Practical Assignment №3

Images Geometric Transformations

20321233 zou letian

20321308 cao xinyang

20321309 chen hanzheng

Purpose

1. *Noises types.* Select an arbitrary image. Get images distorted by various noises using the imnoise() function with parameters other than the default values.

2. *Low-pass filtering.* Process the distorted images obtained in the previous paragraph with a Gaussian filter and a counterharmonic mean filter with different values of the parameter .

3. *Nonlinear filtering.* Process the distorted images obtained in the first point with median, weighted median, rank and Wiener filtering for different sizes of the mask and its coefficients. Implement adaptive median filtering.

4. *High-pass filtering.* Select source image. Detect edges with Roberts, Prewitt, Sobel, Laplace filters, Canny algorithm.

Note. Please note that when doing the practical assignment you are not allowed to use the *"Lenna"* image or any other image that was used either in this book or during the presentation.

Part1 Noises Types

1. Impulse Noise

With impulse noise, the signal is distorted by spikes with very large negative or positive values of short duration and can arise, for example, due to decoding errors. This noise results in white («salt») or black («pepper») dots in the image, which is why it is often called *dot* noise. To describe it, one should take into account the fact that the appearance of a noise spike in each pixel () does not depend on the quality of the original image or on the presence of noise at other points and has the probability of , and the value of the pixel intensity ()will be changed to value ∈ [0,255]:

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

where — the pixel intensity of the original image, — noisy image, if d = 0 — noise like «pepper», if d = 255 --noise like «salt».

2. Additive Noise

Additive noise is described by the following expression:

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

where — noisy image, — original image, signal-independent additive noise with Gaussian or any other probability density function.

3. Multiplicative Noise

Multiplicative noise is described by the following expression:

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

wheew — noisy image, — original image, — signal indepen dent multiplicative noise that multiplies the recorded signal. Examples include film grain, ultrasound images, etc. A special case of multiplicative noise is *speckle* noise. This noise appears in images captured by coherent imaging devices such as medical scanners or radars. In such images, one can clearly observe light spots, speckles, which are separated by dark areas of the image.

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 () of the random variable is described by the following expression:

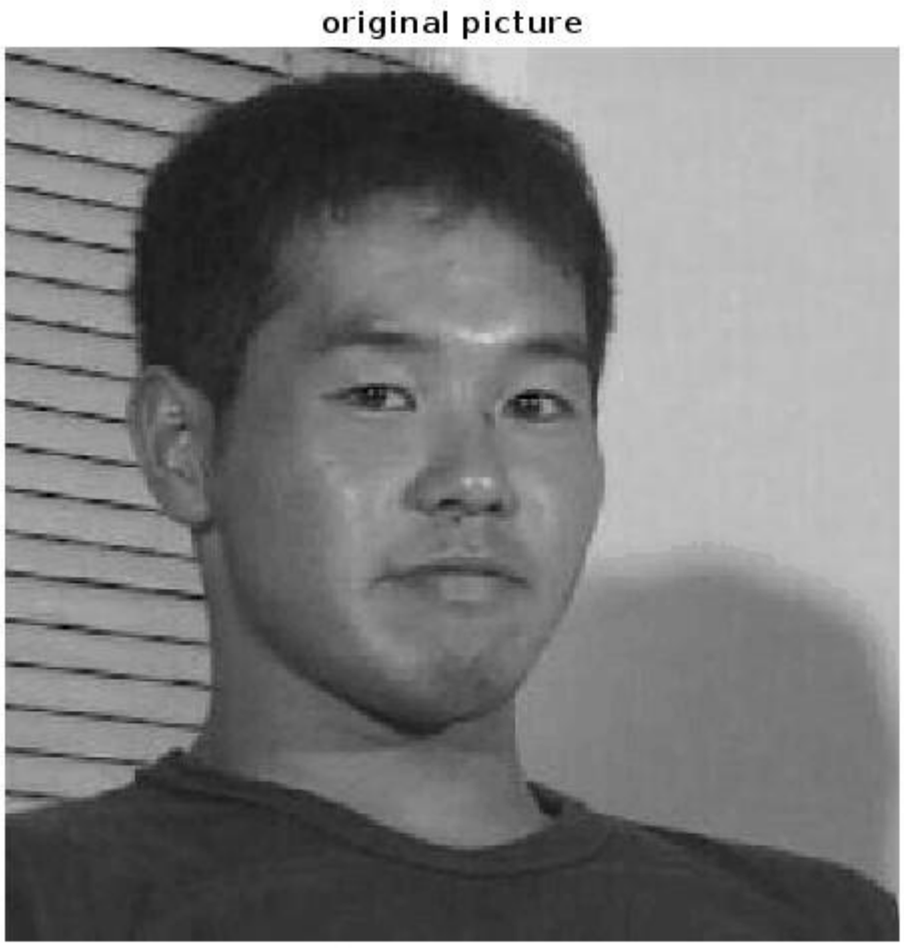
$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}},$$

where — the intensity of the image (for example, for a grayscale image $\in$ [0,255]), — mean (mathematical expectation) of a random variable , — standard deviation, variance 2 determines the power of the introduced noise.

