EE569 Digital Image Processing
Name: Shuo Wang
USC ID: 8749390300
Email: wang133@usc.edu
Date: Feb. 3, 2017

HOMEWORK #1

Issued: 1/13/2017 Due: 11:59PM, 2/5/2017

Problem 1: Basic Image Manipulation (30%)

In this problem, you will conduct a series of simple manipulations on the grayscale and color images to get familiar with image data access, processing and output.

(a) Mirroring, Resizing and Compositing (Basic: 15%)

Motivation

We can use another matrix to represent the modified picture.

For the Dog Mirroring, we can just make the components of the original picture matrix mirrored across the vertical axis and send those entries into the new picture and write it.

For the Dog Resizing question, we can resize the coordinates in the new matrix into the ones in matrix representing the old picture and get the pixel value in the coordinates in the old picture. If the coordinate is not an integer, we can use the formula below to calculate the pixel at such point:

$$F(p',q') = (1-\Delta x)(1-\Delta y)F(p,q) + (\Delta x)(1-\Delta y)F(p+1,q) + (1-\Delta x)(\Delta y)F(p,q+1) + (\Delta x)(\Delta y)F(p+1,q+1)$$

$$\Delta x = p' - p \qquad \Delta y = q' - q$$

In the formula, (p',q') is the coordinates in the old pictures corresponding to the points in the new picture.

For the Image Compositing question, we should leave out the blue background of the dog first. Then, we can put the dog into the specific place at the beach.

Approach and Procedures

For the Dog Mirroring, we can use such transformation below to make the original picture vertical in the new picture:

```
Imagedata1_a_1[3 * (Size1 * i + j) + 0] = Imagedata[3 * (Size1 * i + Size1 - 1 - j) + 0];
Imagedata1_a_1[3 * (Size1 * i + j) + 1] = Imagedata[3 * (Size1 * i + Size1 - 1 - j) + 1];
Imagedata1_a_1[3 * (Size1 * i + j) + 2] = Imagedata[3 * (Size1 * i + Size1 - 1 - j) + 2];
```

The Size1 means the width of the picture, and 0,1 and 2 represent the location of the value of R, G and B. for the question, we can just put the pixel from `Size1 - 1 - j` to `j` to mirror the picture.

For the Dog Resizing question, we should calculate the ratio of new picture to old picture, and then, we can use the ratio to map the pixel points in the new picture to the one in old picture. Finally, use the formula above to get the pixel value at the points of new picture.

For the Image Compositing question, I copy the beach onto the new picture first. Then, I will copy the dog picture onto the beach background by filtering the blue background through such code:

```
if((ImagedataPlus[i][j][0] <= x1)&&(ImagedataPlus[i][j][1] <= x2)&&(ImagedataPlus[i][j][2] >= 255 - x3))
{
}
else
{
    j1 = j + 1100;
```

```
        i1 = i + 400;
        Imagedata1_3[i1][j1][0] = ImagedataPlus[i][j][0];
        Imagedata1_3[i1][j1][1] = ImagedataPlus[i][j][1];
        Imagedata1_3[i1][j1][2] = ImagedataPlus[i][j][2];
}
```

X1 is the value of the filtered pixel of R, x2 is the one of G, 255-x3 is the one of the B. If three values satisfy the condition, we will not copy the pixel value.

Experimental Results

For the Dog Mirroring, it is shown below:

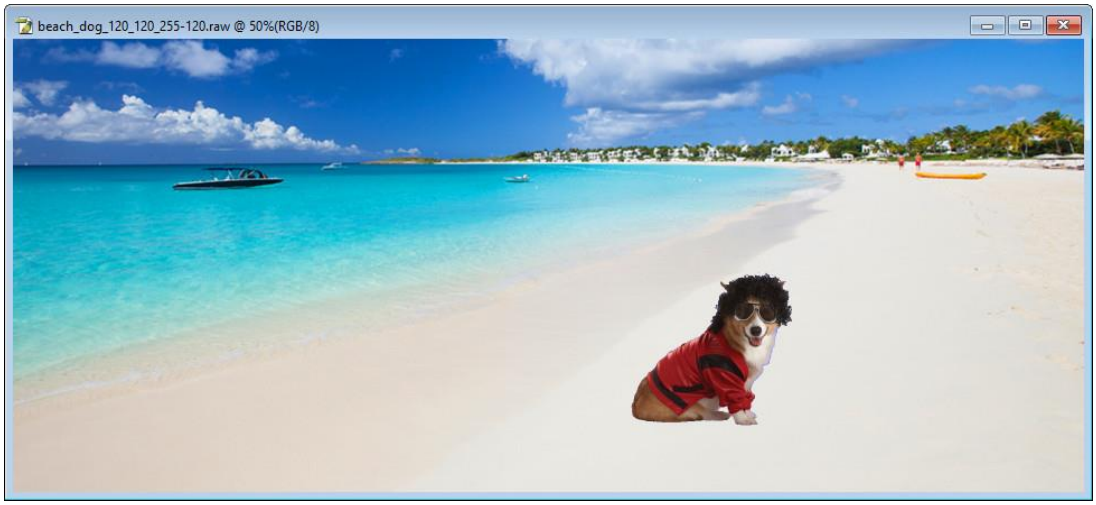
The mirror version of dog.raw

For the Resizing, the pictures are:

The 200x200 version of dog.raw (left) and The 400x400 version of dog.raw (right)

The 600x600 version of dog.raw

For the Image Compositing question, the result is



The beach and dog.raw(remove the color when R<120, G<120, B>135)

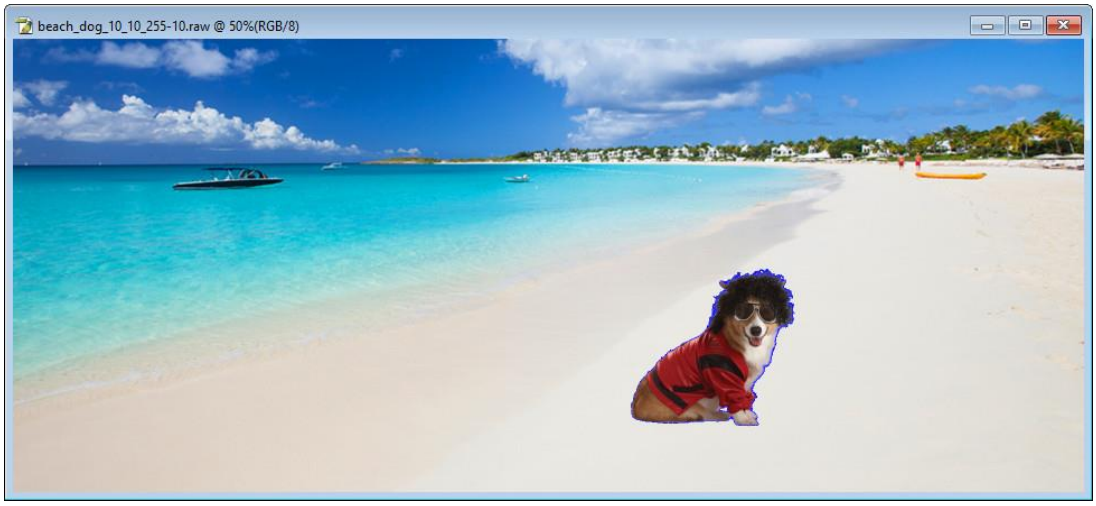
Discussion

In the Resizing questions, we use the bilinear implement to get the pixel value in the new picture, which means that we will get the pixel value through the linear change on the two directions, which will make the boundary un clear. As a result, I want to apply the weight scalars. That is, if the mapping coordinates is close to some point, we can make the pixel value at the point time a larger scale; if not, we will time a small scale. Having done the sum, we can divide it by the sum of the weight scalars.

In the Image Compositing question, we can find that the range of removable color will make a big

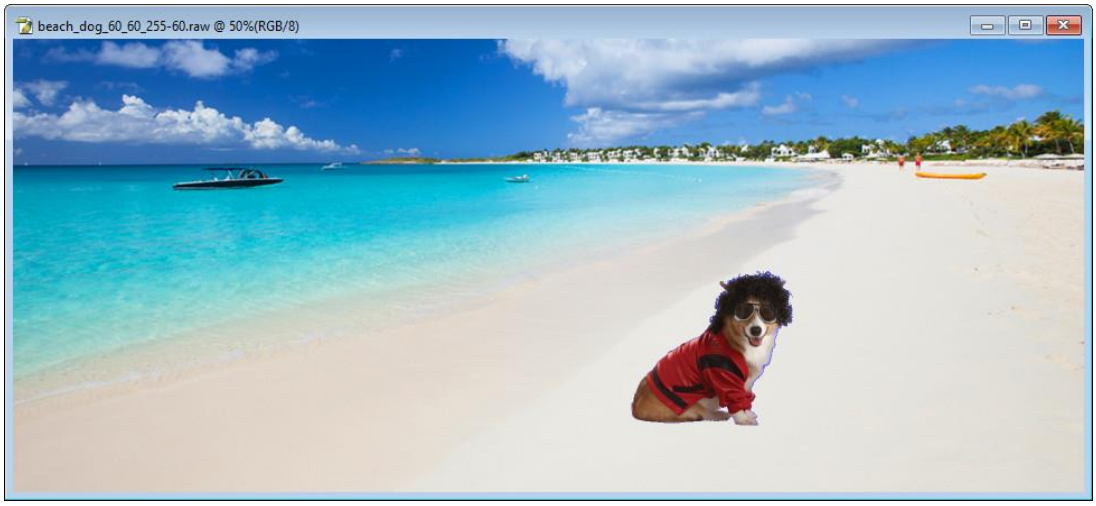
difference since the color of background is not a single color.

If we remove the color when R<10, G<10, B>245, we can find that the blue background cannot be removed completely, which can be shown below:



The beach and dog.raw(remove the color when R<10, G<10, B>245)

If we remove the color when R<60, G<60, B>195, we can find that the blue background will be removed much more than the picture above



The beach and dog.raw(remove the color when R<60, G<60, B>195)

When we remove the color when R<120, G<120, B>135, we will find that the blue can be removed, but some of the margin message is damaged

The beach and dog.raw(remove the color when R<120, G<120, B>135)

When doing the Image Compositing question, the most serious question is the stack overflow. That is, we cannot run the program even though the code is correct since the volume of array is less than the size of the beach.raw. As a result, I decide to make the array saving the picture be global array, whose volume is much larger than the array in main function.

(b) Color Space Transformation (Basic: 15%)

Motivation

For the CMY Color space, we should use the 0 – 255 to depict the value of C, M and Y. First, we use the formula below to get the CMY.

$$\begin{cases} C = 1 - R \\ M = 1 - G \\ Y = 1 - B \end{cases}$$

Because the original range of CMY is from 0 to 1, We should multiply the CMY value with 255.

$$M = \max(R, G, B)$$
$$m = \min(R, G, B)$$
$$C = M - m$$

$$H = \begin{cases} 0 & C = 0 \\ 60\left(\frac{G-B}{C} mod 6\right) & M = R \\ 60\left(\frac{B-R}{C} + 2\right) & M = G \\ 60\left(\frac{R-G}{C} + 4\right) & M = B \end{cases}$$

$$L = \frac{M + m}{2}$$

$$S = \begin{cases} 0 & L = 0 \\ \frac{C}{2L} & 0 < L < 0.5 \\ \frac{C}{2 - 2L} & otherwise \end{cases}$$

Form the HSL color space formula above, we should calculate separately. For H, when $M = R$, the H is from -60 degrees to 60 degrees. However, the H can only be defined from $[0^{\circ}, 360^{\circ}]$, so we should consider when $G - B > 0$ and $G - B < 0$ two situations when M is equal to R. Also, when calculate the H, we should convert it from $[0^{\circ}, 360^{\circ}]$ to $[0, 255]$. For L and S, we should convert them from

$[0,1]$ to $[0,255]$.

For the Sepia filter, we should calculate the I for each pixel first, then use the matrix to calculate the RGB for each pixel. The Sepia filter can be shown below:

$$\begin{bmatrix} S_F(R) \\ S_F(G) \\ S_F(B) \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \begin{bmatrix} I \\ I \\ I \end{bmatrix}$$

$$I = 0.21R + 0.72G + 0.07B$$

Approach and Procedures

For the CMY Color space, we divide the origin RGB by 255 first, then, we do as follow to turn to CMY (Take the R sequence for example):

```
temp = Imagedata[3 * (Size1 * i + j) + 0];
temp = temp / 255;
temp = 1 - temp;
```

Then we will get the grayscale picture for each C, Y and M.

For the HSL Color space, the L and S sequence are easy to calculate, while the H is much difficult, we should discuss situations as the Motivation said (we divide the origin RGB by 255 first)

```
if (C == 0)
{
    Imagedata2_2_temp[i][j][0] = 0;
}
else
{
    if ((M == R) && (G >= B))
    {
        temp = (G - B) / C;
        Imagedata2_2_temp[i][j][0] = 60 * (temp) * ((double)255 / 360);
    }
    else if ((M == R) && (G < B))
    {
        temp = (G - B) / C;
        Imagedata2_2_temp[i][j][0] = (60 * (temp) + 360) * ((double)255 / 360);
    }
    else if (M == G)
    {
        temp = (B - R) / C;
        Imagedata2_2_temp[i][j][0] = (60 * (temp)+120) * ((double)255 / 360);
    }
    else if (M == B)
    {
        temp = (R - G) / C;
        Imagedata2_2_temp[i][j][0] = (60 * (temp)+240) * ((double)255 / 360);
    }
}
```

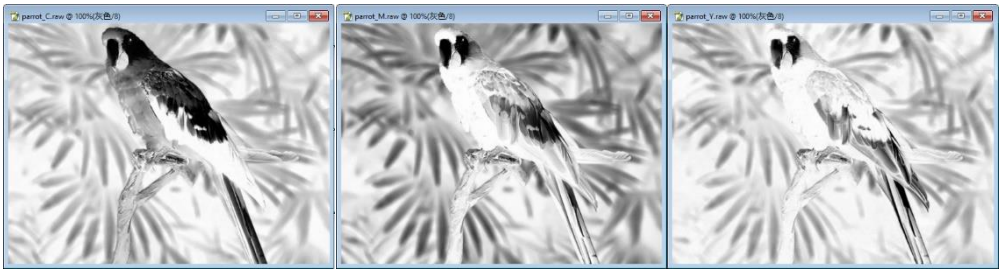
For the Sepia filter, after calculating each I, we will use such formula to calculate the value of R, G and

B, which can be shown below. Also, if the output value is larger than 255, we should convert it to 255:

```
temp01 = (0.393 + 0.769 + 0.189) * (0.21 * temp1 + 0.72 * temp2 + 0.07 * temp3);
if (temp01 >= 255)
{
    Imagedata2_4[i][j][0] = 255;
}
else
{
    Imagedata2_4[i][j][0] = temp01;
}
temp02 = (0.349 + 0.686 + 0.168) * (0.21 * temp1 + 0.72 * temp2 + 0.07 * temp3);
if (temp02 >= 255)
{
    Imagedata2_4[i][j][1] = 255;
}
else
{
    Imagedata2_4[i][j][1] = temp02;
}
temp03 = (0.272 + 0.534 + 0.131) * (0.21 * temp1 + 0.72 * temp2 + 0.07 * temp3);
if (temp03 >= 255)
{
    Imagedata2_4[i][j][2] = 255;
}
else
{
    Imagedata2_4[i][j][2] = temp03;
}
```

Experimental Results

For the CMY images, it is shown below:

