Image Processing

Images Geometric Transformations
Practice 3

Wang Han ISD : 321388
QIU Guangzheng ISD : 321385
Pei Youran ISD : 321384

Contents

1 Purpose **5**

2 Theoretical substantiation **5**

2.1 Noise . . . 5

2.1.1 Impulse Noise . . . 5

2.1.2 Additive Noise . . . 5

2.1.3 Multiplicative Noise . . . 5

2.1.4 Gaussian (Normal) Noise . . . 5

2.1.5 Quantization Noise . . . 5

2.2 Low-pass Filtering . . . 6

2.2.1 Arithmetic Mean Filter . . . 6

2.2.2 Geometric Mean Filter . . . 6

2.2.3 Harmonic Mean Filter . . . 6

2.2.4 Counterharmonic Mean Filter . . . 6

2.2.5 Gaussian Filter . . . 6

2.3 Nonliner Filtering . . . 6

2.3.1 Median Filtering . . . 6

2.3.2 Weighted Median Filtering . . . 7

2.3.3 Adaptive Median Filtering . . . 7

2.3.4 Rank Filtering . . . 7

2.3.5 Wiener Filtering . . . 7

2.4 High-pass Filtering . . . 7

2.4.1 Roberts Filterr . . . 7

2.4.2 Prewitt Filter . . . 8

2.4.3 Sobel Filter . . . 8

2.4.4 Laplace Filter . . . 8

2.4.5 Canny Algorithm . . . 8

3 Practical Assignment:Geometric Transformations **8**

3.1 Original images . . . 8

3.2 Impulse Noise . . . 9

3.2.1 The scripts of Impulse Noise . . . 9

3.2.2 Comments . . . 9

3.2.3 Resulting images . . . 9

3.3 Additive Noise . . . 10

3.3.1 The scripts of Additive Noise . . . 10

3.3.2 Comments . . . 10

3.3.3 Resulting images . . . 10
3.4 Multiplicative Noise . . . 10
3.4.1 The scripts of Multiplicative Noise . . . 10
3.4.2 Comments . . . 11
3.4.3 Resulting images . . . 11
3.5 Gaussian (Normal) Noise . . . 11
3.5.1 The scripts of Gaussian (Normal) Noise . . . 11
3.5.2 Comments . . . 11
3.5.3 Resulting images . . . 12
3.6 Quantization Noise . . . 12
3.6.1 The scripts of Quantization Noise . . . 12
3.6.2 Comments . . . 12
3.6.3 Resulting images . . . 13

4 Practical Assignment:Low passing Filter 13

4.1 Preparation . . . 13
4.1.1 The scripts of Noise image import . . . 13
4.1.2 The scripts of Expand matrix function . . . 14
4.1.3 Comments . . . 14
4.2 Arithmetic Mean Filter . . . 15
4.2.1 The scripts of Arithmetic Mean Filter . . . 15
4.2.2 Comments . . . 15
4.2.3 Resulting images . . . 15
4.3 Geometric Mean Filter . . . 16
4.3.1 The scripts of Geometric Mean Filter . . . 16
4.3.2 Comments . . . 16
4.3.3 Resulting images . . . 17
4.4 Harmonic Mean Filter . . . 17
4.4.1 The scripts of Harmonic Mean Filter . . . 17
4.4.2 Comments . . . 18
4.4.3 Resulting images . . . 18
4.5 Counterharmonic Mean Filter . . . 18
4.5.1 The scripts of Counterharmonic Mean Filter . . . 18
4.5.2 Comments . . . 19
4.5.3 Resulting images . . . 20
4.6 Gaussian Filter . . . 20
4.6.1 The scripts of Gaussian Filter . . . 20
4.6.2 Comments . . . 21

4.6.3 Resulting images 21

5 Practical Assignment:Nonliner Filter 21

5.1 Median Filtering 21

5.1.1 The scripts of Median Filtering 21

5.1.2 Comments 22

5.1.3 Resulting images 22

5.2 Weighted Median Filtering 22

5.2.1 The scripts of Weighted Median Filtering 22

5.2.2 Comments 23

5.2.3 Resulting images 23

5.3 Adaptive Median Filtering 23

5.3.1 The scripts of Adaptive Median Filtering 23

5.3.2 Comments 25

5.3.3 Resulting images 25

5.4 Rank Filtering 26

5.4.1 The scripts of Rank Filtering 26

5.4.2 Comments 26

5.4.3 Resulting images 27

5.5 Wiener Filtering 27

5.5.1 The scripts of Wiener Filtering 27

5.5.2 Comments 28

5.5.3 Resulting images 28

6 Practical Assignment:High-pass filtering 28

6.1 Roberts Filter 28

6.1.1 The scripts of Roberts Filter 28

6.1.2 Comments 29

6.1.3 Resulting images 29

6.2 Prewitt Filter 29

6.2.1 The scripts of Prewitt Filter 29

6.2.2 Comments 30

6.2.3 Resulting images 30

6.3 Sobel Filter 30

6.3.1 The scripts of Sobel Filter 30

6.3.2 Comments 30

6.3.3 Resulting images 31

6.4 Laplace Filterr 31

6.4.1 The scripts of Laplace Filter 31
6.4.2 Comments 31
6.4.3 Resulting images 32
6.5 Canny Algorithm 32
6.5.1 The scripts of Canny Algorithm 32
6.5.2 Comments 32
6.5.3 Resulting images 33

7 Questions to Practical Assignment Report Defense **33**
7.1 What are the main disadvantages of adaptive image filtering methods? 33
7.2 For what values of the parameter Q will the counterharmonic filter work as an arithmetic filter, and for what values as a harmonic one? 33
7.3 What operators can be used to detect edges in the image? 33
7.4 Why, as a rule, is low-pass filtering performed at the first step of edge detection . . 33

8 Conclusion **33**

1 Purpose

Studying of the filtering images basic methods and edges detection..