5. Quantization Noise

Depends 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 and in MATLAB it can be done by the function imnoise(I, 'poisson'). In Python it can be applied by using SciPy function called skimage.util.random_noise(I,poisson').

2. Original images;



3. Code of the scripts;

```
%% Noises Types

clc;
close all;
clear;
X=imread('yjsp.jpg');
I=rgb2gray(X);[numRows,numCols,Layers]=size(I);

figure(1);
imshow(I);
imwrite(I,'noise result\original picture.png');
title('original picture');

%% Impulse Noise

p=0.1;
I_impulse=imnoise(I,'salt & pepper',p);
```

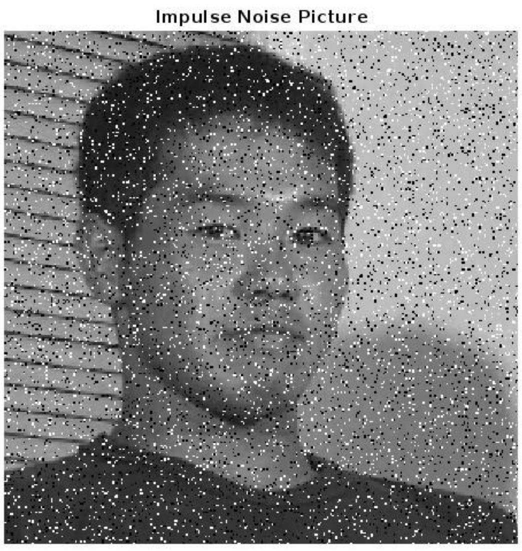
```
figure(2);
imshow(I_impulse);
imwrite(I_impulse,'noise result\Impulse Noise picture.png');
title('Impulse Noise Picture');

%% 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');

%% 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');

%% 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');

%% Quantization Noise
I_Quantization=imnoise(I,'poisson');
```

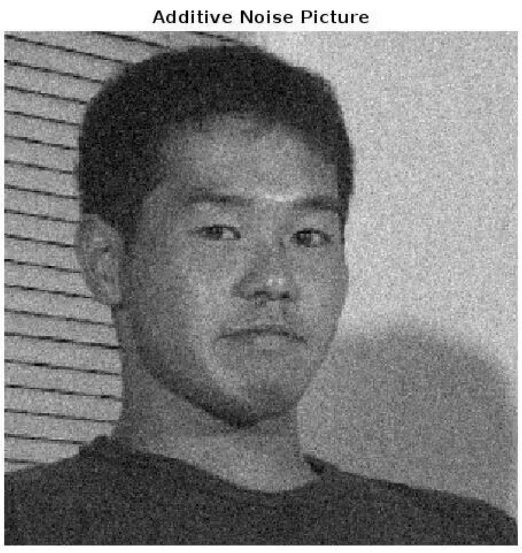
```
figure(6);
imshow(I_Quantization);
imwrite(I_Quantization,'noise result\Quantization Noise picture.png');
title('Quantization Noise Picture');
```

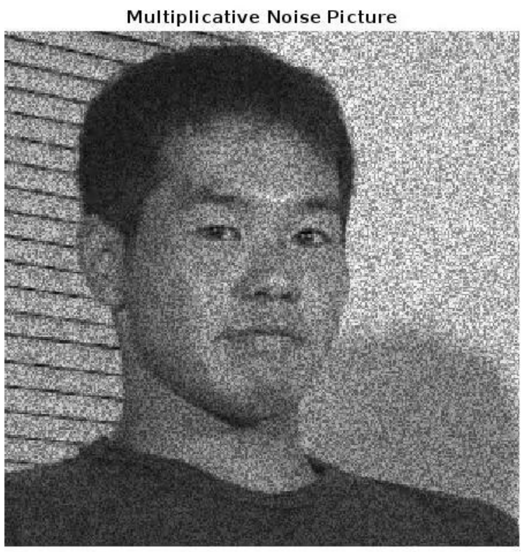
4. Comments;

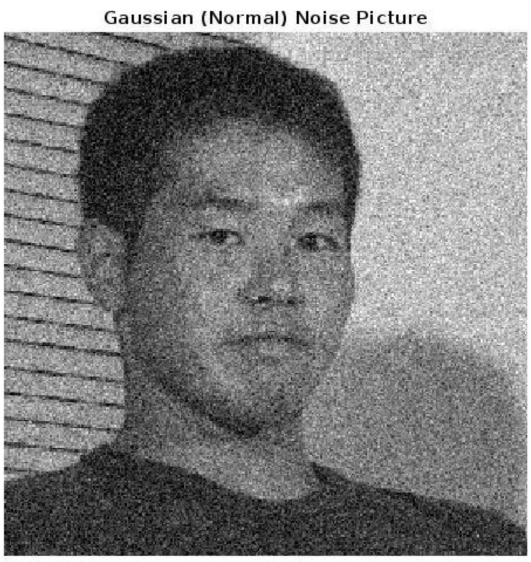
First read the image, and then use the IMNOISE function built into MATLAB to add the corresponding noise.

