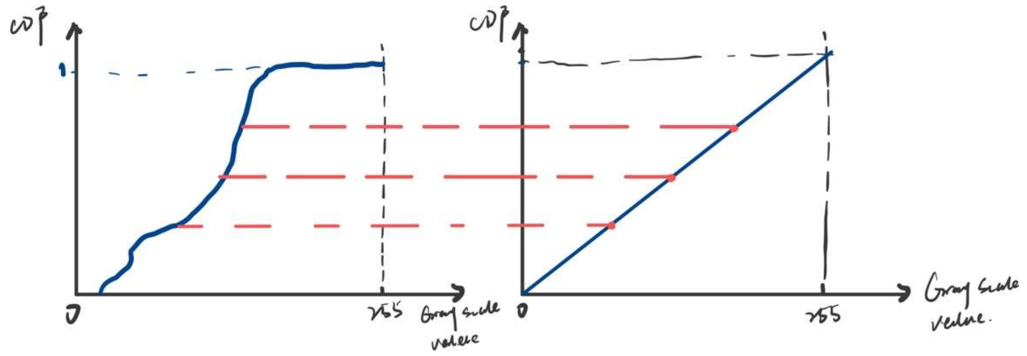


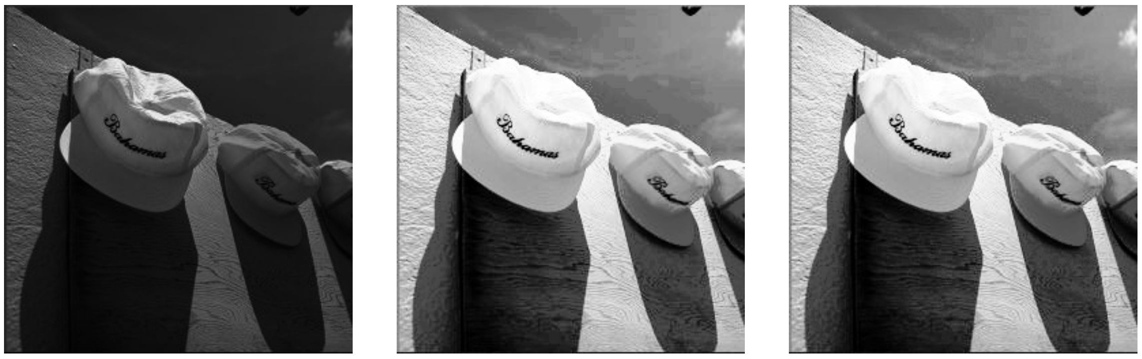
Figure 5. Mapping from a random CDF function (left) to a uniformly distributed CDF function (right).

After the mapping is generated, we can modify every pixel's gray scale value using this mapping function and acquire the histogram-manipulated image. The contrast of the new image should be enhanced.

Method B: the cumulative-probability-based histogram equalization method calculates how many pixels each gray scale intensity value (0-255) can accommodate based on the whole number of pixels in the original image. And then it allocates the pixels into each gray scale value bin in the order of their values until each gray scale value bin holds the same number of pixels. The enhanced image should have a strictly uniform-distributed histogram.

1.2.3 Experiment results

The experimental results are shown in Figure 6. Comparing to the original image (figure a) which is of low contrast, the enhanced images using method A (figure b) & method B (figure c) are way higher in contrast. Many details of the enhanced images are amplified and they are better for us to observe.

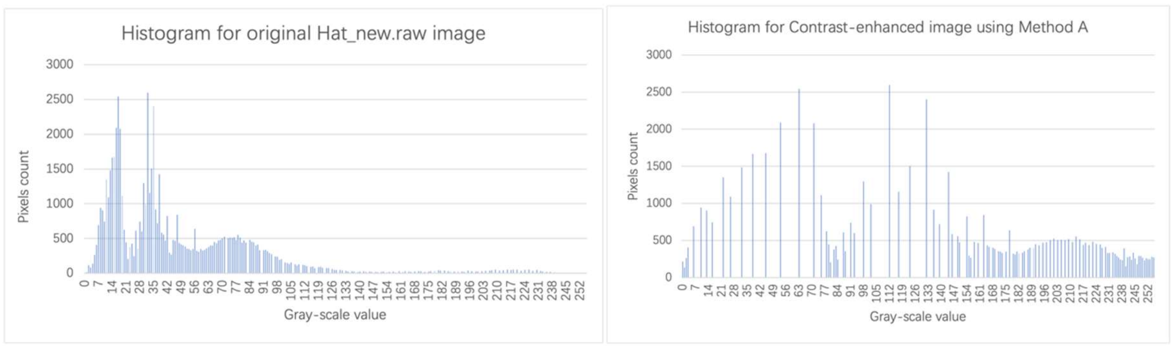


(a) (b) (c)

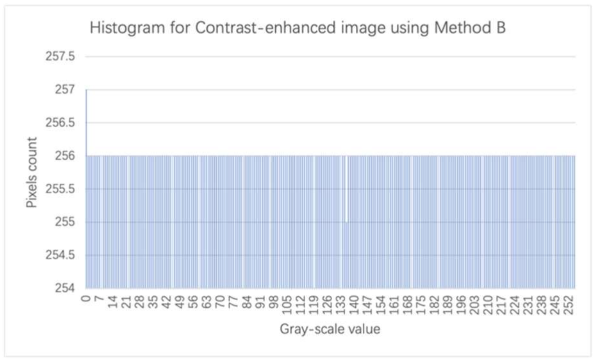
Figure 6. Histogram Manipulation results: (a) the original image, (b) enhanced using method A, (c) enhanced using method B

1.2.4 Discussions

The histograms of the original image and enhanced images are displayed below in Figure 7. The histogram of the original image has a very compactly distributed in a very narrow range, while both the histogram manipulation methods make the pixels distributed in a wider range on the whole gray scale space, from 0 to 255.



(a) (b)



(c)

Figure 7. The histograms for the original image (a) and enhanced images using method A (b) & method B (c)

However, there are still some significant differences between these two histogram manipulation method. For method A, since it only utilizes a simple mapping, converting all the pixels from one gray scale value to another gray scale value, the manipulated histogram has several gaps between each gray scale value. And the numbers of pixels in each gray scale value are not the same. But for method B, the numbers of pixels in each gray scale value are strictly the same, which makes the whole histogram seriously uniformly distributed.

