## 【Experiment Name】

# Lab 5 color image processing, facial image processing

## 【Purpose】

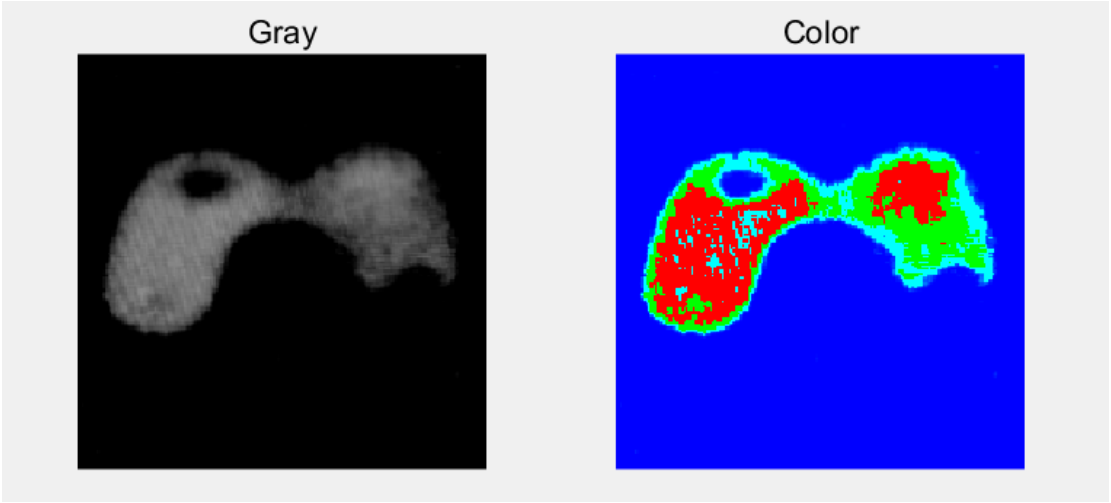
Master the color image processing and face image processing methods in digital image processing based on matlab .

## 【Experimental content】

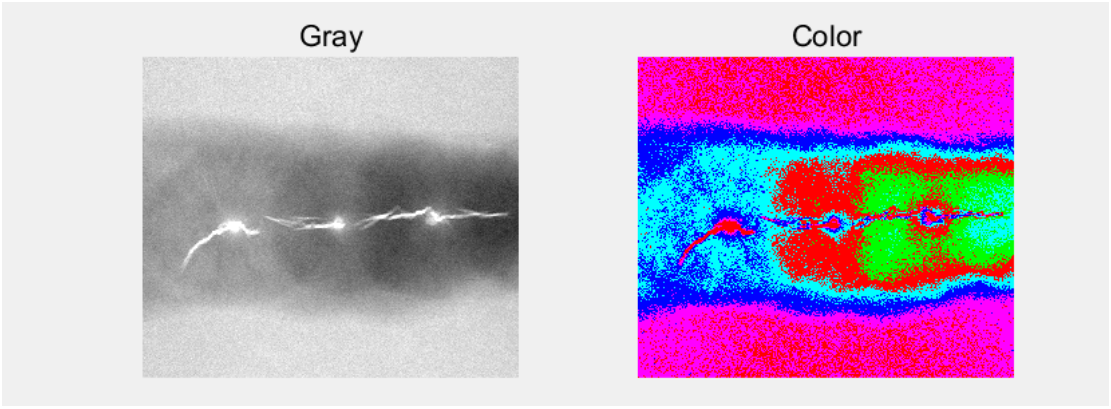
Use the given image to complete the method of image frequency domain filter and median filter .