5. Resulting images.









Quantization Noise Picture

Part2 Low-pass Filtering

1.Arithmetic Mean Filter

This filter averages the pixel intensity value over the neighborhood using a mask with the same coefficients, for example, for a mask with the size 3 × 3 the coefficients are 1/9, if 5 × 5 — 1/25. Thanks to this 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:

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

2. Geometric Mean Filter

Geometric averaging is calculated using the formula:

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

The effect of applying this filter is similar to the previous method, but individual objects in the original image are less distorted. The filter can be used to suppress high frequency additive noise with better statistical performance than an arithmetic averaging filter.

3. Harmonic Mean Filter

The filter is based on the expression:

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

The filter suppresses «salt» noise well and does not work with «pepper» noise.

4. Counterharmonic Mean Filter

The filter is based on the expression:

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

where — filter order. The counterharmonic filter is a generalization of the averaging filters and for 0 suppresses noises like «pepper», and if 0 — noises like «salt», however, it is not possible to remove white and black points at the same time.

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. During images filtering, a two-dimensional Gaussian filter is used:

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

The larger the parameter , the more the image is blurred. Typi cally the filter radius $r = 3\sigma$. In this case, the mask size $2r+1 \times 2r+1$ and the matrix size is $6\sigma+1 \times 6\sigma+1$. Outside this neighborhood, the values of the Gaussian function will be negligible. In the MATLAB environment, filtering an image with a Gaussian filter can be performed using the function imgaussfilt(I). OpenCV also provides a function for Gaussian filter called cv::GaussianBlur(I, I_out) in C++ and cv2.cv::GaussianBlur(I) in Python .

2. Original images;

3. Code of the scripts;

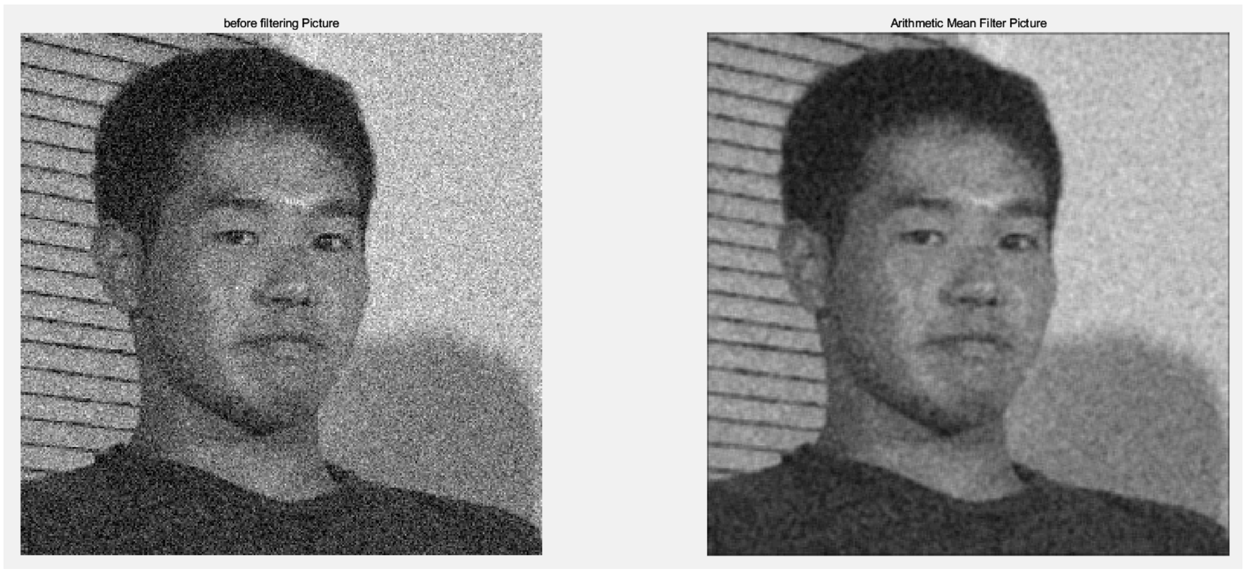
```
%% Images Filtering
clc;
close all;
clear;

X=imread('yjsp.jpg');
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\Multiplicative 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\Quantization Noise
picture.png');%Quantization noise picture as example

%run this after running p3.m
I_imp_salt=imread('Nonlinear filtering\Adaptive Median Filtering
picture.png');

%% Low-pass Filtering
%% 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');
%% 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');
```