2 Theoretical substantiation

2.1 Noise

2.1.1 Impulse Noise

Impulse noise is non-continuous and consists of irregular pulses or noise spikes of short duration and large amplitude. Impulse noise can occur for a variety of reasons, including electromagnetic interference and faults and defects in communication systems, and can also occur when electrical switches and relays in communication systems change state.

$$I(x,y) = \begin{cases} d, with probability \quad p, \\ s_{x,y}, with probability \quad (1-p), \end{cases} \tag{1}$$

2.1.2 Additive Noise

An interference added to the signal during its transmission over a communication channel. More precisely, one says that a given communication channel is a channel with additive noise if the transition function of the channel is given by a density the spaces of the values of the signals at the input and output of the channel, respectively) depending only on the difference. In this case the signal at the output of the channel can be represented as the sum of the input signal and a random variable independent of it, called additive noise.

$$I_{new}(x,y) = I(x,y) + \eta(x,y) \tag{2}$$

2.1.3 Multiplicative Noise

A special case of multiplicative noise is speckle noise. This noise appears in images captured by coherent imaging devices, such as medical scanners or radar. In such images, one can clearly observe points of light, blobs, which are separated by dark areas of the image.

$$I_{new}(x,y) = I(x,y) \cdot \eta(x,y) \tag{3}$$

2.1.4 Gaussian (Normal) Noise

Gaussian noise in the image can occur due to a lack of scene illumination, high temperature, etc. The noise model is widely used in low-pass filtering of images. The probability density distribution function $p(z)$ of the random variable z is described by the following expression:

$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}} \tag{4}$$

2.1.5 Quantization Noise

Depending on the selected quantization step and on the signal. Quantization noise can lead, for example, to the appearance of false contours around objects or to remove low-contrast details in the image. Such noise is not eliminated. Quantization noise can be approximately described by the Poisson distribution.

2.2 Low-pass Filtering

2.2.1 Arithmetic Mean Filter

for example, for a mask withthe size 3 × 3 the coefficients are 1/9, if 5 × 5 — 1/25. Thanks tothis normalization, the value of the filtering result will be reduced to the original image intensities range. Graphically, the two-dimensional function describing the filter mask looks like a parallelepiped, so the name is used in English literature box -filter. Arithmetic averaging is achieved using the following formula:

$$I_{new}(x,y) = \frac{1}{m \cdot n} \sum_{i=0}^{m} \sum_{j=0}^{n} I(i,j) \tag{5}$$

2.2.2 Geometric Mean Filter

The projection map is straight-linear, and the original line will not be deformed after transformation. However, the original outline will change. Therefore, the lines that were originally parallel may not be parallel, and the shape of the image will change significantly.

$$I_{new}(x,y) = \left[\prod_{i=0}^{m} \prod_{j=0}^{n} I(i,j) \right]^{\frac{1}{m \cdot n}} \tag{6}$$

2.2.3 Harmonic Mean Filter

$$I_{new}(x,y) = \frac{m \cdot n}{\sum_{i=0}^{m} \sum_{j=0}^{n} \frac{1}{I(i,j)}} \tag{7}$$

2.2.4 Counterharmonic Mean Filter

$$I_{new}(x,y) = \frac{\sum_{i=0}^{m} \sum_{j=0}^{n} I(i,j)^{Q+1}}{\sum_{i=0}^{m} \sum_{j=0}^{n} I(i,j)^{Q}} \tag{8}$$

2.2.5 Gaussian Filter

The pixels in the sliding window that are closer to the analyzed pixel should have a greater influence on the filtering result than the extreme ones. Therefore, the mask weight coefficients can be described by the bell-shaped Gaussian function.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{9}$$

2.3 Nonliner Filtering

2.3.1 Median Filtering

The classical median filter uses a mask with unit coefficients. An arbitrary window shape can be set using zero coefficients. The pixel intensities in the window are represented as a column vector and sorted in ascending order. The filtered pixel is assigned the median (mean) intensity value in the series. The median element number after sorting can be calculated by the formula $n = (N + 1)/2$, where N — the number of pixels involved in sorting

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{10}$$

2.3.2 Weighted Median Filtering

In this median filtering modification in the mask, weights are used to reflect more influence on the filtering result of pixels located closer to the element to be filtered. Median filtering qualitatively removes impulse noise, and also does not introduce new intensity values in grayscale images. Increasing the size of the window increases the filter noise canceling ability, but the objects outlines begin to distort.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{11}$$

2.3.3 Adaptive Median Filtering

In this filtering modification, a sliding window of size s x s increases adaptively based on the filter results. Let's use Zmin to depresent: Zmax, minimum, maximum and median values of intensity in Zmed window, Zi, j pixel intensity value, coordinates (i, j), Smar maximum allowable window size.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{12}$$

2.3.4 Rank Filtering

A median filtering generalization is a rank filter of order r, selects a pixel with the number from the resulting column vector of mask elements, which will be the result of filtering. Sometimes rank is written as a percentage, for example, for all filter rank is 0%, median filter — 50%, max-filter — 100%.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{13}$$

2.3.5 Wiener Filtering

Uses Wiener's pixel-adaptive method based on statistics estimated from the local neighborhood of the each pixel.

$$\mu = \frac{1}{n \cdot m} \sum_{i=0}^{m} \sum_{j=0}^{n} I(i,j) \tag{14}$$

$$\sigma^2 = \frac{1}{m \cdot m} \sum_{i=0}^{m} \sum_{j=0}^{n} I^2(i,j) - \mu^2 \tag{15}$$

$$I_{new}(x,y) = \mu + \frac{\sigma^2 - v^2}{\sigma^2}(I(x,y) - \mu) \tag{16}$$

2.4 High-pass Filtering

2.4.1 Roberts Filterr

The Roberts filter works with the minimum dimensionality mask allowed for the derivative calculation 2×2, therefore it is fast and quite efficient. Possible options for masks for finding the gradient along the axes Ox and Oy.

$$G_x = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \tag{17}$$

$$G = \sqrt{G_x^2 + G_y^2} = |G_x| + |G_y| \tag{18}$$

$$grad = \arctan\left(\frac{G_y}{G_x}\right) \tag{19}$$

2.4.2 Prewitt Filter

This approach uses two orthogonal masks of size 3×3, allowing you to more accurately calculate the derivatives along the axes Ox and Oy.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{20}$$