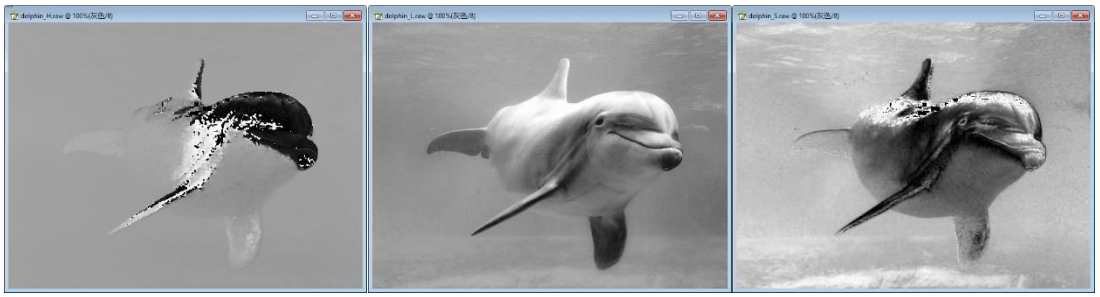


C of the bird.raw (left) M of the bird.raw (middle) Y of the bird.raw (right)

C of the building.raw (left) M of the building.raw (middle) Y of the building.raw (right)

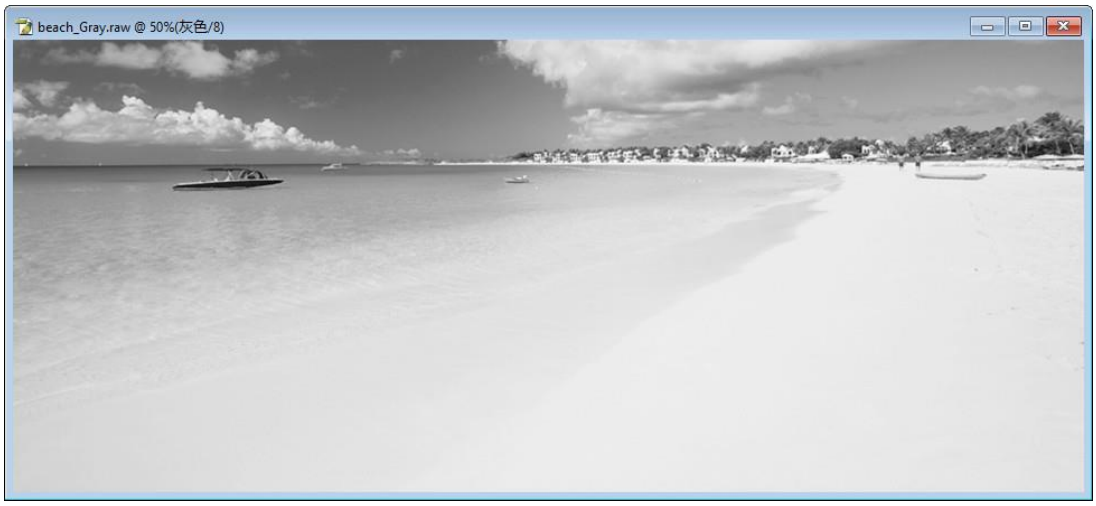
For the HSL images, the result is as follow:

H of the cat.raw (left) L of the cat.raw (middle) S of the cat.raw (right)



H of the dolphin.raw (left) L of the dolphin.raw (middle) S of the dolphin.raw (right)

For the Sepia filter, we can get the grayscale picture and the final output image:



Grayscale of the beach.raw

Final output of the beach.raw

Discussion

From the HSL result, we can find that the H pictures describes the type of the color in the picture; L picture represents the luminance of the picture, the brighter the gray picture is, the brighter this region will be; the S represents the purity of the picture, the brighter the gray image is, the purer the color is.

(c) Layer Blending Mode Implementation (Bonus: 5%)

Motivation

For this question, I choose the Soft Light technique to deal with the figure.

The soft Light of Photoshop can be accomplished through the formula below [1]:

$$f(a,b) = \begin{cases} 2ab + a^2(1-2b) & a < 0.5 \\ 2a(1-b) + \sqrt{a}(2b-1) & otherwise \end{cases}$$

In the formula, $f(a,b)$ represents the output value of pixel, the a is the relative pixel on the bottom, the b is the relative pixel value on the top. For each pixel, we will use the formula to get the soft light effect.

Approach and Procedures

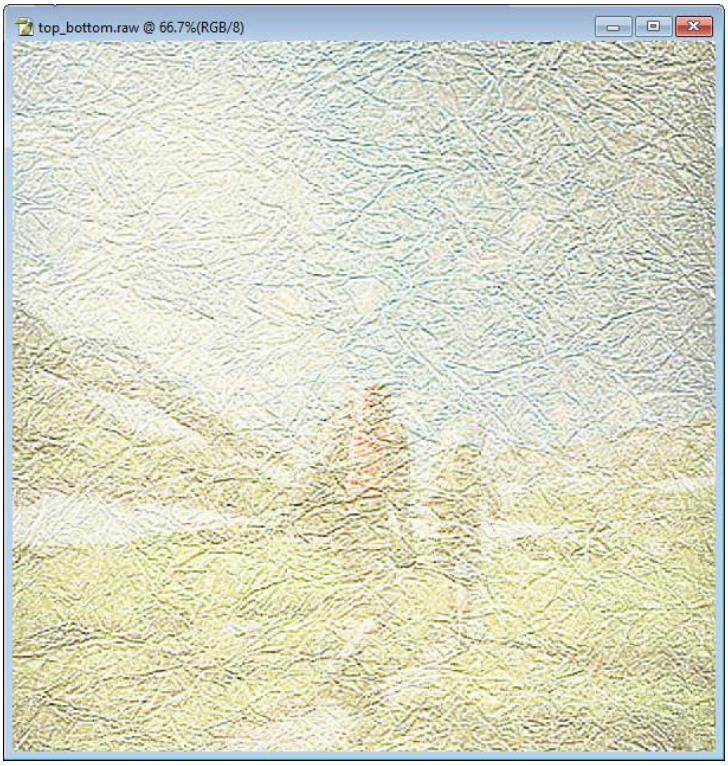
The formula can be written as follows:

```
tempa1 = (float)(Imagedata[3 * (Size1 * i + j) + 0]) / 255;
tempa2 = (float)(Imagedata[3 * (Size1 * i + j) + 1]) / 255;
tempa3 = (float)(Imagedata[3 * (Size1 * i + j) + 2]) / 255;
tempb1 = (float)(Imagedata1_c[3 * (Size1 * i + j) + 0]) / 255;
tempb2 = (float)(Imagedata1_c[3 * (Size1 * i + j) + 1]) / 255;
tempb3 = (float)(Imagedata1_c[3 * (Size1 * i + j) + 2]) / 255;
if (tempb1<0.5)
{
temp1 = 2 * tempa1*tempb1 + tempa1*tempa1*(1 - 2 * tempb1);
Imagedata_3_1[i][j][0] = 255 * temp1;
}
else
{
temp1 = 2 * tempa1*(1 - tempb1) + sqrt(tempa1)*(2 * tempb1 - 1);
Imagedata_3_1[i][j][0] = 255 * temp1;
}
if (tempb2<0.5)
{
temp2 = 2 * tempa2*tempb2 + tempa2*tempa2*(1 - 2 * tempb2);
Imagedata_3_1[i][j][1] = 255 * temp2;
}
else
{
temp2 = 2 * tempa2*(1 - tempb2) + sqrt(tempa2)*(2 * tempb2 - 1);
Imagedata_3_1[i][j][1] = 255 * temp2;
}
if (tempb3<0.5)
{
temp3 = 2 * tempa3*tempb3 + tempa3*tempa3*(1 - 2 * tempb3);
```

```
Imagedata_3_1[i][j][2] = 255 * temp3;
}
else
{
temp3 = 2 * tempa3*(1 - tempb3) + sqrt(tempa3)*(2 * tempb3 - 1);
Imagedata_3_1[i][j][2] = 255 * temp3;
}
```

Experimental Results

For the Soft light images, it is shown below:



The combination of top and bottom.raw

Problem 2: Histogram Equalization (40 %)

(a) Histogram Enhancement for Grayscale Images (Basic: 10%)

Motivation

For the method A, we should build the transfer function, and we will find the area where the pixel values concentrate on first for the image. We can draw the histogram. Having got the histogram, we should convert it into the pdf graph. Then, we build the graph of CDF. From the CDF, we should find the two points whose probabilities are 0.1 and 0.9, which means that the area between two points are where the pixel values concentrate on. We mark the points as f_{max} and f_{min}.

Then, we do the linear transfer function mapping the $[f_{max}, f_{min}]$ to the area of $[255 \times 0.1,\ 255 \times 0.9] = [25, 229]$ by the formula below:

$$y = kx + h$$

$$k = \frac{229 - 25}{f_{max} - f_{min}} \quad h = 229 - kf_{max}$$

In the formula, the y is the output of the pixel value, x is the origin pixel value at the same point.

For the method B, we can use the CDF of the origin image to map the input pixel to a specific probability. Then, we can use the CDF of uniform distribution function $F(x) = x/255$ to map the probability to a new output pixel value.

Approach and Procedures

We can use the code below to get the histogram. The pdf is to divide every element in the histogram matrix by the number of pixel points.

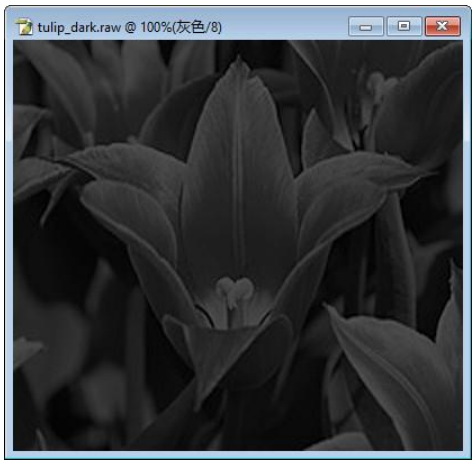
```
for (int i = 0; i<Size2; i++)
{
    for (int j = 0; j<Size1; j++)
    {
        for (int num = -256; num<256; num++)
        {
            if (Imagedata3_a_1[i][j][s] == num)
            {
                his[num + 256] = his[num + 256] + 1;break;
            }
        }
    }
}
```

For the ith CDF values, we can add the value of pdf from 0 to i to get.

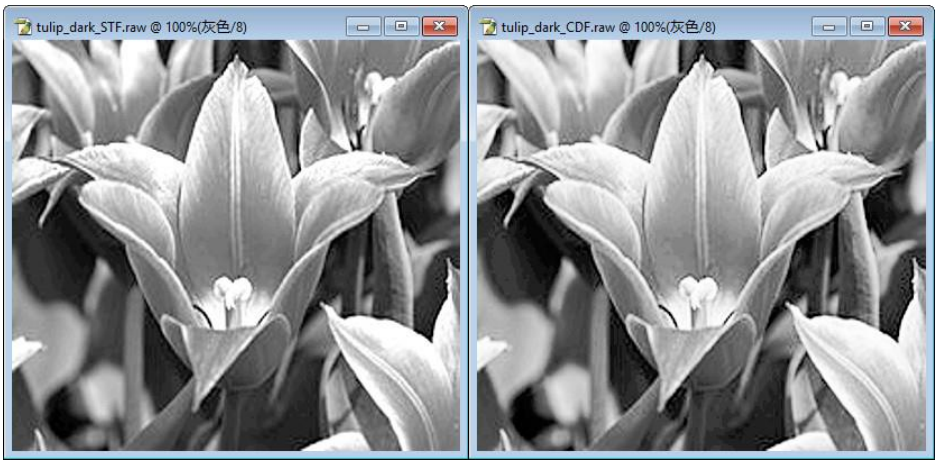
For the method A, I will use the formula above to calculate the parameters of the transfer function first, then man the function to get the desired output of pixel value. If the value is beyond the 255 or below the 0, we well rewrite the output to be 255 and 0.

For the method B, I acquire the CDF output according to the input pixel value. Then, I will make the Probability value time the 255, which is the output pixel value for the point.

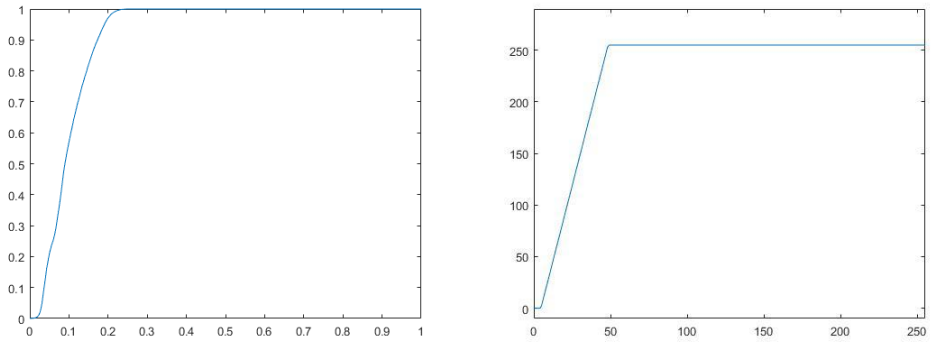
Experimental Results



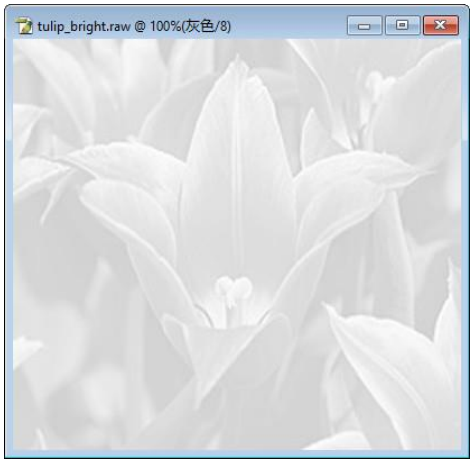
The origin picture of tulip_dark.raw



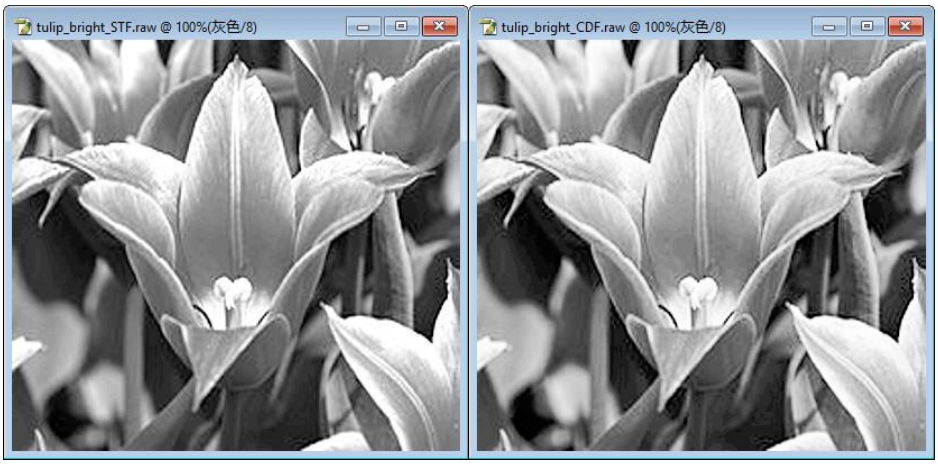
The Method A for tulip_dark.raw (Left) and the Method B for tulip_dark.raw (Right)



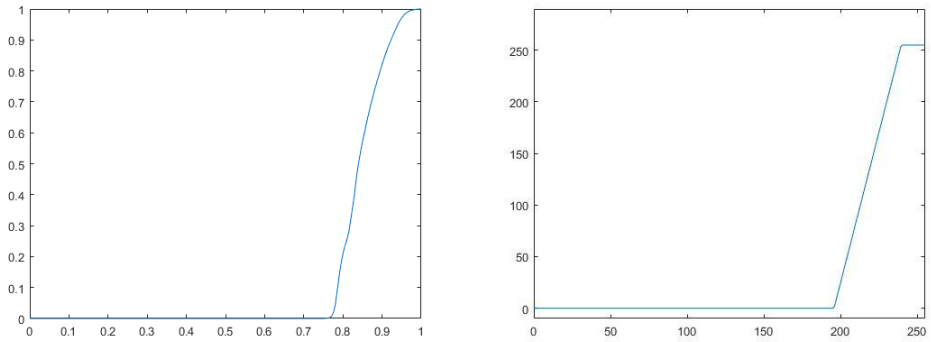
The CDF of the tulip_dark.raw(left) and the transfer function of the tulip_dark.raw(right)