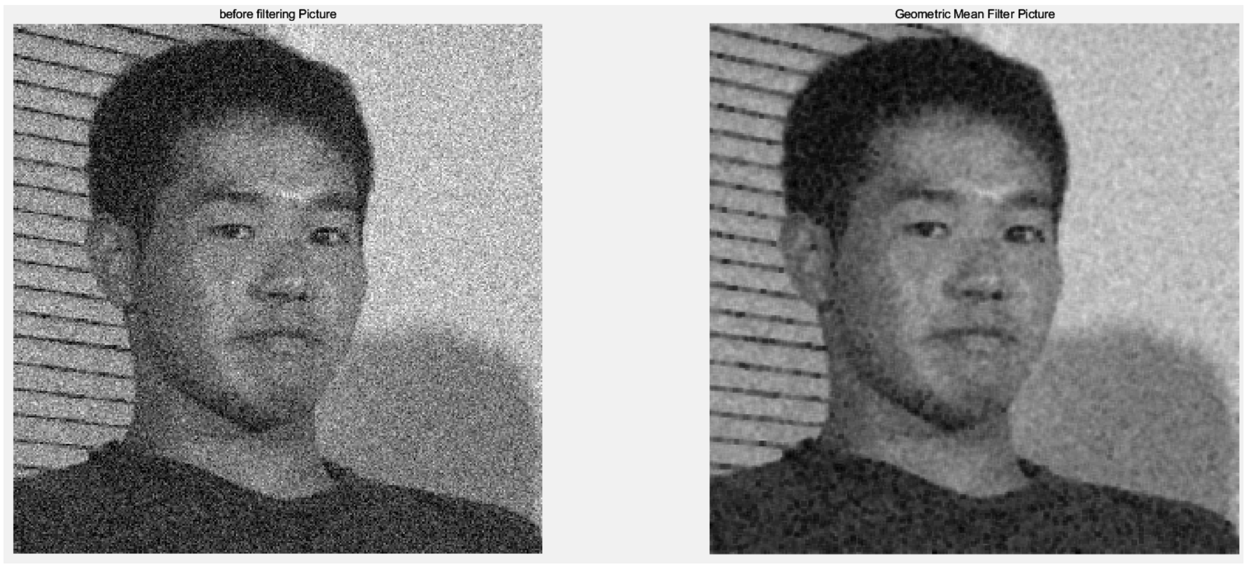
```
saveas(3,'Low-pass Filtering\compare Geometric Mean Filter picture.png');

%% Harmonic Mean Filter

I_extra2=ext(I_imp_salt,2);% 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');

%% Counterharmonic Mean Filter

I_extra3=ext(I_Normal,2);
Q=0;
%Q > 0 suppresses noises like «pepper»,
%and if Q < 0 — noises like «salt», however, it is not possible to remove
%white and black points at the same time. If Q = 0 the filter turns into
%an arithmetic one, and if Q = -1 — to harmonic.

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
```

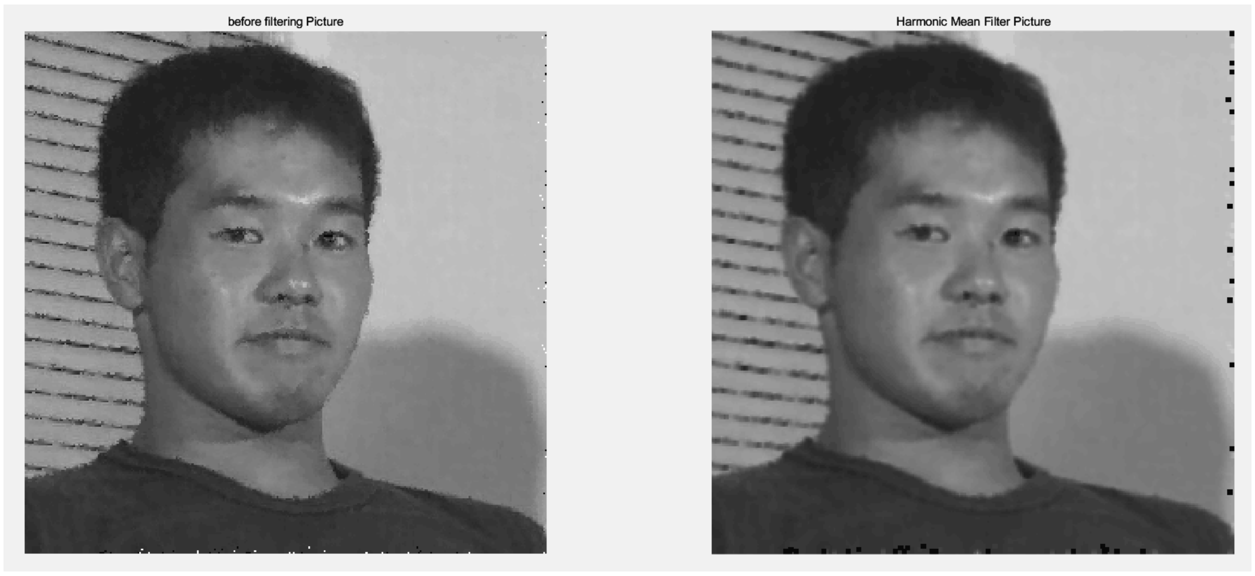
```
I_Counterharmonic=uint8(I_Counterharmonic);

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');

%% 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');
%% 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
```

