

## HOMEWORK #1

Issued: 1/7/2019 Due: 11:59PM, 1/22/2019

Yan Jiao

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

#### 1.1 Bilinear Demosaicing (10%)

##### 1.1.1 Motivation

The original image is a grey image, only contains one channel information. We can use bilinear demosaicing to rebuild the colorful image with three channel R, G and B. This procedure is very simple, so the processing speed will be very fast. Because of these, we can store a color image using less memory space.

Extend the boundary of the picture in order to apply the demosaicing process to each pixel in the original picture unified.

For each pixel, decide which color it is, and then apply the bilinear interpolation equation to get the other two color values.

##### 1.1.2 Approach

Using C++.

Extend the boundary of the picture by 1px at each side, using the reflection way (Figure 1.1).

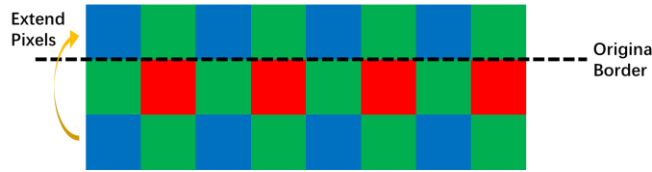


Figure 1.1: reflection border extension

(a) For original green pixel  $(i, j)$ ,  $i$  is odd,  $i = 1, 2, 3, \dots, N$ :

$$\text{Red value: } R_{i,j} = \frac{1}{2}(R_{i,j-1} + R_{i,j+1})$$

$$\text{Blue value: } B_{i,j} = \frac{1}{2}(B_{i-1,j} + B_{i+1,j})$$

(b) For original green pixel  $(i, j)$ ,  $i$  is even,  $i = 1, 2, 3, \dots, N$ :

$$\text{Red value: } R_{i,j} = \frac{1}{2}(R_{i-1,j} + R_{i+1,j})$$

$$\text{Blue value: } B_{i,j} = \frac{1}{2}(B_{i,j-1} + B_{i,j+1})$$

(c) For original red pixel  $(i, j)$ ,  $i$  is always odd,  $i = 1, 2, 3, \dots, N$ :

$$\text{Green value: } G_{i,j} = \frac{1}{4}(G_{i-1,j} + G_{i+1,j} + G_{i,j-1} + G_{i,j+1})$$

$$\text{Blue value: } B_{i,j} = \frac{1}{4}(B_{i-1,j-1} + B_{i+1,j+1} + B_{i-1,j+1} + B_{i+1,j-1})$$

(d) For original blue pixel  $(i, j)$ ,  $i$  is always even,  $i = 1, 2, 3, \dots, N$ :

$$\text{Red value: } R_{i,j} = \frac{1}{4}(R_{i-1,j-1} + R_{i+1,j+1} + R_{i+1,j-1} + R_{i-1,j+1})$$

$$\text{Green value: } G_{i,j} = \frac{1}{4}(G_{i,j-1} + G_{i,j+1} + G_{i-1,j} + G_{i+1,j})$$

##### 1.1.3 Results



Figure 1.2: output picture of the cat.raw

#### 1.1.4 Discussion

In the Figure 1.3, we can find there are obvious zipper effect and false color in the output image.



Figure 1.3: detail contrast between the output picture and the cat\_ori.raw

In the Figure 1.4, we can see the formation procedure of these artificial effects. The very sharp edge will become zipper shape edge (c) after the bilinear interpolation (d, e, f).

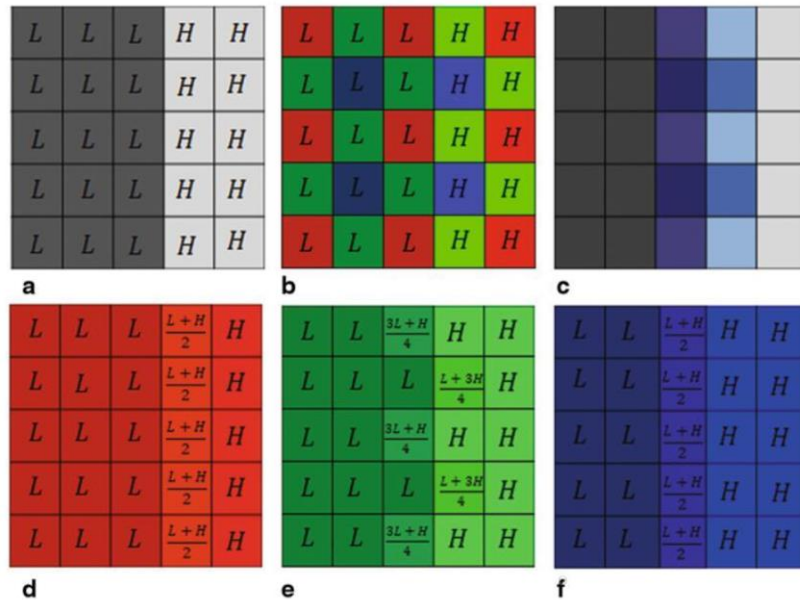


Figure 1.4: artificial effects formation procedure

The artificial effects mainly because lesser surrounding pixel value, so the error between the original image and the demosaicing image will be a little bit large, especially at the color region edges. One way to improve this is to take more surrounding pixel values into the decision of the target pixel value, and let closer surrounding values have larger weight coefficient. Bilinear method only use the same channel color information to calculate. But other channels' color information will also have effects on the target pixel, we should also take that into consideration.

## 1.2 Malvar-He-Cutler (MHC) Demosaicing (20%)

### 1.2.1 Motivation

The simple bilinear demosaicing process has some less-than-ideal artificial effect. In order to have a clearer output image, we should take more surrounding pixels to calculate the center pixel value.

Outer surrounding pixels should have less effect on the center pixel. The information in other color channel cannot be ignored.

### 1.2.2 Approach

Using C++

Extend the border by 2px at each side, because we need the 1px outer (compare to bilinear demosaicing) to reduce the color value error.