| PRO JECT 06 |
|---|

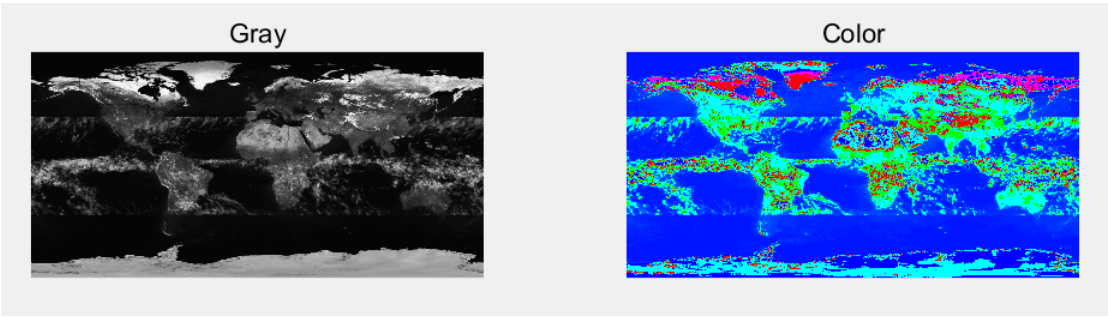
picker_phantom.tif :




weld- original.tif :




tropical_rain_grayscale.tif




PRO JECT Final _

In this experiment, based on the requirements of the experiment, I carefully realized the whole process of step0~3 . And achieved ideal experimental results as follows:


Specific analysis:

①s step1: corresponding to the second picture

Grayscale image conversion part:

```
%% 灰度图像转换
G=uint8(zeros(row,col));    %新灰度图像存储在G中
for i=1:row
    for j=1:col
        G(i,j)=0.3*I(i,j,1)+0.59*I(i,j,2)+ 0.11*I(i,j,3);    %灰度转换公式
    end
end
subplot(showX,showY,2);imshow(G);title('Gray image')
disp('灰度图像转换完成')

```

②s step2: corresponding to the 3rd and 4th pictures

The third picture is the result of my salt and pepper noise generation function.

The fourth picture is the noise generation result of calling imnoise , which is used for comparison with the third picture

Salt and pepper noise generation function: (p 1=p2=0.1 )

```
%% 椒盐噪声函数（输入：需操作的图像image，概率值p1,p2）
function [result]=SaltPepper(image,p1,p2)
[width,height]=size(image);
result=image;
kern1=rand(width,height)<p1;
kern2=rand(width,height)<p2;
result(kern1&kern2)=0;
result(kern1&~kern2)=255;
disp('椒盐噪声处理完成');
end
```

②s step3: corresponding to the 5th, 6th, 7th, and 8th pictures

According to the theoretical guidance in the textbook, I implemented 4 filters and displayed the filtering effect.

Arithmetic mean filter, modified alpha mean filter function:

Because the implementation methods are very similar, I directly use the modified alpha mean filter to realize the arithmetic mean filter (d = 0 ) and the modified alpha mean filter function of d = 5 at the same time.

```
%% 修正的alpha均值滤波器(输入：需操作的图像SP，滤波器边长ff，修正值d)
function [result]=fl(image,ff,d)
[x,y]=size(image);
result=uint8(zeros(x,y));
pro=funcPadding(image,ff);
all=0;
for i=1:x
    for j=1:y
        for a=i:i+(ff-1)
            for b=j:j+(ff-1)
                all=all+pro(a,b)/(ff^2-d);
            end
        end
        result(i,j)=uint8(all);
        all=0;
    end
end
if d==0
    disp('算数均值滤波完成');
else
    disp('修正的alpha滤波完成');
end
end
```

Geometric mean filter function:

```
%% 几何均值滤波器(输入：需操作的图像SP，滤波器边长ff)
function [result]=f2(image,ff)
n = ff; %模板大小
[M,N]=size(image);
f21 = image + uint8(ones(M,N)); %防止有个像素点为0而导致乘积为0
f12 = f21;
for i = 1:M-n+1
    for j = 1:N-n+1
        g2 = f21(i:i+n-1,j:j+n-1);
        s2 = prod(g2(:));
        %中心点的值用子图像的几何均值代替
        f12(i+(n-1)/2,j+(n-1)/2) = s2.^(1/numel(g2));
    end
end
img2 = uint8(f12);
result=img2;
disp('几何滤波完成')
end
```