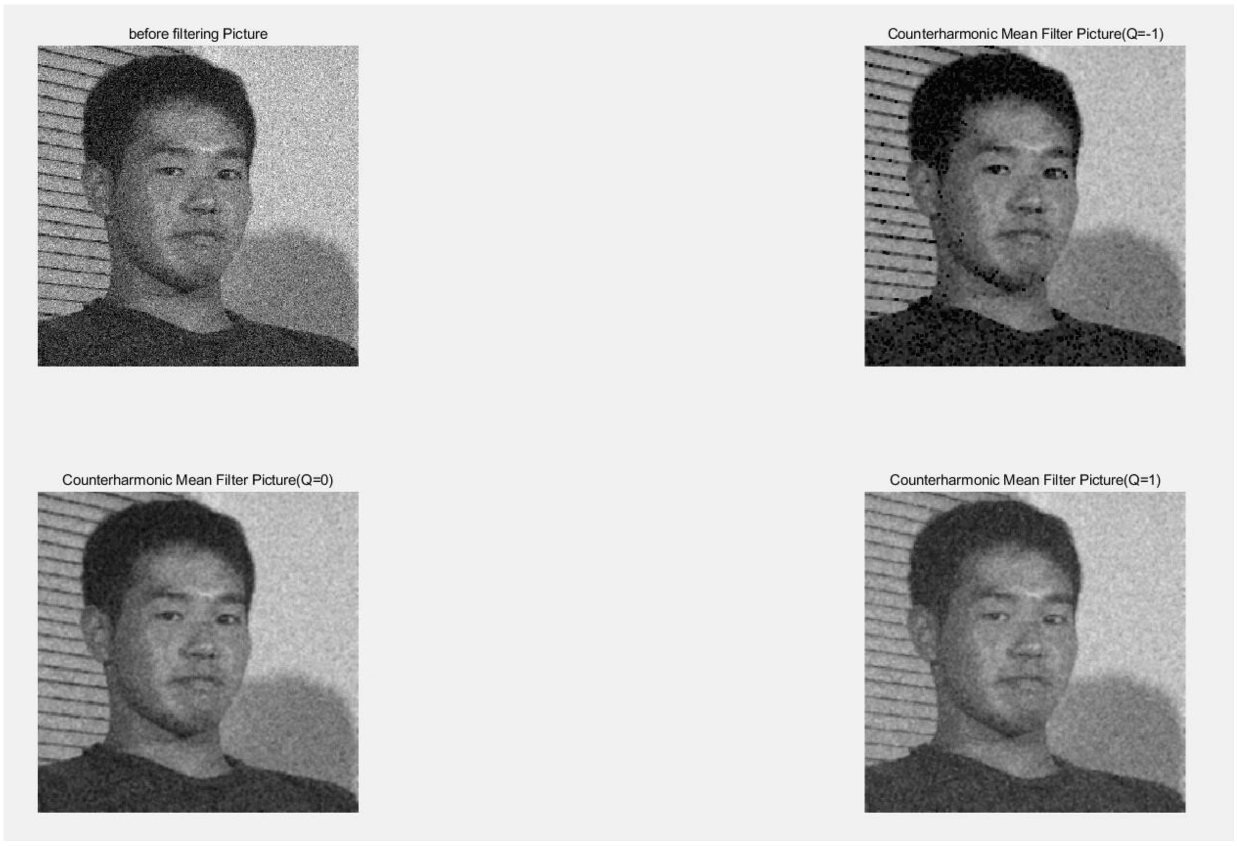
4. Comments: First, read the Impulse noise, Additive noise, Multiplicative noise, Gaussian (Normal) noise and Quantization noise generated in the previous section, and then use a number of different methods to filter them.

5. Resulting images.




before filtering Picture
Geometric Mean Filter Picture


before filtering Picture
Harmonic Mean Filter Picture


before filtering Picture
Counterharmonic Mean Filter Picture(Q=-1)
Counterharmonic Mean Filter Picture(Q=0)
Counterharmonic Mean Filter Picture(Q=1)

Part3

Nonliner Filtering

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

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

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.

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

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.

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

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%.

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

5. **Wiener Filtering**

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

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

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

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

2. Original images;

4. Code of the scripts;

```
%% Nonlinear Filtering
clc;
close all;
clear;

X=imread('yjsp.jpg');
I=rgb2gray(X); %I is an Gray image
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\Multiplicative 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\Quantization Noise
picture.png');%Quantization noise picture as example
[numRows,numCols]=size(I);

%% 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');

%% 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');

%% Adaptive Median Filtering
S_ori=2;%initial size of window(from 0)
S_max=10;
```