The differences specified above can also be observed on the images themselves. Although the whole effects of contrast enhancement of these two methods are very similar, as shown in Figure 6. above, the details can be significantly different if the images are zoomed in, as shown in Figure 8. below. In the area that are marked by the red frames, it is obvious that the results of methd B are much better than method A. More of details of the original pictures are being preserved (e.g. more detailed textures in the sky) if using method B, while the details are much less rich if using method A. The reason for this difference is that method B makes the pixels distributed in more different gray scale bins while method A leaves huge gaps between each gray scale value bin.

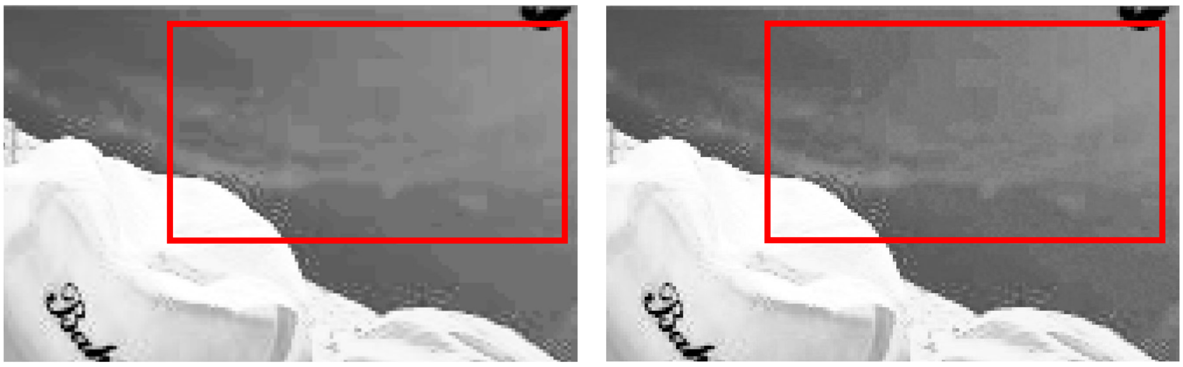


Figure 8. Zoomed pictures from the enhanced images using method A (left) and method B (right)

1.2.5 Other non-programming questions

All the questions have been answered in the contents above!

1.3 Part C. Contrast Limited Adaptive Histogram Equalization

1.3.1 Abstract and motivation

For some simple and gray-scale images, the contrast enhancement methods mentioned in part(b) can work well and can acquire some fair-good results. But for the RGB images with three channels or images with more complex textures, those two methods might cause more noises and more color distortions. In this situation, the Contrast Limited Adaptive Histogram Equalization (CLAHE) algorithm is introduced. CLAHE uses the several minor tiles and does the histogram equalization within these tiles (locally) but not within the whole image range (globally). Since the variances in pixels' gray scale values are not that much comparing to a global range, the histogram equalization results within a local range can reduce the noises and distortions significantly. Therefore, CLAHE is a more popular contrast enhancement method when processing the RGB color images.

1.3.2 Approaches and procedures

When contrast-enhancing an RGB color image, separately processing each channel of R, G and B can hardly get a good result. Since the human eyes are more sensitive to the luminance of the image, converting the image from RGB space into YUV space can be a good method to handle these color images. Hence, in this part's implementation, the color image is firstly converted into a YUV image using the RGB-to-YUV format.

Then the Y-channel pixel data is passed into the CLAHE algorithm function (Image Processing toolbox-> adapthisteq(…) in MATLAB) as the input data. The hyperparameters of "adapthisteq(…)" are chosen, including "NumTiles" (controls the size of minor tiles, default 8*8) and "ClipLimit" (controls the level of contrast enhancement, default 0.01). From my experiments, I choose "NumTiles" to be 8*8 and "ClipLimit" to be 0.0045 to optimize the results.

As a control group, the Y-channel data of the YUV image will also be processed by the two methods implemented in Part(b) and acquires the processed data.

After the all these three methods are operated, the processed YUV image is converted back to an RGB image using the YUV-to-RGB format and get the enhanced RGB image that can be read by normal image readers.

1.3.3 Experimental results

The results of CLAHE algorithms are shown in Figure 9. Compared to the original image with heavy haze (left), the CLAHE algorithm amplifies the luminance of the whole image, makes the contrast higher and removes the haze from the perspective of vision. At the same time, the CLAHE does not amplify the noises of the whole image. Therefore, the result of CLAHE accords perfectly with the expectation.

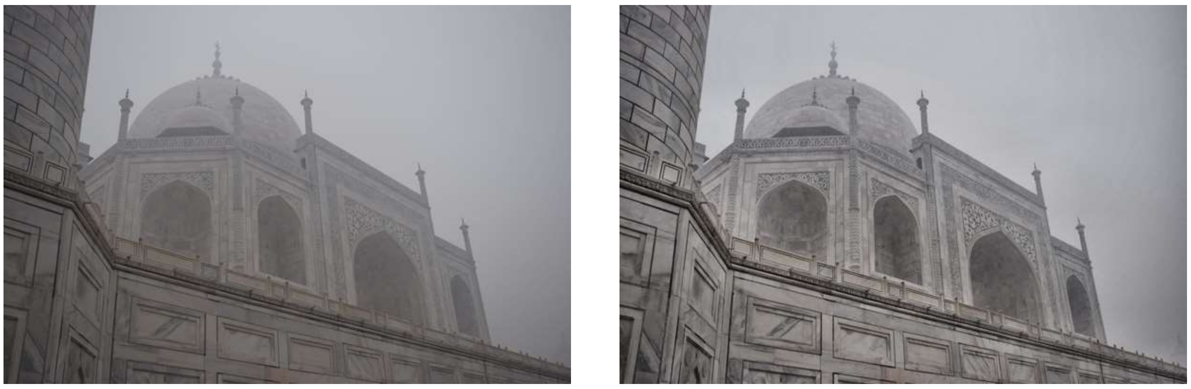


Figure 9. The original image (left) and the processed image after CLAHE processing (right)

The results of the two histogram equalization methods from Part(b) are also displayed below in Figure 10. Both of the enhanced images have color distortions to some level (e.g. the sky and the edges of the lower part of the images). Clearly, these two methods are not suitable to handle this kind of color images.