Median filter function:

```
%% 中值滤波器(输入：需操作的图像SP，滤波器边长ff)
function [result]=f3(image,ff)
[x,y]=size(image);
% 分配空间
list=uint8(1:ff);   %用于从滤波器获取中值
result=uint8(zeros(x,y));
% pro的padding处理
pro=funcPadding(image,ff);
% 中值滤波
f=uint8(1);
for i = 1:x
    for j = 1:y
        for a=i:i+(ff-1)
            for b=j:j+(ff-1)
                list(f)=pro(a,b);
                f=f+1;
            end
        end
        % 冒泡排序法取中值
        for b=1:ff^2-2
            for a=b:ff^2-1
                if list(a)>list(a+1)
                    temp=list(a+1);
                    list(a+1)=list(a);
                    list(a)=temp;
                end
            end
        end
        f=uint8(1);
        result(i,j)=list(uint8(ff^2/2)+1);
    end
end
disp('中值滤波完成');
end
```

Finally, display the running status of the code at any time in the command line window:

```
命令行窗口
    灰度图像转换完成
    椒盐噪声处理完成
    算数均值滤波完成
    几何滤波完成
    中值滤波完成
    修正的alpha滤波完成
fx >>
```

## 【Experiment Summary】

In this experiment, I learned the color image processing and face image processing methods in digital image processing based on matlab . The first half of the experiment mainly examines the understanding of the color image processing part of Chapter 6. First convert the color image to a grayscale image, which can be better achieved based on the given formula. Next, I developed a color conversion method by myself, and successfully converted the grayscale image into a color image, making the image details more obvious. In the final face processing experiment, I successfully implemented three experimental steps in sequence according to the experimental process, and generated an ideal image. Through this experiment, I have deepened my code and application ability well, laid a good foundation for future experiments, enhanced my understanding of color image processing and face image processing methods, and improved my understanding of digital image processing this semester . Ability to integrate technology .

appendix:

PROJECT 04

```
%% PROJECT 04
close all;
clear all;
clc ;

%% Get image and parameters
showX = 3;
```

```
showY =6;
I = imread (' Fig_characters_test_pattern.tif ');
subplot( showX,showY ,1+showY*0); imshow (I); title('Original');
subplot( showX,showY ,1+showY*1); imshow (I); title('Original');
subplot( showX,showY ,1+showY*2); imshow (I); title('Original');
D0=80;
N=2;
alf = [ 5,15,30,80,230];


%% Call the filter function and display the result
for k= 1:showY
D0= alf (k);
result = Bfilter ( I,D 0,N,1);
subplot( showX,showY ,k+showY *0+1); imshow (result); title('ILPF');
result = Bfilter ( I,D 0,N,2);
subplot( showX,showY ,k+showY *1+1); imshow (result); title('BLPF');
result = Bfilter ( I,D 0,N,3);
subplot( showX,showY ,k+showY *2+1); imshow (result); title('GLPF');
end

%% frequency domain filter
function [ image_out ] = Bfilter ( image_in , D0, N, choose)
% filter in the frequency domain
% The input is the grayscale image that needs to be filtered, the cut-off frequency D0, the order N of the Butterworth filter
% The output is the grayscale image after filtering

[m, n] = size( image_in );
P = 2*m;
Q = 2 * n;

fp = zeros( P, Q);
% fill the image with 0, and multiply by (-1)^( x+y ) to move to the transform center
for i = 1 : m
for j = 1 : n
        fp ( i , j) = double( image_in ( i , j)) * (-1)^( i+j );
end
end
% Perform Fourier transform on the filled image
F1 = fft2( fp );

% Generate a Butterworth filter function, centered at (m+1,n+1)
if choose==1
    Bw =ILPF(D 0,P ,Q,m,n);
elseif choose==2
    Bw =BLPF(D 0,N ,P,Q,m,n);
elseif choose==3
    Bw =GLPF(D 0,P ,Q,m,n);
end

% to filter
G = F 1 . * Bw ;
% inverse Fourier transform
gp = ifft2(G);
% processed image
image_out = zeros( m, n, 'uint8');
gp = real( gp );
```

```
g = zeros( m, n);
for i = 1 : m
for j = 1 : n
            g( i , j) = gp ( i , j) * (-1)^( i+j );

end
end
mmax = max( g(:));
mmin = min( g(:));
range = mmax-mmin ;
for i = 1 : m
for j = 1 : n
            image_out ( i,j ) = uint8(255 * (g( i , j)- mmin ) / range);
end
end
end

%% ideal low-pass filter function
function[ Bw ]=ILPF(D0,P,Q,m,n)
      Bw = zeros( P, Q);
for u = 1 : P
for v = 1 : Q
temp = ((u-(m+1.0 ))^ 2 + (v-(n+1.0))^2)^(1/2);
if temp<=D0
                        Bw ( u,v )=1;
end
end
end
end
%% Butterworth filter function
function[ Bw ]=BLPF(D0,N,P,Q,m,n)
      Bw = zeros( P, Q);
a = D0 ^( 2 * N);
for u = 1 : P
for v = 1 : Q
temp = (u-(m+1.0 ))^ 2 + (v-(n+1.0))^2;
                  Bw ( u, v) = 1 / (1 + ( temp^N ) / a);
end
end
end
%% Gaussian low- pass filter function
function[ Bw ]=GLPF(D0,P,Q,m,n)
      Bw = zeros( P, Q);
a = 2*D0^2;
for u = 1 : P
for v = 1 : Q
temp = ((u-(m+1.0 ))^ 2 + (v-(n+1.0))^2)^(1/2);
                  Bw ( u, v) = exp(-temp^2/a);
end
end
end
```

```
% PROJECT 06-02
% I= imread (' picker_phantom.tif ');
I= imread ('weld- original.tif ');
% I= imread (' tropical_rain_grayscale.tif ');
subplot(1,2,1 ); imshow (I); title('Gray');
```

```
I=double(I);
[ m,n ]=size(I);
L=256;
for i =1:m
for j= 1:n
if I( i,j )<L/8
R( i,j )=0;
G( i,j )=4*I( i,j )+0*L;
B( i,j )=L;
elseif I( i,j )<=L/4
R( i,j )=0;
G( i,j )=L;
B( i,j )=-4*I( i,j )+2*L;
elseif I( i,j )<=3*L/8
R( i,j )=4*I( i,j )-2*L;
G( i,j )=L;
B( i,j )=0;
elseif I( i,j )<=L/2
R( i,j )=L;
G( i,j )=-4*I( i,j )+0*L;
B( i,j )=0;
elseif I( i,j )<=5*L/8
R( i,j )=0;
G( i,j )=L;
B( i,j )=8*I( i,j )+2*L;
elseif I( i,j )<=3*L/4
R( i,j )=0;
G( i,j )=-8*I( i,j )-2*L;
B( i,j )=L;
elseif I( i,j )<=7*L/8
R( i,j )=L;
G( i,j )=0;
B( i,j )=8*I( i,j )-2*L;
else
R( i,j )=L;
G( i,j )=-8*I( i,j )+2*L;
B( i,j )=0;
end
end
end
for i =1:m
for j= 1:n
Color( i,j ,1)=R( i,j );
Color( i,j ,2)=G( i,j );
Color( i,j ,3)=B( i,j );
end
end
Color=Color/256;
subplot(1,2,2 ); imshow (Color); title('Color');
```