The origin picture of tulip_bright.raw



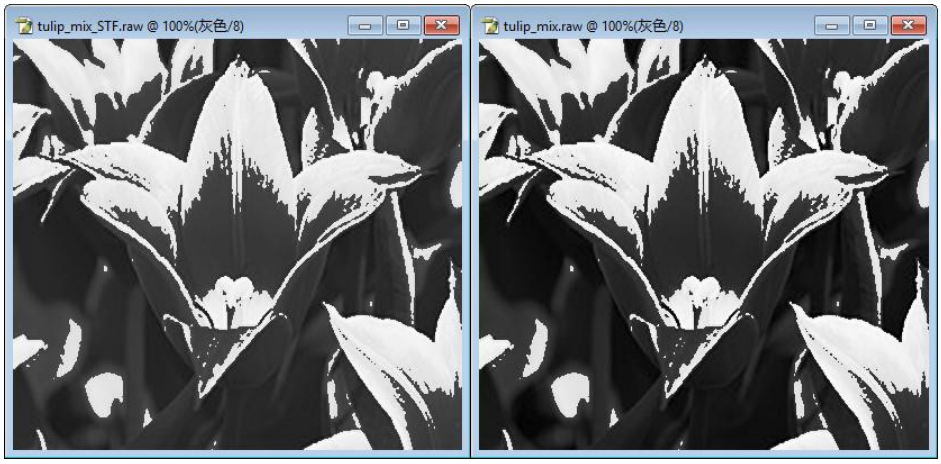
The Method A for tulip_bright.raw (Left) and the Method B for tulip_bright.raw (Right)



The CDF of the tulip_dark.raw(up) and the transfer function of the tulip_dark.raw(down)

Discussion

I have proposed the method A and B to enhance the picture. However, if I apply the method A and B to the image of tulip_mix, I can find the strange things, especially for the transfer function method:



using the conventional transfer function image (Left) and origin image of tulip_mix.raw (Right)

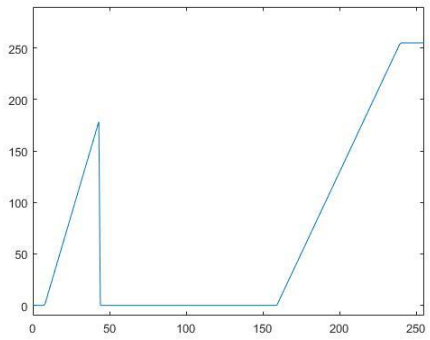
When we compare this with the origin image, we can find that there is no difference between those two images. The reason is that f_{max} is at the brightest point while the f_{min} is at the darkest point since there are two peak values shown in the histogram at a great distance. As a result, we should propose an algorithm to detect the number and location of peaks.

For this algorithm, I will construct two vectors: p_{max} and p_{min} to save the left and right boundary of

the peaks. Firstly, I will save the f_{max} into the end of p_{max} and f_{min} into the beginning of p_{min}. Then, we will check the pixel value from f_{min} to f_{max}. If the pdf becomes zero, we will save the number into p_{max}; If from the latest element of the p_{max}, the CDF has grown 0.05, we will save the number into p_{min}. We will keep on doing this until the f_{max}. Then, we will do the transfer function according to the different region. For example, we can do the transfer function like this:

$$y = \begin{cases} 5.05763x - 38.9783 & 0 \le x < 43 \\ 3.16977x - 503.954 & 0 \le x < 43 \end{cases}$$

And the plot of transfer function is



Therefore, we will use the group of transfer functions to get the output picture which can be shown below, we can see that the enhancement is much better than before:

using the advanced transfer function image of tulip_mix.raw

(b) Histogram Equalization for Color Images (Basic: 16%)

Motivation

For the first part of this question, we should use the methods A and B on each sequence of RGB. Then, we combine three sequences into one image.

For the new method, I think I can use the HSL to convert the RGB to the HSL images, and use the method A and B to enhance the L sequence. Then, I will use a series of algorithms to get the new RGB image.

Approach and Procedures

For the method B, I will use a new way, the Bucket-Filling Method, to get the image with uniform distribution. First, we can get the histogram of the pixel value for all points. Then, we can rearrange the number of points with specific pixel values to be same. At last, we will get the image whose pixel values are uniform distribution (the code can be shown below).

```
long stand;
```

```
stand = (Size1 * Size2) / 256;
int i0 = 0;
int n = 0;
for (int num = 0; num<256; num++)
{
     for (int i = 0; i<Size2; i++)
     {
          for (int j = 0; j<Size1; j++)
          {
               if (Imagedata[BytesPerPixel * (Size1 * i + j) + s] == num)
               {
                    if (i0<stand)
                    {
                         i0++;
                    }
                    else
                    {
                         n++;
                         i0 = 1;
                    }
                    if (n >= 255)
                    {
                         Imagedata2_a[i][j][s] = 255;
                    }
                    else
                    {
                         Imagedata2_a[i][j][s] = n;
                    }
               }
          }
     }
}
```

For the new method, we will use the algorithm above to get the HSL sequence of the image. Then, we can use either method A or method B to enhance the L sequence. Then, I will use the algorithm below to turn the HSL to be RGB image.

$$q = \begin{cases} l \times (1+s) & l < 0.5 \\ l + s - (l \times s) & l \geq 0.5 \end{cases}$$

$$p = 2 \times l - q$$

$$h_k = \frac{h}{360}$$

$$t_R = h_k + \frac{1}{3}$$

$$t_G = h_k + \frac{1}{3}$$

$$t_B = h_k - \frac{1}{3}$$

$$if \;\; t_C < 0 \rightarrow t_C = t_C + 1.0 \;\; for \;\; each \;\; C \in \{R, G, B\}$$

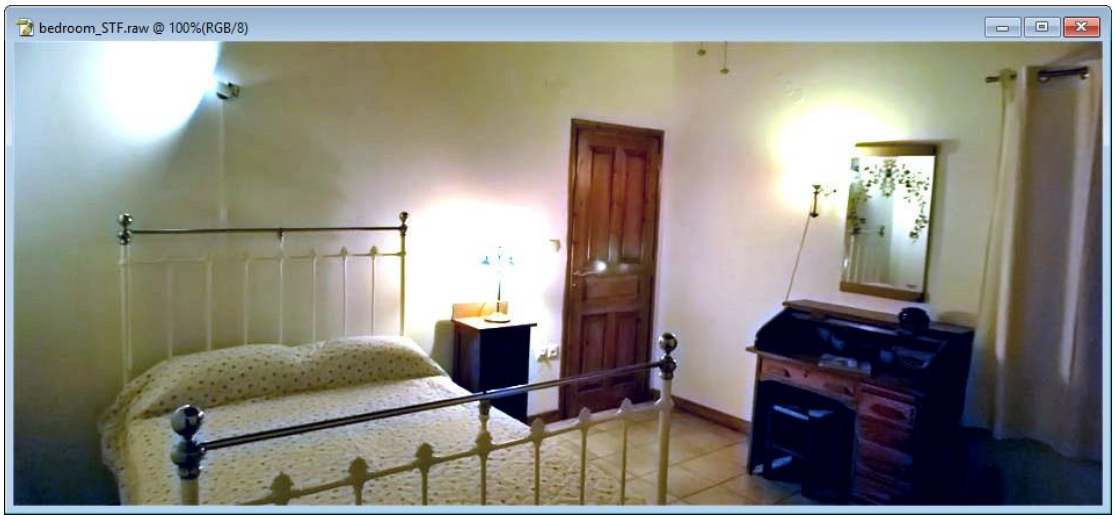
$$if \;\; t_C > 1 \rightarrow t_C = t_C - 1.0 \;\; for \;\; each \;\; C \in \{R, G, B\}$$

For each color $C \in \{R, G, B\}$, we can get that

$$C = \begin{cases} p + \big((q-p) \times 6 \times t_C\big) & t_C < 1/6 \\ q & 1/6 \le t_C < 0.5 \\ p + \big((q-p) \times 6 \times (2/3 - t_C)\big) & 0.5 \le t_C < 2/3 \\ p & otherwise \end{cases}$$

Experimental Results

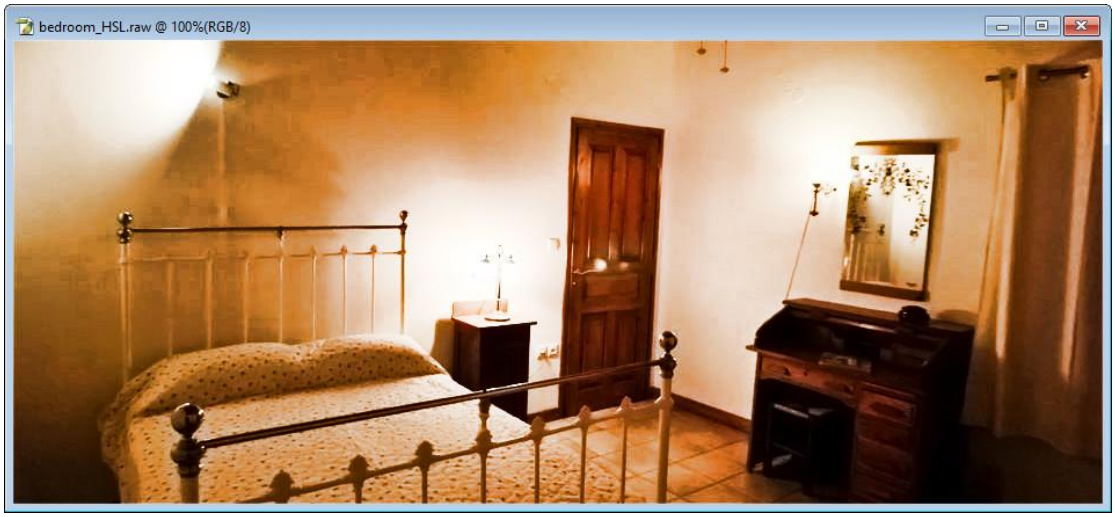
For the first part of question, we can get the image as follow:



Method A for the bedroom.raw

Method B for the bedroom.raw

When applying the new method, we can get new results, which can be shown below:



Use the method B for H sequence bedroom.raw

Discussion

When use the method A and B, we can make the whole RGB expanded and the image can be bright. However, we will get an image with different kinds of color. The most impressive point is that the dark zone will become blue. For the new method, however, we can make the kind of color and the purity of the color unchanged and only expand the lightness of the color, which make the image realer than before compared with the origin image.

(c) Histogram Transform (Advanced: 14%)

Motivation

For the method B, we will make the pixel value be uniformly distributed. However, in this question, we should make the histogram of the pixel value matches the Gaussian function with specific parameters. As a result, we should do the revision for the desired PDF and CDF.

The pdf of the Gaussian is sown below (in the question, μ= 70 and σ = 20):

$$p\left(\frac{x}{\mu}, \sigma\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Approach and Procedures

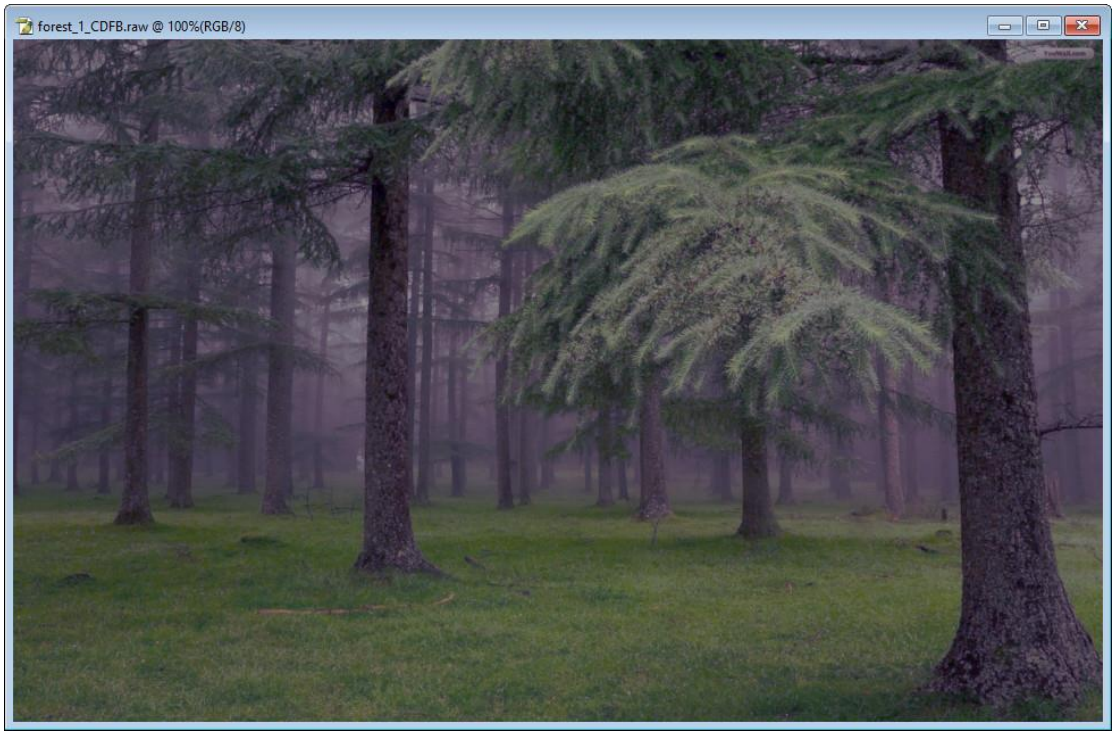
At this time, I will use the Bucket-filling algorithm (which is proposed in the Method B of (b)) to get the picture with desired histogram. We should use the formula of the Gaussian function above to get the relative relation of the number of the pixel points in each bucket. We will use the formula below to get them:

$$N(pixel\ \ value = k) = \frac{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(k-\mu)^2}{2\sigma^2}\right)}{\sum_{s=0}^{255} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(s-\mu)^2}{2\sigma^2}\right)} \times N(total\ \ number\ \ of\ \ pixel\ \ points)$$

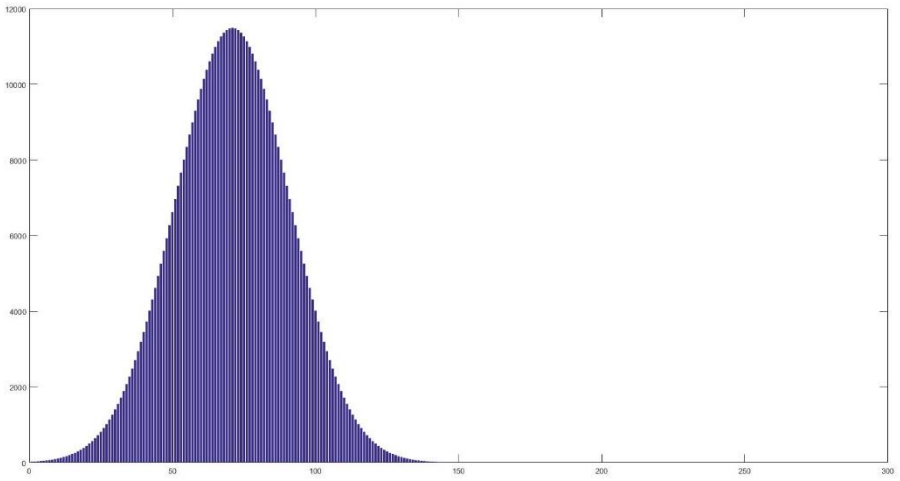
As a result, we can place each pixel points into specific buckets with the growing order of the pixel value, then get the picture what we want.