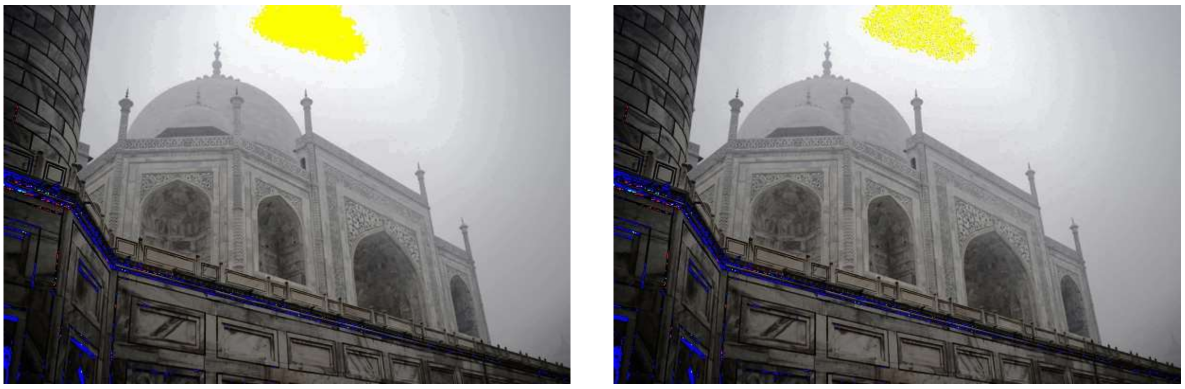


Figure 10. Contrast-enhanced images using Method A (left) and Method B (right)

1.3.4 Discussion

Despite some color distortions in the processed images in Figure 10. using those two methods from Part (b), the whole effects of haze removal are fair good. The whole contrast of the images are enhanced and the whole color of the images is preserved.

However, the color distortion is not tolerated in real practice. Compared to CLAHE, the global histogram equalization implemented in Part(b) equalizes all the pixels based on the whole distribution of pixels in the image. If the image is a gray scale image with only a single channel, the results are good. However, when the image is a RGB color image and is converted to YUV space, the global histogram equalization only processes the Y-channel which stands for the luminance of the image, so some of the pixel might be processed too much (like the dark edges) while the other are processed too less (like the sky). When the YUV image is converted back to the RGB image, the unproperly-processed pixels can appear to be some weird colors, that's how the color distortions occur.

CLAHE, otherwise, avoid this kind of problems because its own characteristics. Since CLAHE only does the histogram equalizations within a narrow range of a minor tile, and the intensity value of pixels within this narrow range can vary not too much from each other, the histogram equalization will not change one pixel's value from its own value into a very different value. Therefore, CLAHE hardly add any noises to the image and can get a good result.

1.3.5 Non-programming questions

Q1: Explain CLAHE in your own words.

A1: This question has been answered in 1.3.1.

Q2: Compare your results between (2) and (3). Discuss your observations.

A2: This question has been answered in 1.3.4.

II. Problem 2: Image Denoising

2.1 Part A: Basic Denoising Method

2.1.1 Abstract and motivation

There are multiple types of noises which can be existing in the digital images. In this part, we only handle two of them, the Additive white Gaussian noise (AWGN) and the Impulse noise (pepper salt noise). Since the noise added to the original signal in AWGN is a Gaussian distribution overall with mean-value of 0, adding up several pixels around a certain area is a effective to remove this kind of noise. The simplest methods to remove AWGN, Low pass filter of mean & Gaussian, will be introduced.

2.1.2 Approaches and procedures

A low-pass filter, also called a "blurring" or "smoothing" filter, averages out rapid changes in intensity. The simplest low-pass filter just calculates the average of a pixel and all of its neighbors within a filter window. The result replaces the original value of the pixel. The process is repeated for every pixel in the image.

The difference between Mean Low pass filter and Gaussian Low pass filter is that the weights for pixels within the filter window are different. As is shown in Figure 11., the mean low pass filter averages the neighboring pixels and the Gaussian low pass filter does the averaging with a Gaussian distribution.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 11. Mean Low pass filter(left) and Gaussian Low pass filter(right)

To implement these two types of low pass filters, two kernel sizes are chosen, which are 3*3 and 5*5. For the boundary of the image, padding is used to prevent the filter kernel exceeding the boundary of the whole image.

2.1.3 Experimental results

Figure 12. shows the images with AWGN processed by the mean low pass filters with kernel sizes 3*3 (left) and 5*5 (right). The PSNR scores for these two filters are 34.0595 and 30.9761. Figure 13. shows the images with AWGN processed by the Gaussian low pass filters with kernel sizes 3*3 (left) and 5*5 (right). The PSNR scores for these two filters are 34.9613 and 33.8815. All of these four filters can reduce the Gaussian noises to some degree, but the noises are still very obvious. At the same time, the filters make the images blurred and indistinct. And with larger size of filter kernels, the image blurs more. However, since the Gaussian low pass filter give more weights to the central pixels and less weights to the marginal pixels, the images processed by the Gaussian filter are less blurred, compared to the mean low pass filters.

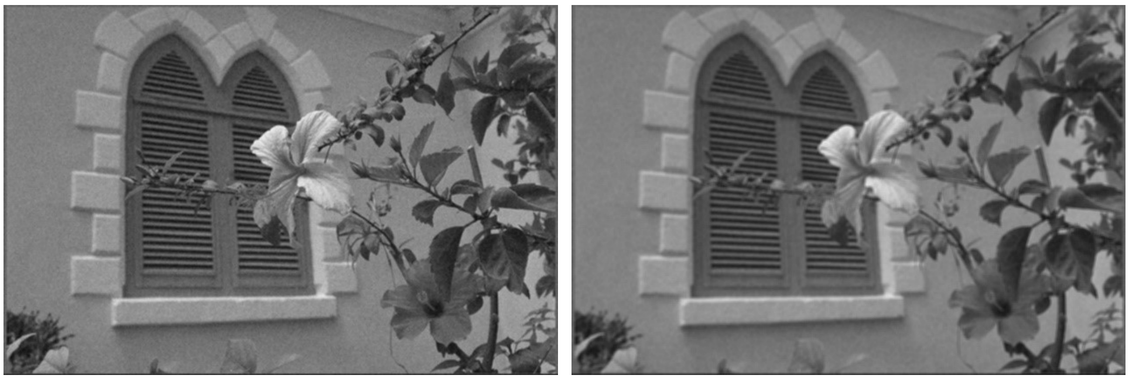


Figure 12. Results of mean low pass filters with kernel size 3*3(left) and 5*5(right)