2.4.3 Sobel Filter

The Sobel operator is a discrete differential operator that combines Gaussian smoothing and differential derivation. The operator uses local differences to find edges, and the obtained is an approximation of a gradient.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{21}$$

2.4.4 Laplace Filter

The Lavacian operator is a second-order derivative operator with rotational invariance and can meet the requirements of image edge sharpening (edge detection) in different directions. Usually, the sum of the coefficients of its operators needs to be zero.

$$L(I(x,y)) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{22}$$

2.4.5 Canny Algorithm

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}} \tag{23}$$

3 Practical Assignment:Geometric Transformations

3.1 Original images

Figure 1: original picture

3.2 Impulse Noise

3.2.1 The scripts of Impulse Noise

```
clc;
close all;
clear;
X=imread('Queen Elizabeth.png');
I=rgb2gray(X); %I is an Gray image
[numRows,numCols,Layers]=size(I);

figure(1);
imshow(I);
imwrite(I,'noise result\original picture.png');
title('original picture');
p=0.1;%p1 is the probability of noise
I_impulse=imnoise(I,'salt & pepper',p);

figure(2);
imshow(I_impulse);
imwrite(I_impulse,'noise result\Impulse Noise picture.png');
title('Impulse Noise Picture');
```

3.2.2 Comments

Impulse noise is the random change of pixel intensity to a specified value with the probability of p in the original image. In general, we use "salt" i.e. d=255, or "pepper" i.e. d=0. or a combination of both. In MATLAB, we can't achieve an effect that contains only one type of noise with the built-in function imnoise(). So the result is an image with both black and white random noise.

3.2.3 Resulting images

Figure 2: original picture

Figure 3: Impulse Noise picture

3.3 Additive Noise

3.3.1 The scripts of Additive Noise

```
a=0;b=200;% a is mean of noise,b is variance
Gaussian_noise=a+sqrt(b)*randn(numRows,numCols);
%create noise of Gaussian distribution
for m=1:1:numRows
    for n=1:1:numCols
        I_additive(m,n)=I(m,n)+Gaussian_noise(m,n);
        %add the original pixel with noise
    end
end

figure(3);
imshow(I_additive);
imwrite(I_additive,'noise result\Additive Noise picture.png');
title('Additive Noise Picture');
```

3.3.2 Comments

In general, noise will conform to a certain distribution, where the noise that directly adds up the influence on the original image is called additive noise. We take a noise signal that conforms to the Gaussian distribution as an example and create a new Gaussian distribution matrix with the same dimension as the original figure, in which the mean and variance can be adjusted by themselves. The corresponding elements of the two matrices are then added. The output result is shown in the figure, when the mean value is greater than 0, the image becomes brighter as a whole, and vice versa. Increase the variance and the image becomes more blurry.

3.3.3 Resulting images

Figure 4: original picture

Figure 5: Additive Noise picture

3.4 Multiplicative Noise

3.4.1 The scripts of Multiplicative Noise

```
I_multiply=imnoise(I,'speckle');

figure(4);
imshow(I_multiply);
imwrite(I_multiply,'noise result\Multiplicative Noise ...
            picture.png');
title('Multiplicative Noise Picture');
```

3.4.2 Comments

Similar to additive noise, but the effect on the original image is multiplication. Using the matlab built-in function imnoise (I,'speckle'), we simulated multiplying the original graph with the corresponding elements of a Gaussian distribution matrix of the same dimension. The final result is shown in the figure.

3.4.3 Resulting images

Figure 6: original picture

Figure 7: Multiplicative Noise picture

3.5 Gaussian (Normal) Noise

3.5.1 The scripts of Gaussian (Normal) Noise

```
I_Gaussian=imnoise(I,'gaussian',0.001);

figure(5);
imshow(I_Gaussian);
imwrite(I_Gaussian,'noise result\Gaussian (Normal)
Noise picture.png');
title('Gaussian (Normal) Noise Picture');
```

3.5.2 Comments

Adding white noise that conforms to the Gaussian distribution to the original image, since the principle is similar to additive noise and multiplicative noise, and the transformed image is also

similar, I will not introduce it too much.

3.5.3 Resulting images

Figure 8: original picture result

Figure 9: Gaussian (Normal) Noise picture

3.6 Quantization Noise

3.6.1 The scripts of Quantization Noise

```
I_Quantization=imnoise(I,'poisson');

figure(6);
imshow(I_Quantization);
imwrite(I_Quantization,'noise result\Quantization
Noise picture.png');
title('Quantization Noise Picture');
```

3.6.2 Comments

Generated by adding a Poisson distribution to the original image. It can simulate the signal noise generated by two levels in life. The output is shown in the figure.

3.6.3 Resulting images

Figure 10: original picture result

Figure 11: Quantization Noise picture

4 Practical Assignment:Low passing Filter

4.1 Preparation

4.1.1 The scripts of Noise image import

```
clc;
close all;
clear;

X=imread('Queen Elizabeth.png');
I=rgb2gray(X); %I is an Gray image
[numRows,numCols]=size(I);

I_Impulse=imread('noise result\Impulse Noise picture.png');

%Impulse noise picture as example
I_Additive=imread('noise result\Additive Noise picture.png');
%Additive noise picture as example
I_Multiplicative=imread('noise result\Multi
plicative Noise picture.png');
%Multiplicative noise picture as example
I_Normal=imread('noise result\Gaussian
(Normal) Noise picture.png');
%Gaussian (Normal) noise picture as example
I_Quantization=imread('noise result\Q
uantization Noise picture.png');
%Quantization noise picture as example

```

```
%run this after running p3.m
I_imp_salt=imread('noise_result\I_imp_salt_picture.png');
```

4.1.2 The scripts of Expand matrix function

```
%% Expand the matrix function

function ext=ext(I_ori,num)
%I_ori: original picture;num:extra circle numbers
[numRows,numCols]=size(I_ori);
ext=I_ori;
for x=1:1:num
    [numRows,numCols]=size(ext);
    ext=cat(1,ext,ext(numRows,:));
    ext=cat(2,ext,ext(:,numCols));
end
ext=double(ext);
for x=1:1:numRows
    for y=1:1:numCols
        if ext(x,y)==0
            ext(x,y)=1;
        end
    end
end
end
```

4.1.3 Comments

Before processing the image, I import the output of the PA1 at the beginning of the program as an image to be processed, which is used to reflect the actual effect of the filter. And establish the function ext(), which is to expand the original graph matrix, copying the last column num times and placing it on the far right, and the last row is also copied num times and placed at the bottom, in case the mask crosses the stack when traversing to the last element of the original graph. Because the default type of imread is unit8, each number cannot exceed 256 maximum, so I converted it to double type for easy calculation.