Use the given filter coefficients shown in the HW instruction.

Need to pay attention to the “already has green” pixel, there are two kinds of filter patterns which need different calculation formulas.

### 1.2.3 Results

(1)



Figure 1.5: MHC demosaicing result of cat.raw

(2)



Figure 1.6: the bilinear demosaicing result (left), original (middle) and the MHC demosaicing result (right)

In the Figure 1.6, we can find that bilinear result is much more blurring than the MHC result.

Table 1.1: comparison between bilinear and MHC

	bilinear	MHC
false color	more	less
zipper effect	more	less
edge	blur	sharp
brightness	higher	true to original image

### 1.2.4 Discussion

Because when deciding the color value of a specific pixel, MHC method uses more pixel information around the target pixel (other color channel's information), also, this method does not only calculate the average of

the surrounding pixel values, it add the weight coefficient into the formula. The result is much clearer than the bilinear output, but takes more program run time.

### 1.3 Histogram Manipulation (20%)

#### 1.3.1 Motivation

Some images' histograms have pixel intensity cluster to one or several regions, so the image's contrast may be very low, and objects in this image will be difficult to identify. In order to make the image have more contrast, we should let the cluster pixel intensity spread to a wider range, make the histogram distribution more like the uniform distribution.

#### 1.3.2 Approach

Using MATLAB.

Method A:

1. Get the CDF of the original picture;
2. Because we want the image distribution to be more like uniform distribution in the range of [0, 255], so we can use the mapping rule:  $x_{\text{output}} = \text{CDF}(x_{\text{input}}) * 255$ .

Method B:

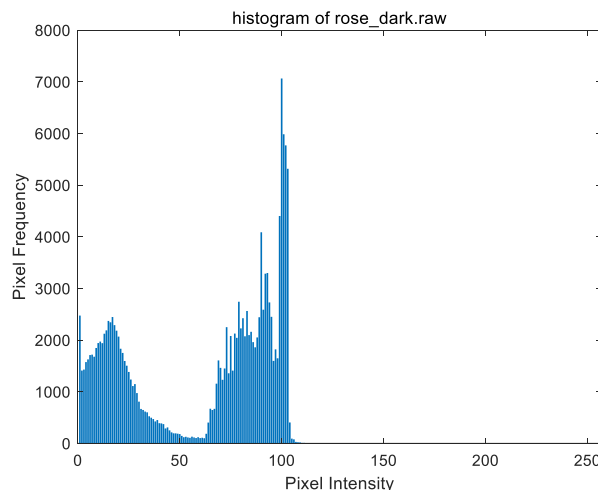
1. Like filling the bucket, we need to spread the pixels into 256 buckets equally, so we should calculate how many pixels should go to each bucket;
2. When the pixels having the same color value need to go to different buckets, we do not care about the order;
3. Like Method A, we need to calculate the CDF of the original image;
4. When deciding which bucket one pixel should go, (suppose the pixel value is  $x$ ) we need calculate  $(\text{cdf}(x-1) * \text{total pixel number} + \text{times this } x \text{ value shows}) / (\text{bucket capacity})$ .

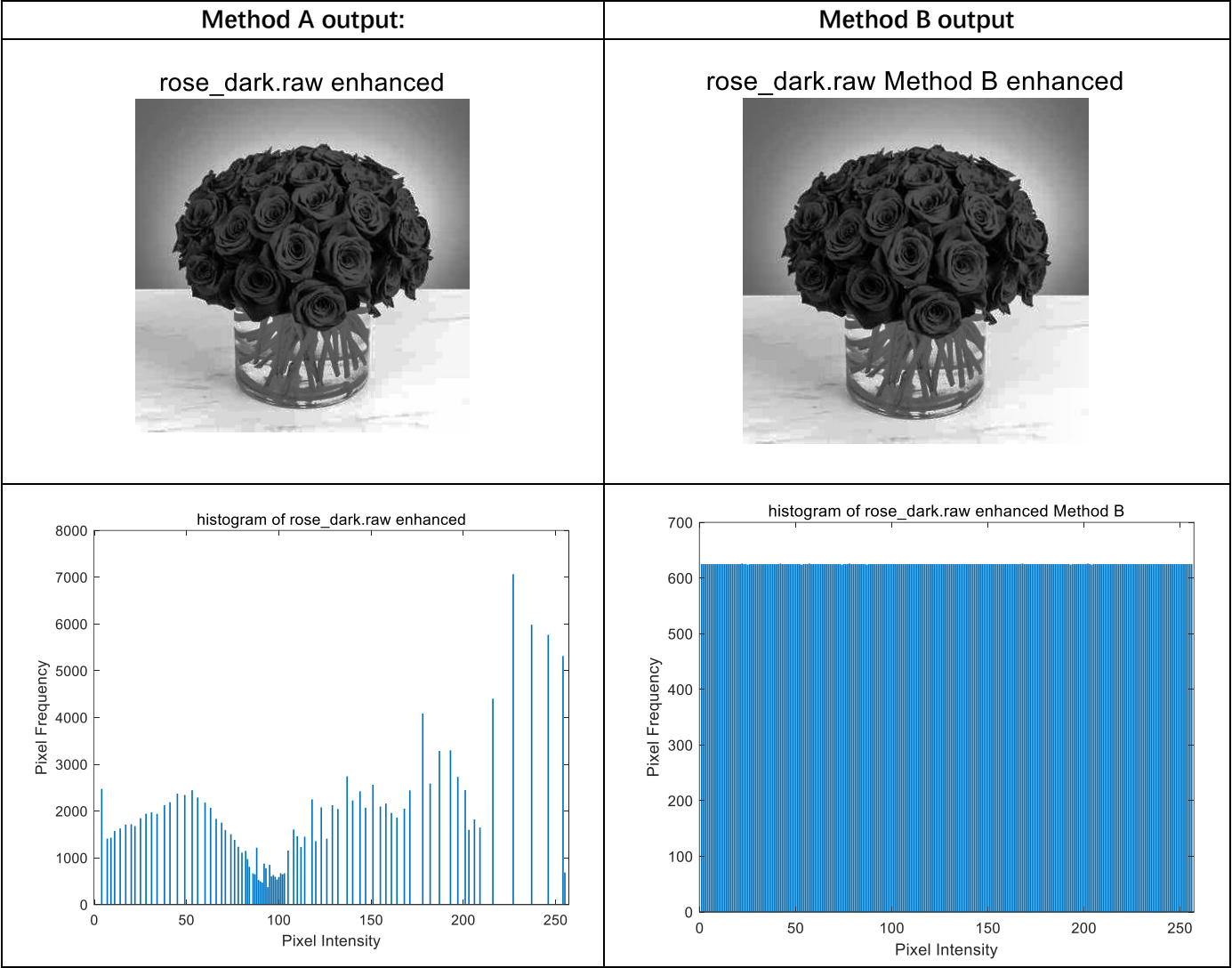
```
for i=1:400
    for j=1:400
        countB(a(i,j)+1)=countB(a(i,j)+1)+1;
        if a(i,j)+1~=1
            b(i,j)=floor((cdfInput(1,a(i,j))*len+countB(a(i,j)+1)-1)/bucVol);
        else
            b(i,j)=floor((countB(a(i,j)+1)-1)/bucVol);
        end
    end
end
```