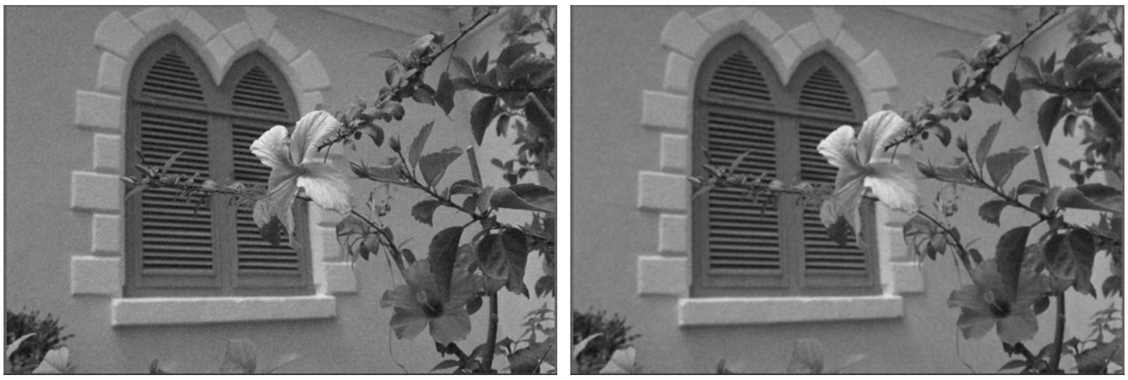


Figure 13. Results of Gaussian low pass filters with kernel size 3*3(left) and 5*5(right)

2.1.4 Discussions

Since the low pass filter is the simplest method to reduce the Gaussian noises in digital images, its effects are limited. To obtain a better effect of noise-removal, we have to choose larger size of filter kernel, because the sample mean value of Gaussian distributed samples are closer to 0 with larger sample numbers. However, the larger size of filter kernel can also make the image blur more, because the filtered pixels' values are influenced by their neighboring pixels. To work out this trade-off problem, we should choose the kernel size more carefully. For example, the Gaussian low pass filter with kernel size 5*5 can be the best choice in this experiment.

2.1.5 Non-programming questions

Q1: What is the type of embedded noise in Figure 5(b)? Justify your answer.

A1: It is additive white Gaussian noise, because there is no obvious white or black dot in the noisy image, which mean that it's not impulse noise. And the noise is evenly distributed among the whole image and noise on each pixel does not influence each other, so it is Gaussian noise.

2.2 Part B: Bilateral Filtering

2.2.1 Abstract and motivation

In part A, two simple low pass filters are applied to the digital images to remove the Gaussian noise. However, the processing results are not that satisfying, because the simple low pass filter can bring massive blurs to the images when doing the averaging of pixels. To improve the results, a more sophisticated method, the Bilateral Filtering, is introduced in this part.

2.2.2 Approaches and procedures

The bilateral filtering also uses filter kernel to slide on the image and calculate the central pixel values using their neighboring pixels within the kernel range. The central pixel value is calculated as the formula below:

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

Where(i,j) is the position of the central pixel, (k,l) is the position of the neighboring pixel, $Y(i,j)$ is the new value of the central pixel and $I(k,l)$ is the original value of the central pixel. The weighted factor $w(i,j,k,l)$ is calculated as the formula below:

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

The weighted factor $w(i,j,k,l)$ is decided by both the position differences between the central pixel and neighboring pixels and the gray-scale value differences between the central pixel and neighboring pixels. The two constant parameters σ_c and σ_s can be chosen according to different patterns of noises. They are related to the variance of position differences and gray-scale value differences.

2.2.3 Experimental results

The Bilateral filtering results are shown in Figure 14., with kernel size 3*3 (left) and 5*5 (right). The σ_c and σ_s for the filter with kernel 3*3 are set to be 20 and 50, and the σ_c and σ_s for the filter with kernel 5*5 are set to be 10 and 20. The PSNR for these two filters are respectively 35.8181 and 35.2788, which are way higher than the simple low pass filters. Most of the Gaussian noises are removed by the Bilateral filters, while the edges of the images are preserved. With larger kernel size, the denoising efficiency is better, but it also makes the image more blurred.

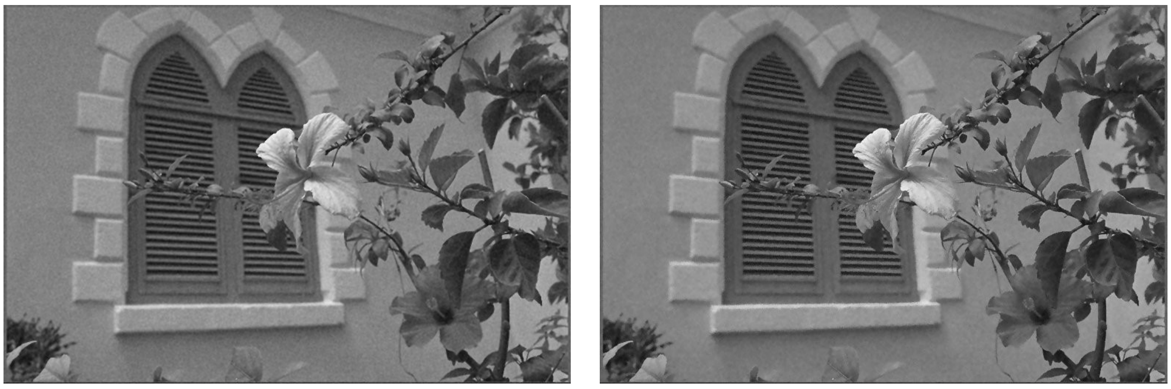


Figure 14. Bilateral Filtering with kernel sizes 3*3(left) and 5*5(right)

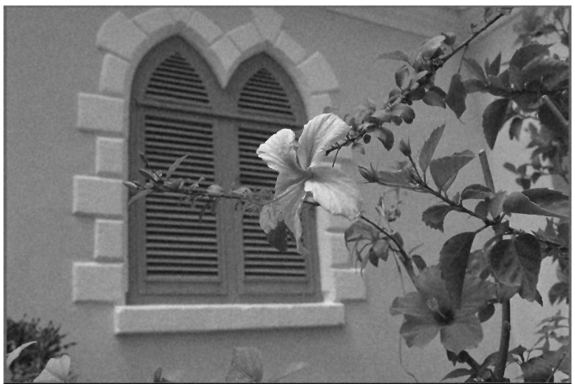
2.2.4 Discussion

In the bilateral filtering, σ_c and σ_s respectively represent the variance of gray-scale value differences and the variance of spatial differences (distances from the central pixel). In the area of edges, the variance of gray-scale values can be very high since the gray-scale value of pixels around those edges' areas can vary a lot from each other. The bilateral filtering counts all these differences into the weighted factor $w(i,j,k,l)$ to figure out where the edges are and to preserve the edges in the images.