4.2 Arithmetic Mean Filter

4.2.1 The scripts of Arithmetic Mean Filter

```
mask1=fspecial('average',3);
I_Arithmetic=uint8(filter2(mask1,I_Normal));

figure(2);
subplot(1,2,1);
imshow(I_Normal);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_Arithmetic);
title('Arithmetic Mean Filter Picture');
imwrite(I_Arithmetic,'Low-pass Filtering\Arithmetic
Mean Filter picture.png');
saveas(2,'Low-pass Filtering\compare Arithmetic
Mean Filter picture.png');
```

4.2.2 Comments

In the arithmetic mean filter, I use Gaussian noise as the original image. Its principle is to take the average number of pixels under the mask and assign them to that pixel. The effect of this processing is to make the original uneven pixel intensity uniform, the advantage is to reduce the difference between noise and surrounding pixels, the disadvantage is to blur the originally clear image and still see the noise.

4.2.3 Resulting images

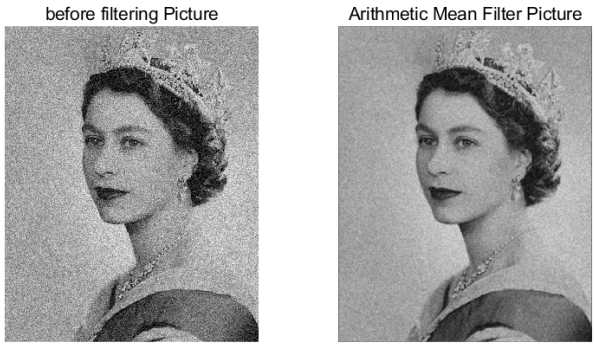

Figure 12: Arithmetic Mean Filter

4.3 Geometric Mean Filter

4.3.1 The scripts of Geometric Mean Filter

```
I_Geometric=zeros(numRows,numCols);

I_extra1=ext(I_Normal,2);

for x=1:1:numRows
    for y=1:1:numCols
        I_Geometric(x,y)=double(I_extra1(x,y)*I_extra1(x+1,y)
        *I_extra1(x,y+2)*I_extra1(x+2,y)*I_extra1(x,y+1)
        *I_extra1(x+1,y+1)*I_extra1(x+2,y+1)
        *I_extra1(x+1,y+2)*I_extra1(x+2,y+2))^(1/9);
    end
end
I_Geometric=uint8(I_Geometric);

figure(3);
subplot(1,2,1);
imshow(I_Normal);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_Geometric);
title('Geometric Mean Filter Picture');
imwrite(I_Geometric,'Low-pass Filtering\Geometric
Mean Filter picture.png');
saveas(3,'Low-pass Filtering\compare Geometric
Mean Filter picture.png');
```

4.3.2 Comments

Since MATLAB does not have a built-in function, I first use ext() to expand and convert the type of the original matrix before editing the main program, and then use the traversed element as the upper left corner of the 3*3 mask, multiply all the 3*3 pixels below the mask, and then open the ninth power. Then iterate through the mask until the last row and the last column. Since imshow cannot recognize decimals, and the range of double type matrix is forcibly converted to 0-1, and the part greater than 1 is directly converted to 1, the output matrix should finally be converted to uint8 type and then output, and the result is shown in the figure, which has a similar effect to the arithmetic mean filter.

4.3.3 Resulting images

Figure 13: Geometric Mean Filter picture

4.4 Harmonic Mean Filter

4.4.1 The scripts of Harmonic Mean Filter

```
I_extra2=ext(I_imp_salt);
% if you didn't run p3.m,you can try other noise pictures
onematrix=ones(3,3);

for x=1:1:numRows
    for y=1:1:numCols
        I_Harmonic(x,y)=9/(sum(sum(onematrix./I_extra2(x:x+2
        ,y:y+2))));
    end
end

I_Harmonic=uint8(I_Harmonic);

figure(4);
subplot(1,2,1);
imshow(I_imp_salt);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_Harmonic);
title('Harmonic Mean Filter Picture');
imwrite(I_Harmonic,'Low-pass Filtering
```

```
\Harmonic Mean Filter picture.png');
saveas(4,'Low-pass Filtering\
compare Harmonic Mean Filter picture.png');
```

4.4.2 Comments

The algorithm framework is similar to the geometric mean filter, first use the ext() function to expand and convert the data type of the original image, and then calculate according to the formula in the book, and finally the data type is converted back to the uint8 output, you can see that the filter handles the "salt" point very well, almost completely restoring the picture.