Experimental Results

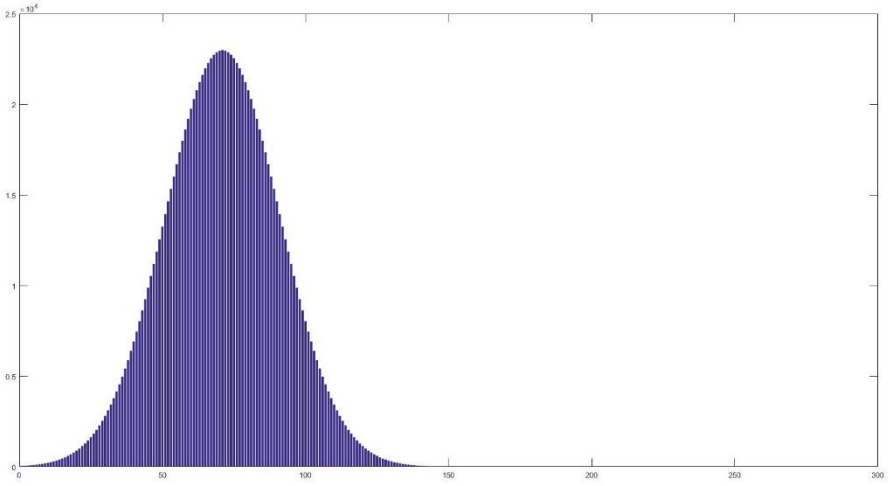
We can get the histogram and the output image below:



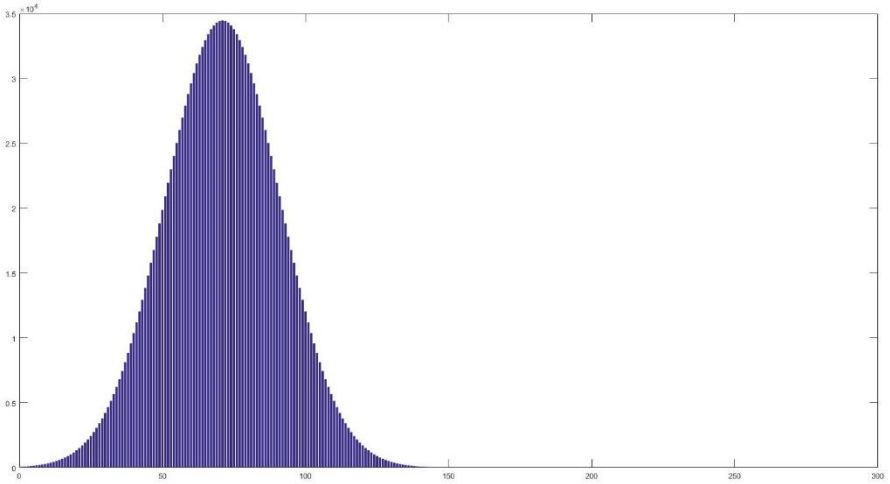
The transferred image of forest_1.raw



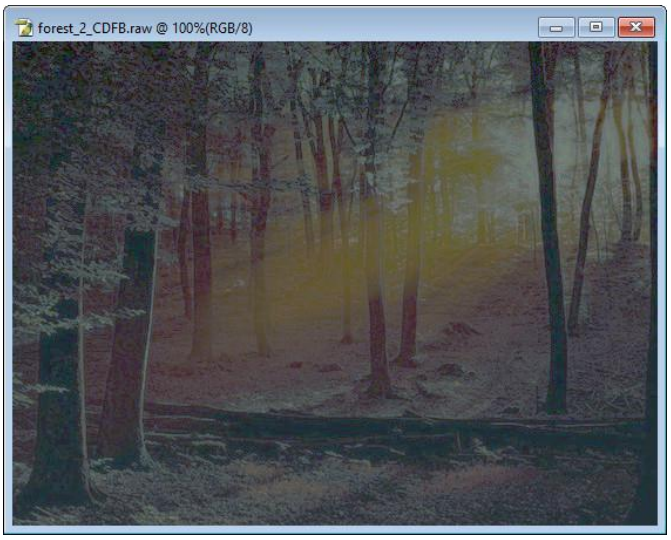
The histogram of R of forest_1.raw



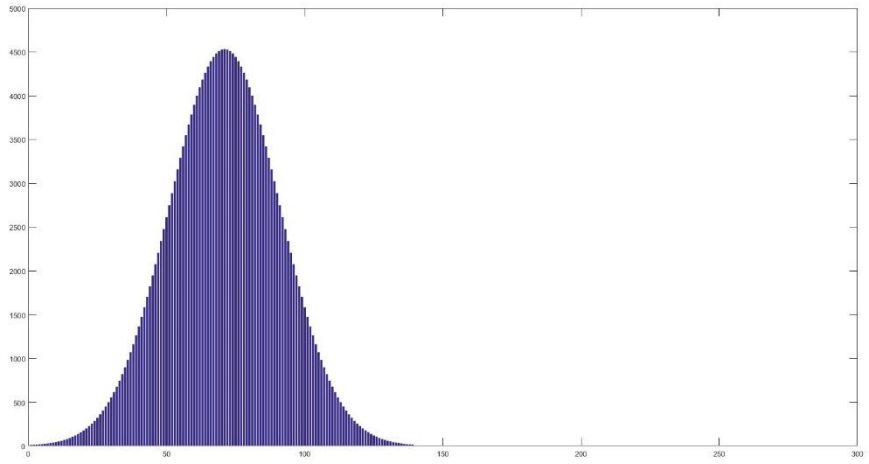
The histogram of G of forest_1.raw



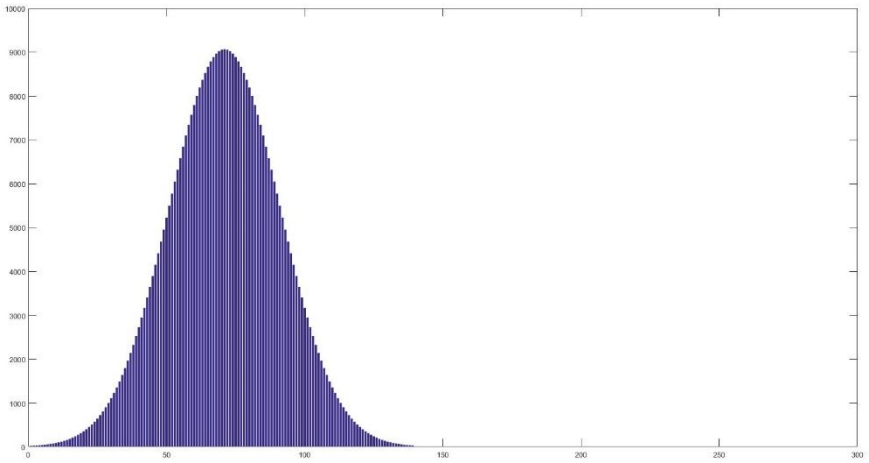
The histogram of B of forest_1.raw



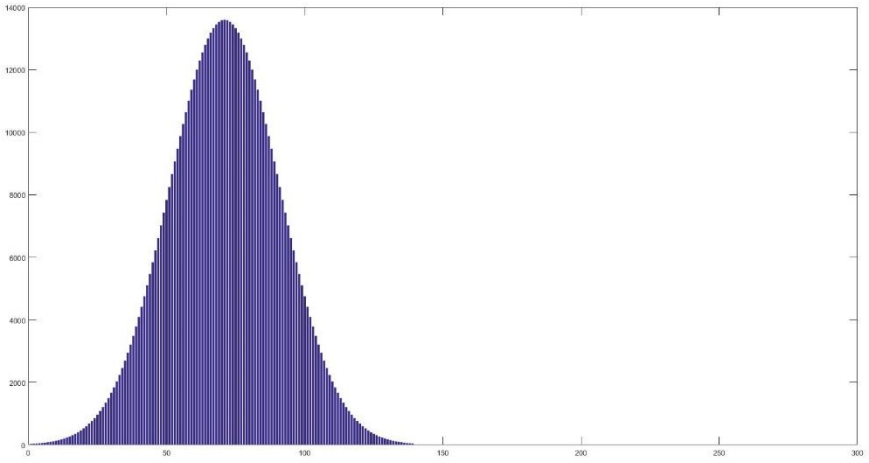
The transferred image of forest_2.raw



The histogram of R of forest_2.raw



The histogram of G of forest_2.raw



The histogram of B of forest_2.raw

Discussion

Because I apply the Bucket-filling algorithm. The histogram for each RGB will be the histogram with

the same standard deviation 20 and mean 70. After such transform, the RGB value will be 70 on average, which means that the image will be dark if the pixel value of origin image is higher than the one of new image.

Problem 3: Noise Removal (30 % + 10 %)

(a) Mix noise in color image (Basic: 15%)

Motivation

In this question, the most important question is to decide the type of the noise. In general, there are two types of noise: random noise and impulse (salt and pepper noise). As a result, we should detect the noise first. Having known the type of the filter, we can decide which filter we should use.

Approach and Procedures

At first, we should detect the type of noise in each RGB sequences. For each sequence, we should decide if there is impulse noise which can make the pixel value be 0 or 255 firstly. In fact, we can use the outlier detection to judge if there is impulse noise since this noise is likely to make the pixel inconsistent. The outlier detection can be shown as follow:

$$\begin{matrix} X_1 & X_2 & X_3 \\ X_4 & X & X_5 \\ X_6 & X_7 & X_8 \end{matrix}$$

The pixel at point X and its neighbor.

Then, we should calculate as follow:

$$\Delta = \left| \frac{1}{8} \sum_{i=1}^{8} X_i - X \right|$$

If the Δ is larger than a constant (in this situation, I choose 20 since the standard deviation of noise is less than 15), we can regard it as an impulse noise pixel. Having detected the pixel, we can remember the number of such pixels and change the value of X with origin pixel value at the same point.

Then, we can detect the random noise. For each pixel, we calculate as follow:

$$\Delta x = |X_{noise} - X_{ori}|$$

We can find that the range of Δx is between -255 and 255. Then, we can do the histogram for those Δx. From the shape of histogram, we can get the type of noise; we can get the mean and deviation by calculating the average and standard deviation of Δx.

Having got the parameters, we can choose the filters. However, we should build those kind of filters.

Average filter

For the picture, we should extend the edge of the image first. Then, we can use the formula below to calculate the value of point at the center point (I take the 3x3 for example):

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian filter

Also, we should extend the edge of the image first. Then, we should use the pixel around times the weight number and add them to get the pixel value at the center point. The weight number of each pixel can be shown as follow (the σ is a constant and take the 3x3 Gaussian filter as example):

$$\frac{1}{\left(4 \times e^{-\left(\frac{2}{\sigma^2}\right)} + 4 \times e^{-\left(\frac{1}{\sigma^2}\right)} + 1\right)} \times \begin{bmatrix} e^{-\left(\frac{2}{\sigma^2}\right)} & e^{-\left(\frac{1}{\sigma^2}\right)} & e^{-\left(\frac{2}{\sigma^2}\right)} \\ e^{-\left(\frac{1}{\sigma^2}\right)} & 1 & e^{-\left(\frac{1}{\sigma^2}\right)} \\ e^{-\left(\frac{2}{\sigma^2}\right)} & e^{-\left(\frac{1}{\sigma^2}\right)} & e^{-\left(\frac{2}{\sigma^2}\right)} \end{bmatrix}$$

Median filter

In this case, I will take the five-point median filter as example.

If I want to calculate the pixel by this filter, I should use the pixel at this point and at the point direct above, below, left and right. Then, for those five points, we should compare the pixel value and get the third largest pixel value and it is the value at the point.

Bilateral filter

For this question, we can scan the pixel points in a $n \times n$ square whose center point is where we want to get the value. Then, we can get the pixel value use the formula below (point (i,j) is the center point of square, point (k,l) is the points in the square, $I(i,j)$ is the pixel value of (i,j)):

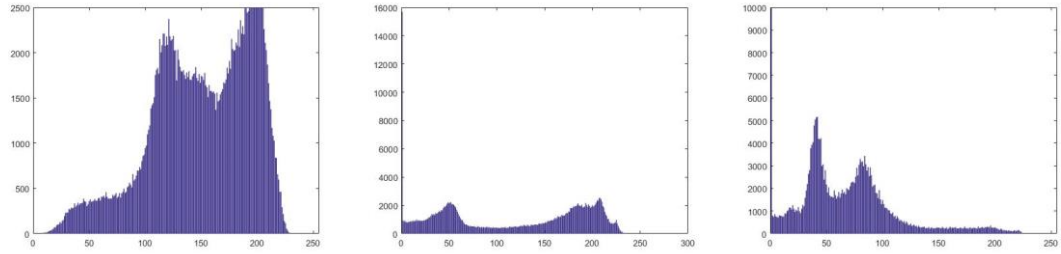
$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}\right)$$

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

Experimental Results

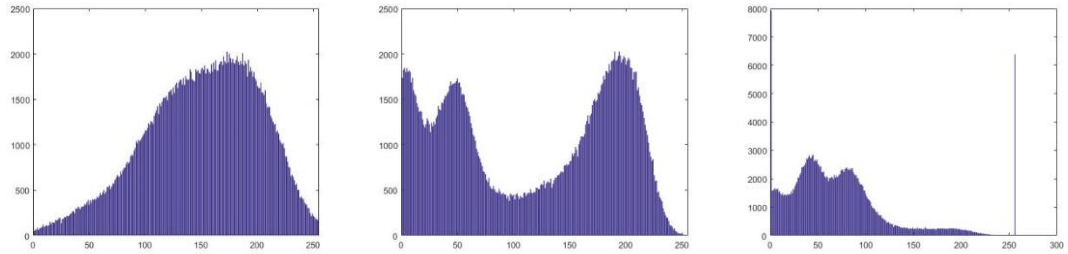
The histogram of R, G and B of the pepper picture

Origin picture



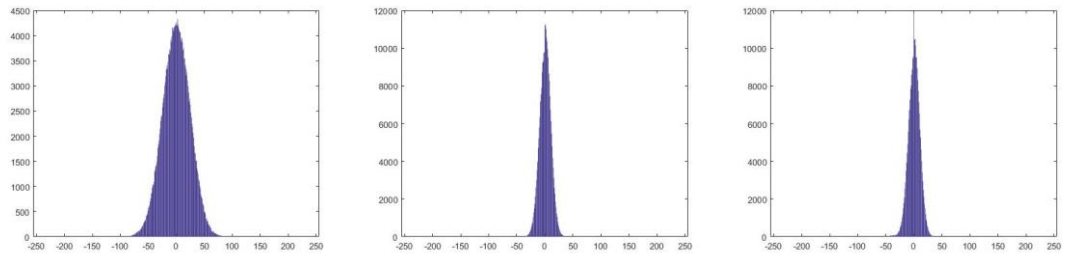
The histogram of R (Left), G(Middle) and B (Right) for pepper.raw

Noisy picture



The histogram of R (Left), G(Middle) and B (Right) for pepper_noisy.raw

From the picture, we can see that for the B sequence, it has impulse noise. Then, we will draw the histogram of random noise



The noise histogram of R (Left), G(Middle) and B (Right) for pepper_noisy.raw

And we can calculate the deviation and mean of random noise, we can find that

For R sequence, $mean = -0.187 \quad \sigma = 12.5772$

For G sequence, $mean = 0.6825 \quad \sigma = 12.5444$

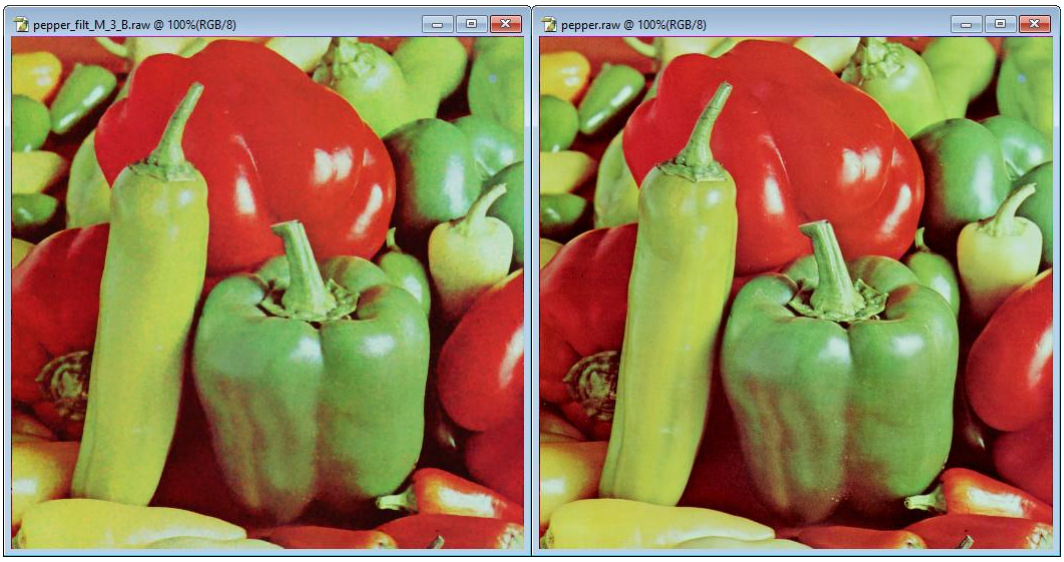
For B sequence, $mean = 0.4386 \quad \sigma = 12.5536$

For this result, we will use the Median Filter to deal with the impulse noise in the B sequence; then, we will use the same filters to delete the random noise in three sequences.