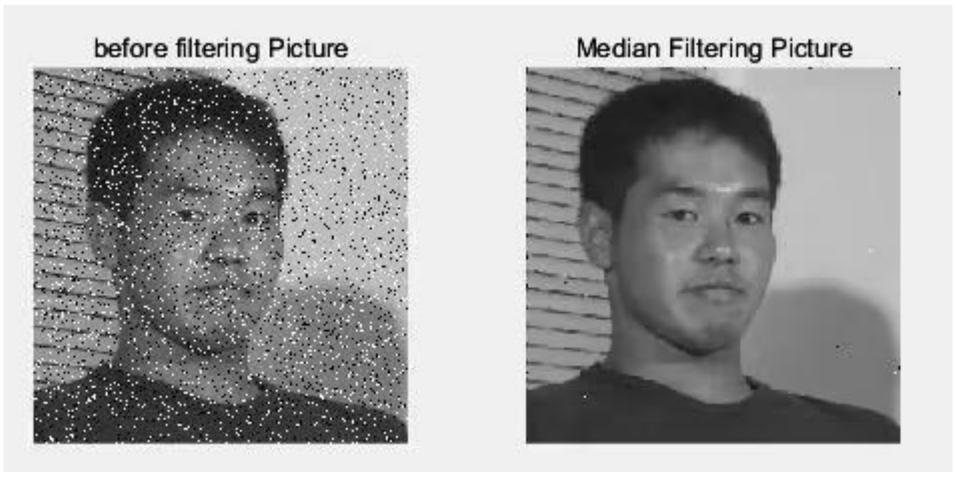
```
%the maxium size of window could be determined by yourself,which I choose 10.
%the bigger constant you choose,the longer time you run
I_ori3=I_Impulse;
I_extra=ext(I_ori3,S_ori);
I_adaptive=zeros(numRows,numCols);
A1=0;A2=0;
c=0;d=0;e=0;%debug counters
for x=1:1:numRows
    for y=1:1:numCols
        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);
```

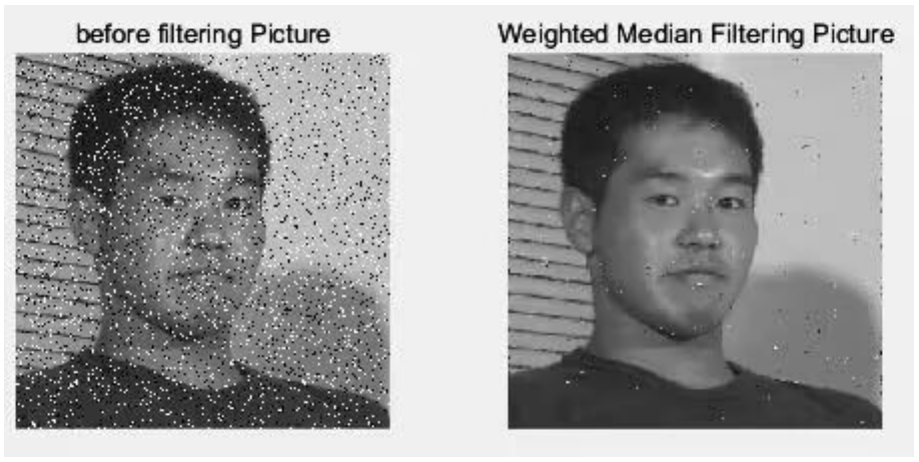
```
figure(4);
subplot(1,2,1);
imshow(I_ori3);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_adaptive);
title('adaptive Filtering Picture');
imwrite(I_adaptive,'Nonlinear filtering\Adaptive Median Filtering
picture.png');
saveas(4,'Nonlinear filtering\compare Adaptive Median Filtering picture.png');
%% 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');
%% Wiener Filtering

I_ori5 = I_Impulse;
m=8;n=8;
I_Wiener = wiener2(I_ori5,[m n]);
```

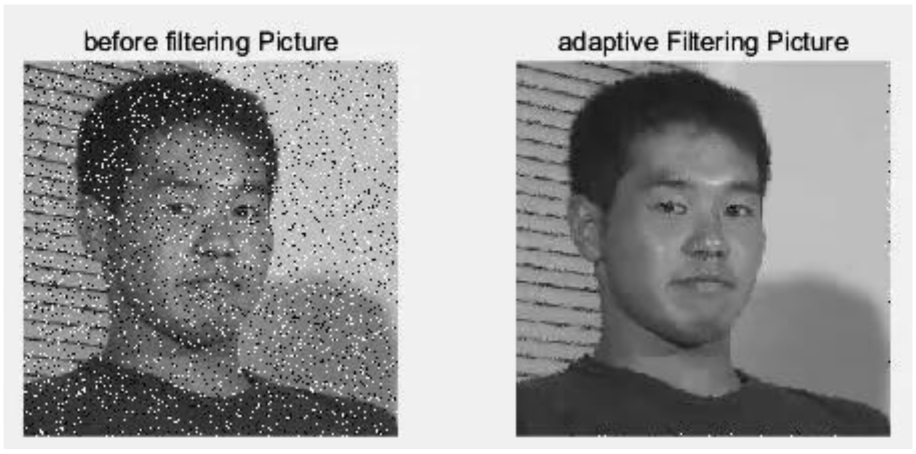
```
figure(6);
subplot(1,2,1);
imshow(I_ori5);
title('before filtering Picture');
subplot(1,2,2);
imshow(I_Wiener);
title('Wiener Filtering');
imwrite(I_Wiener,'Nonlinear filtering\Wiener Filtering picture.png');
saveas(6,'Nonlinear filtering\compare Wiener Filtering picture.png');



%% 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
```