4.4.3 Resulting images

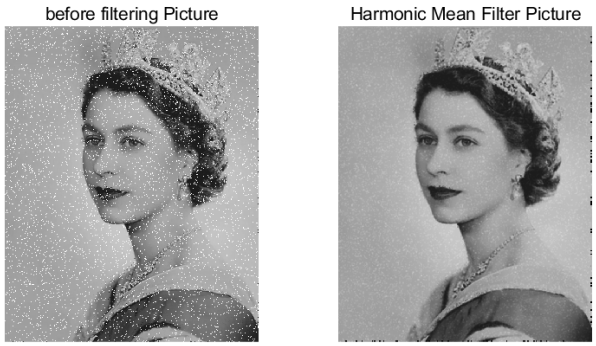

Figure 14: Harmonic Mean Filter

4.5 Counterharmonic Mean Filter

4.5.1 The scripts of Counterharmonic Mean Filter

```
I_extra3=ext(I_Normal,2);

for Q=-1:1:1
    for x=1:1:numRows
        for y=1:1:numCols
            dividend(x,y)=(sum(sum(I_extra3(x:x+2,y:y+2).^(Q+1))));
            divisor(x,y)=(sum(sum(I_extra3(x:x+2,y:y+2).^Q)));
            I_Counterharmonic(x,y,Q+2)=dividend(x,y)/divisor(x,y);
        end
    end
end
```

```
I_Counterharmonic=uint8(I_Counterharmonic);

figure(5);
subplot(2,2,1);
imshow(I_Normal);
title('before filtering Picture');
subplot(2,2,2);
imshow(I_Counterharmonic(:,:,1));
title('Counterharmonic Mean Filter Picture(Q=-1)');
subplot(2,2,3);
imshow(I_Counterharmonic(:,:,2));
title('Counterharmonic Mean Filter Picture(Q=0)');
subplot(2,2,4);
imshow(I_Counterharmonic(:,:,3));
title('Counterharmonic Mean Filter Picture(Q=1)');
imwrite(I_Counterharmonic(:,:,1),'Low-pass Filtering\Counterharmonic
Mean Filter picture(Q=-1).png');
imwrite(I_Counterharmonic(:,:,2),'Low-pass Filtering\Counterharmonic
Mean Filter picture(Q=0).png')
imwrite(I_Counterharmonic(:,:,3),'Low-pass Filtering\Counterharmonic
Mean Filter picture(Q=1).png')
saveas(5,'Low-pass Filtering\compare Counterharmonic
Mean Filter picture.png');
```

4.5.2 Comments

According to the formula in the book, Q has a great influence on its role. Therefore, I listed the three cases of Q=-1,0,1 in the code, superimposed in a three-dimensional matrix through an external for loop, and the third dimension of the matrix only has three layers, 1, 2, and 3, corresponding to the case of Q=-1,0,1. Then the three layers are output separately, and the results are shown in the figure. Although at the digital level, the output matrix corresponding to the three Q values is slightly different, it is not distinguishable from the naked eye.

4.5.3 Resulting images

Figure 15: Counterharmonic Mean Filter

4.6 Gaussian Filter

4.6.1 The scripts of Gaussian Filter

```
I_Gaussian=imgaussfilt(I_Normal);

figure(6);
subplot(1,2,1);
imshow(I_Normal);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_Gaussian);
title('Gaussian Filter Picture');
imwrite(I_Gaussian,'Low-pass Filtering\
Gaussian Filter picture.png')
saveas(6,'Low-pass Filtering\
compare Gaussian Filter picture.png');

function ext=ext(I_ori)
[numRows,numCols]=size(I_ori);
ext=cat(1,I_ori,I_ori(numRows,:),I_ori(numRows,:));
ext=double(cat(2,ext,ext(:,numCols),ext(:,numCols)));

for x=1:1:numRows
    for y=1:1:numCols
        if ext(x,y)==0
            ext(x,y)=1;
        end
```

```
    end
end
end
```

4.6.2 Comments

The image to be processed is filtered using a two-dimensional Gaussian smooth kernel with a standard deviation of 0.5 and the filtered image is returned, and the result is shown in the figure, similar to the averaging filter, reducing the difference between image noise and surrounding pixels.

4.6.3 Resulting images

Figure 16: Gaussian Filter picture

5 Practical Assignment:Nonliner Filter

5.1 Median Filtering

5.1.1 The scripts of Median Filtering

```
I_ori1=I_Impulse;
I_Median=medfilt2(I_ori1);

figure(2);
subplot(1,2,1);
imshow(I_ori1);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_Median);
title('Median Filtering Picture');
```

```
imwrite(I_Median, 'Nonlinear_filtering\
Median_Filtering_picture.png');
saveas(2, 'Nonlinear_filtering
\compare_Median_Filtering_picture.png');
[numRows, numCols]=size(I);
```

5.1.2 Comments

It is implemented through the built-in function medfilt2(), which works by traversing all elements in the m*n neighborhood around each corresponding element of the image to be processed, and then assigning the corresponding element of the new matrix. The output is shown and has an impressive effect on pulsed salt and pepper noise.

5.1.3 Resulting images

Figure 17: Median Filtering

5.2 Weighted Median Filtering

5.2.1 The scripts of Weighted Median Filtering

```
m=2;n=3;
I_ori2=I_Impulse;
I_Weighted=medfilt2(I_ori2 ,[m,n]);

figure(3);
subplot(1,2,1);
imshow(I_ori2);
title('before_filtering_Picture');
subplot(1,2,2);
```

```
imshow(I_Weighted);
title('Weighted Median Filtering Picture');
imwrite(I_Weighted,'Nonlinear filtering\
Weighted Median Filtering picture.png');
saveas(3,'Nonlinear filtering\
compare Weighted Median Filtering picture.png');
```

5.2.2 Comments

Weights (numbers 2, 3, etc.) are used to reflect the effect of pixels closer to the element to be filtered on the filtering results. In the MATLAB function medfilt2(), this method is only reflected in changing the neighborhood range of the median from 3*3 to m*n, and the result obtained is shown in the case of applying m=2 and n=3. The image has subtle black and white noise, and as can be seen from the figure, the weighted median filter is not necessarily better than the median filter, and is more affected by the weight and the shape of the neighborhood matrix.