After experimenting, we find that the two Median filters cascaded with one Bilateral filter is a good choice. That is, for each sequence, I use a Median filter. And then, I use the Median filter again. Finally, I use the Bilateral filter.

For the parameter of the Bilateral filter, I use the 24×24 square window to calculate the pixel value of the center, and $\sigma_d = 15$, $\sigma_r = 13$.

The PNSR of the R, G and B is 30.6886, 28.0852 and 30.4469. Compared with the noisy image whose PSNR of RGB are 20.3157, 28.2002 and 17.8403, we can find that the R and B get a significant improvement in quality while there is little change in G channel. The output picture can be shown below:

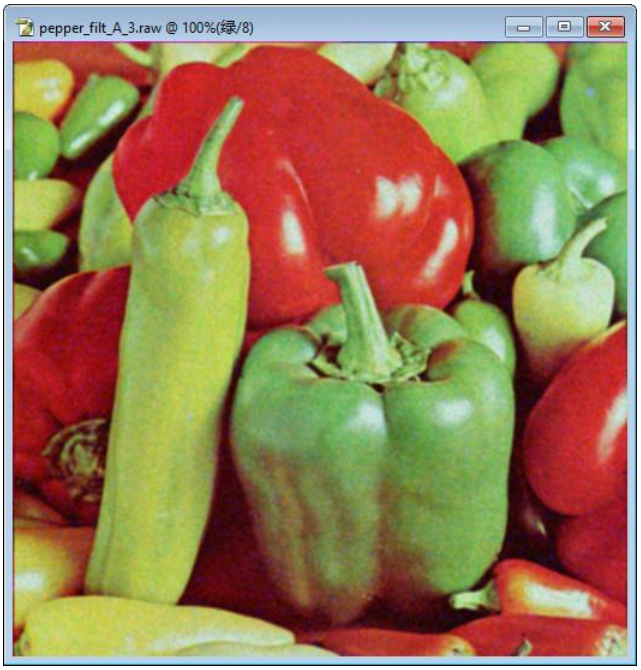


The denoised image (Left) and origin image (Right) of pepper.raw

From the comparison of two pictures, we can see that the noise has been suppressed deeply. In addition, the edge in the picture, such as the red-pepper edge, yellow-pepper edge, has been reserved.

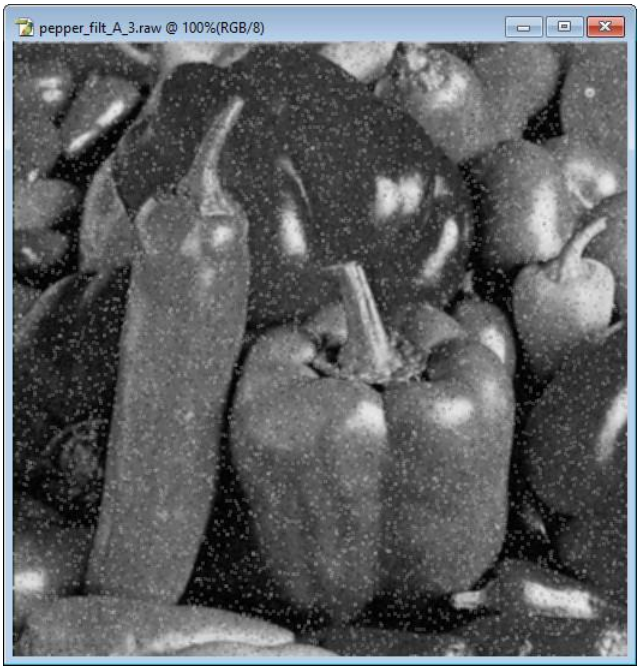
Discussion

When choosing the type of filter, we can find that the type of the filter can make a large difference for the denoising effect. For example, if we use the average filter without cascading with Median filter, we can get the result:



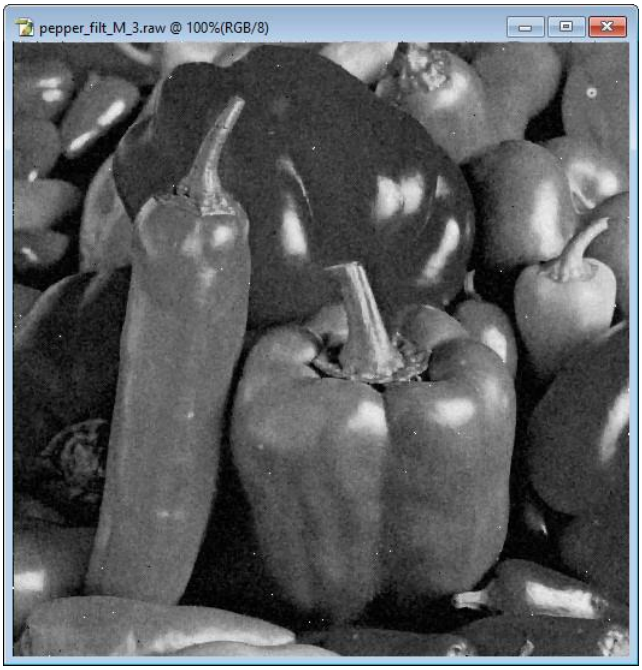
The denoised image by average filter of the pepper.raw

Compared with the original picture above, we can find that the random noise has not been filtered completely. In addition, when we see the B sequence, we can find that:



The denoised image by average filter of the pepper.raw (B sequence)

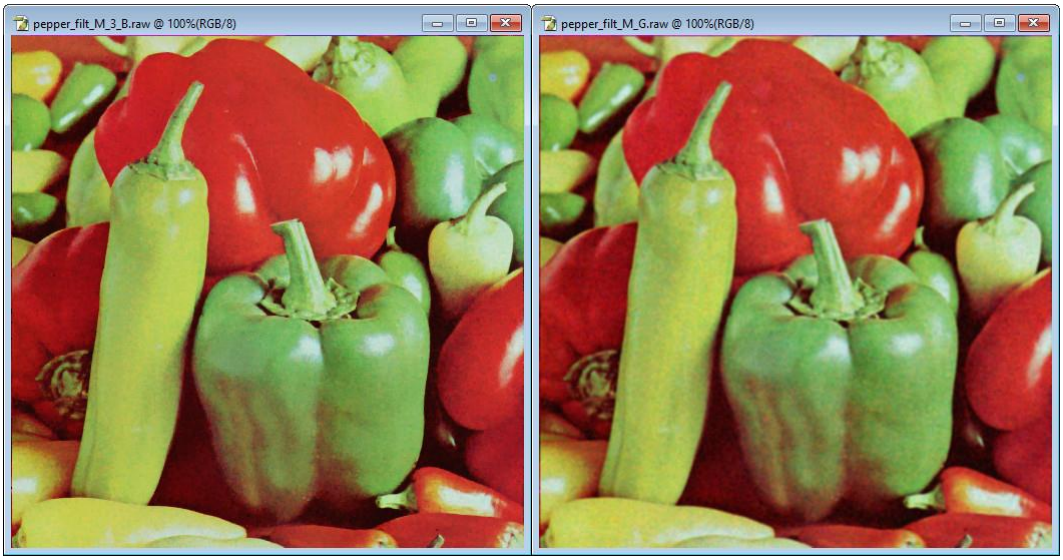
The impulse noise just suppressed. However, the pixel value is still inconsistency. Therefore, I think for the impulse noise, Median filter is really effective than the average filter, which can be shown below:



The denoised image by Median filter of the pepper.raw (B sequence)

In the picture, the Median is an effective solution to filter the pepper noise and reserve the edge of the picture.

Also, for the denoising of the random noise, the Bilateral filter is really effective, we can compare the output of Gaussian filter and Bilateral Filter (both are cascaded with Median filter before)



The denoised image by Bilateral filter (Left) and image by Gaussian (Right) of pepper.raw

From this comparison, we can find that the Bilateral filter can make the image clear when denoising the noise, while the Gaussian filter cannot.

The non-programming question

Question 1

(1) No. From the result above, we can see that all R, G and B have Gaussian random noise and the variation and mean are similar. However, for the B sequence, it has impulse noise.

(2) I think we should perform filtering on individual channels separately. Because different filter can deal with different noise as I have shown above. For example, Median filter can filter the pepper

noise; Gaussian filter can filter the random noise. In addition, the type of noise for different sequence is totally different. Therefore, we should use different filter according to the different type of the noise.

(3) I would like to use the Gaussian filter to remove the mixed noise because it can remove the both the Impulse noise and random noise effectively, which can be shown below (size is 3x3):

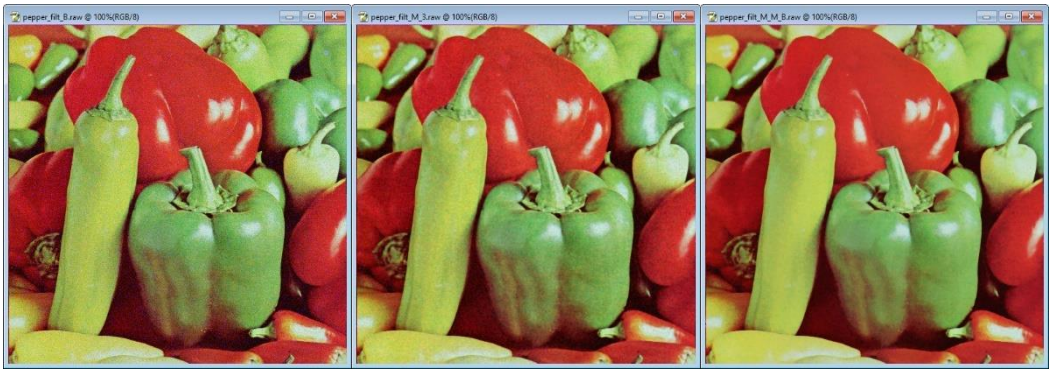
PSNR	R (Random)	G (Random)	B (Random + Impulse)
Average	27.483	28.995	25.1308
Median	24.645	30.3267	29.3284
Gaussian	27.5445	29.6262	25.1544
Bilateral	20.9304	29.5748	17.6354

From the table, we can see that Bilateral and Average is not useful for the impulse noise; the median sometimes cannot be useful for the Random Noise; only the Gaussian can make both the Random noise and Impulse suppressed evenly.

(4) I can cascade the filters. In this section I will use s series of cascade combination to check their performance of PSNR

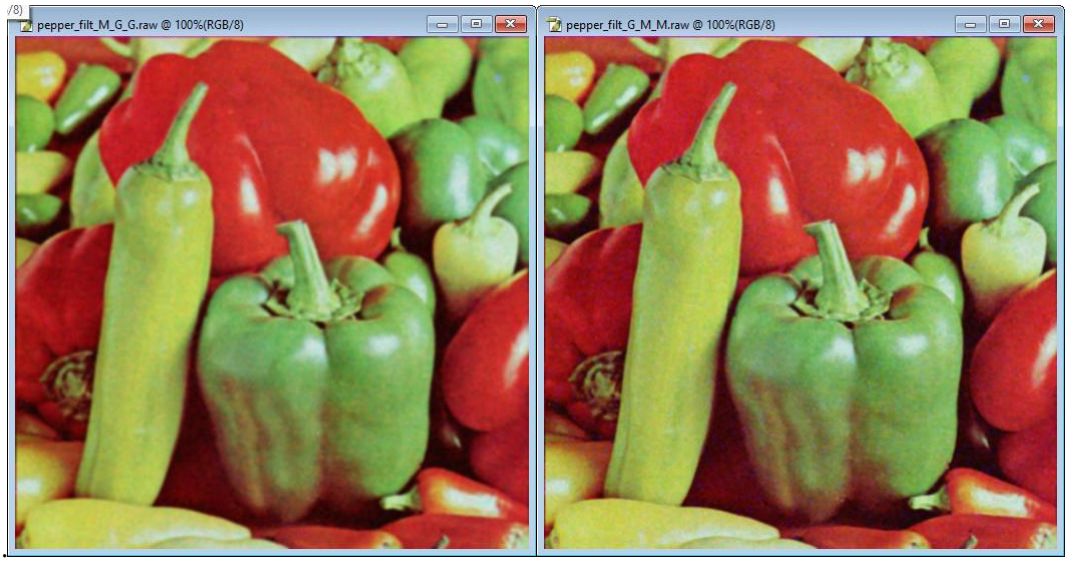
Channel/Combination	R	G	B
Non	20.3157	28.2002	17.8403
3x3 Gaussian	27.483	28.995	25.1308
3x3 Mean	27.5538	29.3937	25.1881
5-points Median (+)	24.645	30.3267	29.3284
3x3 Median + 3x3 Median	26.7131	31.4283	30.7314
3x3 Median + 3x3 Mean	27.9158	29.2039	30.044
3x3 Mean + 3x3 Mean	28.308	28.1347	26.4627
3x3 Median + 3x3 Mean + 3x3 Mean	28.1705	27.9072	29.4459
3x3 Mean + 3x3 Median + 3x3 Median	28.316	29.3466	25.8522
3x3 Median + 3x3 Median + 3x3 Bilateral	28.975	28.137	30.7018

We can see that we can get better performance than single filter if we cascade the filter correctly. For example, I cascade two Median filters and one Bilateral filter to deal with the pepper image. Although there is shortcoming for both of them, when combining together, they can get really good PSNR results, which are 28.0852 (R), 30.6886 (G), 30.4469(B). (Images are shown below)



The Bilateral (Left), Median (Middle) and cascade of two filters (Right)

In addition, we can find that the order of the filters is really important. For example, if we change the order of the median and Gaussian filter, it will leave different result on the sequence with the impulse noise (B). Therefore, for the sequence with the impulse noise, we should place the median filter first, then other kinds of filters (which can be shown below)



Median + Gaussian + Gaussian (Left) and Gaussian + Median + Median (Right)

(5) For the Bilateral filter, the difference of the window size means the different calculation speed and different filtering quality.

We can get the table about the time and RSNR of the image for the different searching window:

L×L	PSNR (R)	PSNR (G)	PSNR (B)	Duration (s)
24	28.137	30.7018	28.975	3.893
48	28.0801	30.6116	29.5688	13.064
72	28.0852	30.6886	30.4469	27.566
96	28.117	30.8415	30.7434	42.877

Therefore, for the Bilateral filter, the larger the size window is, the higher the PSNR is, while the duration will be longer significantly.

However, for the other filters, there will be difference. Take the Mean, Median and Gaussian for example

Channel/Combination	R	G	B
3x3 Gaussian	27.483	28.995	25.1308
5x5 Gaussian	27.8914	26.6049	26.3032
7x7 Gaussian	27.1198	24.8736	25.992
3x3 Mean	27.5538	29.3937	25.1881
5x5 Mean	28.4039	27.8946	26.5404
7x7 Mean	28.3943	27.3177	26.7134
5-points Median (+)	24.645	30.3267	29.3284
9-points Median (+)	26.418	30.4527	30.2457
13-points Median (+)	26.9948	29.9181	29.9899

Therefore, the change of quality is not sure with the change of the size of the window for the Mean, Median and Gaussian filter.

Question 2

(1) It has been discussed in the section of result and discussion

(2) Even though we can get good PSNR, we have to spend 27s on the question, we should consider methods to decrease the time of running.