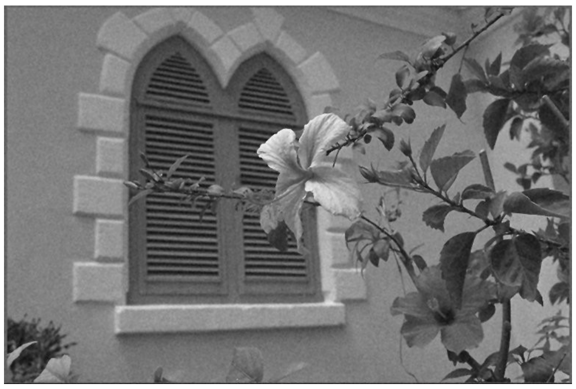
When choosing different σ_c and σ_s parameters, the denoising effects can be different. Figure 15. displays the processed the images through Bilateral filters with kernel size 3*3 and different σ_c and σ_s parameters.

(a)$\sigma_c = 5 \;\; \sigma_s = 50$

(b)$\sigma_c = 50 \;\; \sigma_s = 50$



(c)$\sigma_c = 5 \;\; \sigma_s = 50$



(d)$\sigma_c = 50 \;\; \sigma_s = 50$

Figure 15. Bilateral filtering with kernel size 3*3 with different σ_c and σ_s parameters

From the images above, the denoising effects for different parameters are nearly the same. However, with higher σ_c , the edges for the images are sharper, and with higher σ_s , the textures and details for the images are more preserved. To take a closer look to the denoising effects for different parameters, Table 1. below gives the PSNR scores for different parameters. From the table, changing σ_c value can hardly influence the denoising results but too high or too low σ_s value can make the denoising results worse.

Table 1. PSNR scores for Bilateral Filtering with kernel size 3*3 and different σ_c and σ_s

σ_c , σ_s	PSNR scores
20, 50	35.8181
5, 50	35.8353
50, 50	35.8171
20, 10	32.427
20, 150	34.4555

2.2.5 Non-programming questions

Q1: Explain the roles of σ_c and σ_s. Discuss the change in filter's performance with respect to the values of σ_c and σ_s.

A1: This question has been answered in section 2.2.4.

Q2: Does this filter perform better than linear filters you implemented in Problem 2(a)? Justify your answer in words.

A2: The Bilateral filters perform much better than the filters implemented in Problem 2(a), not only because the denoising effects are better with higher PSNR scores, but also because the edges and textures of the whole images are well preserved, which is better of visual observation. As is specified in section 2.2.4, the Bilateral filtering can detect the edges and complex textures areas using its weighted factor $w(i,j,k,l)$ and to preserve these details.

2.3 Part C: Non-Local Means (NLM) Filtering

2.3.1 Abstract and motivation

The simple low pass filters and the Bilateral filters are both based on a fixed-size filter kernels and local pixels around the central pixel of the filter kernel. But the NLM filter does not only calculate the mean value of the neighboring pixels around the central pixel, but also uses a smaller sliding window inside the filter kernel to correct the mean results to preserve more detailed information of the images when removing the noises.

2.3.2 Approaches and procedures

The central pixel value formula of NLM filter is quite similar to Bilateral filter, as shown below. The difference is at the weighted factor $w(i,j,k,l)$. In the NLM, the factor $w(i,j,k,l)$ is defined as the level of correlation between the central pixel and the neighboring pixels. If the $w(i,j,k,l)$ is higher, it means that the central pixel is much similar to the neighboring pixels, and the filter will make the new central pixel value more like its original value, vice versa. Therefore, when there is an area of edge, the $w(i,j,k,l)$ value will drop very fast, which correct the mean filter's results to preserve the edges.

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

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

In this part, the NLM filter is implemented by the MATLAB Image processing toolbox's function, imnlmfilt(). The parameters for it are set as: "SearchWindowSize" = 21; "ComparisonWindowSize" = 5; "DegreeofSmoothing" is acquired from the noise in the image. How to choose the parameters will be specified in section 2.3.4.

2.3.3 Experimental results

The NLM filtering results with parameters setting mentioned above are displayed in Figure 16. It is obvious that the denoising result is very perfect since there is barely any noise pattern that can be observed. Also, the edges and textures of the images are well preserved.

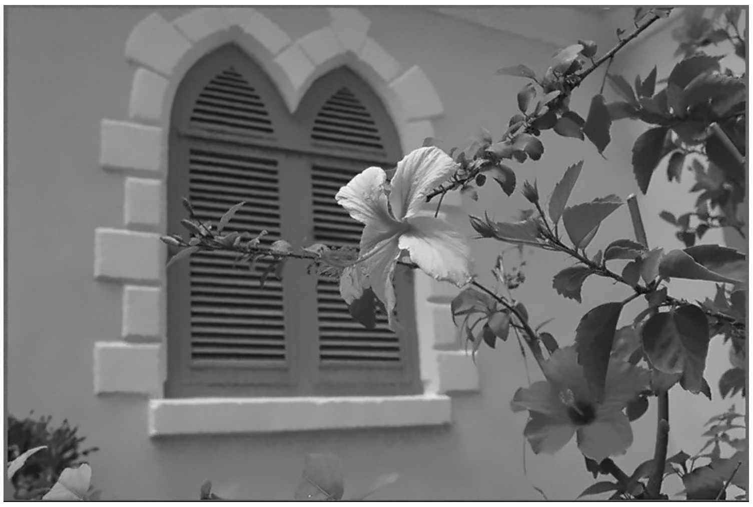
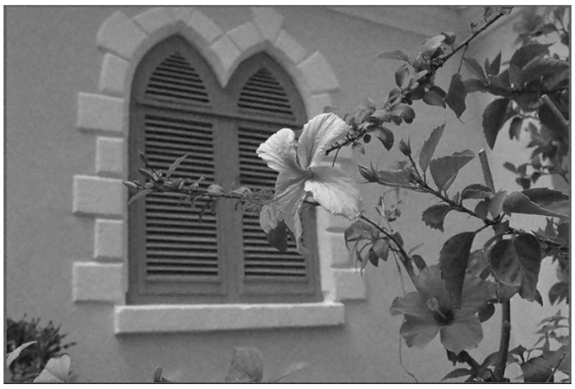