Figure 1.7: critical code when deciding the output pixel value

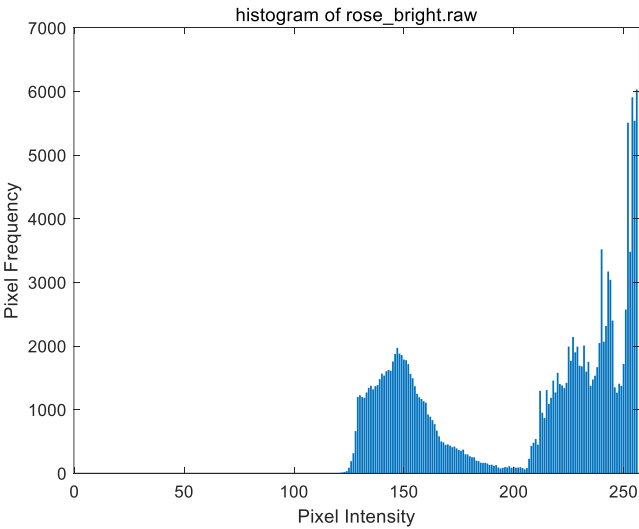
#### 1.3.3 Results

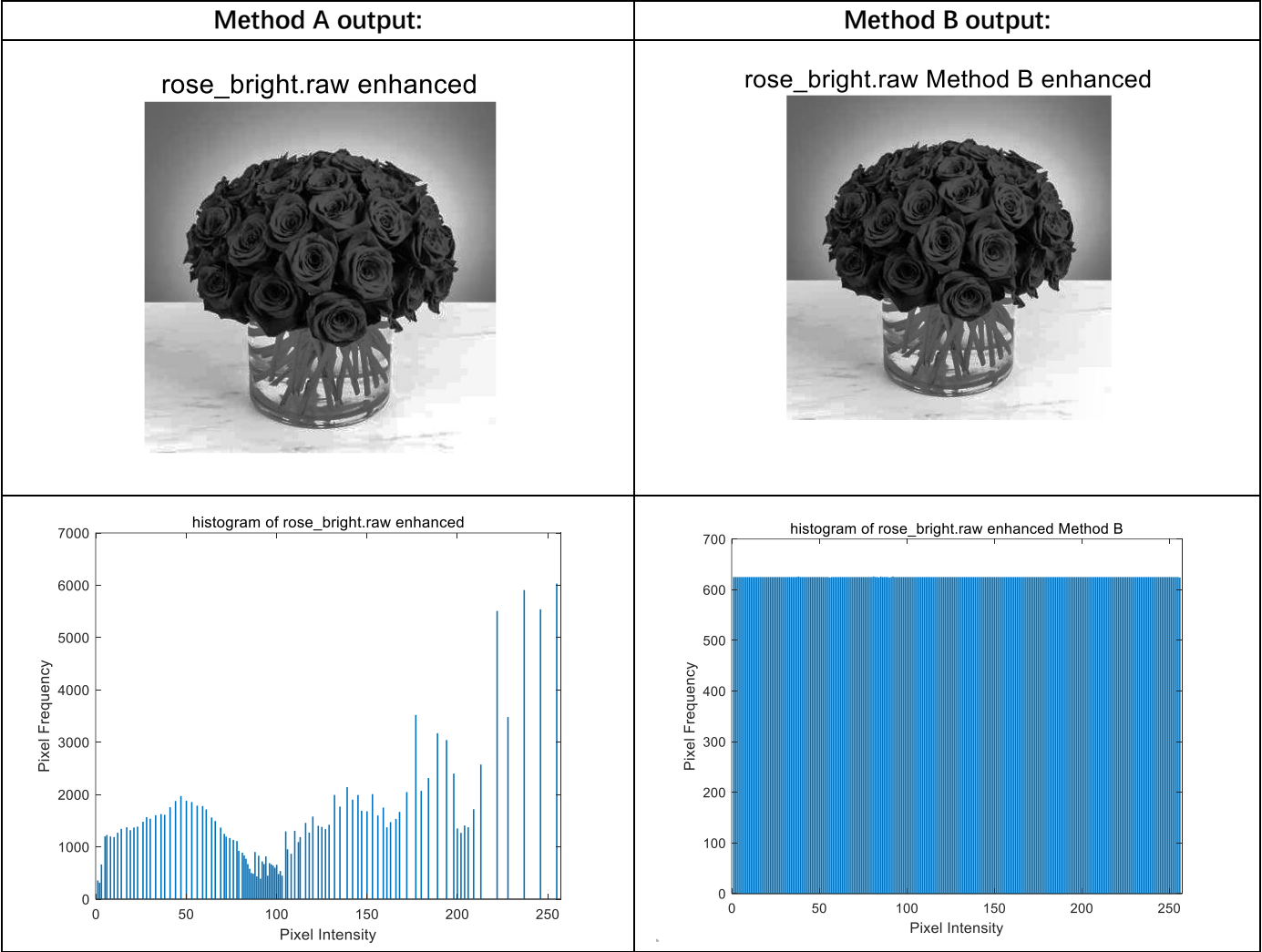
(a) rose\_dark.raw:





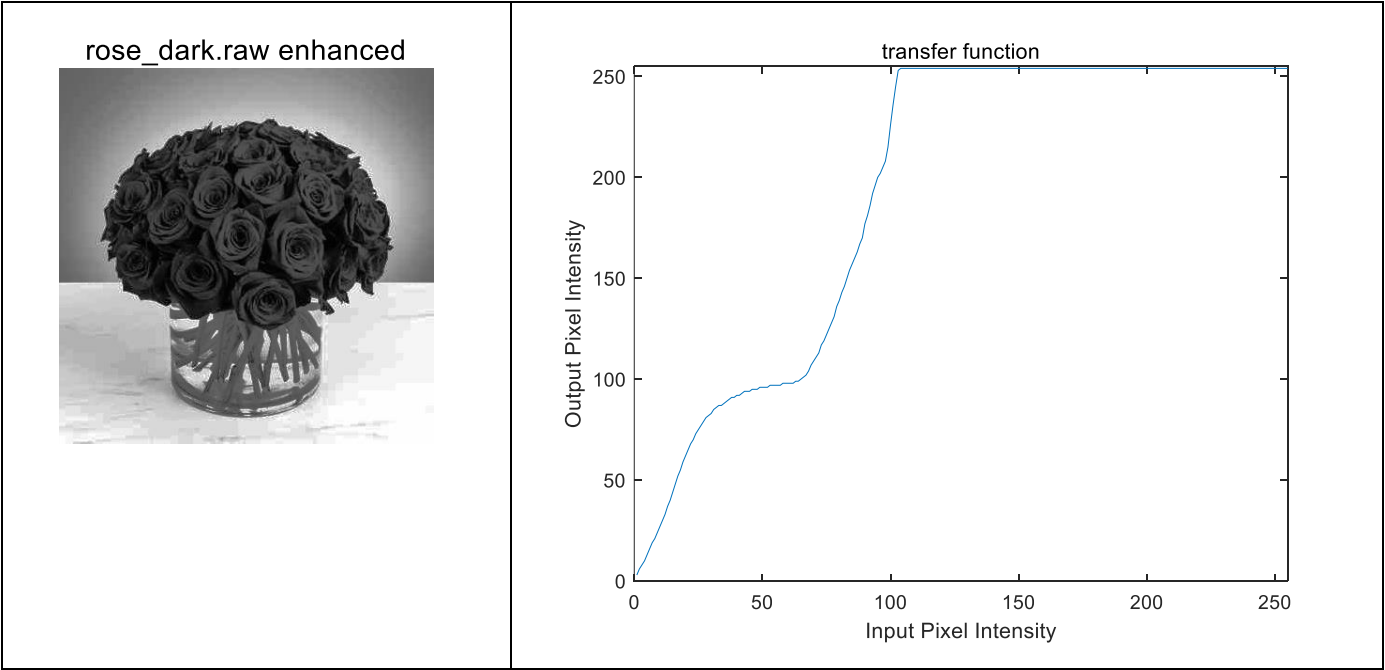
rose\_bright.raw:





(2)

Rose\_dark:



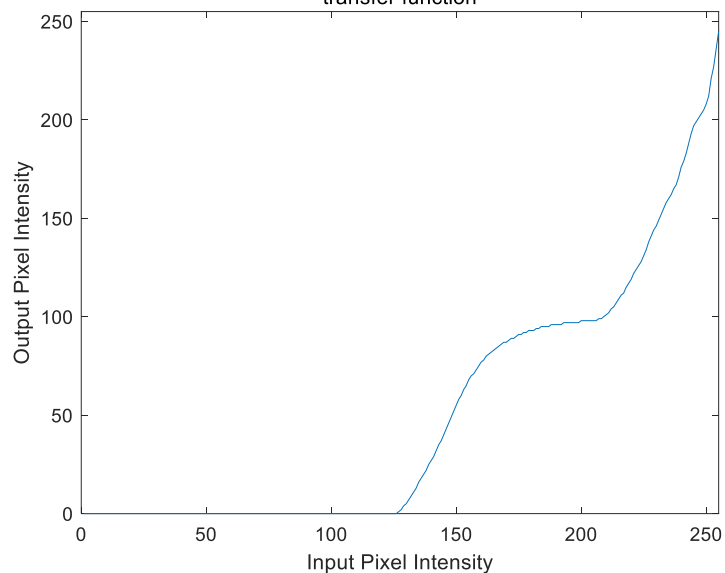
Rose\_bright:



rose\_bright.raw enhanced

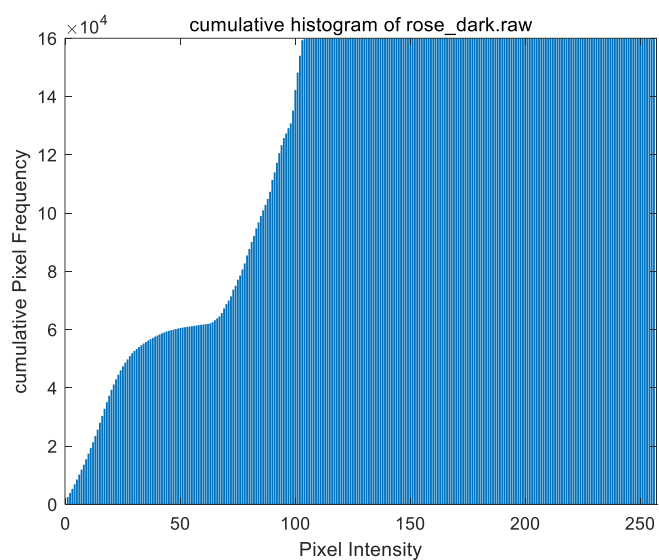


transfer function

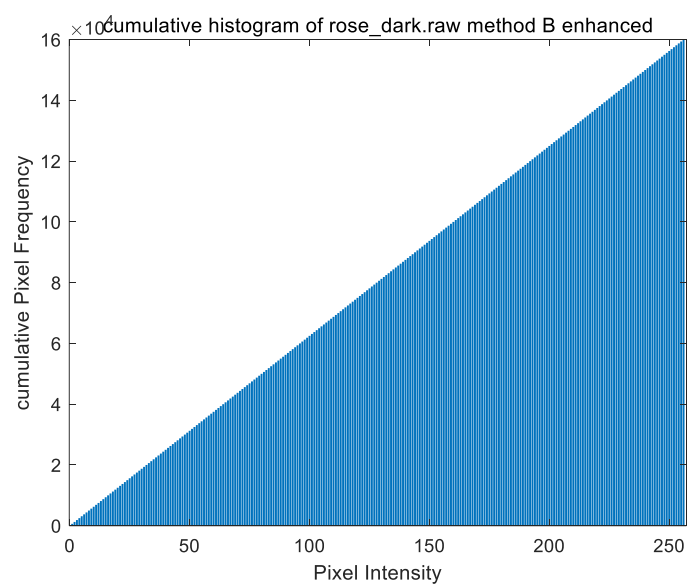


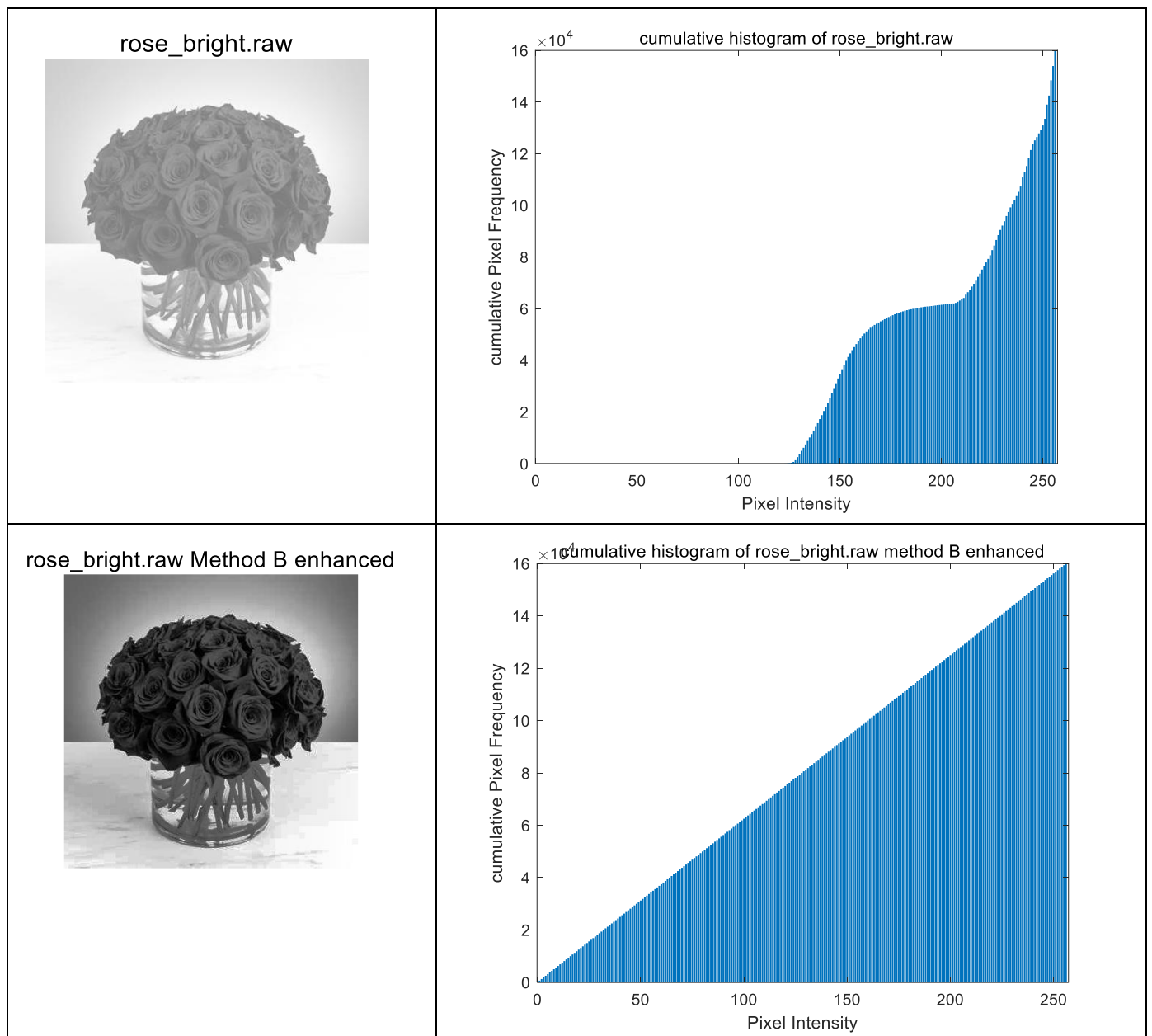
(3)