(3) From the statistic above, we can use smaller window to get faster speed with little decrease of the PSNR. In addition, we can set different parameters of the filter for different variation and mean of

the same type of noise in order to improve the PSNR for each sequences.

(b) Non-local means (NLM) filter (Advanced: 15%)

Motivation

For this question, we should finish two main objectives. Firstly, we should build the filter according to the theory of the Non-local Mean filter; Secondly, we should test the different parameters for the Non-local mean filter.

Approach and Procedures

In the theory, we should get the pixel in a point by calculating the deviation between the gray level vector at the pixel point and the all gray level vectors in the image in order to get the weight number and acquire the pixel value at the point. However, we can just use a specific zone to search for the gray pixel vector. For example, we can use the areas around the pixel point whose pixel value we want to get. We can call the window as searching window. For the gray level vector, we can use the 3×3 or 9 components around the pixel point to form the vector.

Then, we can use the Gaussian-weighted Euclidean distance to get the distance of two vectors. Take the 9-vector as example. If the central vector is $[N(v_{i0}) \quad N(v_{i1}) \quad \cdots \quad N(v_{i8})]$ (the center point is $N(v_{i4})$), the searched vector is $[N(v_{j0}) \quad N(v_{j1}) \quad \cdots \quad N(v_{j8})]$ (the center point is $N(v_{j4})$), the Weighted Euclidean distance can be written as

$$\|v(i) - v(j)\|^2 = \sum_{k=0}^{8} w_k \|N(v_{ik}) - N(v_{jk})\|^2$$

In the equation, the weight number w_i is depended on the distance between the v_{ik} and the center point v_{i4}, which can be depicted as (if it is Gaussian weight number):

$$w_k = \frac{\exp\left(-\frac{(x_{v_{ik}} - x_{v_{i4}})^2 + (y_{v_{ik}} - y_{v_{i4}})^2}{2\sigma^2}\right)}{\sum_{k=1}^{9} \exp\left(-\frac{(x_{v_{ik}} - x_{v_{i4}})^2 + (y_{v_{ik}} - y_{v_{i4}})^2}{2\sigma^2}\right)}$$

If it is average weight number, all the $w_i = 1/9$.

Having got the Euclidean distance, we can get the pixel value for the point v_{i4}.

$$N(v_{i4}) = \frac{1}{Z(i)} \times \sum_j N(v_{j4}) \exp\left(-\frac{\|v(i) - v(j)\|^2}{2h^2}\right)$$

$$Z(i) = \sum_j \exp\left(-\frac{\|v(i) - v(j)\|^2}{2h^2}\right)$$

For the next part, the change of parameters, I decide to change the following parameters: the size of the gray level vector, the value of h, and the weight number.

In theory, the Non-Local Mean cannot be useful to filter the impulse noise. Therefore, I decide to propose a new approach. At first, I leave out the central point whose pixel is impulse noise when scanning in the large searching window. In addition, I skip the central points which is one of elements in central gray level vector when scanning in the searching window, which makes the filter use the pixel value not relevant to the local pixel value. The core code is shown below (for every points in searching window):

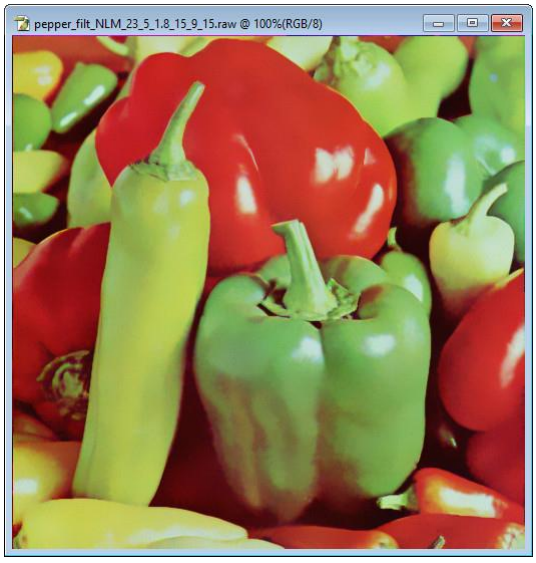
```
double avg;
avg = Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17 - 1) + (j0 + 17 - 1)) + s] + Imagedataex[BytesPerPixel * ((Size1 +
```

```
34) * (i0 + 17 - 1) + (j0 + 17)) + s] + Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17 - 1) + (j0 + 17 + 1)) + s] +
Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17) + (j0 + 17 - 1)) + s] + Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 +
17) + (j0 + 17 + 1)) + s] + Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17 + 1) + (j0 + 17 - 1)) + s] +
Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17 + 1) + (j0 + 17)) + s] + Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 +
17 + 1) + (j0 + 17 + 1)) + s];
avg = avg / 8;
if (abso(i0 - i) <= 1 && abso(j0 - j) <= 1)
{


}
else
{
      if ((Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17) + (j0 + 17)) + s] == 255 || Imagedataex[BytesPerPixel *
((Size1 + 34) * (i0 + 17) + (j0 + 17)) + s] == 0) && (abso(avg - Imagedataex[BytesPerPixel * ((Size1 + 34) * (i0 + 17) + (j0 +
17)) + s]) > 40))//
      {


      }
      else
      {
            //insert the processing code
      }
}
```

Results

One of the good result for the pepper image can be shown below: (h for R is 15, for G is 9, for B is 15, average filter, vector size: 5x5, searching window: 23x23)

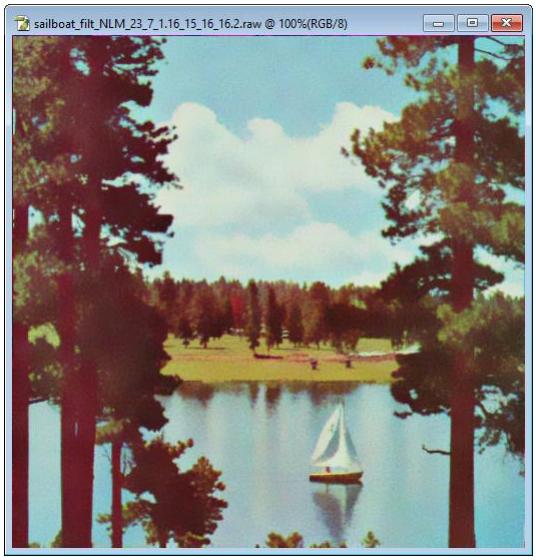
The PSNR of R is 29.29. The PSNR of G is 30.38. The PSNR of B is 28.34.



The denoised image by Non-Local Mean filter of pepper.raw

One of the good result for the sailboat image can be shown below: (h for R is 15, for G is 16, for B is 16.2, average filter, vector size: 7x7, searching window: 23x23, Gaussian kernel $\sigma = 1.12$)

The PSNR of R is 27.43. The PSNR of G is 25.85. The PSNR of B is 26.20.



The denoised image by Non-Local Mean filter of sailboat.raw

Discussion

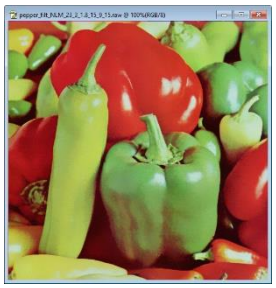
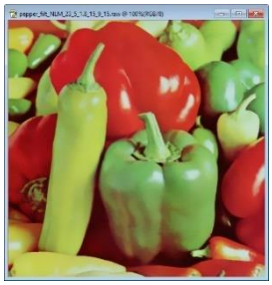
a) Pepper image

For the pepper image, we should discuss the effect of the parameters. In this report, I will analyze the size of the gray level vector, the value of h, the weight number and the size of searching window.

(1) size of the gray level vector

For the question, we use three sizes: 3x3, 5x5 and 7x7, search window is 23x23

At first, without changing other parameters, we can get the PSNR, simulation time and the denoised image of the pepper.

Size	PSNR		h	Time (s)	Image
3x3	R	28.05	15	49.785	
	G	32.03	9		
	B	28.17	15		
5x5	R	29.32	15	60	
	G	31.16	9		
	B	28.61	15		
7x7	R	29.29	15	81.8	

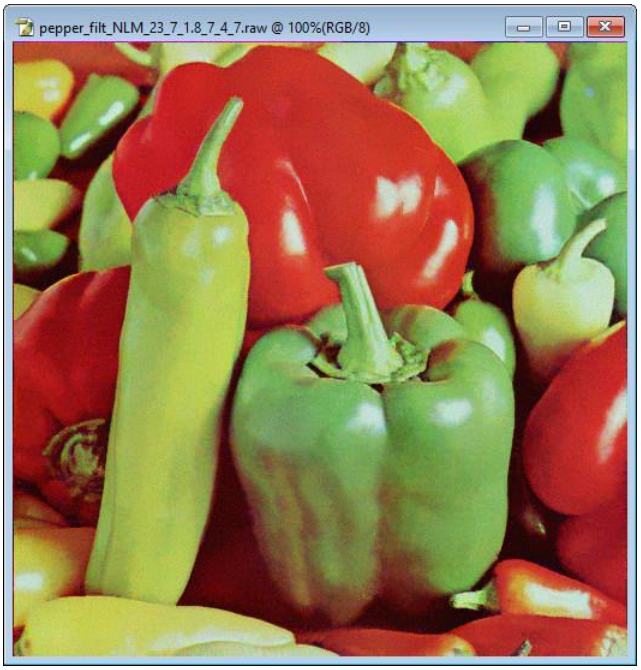
	G	30.38	9		
	B	28.34	15		

From the picture, we can see that the Size of the vector can change the running time significantly, but the quality of the image does not follow the increase of the image size.

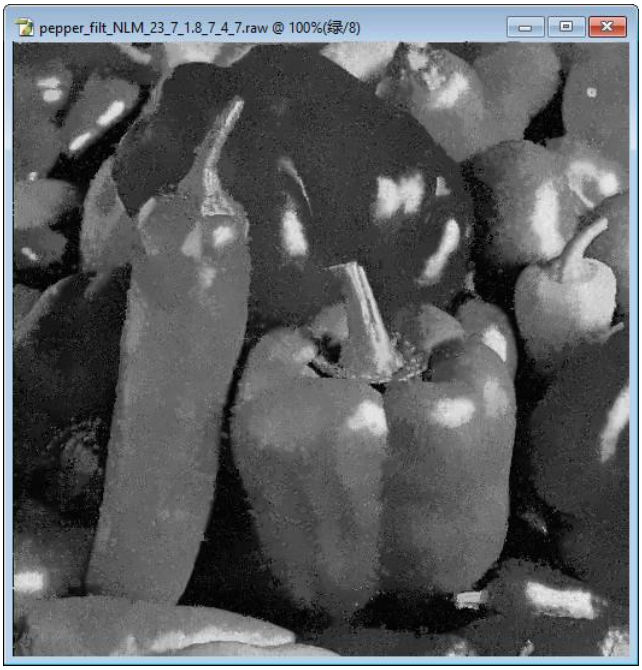
(2) The change of the value of h

We can use different h value to deal with the noisy pepper image. (in this question, we will use the 7x7 vector, 23x23 search window and use $\sigma = 1.8$)

When h of R is 7, h of G is 4 and h of B is 7, we can get the pepper image and PSNR.



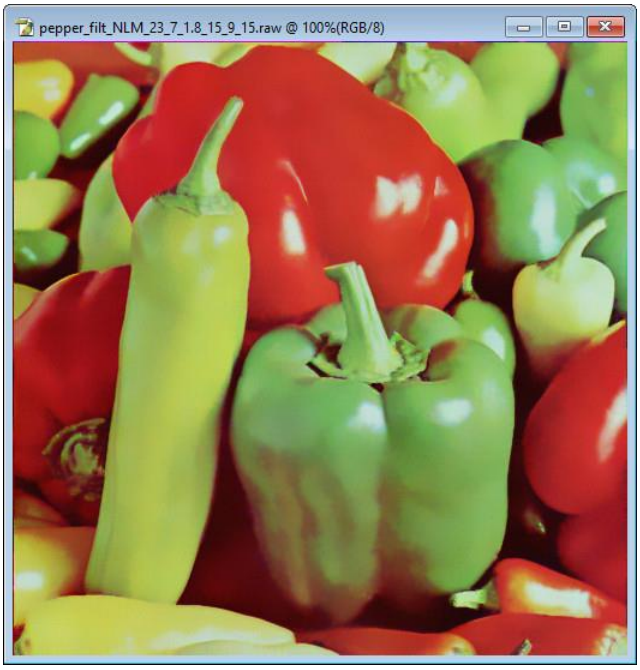
The denoised image by parameter above of pepper.raw



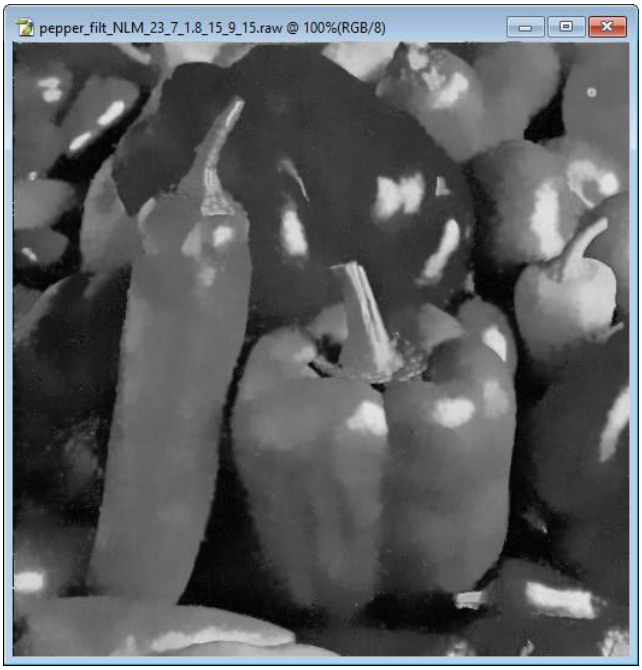
The denoised sequence of B by parameter above of pepper.raw

The PSNR of R is 25.59; PSNR of G is 29.87; PSNR of B is 26.5. We can see that the quality is bad, especially for G where has some speckles because of the impulse noise.

When h of R is 15, h of G is 9 and h of B is 15, we can get the pepper image and PSNR.



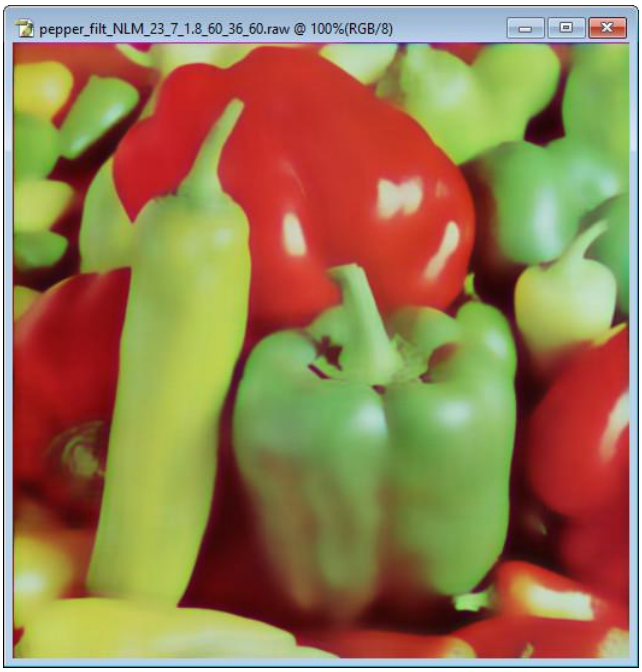
The denoised image by parameter above of pepper.raw



The denoised sequence of B by parameter above of pepper.raw

The PSNR of R is 29.29. The PSNR of G is 30.38. The PSNR of B is 28.34. We can see that the quality is really much better, and the impulse noise in G has been suppressed significantly.