5.2.3 Resulting images

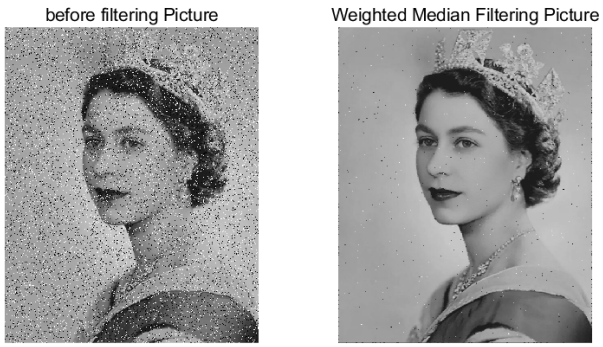

Figure 18: Weighted Median Filtering

5.3 Adaptive Median Filtering

5.3.1 The scripts of Adaptive Median Filtering

```
S_ori=5;%initial size of window(from 0)
S_max=10;%the maxium size of window
I_ori3=I_Impulse;
I_extra=ext(I_ori3,S_ori);
I_adaptive=zeros(numRows,numCols);
A1=0;A2=0;
c=0;d=0;e=0;%debug counters
```

```
for x=1:1:numRows
    for y=1:1:numCols
        s=S_ori;%initialize size of window
        while 1
            Z=double(I_extra(x:x+s,y:y+s));
            z_med=median(Z,"all");
            z_max=max(Z,[],"all");
            z_min=min(Z,[],"all");
            A1=z_med-z_min;
            A2=z_med-z_max;
            s_max=min([numRows-x,numCols-y,S_max]);
            c=c+1;
            if A1>0 && A2<0
                B1=Z(1,1)-z_min;
                B2=Z(1,1)-z_max;
                if B1>0 && B2<0
                    I_adaptive(x,y)=Z(1,1);
                    break
                else
                    I_adaptive(x,y)=z_med;
                    d=d+1;
                    break
                end
            else
                if s<s_max
                    s=s+1;
                else
                    I_adaptive(x,y)=Z(1,1);
                    e=e+1;
                    break
                end
            end
        end
    end
end

I_adaptive=uint8(I_adaptive);

figure(4);
subplot(1,2,1);
imshow(I_ori3);
title('before filtering Picture');
```

```
subplot(1,2,2);
imshow(I_adaptive);
title('adaptive Filtering Picture');
imwrite(I_adaptive,'Nonlinear filtering\Adaptive Median
Filtering picture.png');
saveas(4,'Nonlinear filtering\compare Adaptive Median
Filtering picture.png');
```

5.3.2 Comments

The underlying logic is also to look for medians. But the biggest difference from the median filter is that during traversal, the window size is not fixed. For example, when processing one of the elements, the window will continue to grow until it gets a return value that meets the criteria. I wrote the code the way in the book, and before processing I set the initial size and upper size of the window to prevent it from running too long. The pending image is then expanded according to the initial size of the window. The final output result is as shown in the figure, after many experiments, I found that the maximum size of the window is not significantly different after the output image is raised to 7*7, it will only lengthen the running time. At the same time, I also found that the initial size of the window is not as small as possible, and when the size is 1*1, there are still more peppers and salts in the output that have not been removed. When the size is 2*2, the pepper salt is basically removed, but the image outline begins to deform, and when the size is 3*3, the contour shape becomes more serious.

5.3.3 Resulting images

Figure 19: Adaptive Median Filtering

5.4 Rank Filtering

5.4.1 The scripts of Rank Filtering

```
I_ori4 = I_Impulse;
I_Rank1 = ordfilt2(I_ori4,1,ones(3,3));
I_Rank2 = ordfilt2(I_ori4,5,ones(3,3));
I_Rank3 = ordfilt2(I_ori4,9,ones(3,3));

figure(5);
subplot(2,2,1);
imshow(I_ori4);
title('before filtering Picture');
subplot(2,2,2);
imshow(I_Rank1);
title(' Rank Filter(filter rank is 0%)');
subplot(2,2,3);
imshow(I_Rank2);
title(' Rank Filter(filter rank is 50%)');
subplot(2,2,4);
imshow(I_Rank3);
title(' Rank Filter(filter rank is 100%)');
saveas(5,'Nonlinear filtering\
compare Rank Filter result.png')

imwrite(I_Rank1,'Nonlinear filtering\
Rank Filter(filter rank is 0%) picture.png');
imwrite(I_Rank2,'Nonlinear filtering\
Rank Filter(filter rank is 50%) picture.png');
imwrite(I_Rank3,'Nonlinear filtering\
Rank Filter(filter rank is 100%) picture.png');
```

5.4.2 Comments

is done using the MATLAB built-in function ordfilt2().The output result is shown in the figure. It can be clearly seen that when the filter rank is 50%, the treatment effect is the best. When the filter rank is 0%, the salt noise will be processed but the pepper noise will be amplified. 100% of the time it's the opposite.

5.4.3 Resulting images

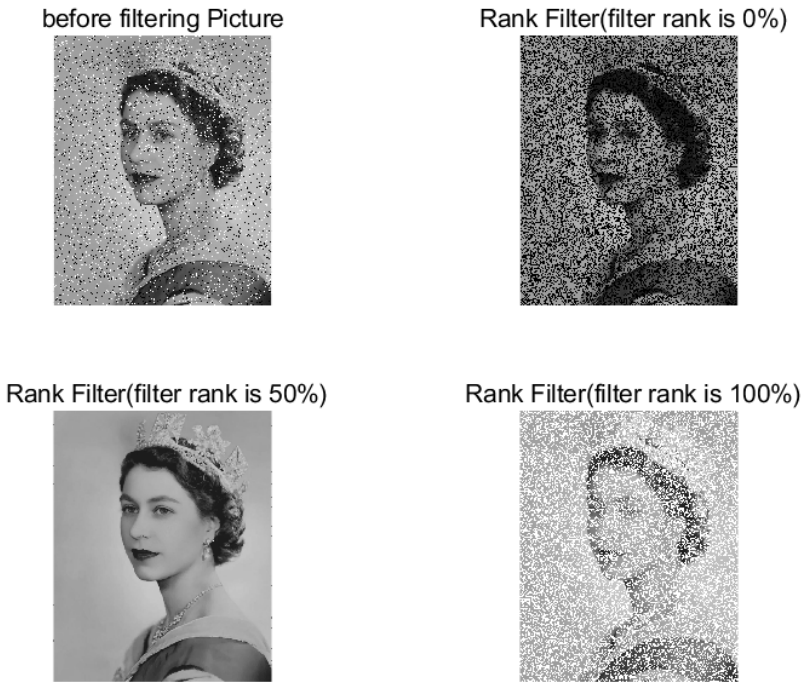

Figure 20: Rank Filtering

5.5 Wiener Filtering

5.5.1 The scripts of Wiener Filtering

```
177
178 I_ori5 = I_Impulse;
179 m=8;n=8;
180 I_Wiener = wiener2(I_ori5,[m n]);
181
182 figure(6);
183 subplot(1,2,1);
184 imshow(I_ori5);
185 title('before filtering Picture');
186 subplot(1,2,2);
187 imshow(I_Wiener);
188 title('Wiener Filtering');
189 imwrite(I_Wiener,'Nonlinear filtering\
190 Wiener Filtering picture.png');
191 saveas(6,'Nonlinear filtering\
192 compare Wiener Filtering picture.png');
```