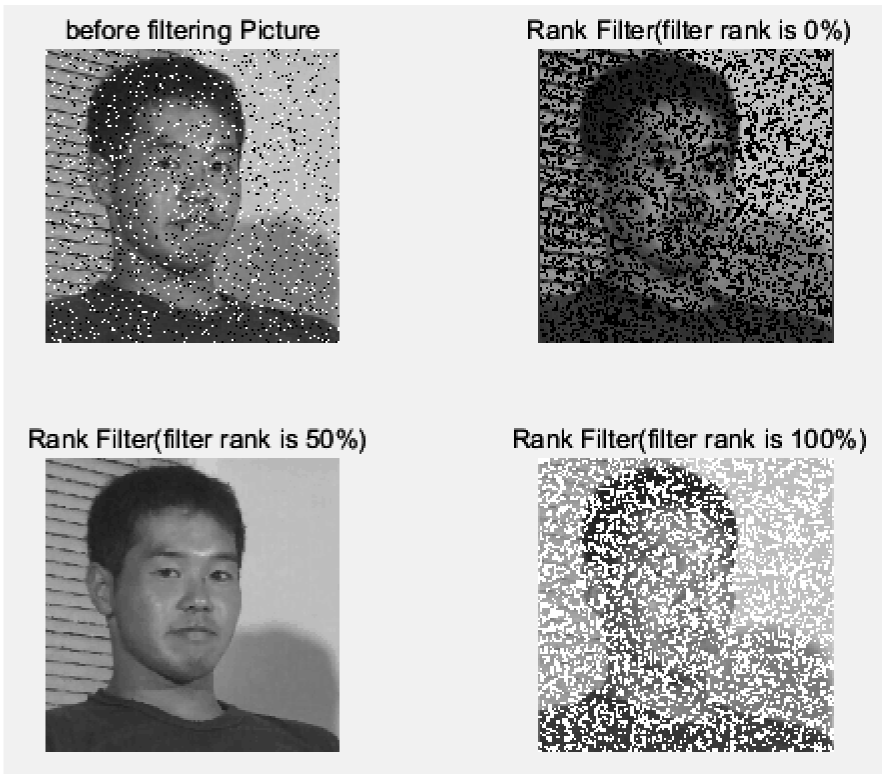
4. Comments; First, read the Impulse noise, Additive noise, Multiplicative noise, Gaussian (Normal) noise, and Quantization noise generated in the first part, and then use a number of different methods to operate the corresponding nonlinear filter.

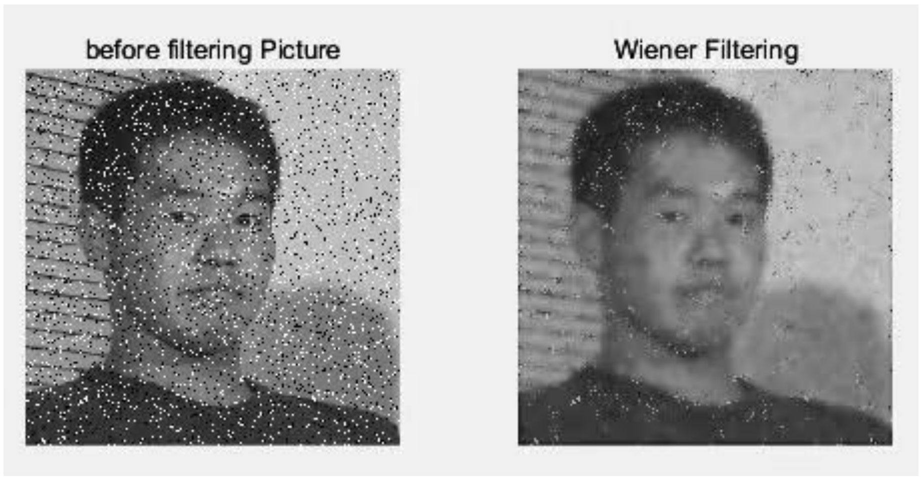
5. Resulting images.


before filtering Picture
Median Filtering Picture


before filtering Picture
Weighted Median Filtering Picture


before filtering Picture
adaptive Filtering Picture


before filtering Picture
Rank Filter(filter rank is 0%)
Rank Filter(filter rank is 50%)
Rank Filter(filter rank is 100%)



Part4 High-pass Filtering

1. Roberts Filter

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.

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

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

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

72.4.2

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.

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

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.

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

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.