rose\_dark.raw



rose\_dark.raw Method B enhanced



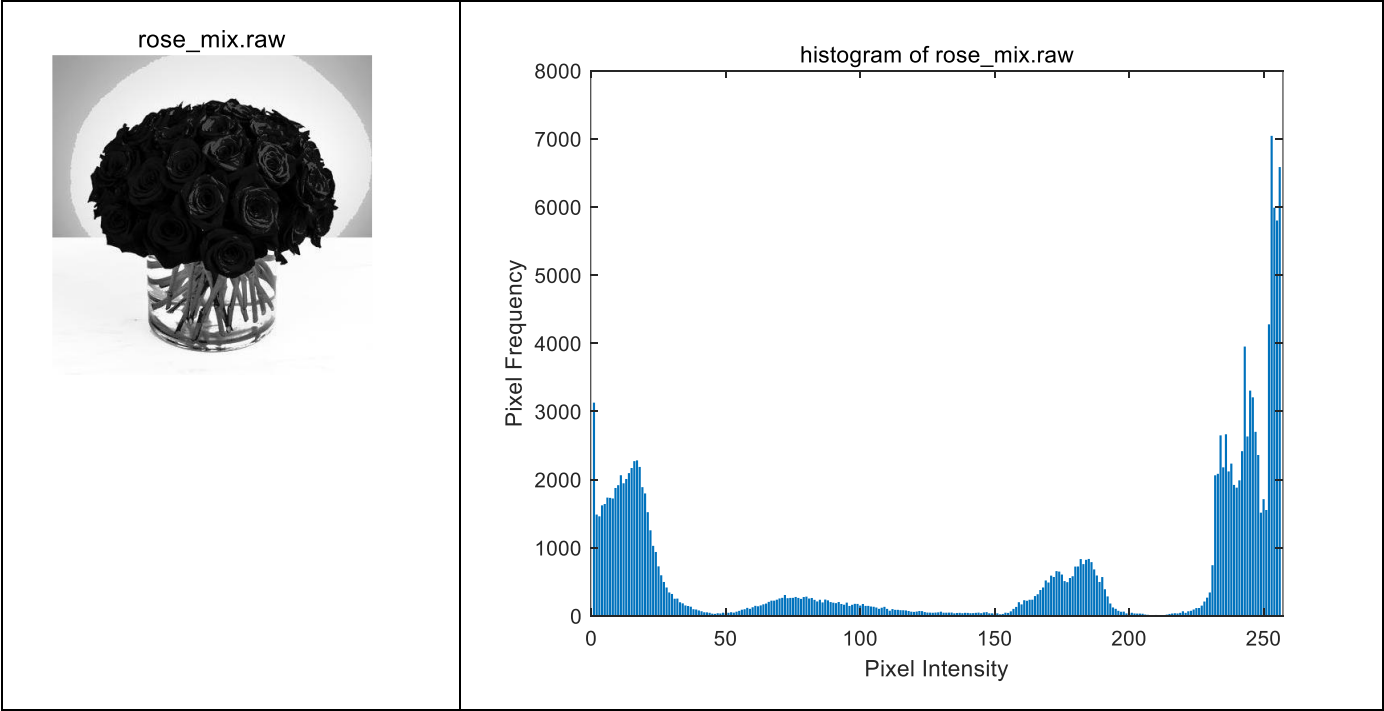


(4)

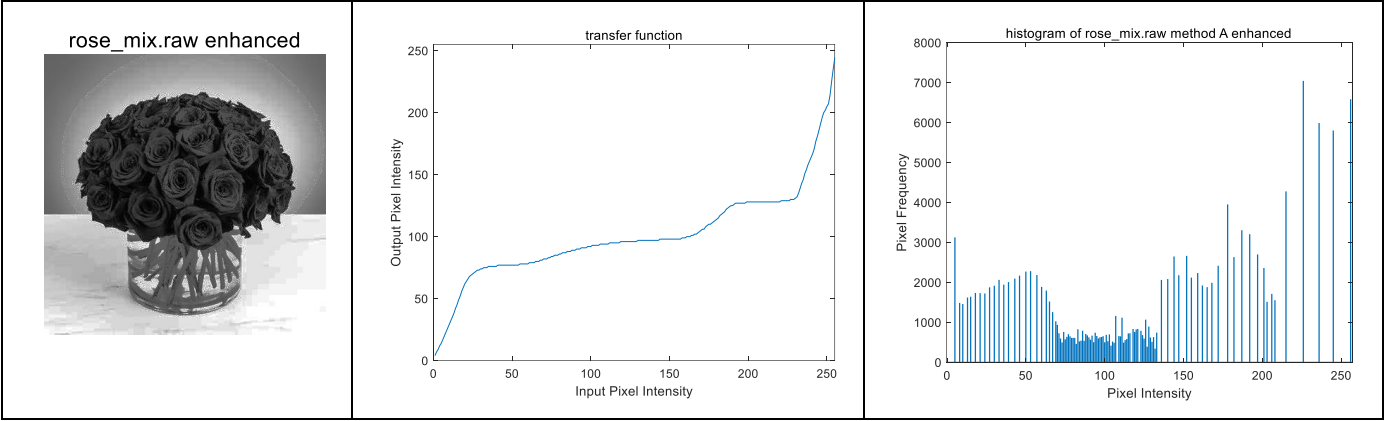
For Method A, we can divide the pixels into several cluster, and apply the transfer process seperately to reduce the noise near the color edges.

(5)

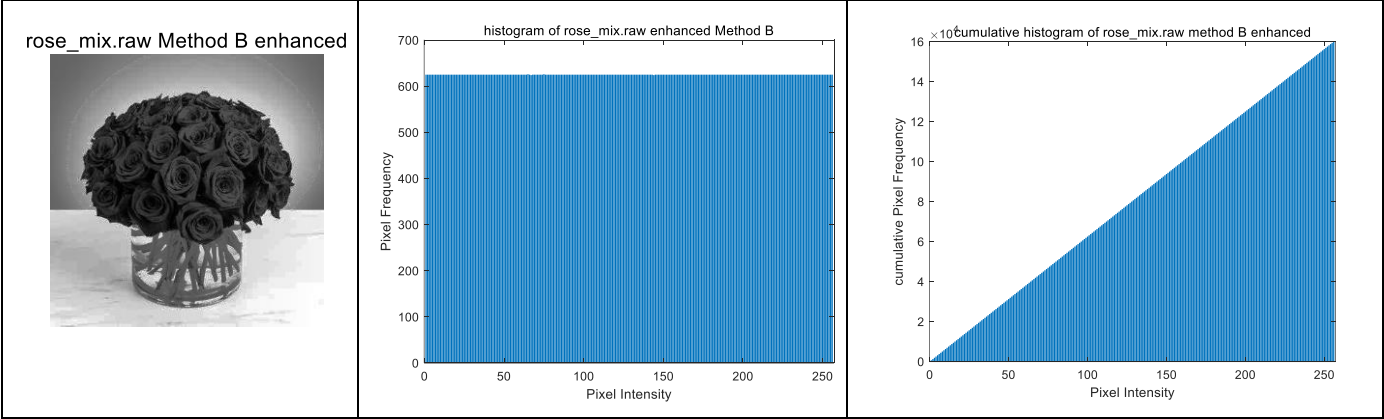




Method A:



Method B:



1.3.4 Discussion

When apply the two methods to the rose\_mix.raw, we can find more noise near the sharp color edges. One way to improve this is to divide the original image histogram into several pieces, and apply these two methods

to these pixel cluster pieces separately.

## Problem 2: Image Denoising (50%)

### 2.1 Gray-level image (20%)

#### 2.1.1 Motivation

In order to denoising the uniform noise image, we can use the pixels surround the target pixel to reduce the uniform noise. We need to choose the suitable filter window size to get the best output image. In addition, taking weight coefficient into consideration is a good choice.

#### 2.1.2 Approach

Using C++ and MATLAB (for plot distribution PDF/CDF)

Extend the image boundary first.

For uniform filter, we just need to get the average color value of all the pixels inside the filter window.

For Gaussian filter:

We use 
$$Y(i, j) = \frac{\sum_{k,l} I(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)} \text{ and } w(i, j, k, l) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(k-i)^2 + (l-j)^2}{2\sigma^2}\right)$$
 to get

the color value at (i, j), where  $\sigma$  is the standard of Gaussian distribution.

#### 2.1.3 Results

(1)

Uniform noise, because the error between the original image and the noise image distribution is shown in Figure 2.1.

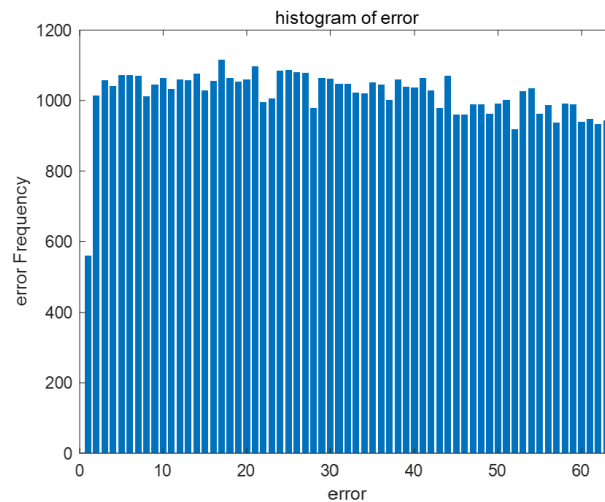





Figure 2.1: histogram of error

(2)