5.5.2 Comments

This is done using the MATLAB built-in function wiener2(). The output is shown in the figure, and I think the output of this filter approximates the processing effect of a low-pass filter. The equalization of image differences is done well, but it blurs an otherwise sharp image. There are also a few concentrated noises that remain uneliminated.

5.5.3 Resulting images

Figure 21: Wiener Filtering

6 Practical Assignment:High-pass filtering

6.1 Roberts Filter

6.1.1 The scripts of Roberts Filter

```
1
2  import cv2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  o = cv2.imread('Queen Elizabeth.png')
6  kernelx = np.array([[-1, 0], [0, 1]], dtype=int)
7  kernely = np.array([[0, -1], [1, 0]], dtype=int)
8  x = cv2.filter2D(o, cv2.CV_16S, kernelx)
9  y = cv2.filter2D(o, cv2.CV_16S, kernely)
10 absX = cv2.convertScaleAbs(x)
11 absY = cv2.convertScaleAbs(y)
12 Roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)
```

```
fig1 , axes = plt.subplots(1,2, figsize=(10,5), dpi=100)
axes[0].imshow(o[:,:,::-1])
axes[1].imshow(Roberts[:,:,::-1])
```

6.1.2 Comments

The Robert operator uses the cross-difference algorithm to calculate the rate of change of gray along the direction of Ox and Oy, and we can see from the figure that the effect on image edge detection is not obvious

6.1.3 Resulting images

Figure 22: Roberts Filter

6.2 Prewitt Filter

6.2.1 The scripts of Prewitt Filter

```

kernelX = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]],
dtype=int)
kernelY = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]],
dtype=int)
x = cv2.filter2D(o, cv2.CV_16S, kernelX)
y = cv2.filter2D(o, cv2.CV_16S, kernelY)
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Prewitt = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

fig2 , axes = plt.subplots(1,2, figsize=(10,5), dpi=100)
axes[0].imshow(o[:,:,::-1])
axes[1].imshow(Prewitt[:,:,::-1])
```

6.2.2 Comments

The Prewitt operator is a differential operator for image edge detection, and its principle is to use the difference generated by the gray value of pixels in a specific region to achieve edge detection. Since the Prewitt operator uses a 3x3 template to calculate the pixel values in the region, while the template of the Robert operator is 2x2, the edge detection results of the Prewitt operator are more obvious in both horizontal and vertical directions than the Robert operator. The Prewitt operator is suitable for identifying noisy, grayscale gradient images, and it can be seen from the figure that the edge detection results of the image are indeed better than the Roberts operator.

6.2.3 Resulting images

Figure 23: Prewitt Filter

6.3 Sobel Filter

6.3.1 The scripts of Sobel Filter

```
Sobelx = cv2.Sobel(o, cv2.CV_64F,1,0)
Sobely = cv2.Sobel(o, cv2.CV_64F,0,1)
Sobelx = cv2.convertScaleAbs(Sobelx)
Sobely = cv2.convertScaleAbs(Sobely)
Sobelxy = cv2.addWeighted(Sobelx,0.5, Sobely,0.5,0)
Sobelxy11 = cv2.Sobel(o, cv2.CV_64F,1,1)
Sobelxy11 = cv2.convertScaleAbs(Sobelxy11)

fig3, axes = plt.subplots(1,3, figsize=(10,5), dpi=100)
axes[0].imshow(o[:,:,::-1])
axes[1].imshow(Sobelxy[:,:,::-1])
axes[2].imshow(Sobelxy11[:,:,::-1])
```

6.3.2 Comments

The Sobel operator is a discrete differential operator for edge detection that combines Gaussian smoothing and differential derivation. This operator is used to calculate the approximation of the brightness and darkness of the image, and to record specific points in the area that exceed a certain

number as edges according to the degree of brightness and darkness next to the edges of the image. The Sobel operator adds the concept of weight on the basis of the Prewitt operator, believing that the influence of the distance of adjacent points on the current pixel is different, and the closer the pixel corresponds to the greater the influence of the current pixel, so as to achieve image sharpening and highlight the edge outline.

The Sobel operator detects the edge according to the gray weighted difference between the upper and lower pixels, the left and right adjacent points, and the extreme value is reached at the edge. It has a smoothing effect on noise and provides more accurate edge direction information. Because the Sobel operator combines Gaussian smoothing and differential derivation (differentiation), the result will have more noise immunity, and when the accuracy requirements are not very high, the Sobel operator is a more commonly used edge detection method. The edge positioning of the Sobel operator is more accurate, and it is often used in images with a lot of noise and grayscale gradient

6.3.3 Resulting images

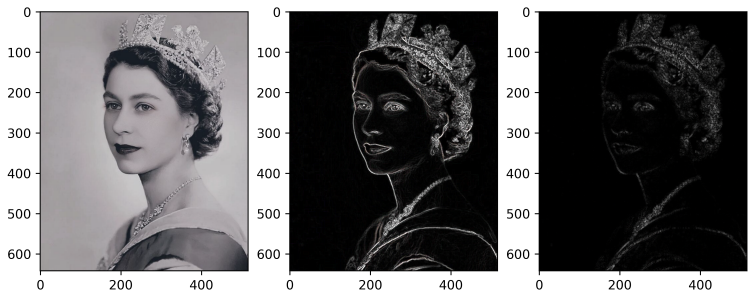

Figure 24: Sobel Filter

6.4 Laplace Filterr

6.4.1 The scripts of Laplace Filter

```
Laplacian = cv2.Laplacian(o, cv2.CV_64F)
Laplacian = cv2.convertScaleAbs(Laplacian)
fig4, axes = plt.subplots(1,2, figsize=(10,5), dpi=100)
axes[0].imshow(o[:,:,::-1])
axes[1].imshow(Laplacian[:,:,::-1])
```

6.4.2 Comments

The Laplace operator is one of the commonly used edge enhancement operators, which also uses partial derivatives calculation, while Laplace, which is slightly different in gradient method, uses a second-order partial derivative, as can be seen from the figure, for the crown above the queen's head, the image edge detection is more obvious.