P ROJECT Final

```
%% DIP Final Project
% initialization
close all;
clear all;
```

```
clc ;

% parameter, used for x,y of the display area
showX =2; % This project displays 8 pictures in total, 2x4 display
showY =4; % This project displays 8 pictures in total, 2x4 display

%% Get the original color image and its length, width and depth, the image size is 1024x1024x 3
I= imread (' yourID.tif ');
subplot( showX,showY ,1); imshow (I); title('Original image')
[ row, col, deep ]=size(I); % length, width, and depth respectively exist in row, col, deep

%% grayscale image conversion
G=uint8(zeros( row,col )); % The new grayscale image is stored in G
for i = 1: row
for j= 1:col
G( i,j )=0.3*I(i,j,1)+0.59*I(i,j,2)+ 0.11*I(i,j,3); % gray scale conversion formula
end
end
subplot( showX,showY ,2); imshow (G); title('Gray image')
disp ('Grayscale image conversion complete')

%% Part 2: Add salt and pepper noise
%parameter:
p1=0.1; % salt and pepper rate (horizontal)
p2=0.1; % salt and pepper rate (longitudinal)

% process and compare
SP= Salt Pepper ( G,p 1,p2);
subplot( showX,showY ,3); imshow (SP); title(' Salt&pepper noise');
SP_im = imnoise ( G, 'salt & pepper', 0.1);
subplot( showX,showY ,4); imshow ( SP_im ); title('Noise by imshow_Func ');
imshow ( SP);

%% Part III: Filtering
% parameter: filter side length ff
ff=5;

% The four filters are processed sequentially
result=f1(SP,ff,0); % arithmetic mean filtering (shares the same function as alpha)
subplot( showX,showY ,5); imshow (result); title('Arithmetic mean filter')
result=f2( SP,ff ); %geometric mean filtering
subplot( showX,showY ,6); imshow (result); title('Geometric mean filter')
result=f3( SP,ff ); % Median filtering
subplot( showX,showY ,7); imshow (result); title('Median filter')
result=f1(SP,ff,5); % corrected alpha mean filter
subplot( showX,showY ,8); imshow (result); title('Alpha-trimmed mean filter')

%% salt and pepper noise function (input: image image to be operated, probability values p1, p2)
function [result]= SaltPepper ( image,p 1,p2)
[ width,height ]=size(image);
result=image;
kern1=rand( width,height )<p1;
kern2=rand( width,height )<p2;
result(kern1&kern2 ) = 0;
result(kern1&~kern 2)= 255;
disp ('salt and pepper noise processing completed');
end
```

```
%% Corrected alpha mean filter (input: image SP to be operated, filter side length ff, correction value d)
function [result]=f1( image,ff ,d )
[ x,y ]=size(image);
result=uint8(zeros( x,y ));
pro= funcPadding ( image,ff );
all=0;
for i =1:x
for j= 1:y
for a= i:i +(ff-1)
for b= j:j +(ff-1)
                    all= all+pro ( a,b )/(ff^2-d);
end
end
result( i,j )=uint8(all);
all=0;
end
end
if d==0
    disp ('Arithmetic mean filter complete');
else
    disp ('Corrected alpha filter completed');
end
end

%% Geometric mean filter (input: image SP to be operated, filter side length ff)
function [result]=f2( image,ff )
n = ff; % template size
[ M,N ]=size(image);
f21 = image + uint8(ones(M,N)); % prevent a pixel from being 0 and resulting in a product of 0
f12 = f21;
for i = 1:M-n+1
for j = 1:N -n+1
g2 = f21( i:i +n-1,j:j+n-1);
s2 = prod( g2(:));
The value of the center point is replaced by the geometric mean of the subimage
f12( i +(n-1)/ 2,j +(n-1)/2) = s2.^(1/ numel (g2));
end
end
img2 = uint8(f12);
result=img2;
disp ('geometry filtering complete')
end

%% median filter (input: image SP to be operated, filter side length ff)
function [result]=f3( image,ff )
[ x,y ]=size(image);
% allocate space
list=uint8(1:ff); % is used to get the median value from the filter
result=uint8(zeros( x,y ));
%pro's padding processing
pro= funcPadding ( image,ff );
% median filter
f=uint8(1);
for i = 1:x
for j = 1:y
for a= i:i +(ff-1)
for b= j:j +(ff-1)
```

```
list(f)=pro( a,b );
f=f+1;
end
end
% The bubble sort method takes the median value
for b= 1:ff ^2-2
for a= b:ff ^2-1
if list(a)>list(a+1)
temp=list(a+1);
list(a+ 1) = list(a);
list(a)=temp;
end
end
end
f=uint8(1);
result( i,j )=list(uint8(ff^2/2)+1);
end
end
disp ('median filter completed');
end

%% padding function (input: image SP to be operated, filter side length ff)
function [result]= funcPadding ( image, ff)
[ x,y ]=size(image);
edge=ff/2-1;
result=uint8(zeros(x+ff- 1,y +ff-1));
for i =1:x
for j= 1:y
result(uint16( i+edge ),uint 16( j+edge ))=image( i,j );
end
end
% imshow (result);
% disp ( 'padding finished');
end
```