Table 2.1: Uniform and Gaussian filter output image

window size	Uniform	Gaussian, $\sigma=10$	Gaussian, $\sigma=2$
3			

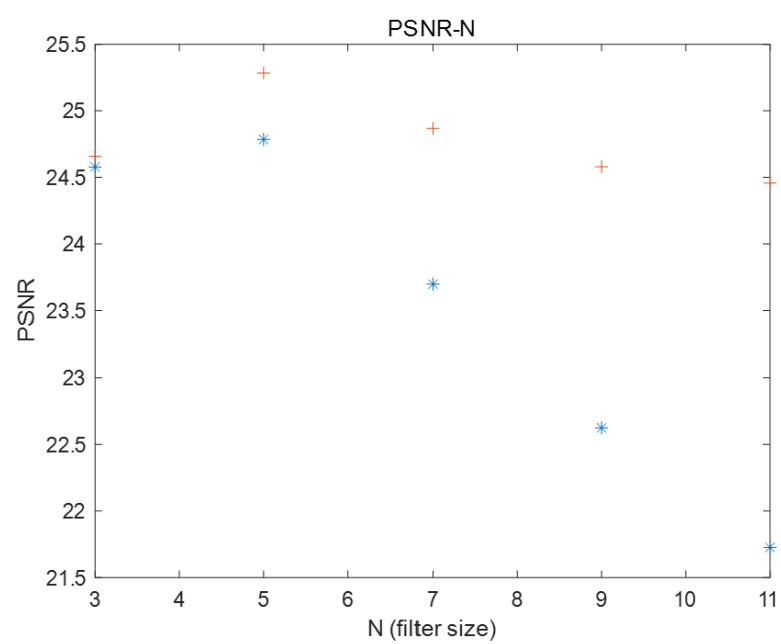
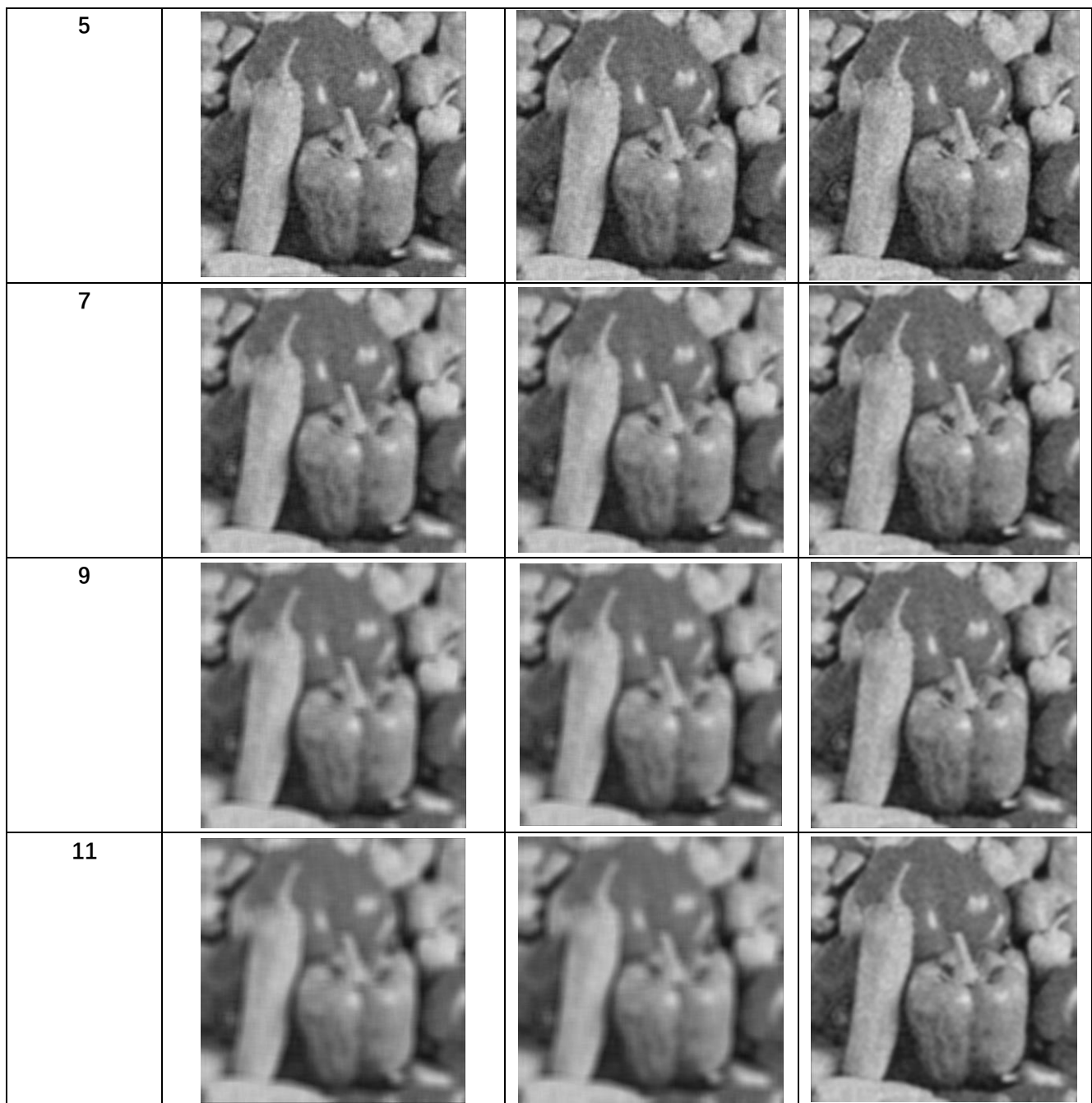

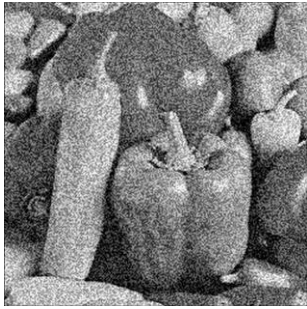

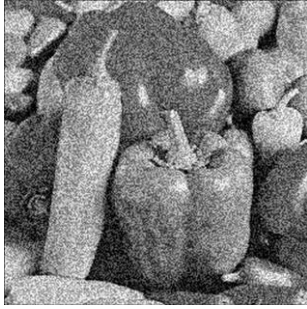
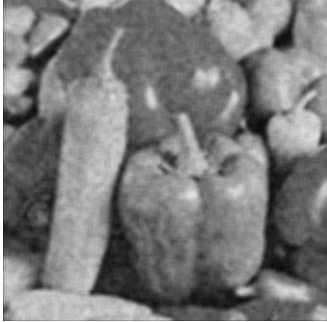


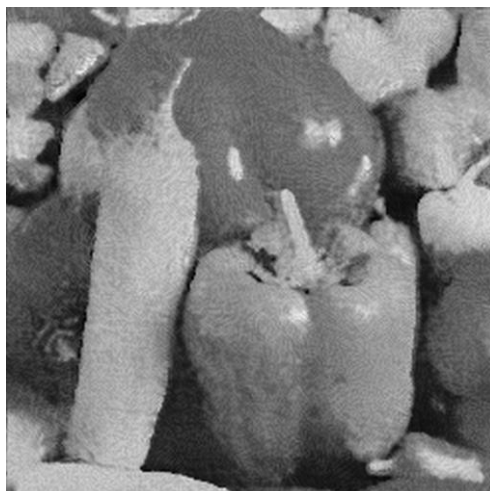
Figure 2.2: PSNR of different filter window size (blue: uniform; red: Gaussian  $\sigma=2$ )

(3) Bilateral, filter size =  $5 * 5$

$\sigma_c$	$\sigma_s$	Output image	$\sigma_c$	$\sigma_s$	Output image
2	200		2	10	
100	200		100	10	
200	200				

(4)

Non-local-mean:  $h=10$ ,  $\text{sig}=50$



#### 2.1.4 Discussion

In Table 2.1, we can see that the Gaussian filter ( $\sigma=2$ ) output is much clearer than the uniform filter.

However, if we set the  $\sigma$  to a much larger value, the output image will be much more like the uniform output, because the weight coefficient of closer pixel and further pixel will be very similar.

In Figure 2.2, we can see that when the window size is 5, the output denoising images' PSNR are highest, it means that the denoising image is more like the original image. Along with the increasing of the filter window size, the output image will be much more blur. So, for this specific image (this size and amount of image details) filter size 5 is a good choice.

## 2.2 Color image (20%)

### 2.2.1 Motivation

### 2.2.2 Approach

### 2.2.3 Results

#### (1) uniform noise

### 2.2.4 Discussion

## 2.3 Shot noise (10%)

### 3.3.1 Motivation

### 3.3.2 Approach

### 3.3.3 Results

### 3.3.4 Discussion