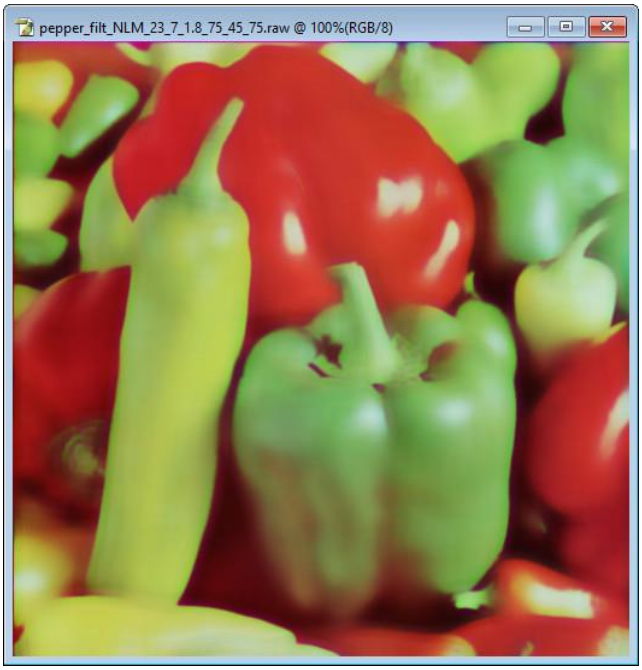
When h of R is 60, h of G is 36 and h of B is 60, we can get the pepper image and PSNR.



The denoised image by parameter above of pepper.raw

The PSNR of R is 25.42 PSNR of G is 26.93; PSNR of B is 24.63. We can see that the image becomes unclear and the quality of the image is decreased.

When h of R is 75, h of G is 45 and h of B is 75, we can get the pepper image and PSNR.



The denoised image by parameter above of pepper.raw

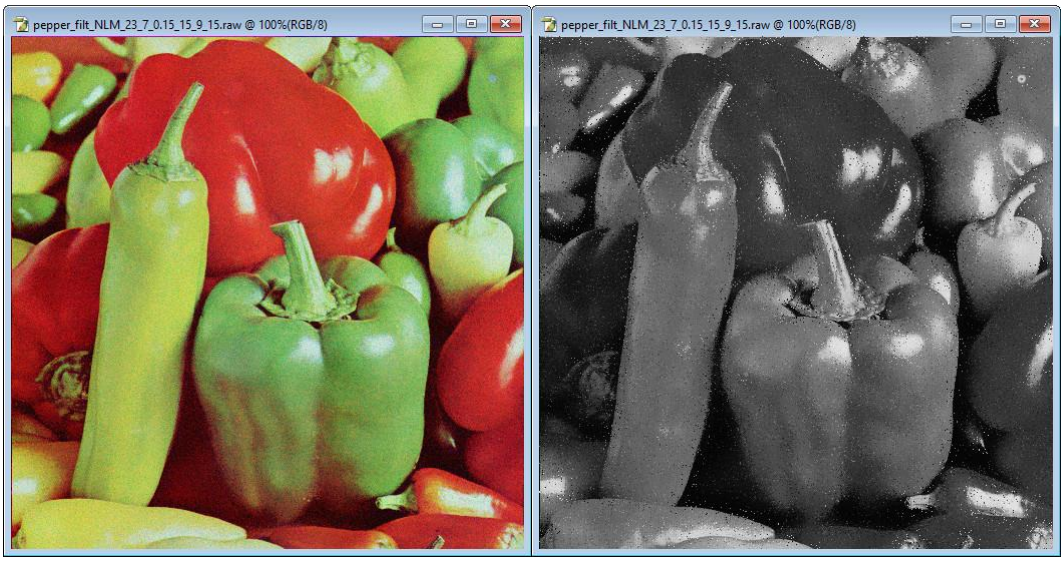
The PSNR of R is 24.76 PSNR of G is 26.00; PSNR of B is 24.01. We can see that the image becomes unclear and the quality of the image is decreased.

Therefore, if we set too low h value, we will not filter the noise of image; if we set too high h value, the image will be unclear.

(3) weight number

In the question before, I have used Gaussian Weight Number with $\sigma = 1.8$; now, we can change the σ and do it again. For example, we can use the $\sigma = 0.15$ and try it again.

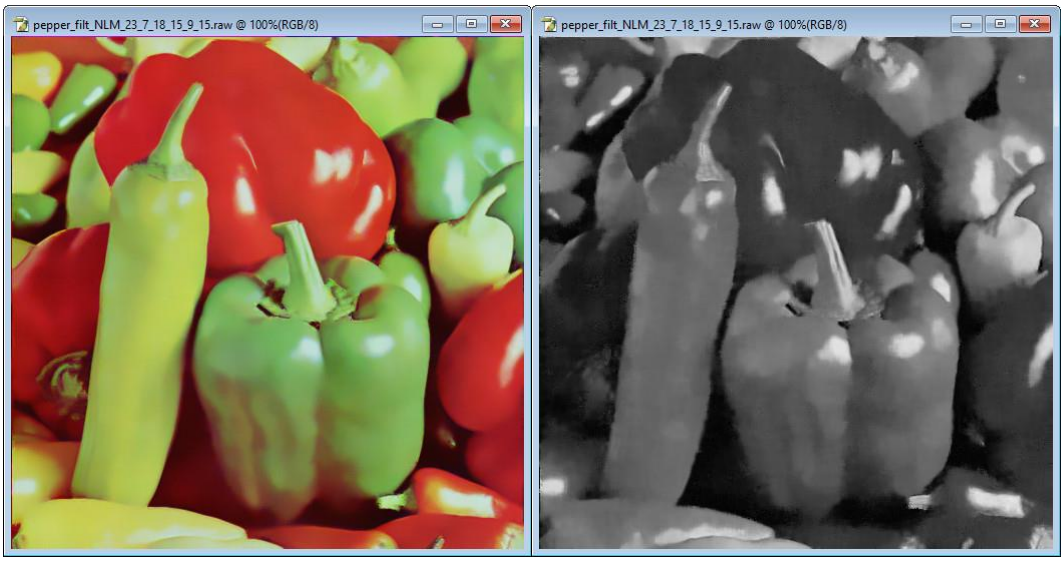
The result for $\sigma = 0.15$ can be shown below (all other parameters are same): (Second image is the B sequence image for pepper)



The denoised image by parameter above of pepper.raw

The PSNR of R is 22.31; PSNR of G is 30.18; PSNR of B is 24.57. we can see that there is still the existence of impulse noise.

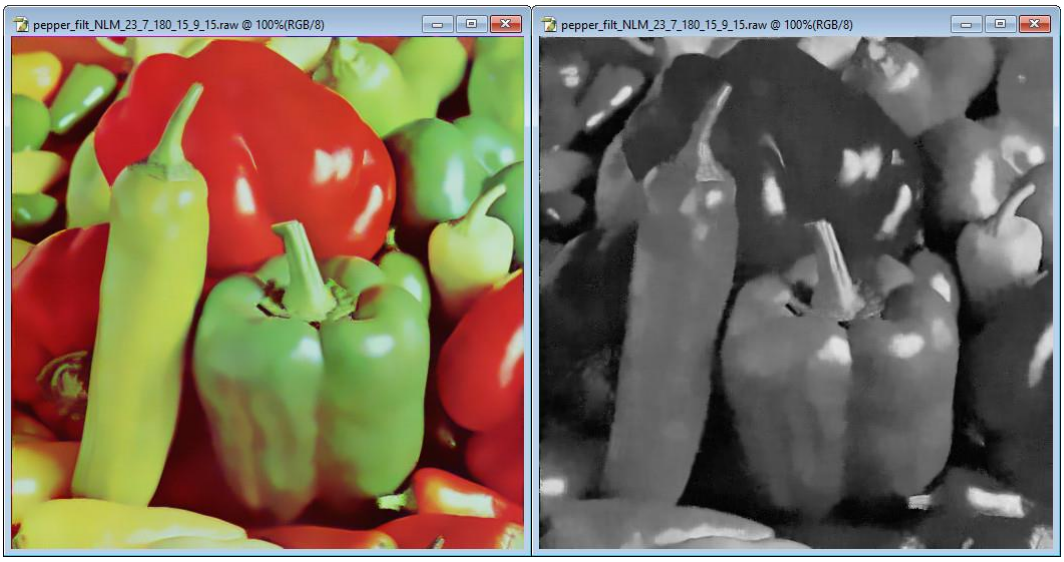
If we use $\sigma = 18$ Gaussian kernel function, we can get:



The denoised image by parameter above of pepper.raw

The PSNR of R is 28.84; PSNR of G is 28.98; PSNR of B is 27.68.

If we use $\sigma = 180$ Gaussian kernel function, we can get:



The denoised image by parameter above of pepper.raw

The PSNR of R is 28.83; PSNR of G is 28.96; PSNR of B is 27.67.

Comparing the image with the PSNR when $\sigma = 1.8$, we can see that the too little or too much σ will affect the quality of image, so we should consider the σ with caution.

(4) The size of searching window

For this question, we can choose three type of searching window: 13x13, 23x23 and 47x47, and the result of running time and PSNR can be shown below: (in this section, the h value for R, G and B will be 15, 9 and 15; $\sigma = 1.8$; vector size is 7x7)

Size of searching window	Seq.	PSNR	running time
19x19	R	29.17	15.20
	G	30.19	15.94
	B	28.20	14.15
23x23	R	29.29	27.16

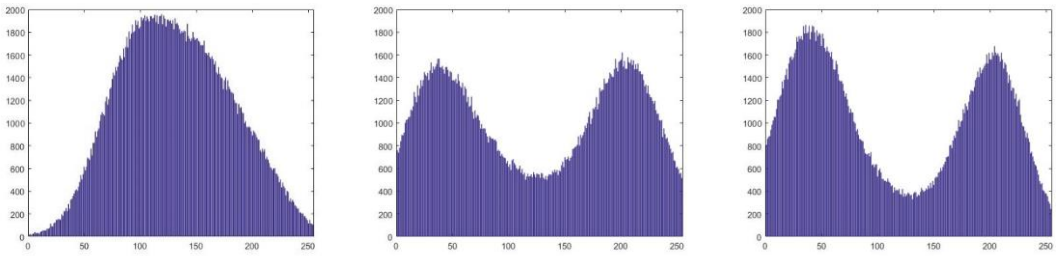
	G	30.37	27.61
	B	28.33	26.83
35x35	R	29.09	102.20
	G	30.50	95.91
	B	28.19	95.95
47x47	R	28.87	180.82
	G	30.48	190.38
	B	27.95	179.81

From the result, we can find that larger size searching window does not mean the improvement of quality; In contrast, it means the increasing of time significantly.

b) Sailboat image

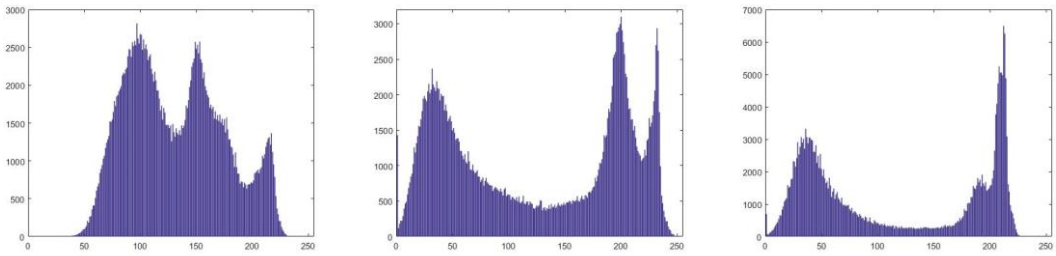
For the sailboat image, we should detect the noise type first. From the histogram of R, G and B, we can see that

Noisy picture



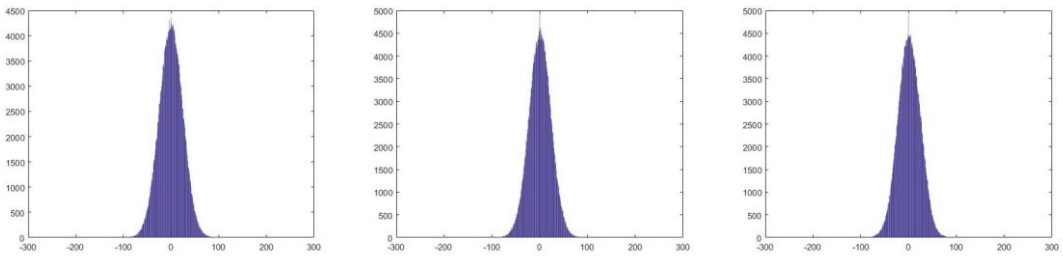
The histogram of R (Left), G(Middle) and B (Right) for sailboat.raw

Origin picture



The histogram of R (Left), G(Middle) and B (Right) for sailboat _noisy.raw

From the picture, we can all sequences have no impulse noise. Then, we will draw the histogram of random noise



The noise histogram of R (Left), G(Middle) and B (Right) for sailboat _noisy.raw

And we can calculate the deviation and mean of random noise, we can find that

For R sequence, $mean = -0.1792 \quad \sigma = 11.8258$

For G sequence, $mean = 0.8432 \quad \sigma = 11.7853$

For B sequence, $mean = 1.30627 \quad \sigma = 11.7669$

In total, all the noise in three sequences are random noise. Therefore, we will test the effect of single filter. The result can be shown below:

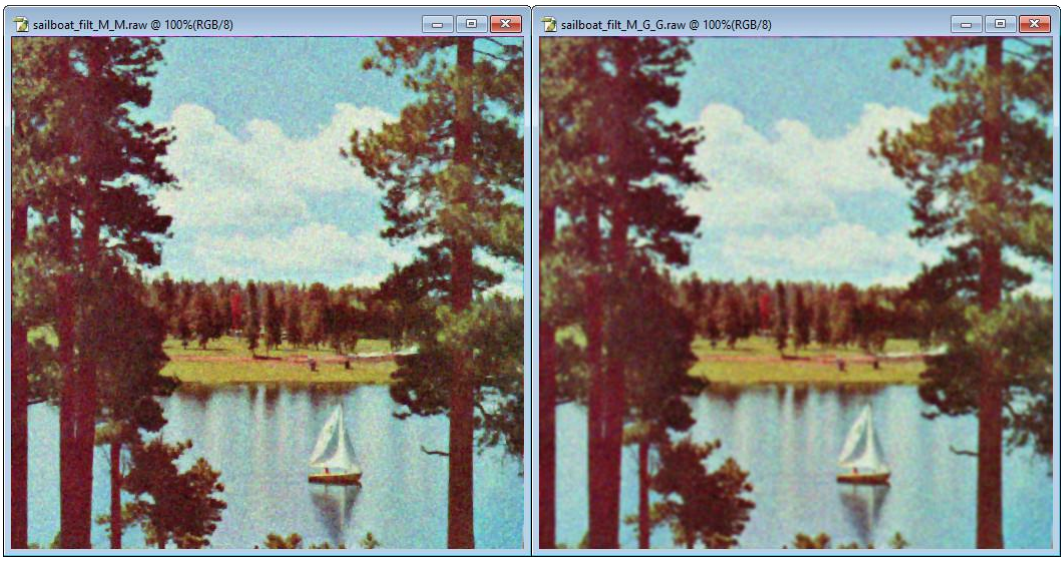
PSNR	R (Random)	G (Random)	B (Random + Impulse)
Average 3x3	27.0243	25.0819	24.9349
Median 5-points	26.1919	25.1661	25.07
Gaussian 3x3	27.1816	25.4294	25.38
Median + Median	26.1919	25.1661	25.07
Median + Gau + Gau	27.4689	24.4003	24.6442
Median + Bilateral	27.2708	24.7877	25.8116
Non-local mean	27.4315	25.9514	26.3067

P.S. The value h of non-local mean is 15 (R), 16.2 (G) and 16.2 (B), vector size is 7x7, size of searching window is 23x23 and the $\sigma = 1.16$.