Figure 16. NLM filtering processing results

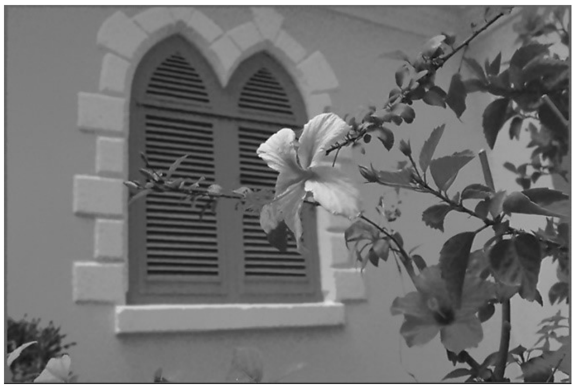
2.3.4 Discussion

Although the processing results of NLM filtering are quite satisfying according to Figure 16., we can still optimize the results by selecting the parameters for NLM filter more wisely. There are three parameters that can be changed in MATLAB imnlmfilt() function: "SearchWindowSize" = 21; "ComparisonWindowSize" = 5; "DegreeofSmoothing". Among all these three parameters, the "DegreeofSmoothing" is acquired by the function automatically, which equals to the standard deviation of the Gaussian noise existing in the image. We can consider that the default "DegreeofSmoothing" value is already optimized, so only the other two parameters will be discussed in this part, the big search window size N and the small comparison window size

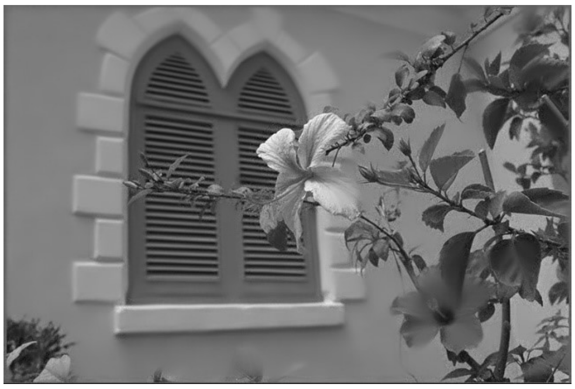
N′.



(a) N = 5, N′ = 5



(b) N = 99, N′ = 5



(c) N = 21, N′ = 21

Figure 17. Results of NLM filtering with different parameters setting

Results of NLM filtering with different parameters settings are shown in Figure 17. above. From Figure 17(a) we can find that: if the big searching window is too small, the denoising effects will get worse and more Gaussian noise will appear even after applying the filter to the image.

From Figure 17(b) we can find that: with big searching window that is too large, the denoising effects actually is very similar to a relatively smaller searching window (e.g., N = 21), while the image can blur more. In addition, a larger searching window can consume way more time to do the calculation for the program, so it is very inefficient to choose a very large searching window.

Figure 17(c) shows the results when the small comparison window size is as same as that of the big searching window size. We can find that the overall denoising effects are still good, but that some areas can blur a lot (such as the leaves). Therefore, the chosen comparison window size should match the big searching window size to obtain a better processing result.

Therefore, for this certain image, the big window size N is set as 21 and the small comparison window size N′ is set as 5, and we get a optimized result as shown in Figure 16. above.

2.3.5 Non-programming questions

Q1: Try several filter parameters and discuss their effect on filtering process. Clearly state your final choice of parameters in your report.

A1: This question has been answered in section 2.3.4.

Q2: Compare the performance of NLM with filters used in Problem 2(a) and Problem 2(b).

A2: This question has been answered in section 2.3.3.

2.4 Part D: Mixed noises in color image

2.4.1 Abstract and motivation

In this part, more complex situations will occur, that is the RGB images with mixed noises of both Gaussian noise and impulse pepper/salt noise. For the RGB images, we can apply the mono-channel filters for R, G and B channels separately and then merge them. For the mixed noises, we can apply multiple types of filters in a certain order according to the types of the noises.

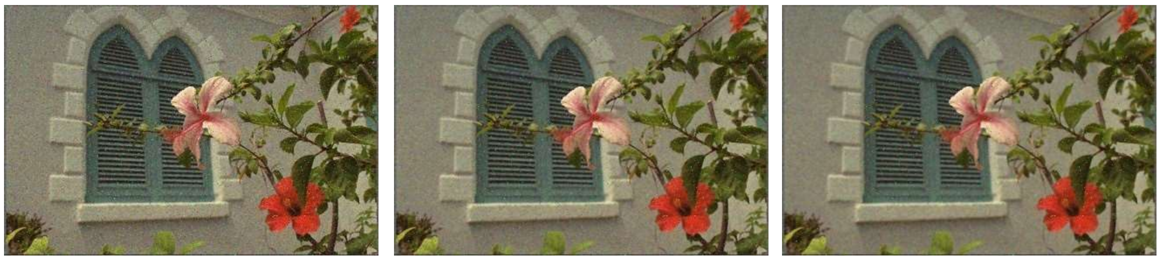
2.4.2 Approaches and procedures

The noisy images provided in this part has a mixed noise of both the Gaussian noise and impulse noise. How to identify this mixed noise will be specified in section 2.4.5. For this kind of noises, we can apply both the bilateral filters and NLM filters to the images to find the results.

Firstly, the image is separated into three channels of R, G and B, and then each channel is processed by a series of filters separately. After the processing procedures, three channels are merged to get a new RGB image, which is the denoised image. The experiments in this part will apply different combinations of filters to the noisy RGB images and show the results of them.

2.4.3 Experimental results

Firstly, only one single type of filter is applied to the noisy RGB image. Figure 18. shows the results of several times of Bilateral filtering. The bilateral filters can remove some of the Gaussian noise, but it cannot handle the impulse noise. No matter how many times the Bilateral filters are applied to the image, the impulse noise still exists. Therefore, the single Bilateral filtering is not suitable for this situation.