6.4.3 Resulting images

Figure 25: Laplace Filter

6.5 Canny Algorithm

6.5.1 The scripts of Canny Algorithm

```
r1=cv2.Canny(o,128,200)
r2=cv2.Canny(o,32,128)
o_grey=cv2.cvtColor(o,cv2.COLOR_BGR2GRAY)
fig5, axes = plt.subplots(1,3, figsize=(10,5), dpi=100)
imgs=np.hstack([o_grey,r1,r2])
cv2.imshow("Canny Algorithm",imgs)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imwrite('High-pass_filtering/Canny_Algorithm.png', imgs)
```

6.5.2 Comments

The Canny operator is the best edge detection operator practiced so far. Detecting and connecting edges by applying non-maximum suppression and double threshold processing and connectivity to gradient amplitude images ensures that all edges should be found, that there should be no false responses, and that edge points should be well positioned, and from the processing results, almost all the edges of the image are detected, and the double threshold at 32 and 128 (smaller) can capture more edge information.

6.5.3 Resulting images

Figure 26: Canny Algorithm

7 Questions to Practical Assignment Report Defense

7.1 What are the main disadvantages of adaptive image filtering methods?

The adaptive median filter runs two processes A and B. Both processes need to be judged, and then the convolution kernel is added, which takes a long time, and there are multiple loops in the algorithm design process, and the complexity is high.

7.2 For what values of the parameter Q will the counterharmonic filter work as an arithmetic filter, and for what values as a harmonic one?

when Q= 1,the counterharmonic filter work as an harmonic filter,Q = 0 the filter turns into arithmetic filter an arithmetic one.

7.3 What operators can be used to detect edges in the image?

Roberts operator, Prewitt operator, Laplace operator, Sobel operatorscharr operatorcanny operator can be used to detect edges in the image.

7.4 Why, as a rule, is low-pass filtering performed at the first step of edge detection

Edge detection algorithms are primarily based on the first and second derivatives of image intensity, but derivatives are often sensitive to noise, so noise inside the image must be removed by a low-pass filter. The goal of the low-pass filter is to reduce the rate of change of the image. For example, replace each pixel with the mean of the pixels around that pixel. This smooths and replaces areas where the intensity varies significantly. Therefore, low-pass filtering is required before detecting the edge.

8 Conclusion

Through this practice, we learned how to add various types of noise to an image, including the common noise uses of impulse noise, additive multiplicative noise, Gaussian, and quantization noise. I also learned how to process noisy pictures with a low-pass filter. Among them, low-pass filters are divided into linear and non-linear types.

Linear filters are named because their mask is a matrix, which is divided into arithmetic mean filters, geometric mean filters, harmonic mean filters and antiharmonic mean filters, and Gaussian filters. Among them, the arithmetic mean filter is to average the pixels in the mask, and the

geometric mean filter is to find the geometric mean of the pixels in the mask, and the two methods are similar and the results are similar. The harmonic averaging filter is done by adding and dividing the pixels in the mask by 9 and then the reciprocal again. Because this transformation requires multiple countdowns, images with pixel intensity 0 in the original image cannot be processed, so they are useful for white noise but not for black noise. The antiharmonic averaging filter is a composite of multiple averaging filters, and by changing the size of Q, each corresponds to one of the above filters. These linear filters are generally effective for uniform noise, i.e. noise such as additive and multiplication, but less pronounced for impulse noise.

Nonlinear filters do not have various restrictions on masks, but instead replace masks with freer windows. It is divided into median filter, weighted median filter, adaptive median filter, rank filter, and Vienna filter. The median filter takes the median output within the window, thus avoiding the effect of the maximum and minimum values at that position. The weighted median filter takes the processed median output by changing the shape and weights of the window. More directional and selective. The adaptive median filter is more free, and the size of the window is constantly changing, making the results more accurate and avoiding some extreme cases (e.g. the pixel intensity in the window is 0 or 255, which other median filters cannot accurately handle). As well as self-contained rank filters and Viennese filters, etc. These nonlinear filters filter the maximum and minimum values in the window instead of spreading them evenly across the surrounding pixels. Therefore, nonlinear filters are very effective in processing impulse noise, but not so effective at uniform noise.

In the process of learning and practicing the high-pass filter operator, we know that the Roberts operator is a cross-difference method used, and the Prewitt operator is a differential operator for image edge detection, and its principle is to use the differential generated by pixel gray values in a specific area to achieve edge detection. Since the Prewitt operator uses a 3*3 template to calculate the pixel values in the region, while the template of the Robert operator is 2*2, the edge detection results of the Prewitt operator are better than Robert in both horizontal and vertical directions. On the basis of the Prewitt operator, the Sobel operator adds the concept of weights, bel that the closer to the pixel the larger the weight, thereby enhancing image sharpening and edge curve extraction. The Laplace operator is a second-order derivative that is more pronounced for gradient variations, but is also more susceptible to noise. In the final Canny algorithm, non-maximum suppression and double threshold detection are used, which completely avoids noise while preserving continuous edge curves. By comparing the code implementation and results of several operators that implement high-pass filtering, we can find that under the condition of lower thresholds, edge information is retained more, and the Canny algorithm has the best processing effect.