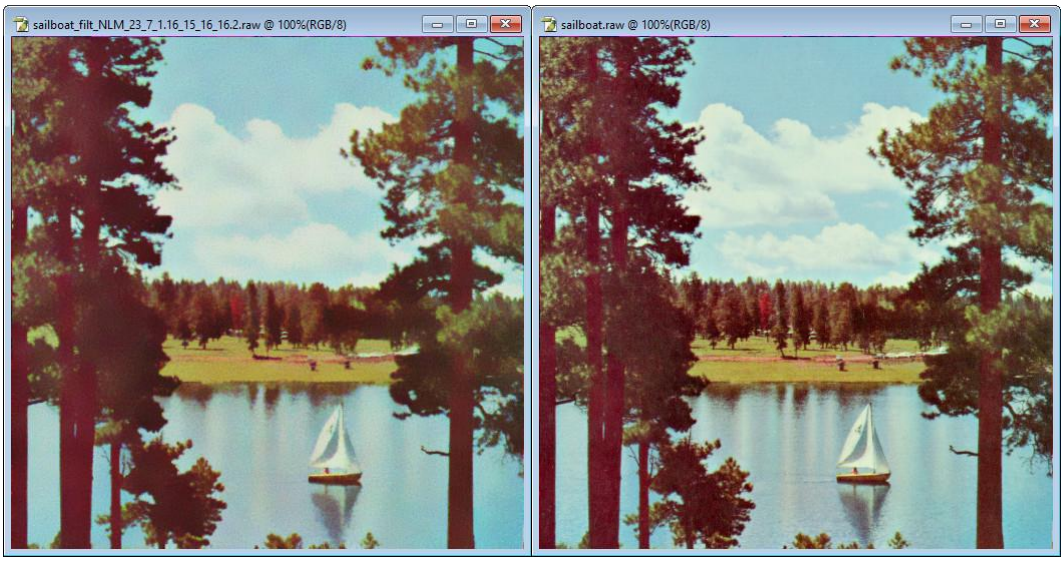
The filtered image can be shown below:

The image of Average (3x3) filter (Left) and Gaussian (3x3) filter (Right)

The image of Median (5-point) filter (Left) and Bilateral filter (Right)



The image of Median + Median (5-point) filter (Left) and Median + Gaussian + Gaussian filter (Right)



The image of Non-Local mean filter (left) and origin (right)

From the result of PNSR above, we can find that the Non-local mean filter can filter the random noise with good effect compared with other filters in the (a). In addition, From the output image, we can see that the random dark speckles have been destroyed completely when using the NLM filter, while there is still dark speckle in image filtered by the other filter, which means that the random noise has not been removed completely.

The reason for such phenomenon is for the **theory of the non-local mean.**

For the non-local mean, the most important task is to find the similarity between vectors and the vector we want to filter. If those two vectors are similar, we can time a high weight number on the vector, and if they are same, we can time one.

If we have some same patch vectors with i.i.d. random noise, we can time them and divide them, we can find that the mean will be zero and the deviation will be divided by the number of patch vector, which will finally approach zero.

If we have N similar vectors, they are

$$v_1 = u + n_1$$

$$v_2 = u + n_2$$
$$v_3 = u + n_3$$
$$\vdots$$
$$v_n = u + n_n$$

n_i are iid random noise with mean 0 and variation σ^2.

If we sum them and divide it by n, we can get that

$$\frac{1}{n} \times \sum_{i=1}^{n} u + n_i = u + \frac{1}{n} \times \sum_{i=1}^{n} n_i$$

The average of $\sum_{i=1}^{n} n_i$ is 0 and variation is $\sigma^2/n \to 0$, which means that the noise become little.

However, from the theory of the non-local mean and the result of the pepper, we can find that the impulse noise cannot be filtered effectively. In addition, the complexity of the NLM filter is really large since for each pixel, we have to search a series of pixels in the other location of this image. In addition, we can find that some gray margin can be unclear through the

For the problem that the Non-Local Mean cannot be useful to filter the impulse noise, I decide to leave out the central point with such noise when scanning in the searching window. In addition, I will skip the central vector when scanning in the searching window, which make the filter use the pixel value not relevant to the local pixel value.

The non-programming question

(1) Describe the Non-Local Mean filter

It is depicted in the section of Approach and Procedures.

(2) Plot the best denoised pepper image and the parameter you choose

The image in shown in the Result section and the parameter can be shown below:

The h for R is 15, for G is 9, for B is 15;

The weight number is Gaussian Kernel with $\sigma = 1.8$;

Vector size is 5x5

Size of searching window: 23x23.

During denoising, we can get the PSNR of R is 29.32; the PSNR of G is 31.16 and the PSNR of B is 28.61.

(3) The advantage of the NLM filter and why

It is shown in the last part of Discussion section.

(c) Block matching and 3-D (BM3D) transform filter (Bonus: 10%)

Motivation

In this question, we will denoise the image pepper and sailboat image through the BM3D filter. In addition, we should change some parameters of the BM3D filter and acquire the effect. Finally, we would discuss the difference between the BM3D and NLM filter.

Approach and Procedures

For this question, we will use the BM3D filter. The basic **Algorithm** of the BM3D is that:

(1) Basic estimate

1. Acquire many groups of pixel patches in the image, and one group of patches is formed by a series of similar patches.
2. Make each group of patches into a three-dimension matrix and do the 3-D transform on each matrices.
3. Implement the hard-thresholding filter to suppress the noise part in the spectrum.
4. Do the 3-D inverse transform and return those patches to their origin positions in the image.

(2) Final estimate

1. Acquire many groups of pixel patches in both the noisy and basic-estimate image, and one group of patches is formed by a series of similar patches.
2. Make each group of patches into a three-dimension matrix and do the 3-D transform on each matrices.
3. Implement the Wiener filter to suppress the noise part in the spectrum by using the spectrum acquired from the basic estimate.
4. Do the 3-D inverse transform and return those patches to their origin positions in the image.

When using the filter, we should research on the effects of several parameters: the sliding step to process every next reference block for the Wiener filter and hard-thresholding; The block size of Wiener filter and HT filter and threshold for the block-distance of HT and Wiener filter.

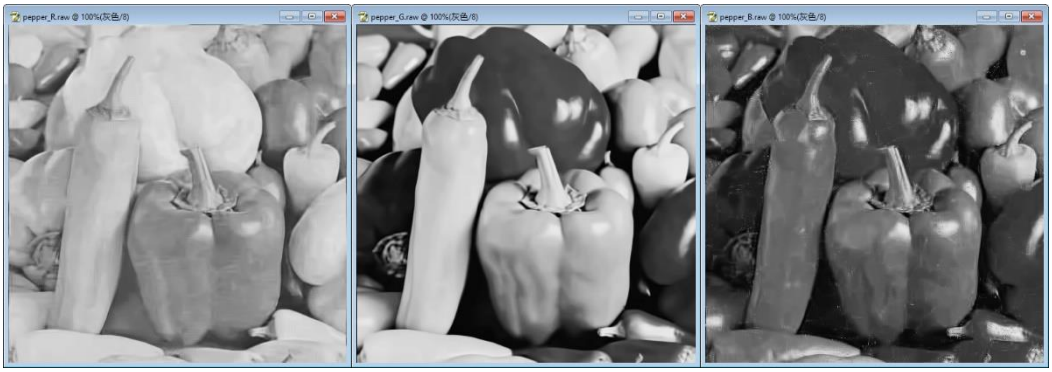
In this question, I use the MATLAB code searched from the website [2].

Results

One of the good result for the pepper image can be shown below: (my profile is chosen as vn_old)

The PSNR of R is 30.47. The PSNR of G is 31.18. The PSNR of B is 28.25.

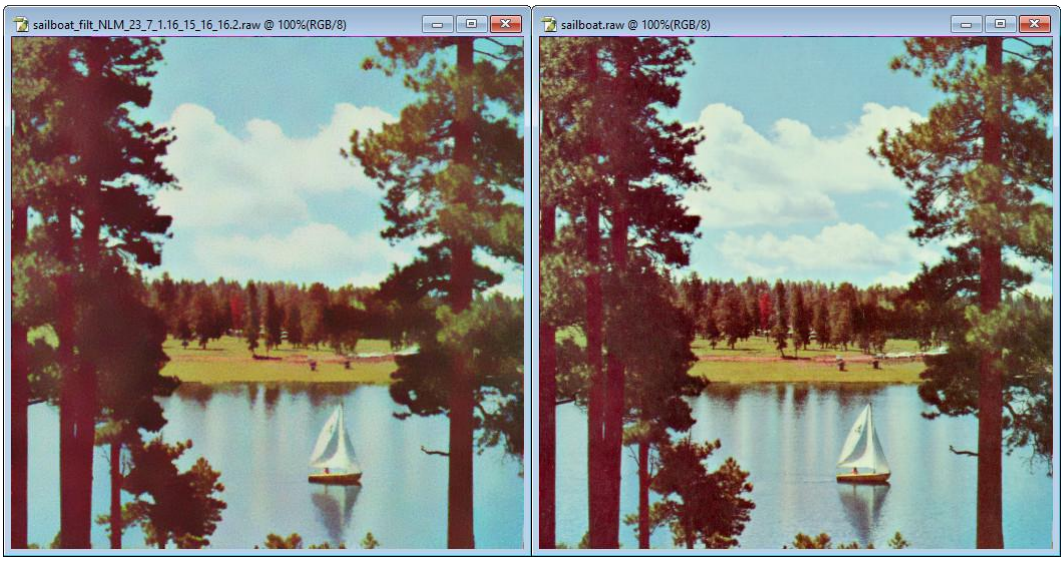
The denoised image by BM3D filter and origin image of pepper.raw



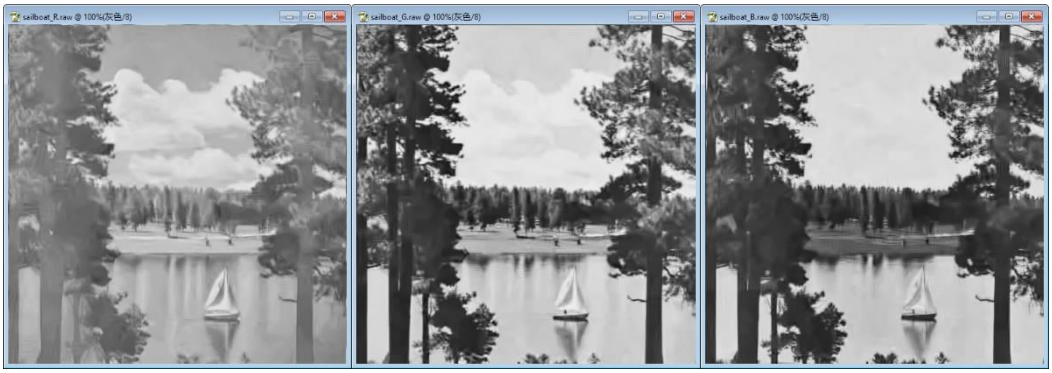
The R (Left), G (Middle) and B (Right) of denoised image by BM3D filter of pepper.raw

One of the good result for the sailboat image can be shown below: (my profile is chosen as vn_old)

The PSNR of R is 28.74. The PSNR of G is 26.72. The PSNR of B is 26.99.



The denoised image by BM3D of sailboat.raw



The R (Left), G (Middle) and B (Right) of denoised image by BM3D filter of sailboat.raw

Discussion

As mentioned above, I will change three kinds of parameters (for this part, I will only use the Winner filter to test):

- the sliding step to process every next reference block for the Wiener filter and hard-thresholding;
- The block size of Wiener filter and HT filter
- Threshold for the block-distance of HT and Wiener filter

For the sliding step for the Winner filter, I set four groups of statistics, we can find the result as follow (use the B sequence of the sailboat as example):

N_{step}	PSNR	Running time (s)
4	26.99	9.0
6	26.72	6.8
8	26.97	6.4
10	26.95	6.1

From the table above, we can find that the change of N_{step} cannot easily affect the value of PSNR, but it can change the running time significantly. The more the N_{step} is, the less the time will be used.

For the block size of Wiener filter, I will set up five groups of statistics, we can find the result as follow (use the B sequence of the sailboat as example):

N_1	PSNR	Running time (s)
9	27.02	5.7

11	26.99	6.9
13	26.96	8.3
15	26.95	10.5
17	26.94	12.6

From the table above, we can find that the change of N_1 will decrease value of PSNR, and it can change the running time significantly. The more the N_1 is, the more the time will be used.

For the threshold for the block-distance of Wiener filter, we will also set up four groups of statistics, we can find the result as follow (use the B sequence of the sailboat as example):

τ_{wiener}	PSNR	Running time (s)
35	26.88	6.6
350	26.97	6.8
3500	26.99	7.0
5000	26.99	7.1

From the table above, we can find that the change of τ_{wiener} will increase the PSNR and the running time slightly.

The non-programming question

(1) Explain the BM3D algorithm

It has been explained in the Approach and Procedure.

(2) Classification of BM3D

It is both the spatial domain and frequency domain filter. The reason is that we should search the patches with similar pixel values, which means that it is spatial domain; then, we will use the Wiener and HT filter to filter the high-frequency noise, which is the frequency domain. So the BM3D belongs to both domain filter.

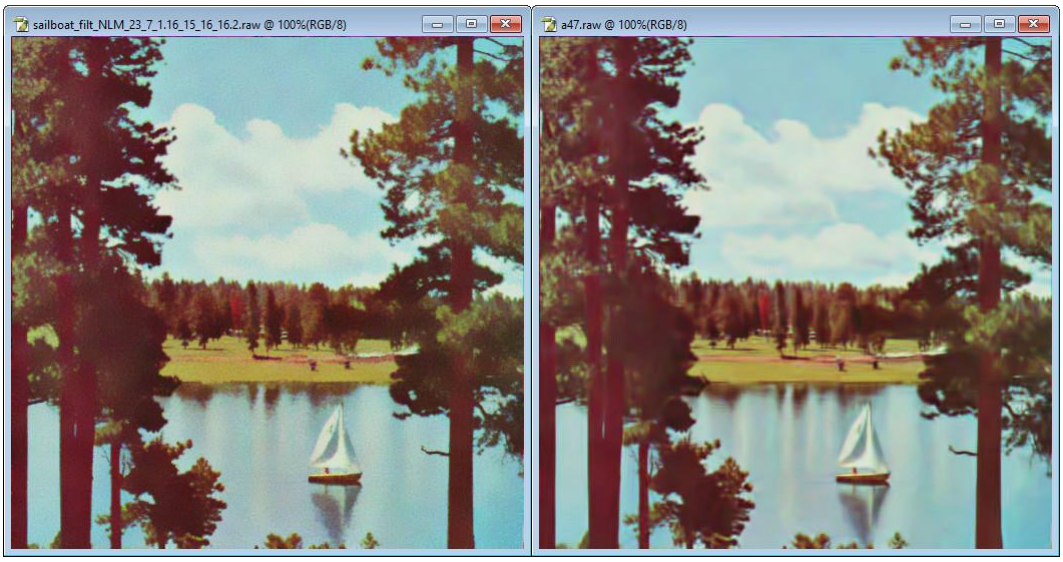
(3) Comparison of NLM and BM3D

Running speed: the NLM will require much time to get the output image, while most of the output for BM3D can be finished in led than ten seconds.

Quality of the picture: we can compare the output image and the PSNR of BM3D and NLM

		PSNR(R)	PSNR(G)	PSNR(B)
pepper	NLM	28.87	30.93	26.91
	BM3D	30.47	31.18	28.25
sailboat	NLM	27.8715	25.9749	26.401
	BM3D	28.74	26.72	26.99

Output image of Pepper for NLM (Left), BM3D (Right)



Output image of Sailboat for NLM (Left), BM3D (Right)

From the image and the PSNR of two images and two methods, we can find that the image output by BM3D is a bit clearer than the one by NLM, and BM3D can keep much more message in detail than the BM3D, especially from the edge of pepper and the details of the cloud in sailboat image. In addition, from the PSNR, we can find that the BM3D can deal with the impulse noise more effective than NLM. From the gray image of B sequence below, we can see that the NLM can only suppress the impulse noise, while the BM3D can nearly denoise such noise completely and the margin of the image can be kept clear:

The denoised sequence of B Using NLM (Left) and BM3D (Right) pepper.raw

References

[1] [Online] https://en.wikipedia.org/wiki/Blend_modes

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