(a) Bilateral x 1 (b) Bilateral x 2 (c) Bilateral x 3

Figure 18. Results of several times of Bilateral filtering

Figure 19. shows the results of a single NLM filter applied to the noisy RGB image. The denoising result is quite good, since a single NLM filter can already remove most noises, including the Gaussian noise and the impulse. However, there are still some noisy pixels existing on the image, especially for the complex areas, such as the window in the background. Also, the image is getting more blurred.

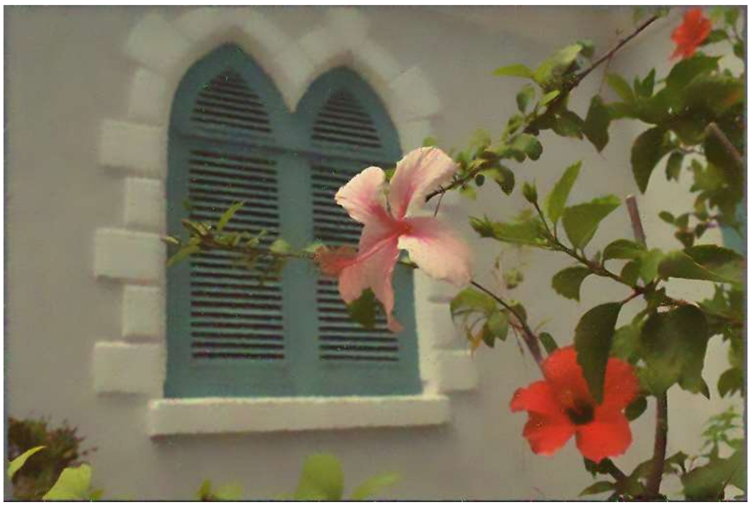


Figure 19. Results of a single NLM filter applied to the noisy RGB image

Finally, combinations of Bilateral filters and NLM filters are applied to the noisy RGB images and the results are shown in Figure 20. When applying Bilateral filter first and then the NLM filter, the result is as the figure on the left. The image is less blurred and the Gaussian noises are almost removed, but most of the impulse noise are not removed. Clearly this is not we expect. When applying NLM filter first and then the Bilateral filter, the result is shown as the figure on the right. The mixed noise of both Gaussian noise and impulse noise are almost cleared. There is only a few minor noisy areas on the image. But the image is more blurred, compared to Figure 19.

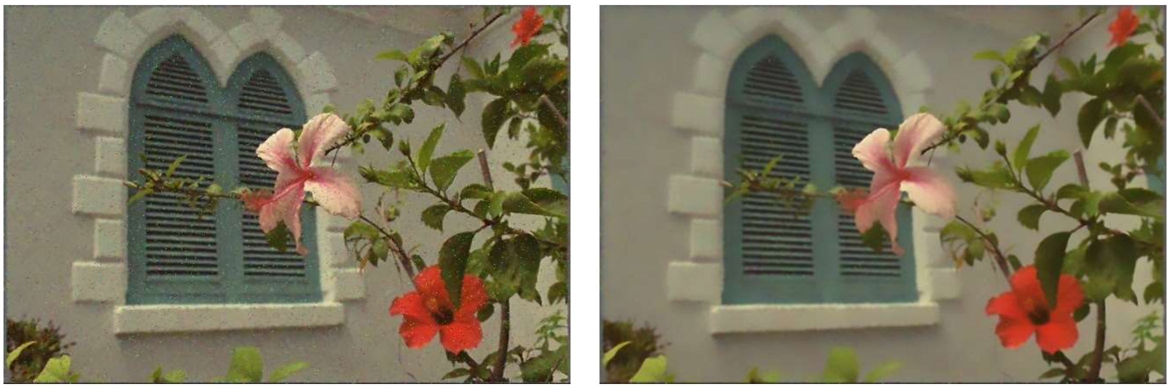


Figure 20. Results of Bilateral + NLM (left) and NLM + Bilateral (right)

To conclude, when denoising a RGB image with mixed noises of both Gaussian noise and impulse noise, we can apply a NLM filter first to see the result. If there are still some impulse noises existing, we can then apply a Bilateral filter to the image. This can remove most of the mixed noise while the edges and details of the image are still preserved.

2.4.4 Discussion

Although the combination of NLM and Bilateral filters has already obtained a good result when handling the RGB images with mixed noises of both Gaussian noise and impulse noise, there are still some pepper/salt noisy dots sticking to the image. This makes sense because both the NLM filter and Bilateral filter are based on averaging the neighboring pixels. They are very effective when facing the Gaussian noise only, because the mean value of a Gaussian noise within a range converges to 0. However, the impulse noise does not work in this way. When do the averaging to a window when there are several impulse noisy pixels in the window, it can make the central pixel value much higher if do the averaging. Therefore, the impulse noise can hardly be removed if only using these two filters. The best way to handle impulse noise is to use the median filtering or fancier filtering methods.

Also, processing each channel of R, G, B separately can also bring some color distortions to the whole image, because for the same pixel position, the pixel changes are not synchronized. Therefore, the color of different areas can vary a lot. To solve this problem, some algorithms designed for RGB images can be take into consideration, such as BM3D.

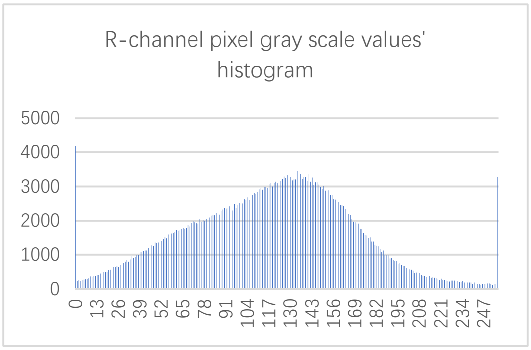
2.4.5 Non-programming questions

Q1: What types of noises are there? Justify your answer.

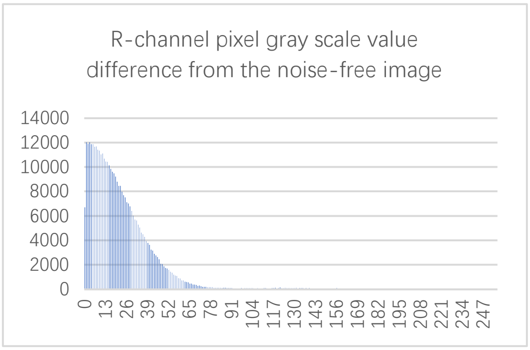
A1: In this image, there are at least two types of noises: Gaussian noises and impulse noises.