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

5. Canny Algorithm

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

2. Original images;

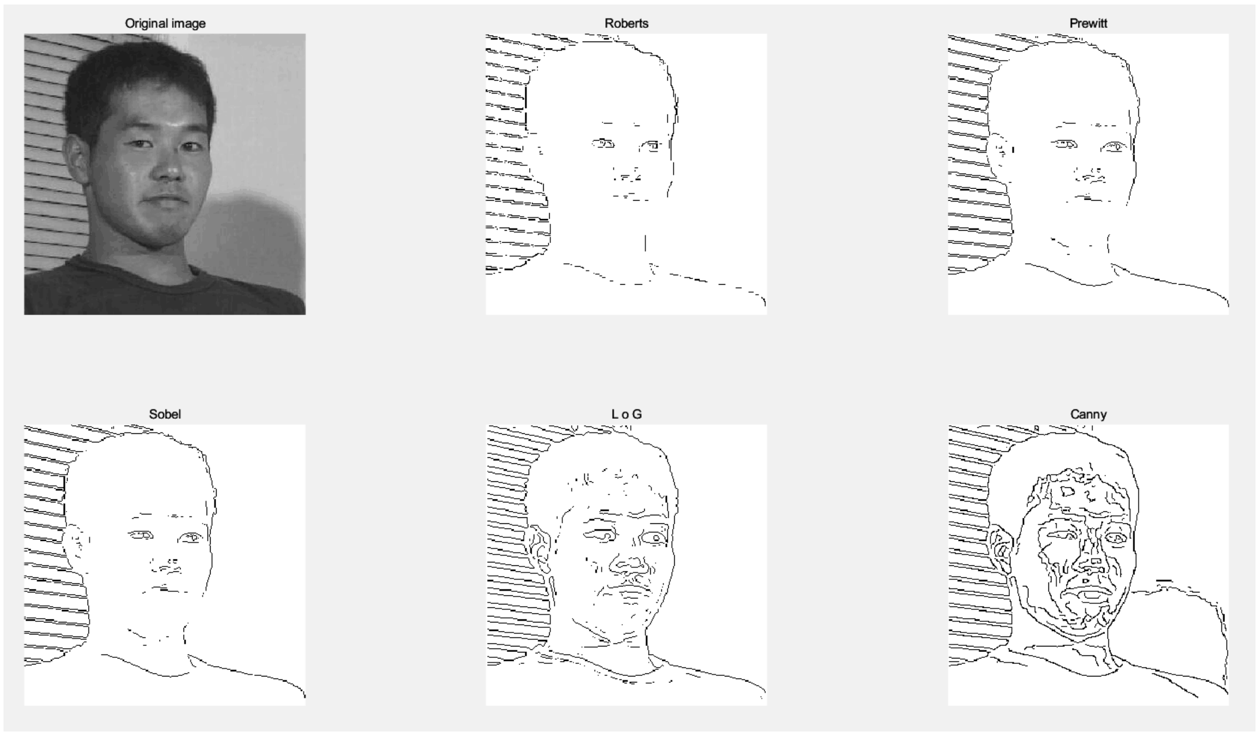
3. Code of the scripts;

```
clc
clear
I = imread('yjsp.jpg');
I = rgb2gray(I);

figure

subplot(2,3,1);
imshow(I);
title('Original image');

Iroberts = edge(I,'Roberts');
subplot(2,3,2);
%figure
imshow(~Iroberts);
title('Roberts');

Ipr = edge(I,'Prewitt');
subplot(2,3,3);
%figure
imshow(~Ipr);
title('Prewitt');

Isobel = edge(I,'Sobel');
subplot(2,3,4);
%figure
imshow(~Isobel);
```

```
title('Sobel');

Ilog = edge(I,'log');
subplot(2,3,5);
%figure
imshow(~Ilog);
title('L o G');

Iedge = edge(I,'Canny');
subplot(2,3,6);
%figure
imshow(~Iedge);
title('Canny');
```

4. Comments; Read the image and use the MATLAB built-in edge function to perform the corresponding High-pass Filter operation on the image to generate the corresponding image.

5. Resulting images.



Conclusion

Through this exercise, we learned how to add various types of noise to the image, such as impulse noise, additive multiplicative noise, etc. I also learned how to process noisy images with low pass filters (linear and nonlinear).

The mask of linear filter is a matrix, which can be divided into arithmetic average filter, geometric average filter, harmonic average filter, anti-harmonic average filter and Gaussian filter.

The nonlinear filter is divided into median filter, weighted median filter, adaptive median filter, rank filter and Vienna filter. These nonlinear filters filter the maximum and minimum values in the window, rather than distributing them evenly

across the surrounding pixels. Nonlinear filters work well for pulsed noise, but poorly for uniform noise.
In the process of learning and practicing high-pass filter operators, we know Roberts operator, Prewitt operator, Prewitt operator, Robert operator, Sobel operator, Laplacian operator and Canny operator, among which the Canny operator is the best.

Questions to Practical Assignment Report Defense

1. What are the main disadvantages of adaptive image filtering methods?

Other parameters of variable step size need to be determined by experiment, so it is not convenient to use.

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

When Q value is 0, the inverse harmonic filter bec omes arithmetic mean filter. When Q is -1, the inv erse harmonic mean filter reverts to the harmonic m ean filter.

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

Roberts operator, Prewitt operator, Sobel operator, C anny operator, LOG operator

4. Why, as a rule, is low-frequency filtering performed at the first step of edge detection?

Edge detection algorithms are mainly based on the f irst and second derivatives of image intensity, but the derivatives are usually very sensitive to noise, so filters must be used to improve the performanc e of edge detectors related to noise.

.