When we plot the histogram of R-channel pixel gray scale values, as shown in the left figure below, we can find that there are much more pixels in value of 0 and 255, which are exactly the impulse noisy pixels (the pepper and salt pixels). Therefore, the image contains impulse noises.

When we plot the histogram of R-channel pixel gray scale value difference from the pixels of the original noise-free image (the absolute value of the difference between the pixel from the noisy image and the pixel of the same position from the original noise-free image), as shown in the right figure below, we can find that the difference is Gaussian distributed, which means that the noise is distributed as a Gaussian distribution. Therefore, the image contains both the Gaussian noises and the impulse noises.



(a)



(b)

Q2 What filters would you like use to remove mixed noise? Can you cascade these filters in any order? Justify your answer.

A2: This question has already been answered in section 2.4.2 and 2.4.3.

Q3-1: Describe your method and show its results

A3-1: This question has been already answered in section 2.4.3.

Q3-2: Discuss its shortcomings.

Q3-3. Give some suggestions to improve its performance.

A3-2 & A3-3: These two questions are answered in section 2.4.4.

III. Problem 3: Special Effect Image Filters: Creating Frosted Glass Effect

3.1 Abstract and motivation

In this Problem, a new filter will be introduced, which is called the frosted glass filter. This is a widely used filter, especially for some special effects for the pictures and images. When the frosted glass filter is applied to a noisy picture, it can even reduce the noise to some degree. In this part, the frosted glass filter will be implemented and applied to the noisy images to find the results.

3.2 Approaches and procedures

The principle for the frosted glass filter is very simple. It chooses a certain window size, and replace the central pixel value with a random pixel within the window. In this part, the window size will be chosen as N=5. And the program will use the random number generator in C++ to randomly choose a pixel to replace the central pixel.

3.3 Experimental results

Firstly, the frosted glass is applied to a noise-free image. Figure 21.(left) shows the results. The results are exactly as expected.

Then, the frosted glass filter is applied to a noisy RGB image. The results are shown in Figure 21.(right). The image is still noisy because the frosted glass filter only replace the pixel value with some value from another pixel. Therefore, the noise still exists.

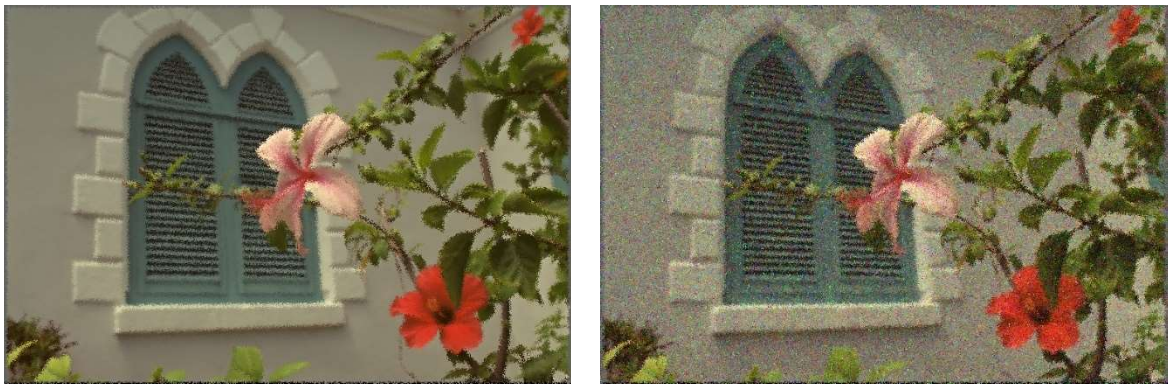


Figure 21. Apply the frosted glass filter to a noise-free image(left) and a noisy image

Finally, the combination of Bilateral filter and frosted glass filter is applied to the noisy image. Figure 22. shows the results. When apply the frosted glass filter first and then the bilateral filter, as shown in Figure 22 (left), the denoising result is not good enough. There are still some severe Gaussian noisy patterns existing on the image. However, when apply the bilateral filter first and then the frosted glass filter, as shown in Figure 22 (right), the denoising effects are much better. Most of the Gaussian noises are removed and the overall effects of the image is as same as the results from a noise-free image, as shown in Figure 21 (left).

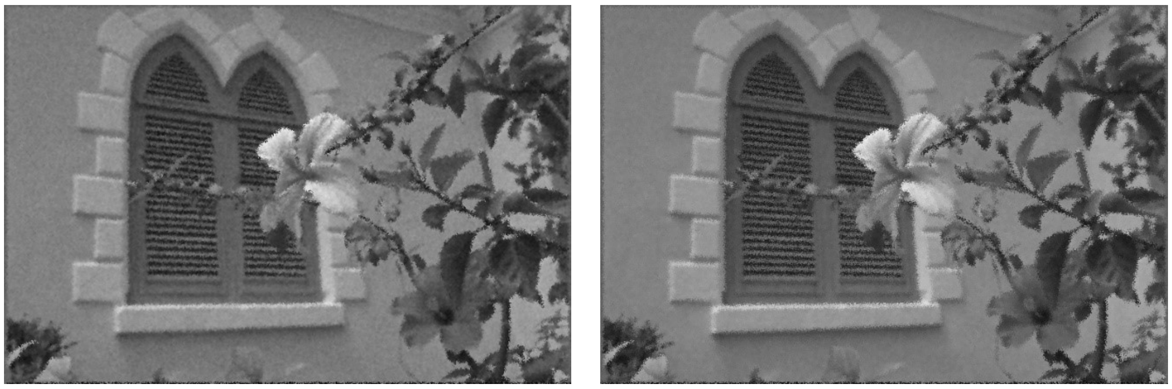


Figure 21. Apply the frosted glass filter + Bilateral filter(left) and Bilateral filter + frosted glass filter(right) to a noisy gray scale image

3.4 Non-programming questions

Q1: Repeat the process for the noisy image Flower_noisy.raw. Describe your observations and discuss whether the noise leads to any difference.

A1: This question is answered in section 3.3.

Q2: Considering the Bilateral filtering denoising algorithm. Try the following two processes:
a. First, perform denoising on Flower_gray_noisy.raw. Then, apply the frosted-glass filtering.
b. First, apply the frosted-glass filtering on Flower_gray_noisy.raw. Then, perform denoising.
Compare your results and discuss your observations.

A2: This question is answered in section 3.3.