数字图像处理与分析 实验报告

实验一 图像几何变换

图像的平移

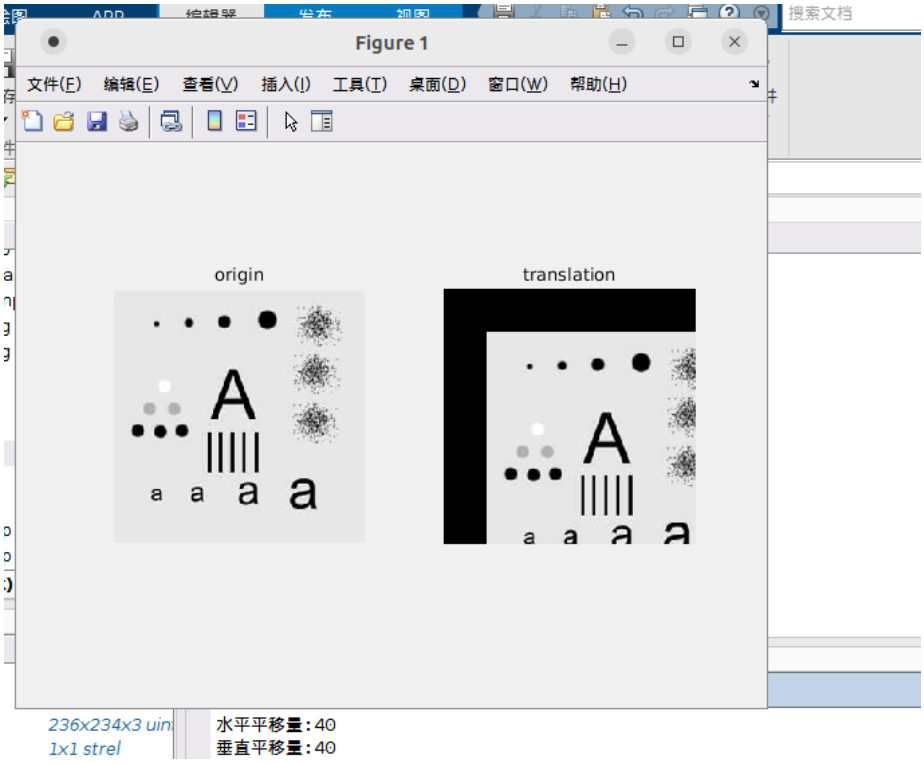
- imput函数来进行输入读入
- translate函数来生成一个基于输入位移大小的方式：

```
se = translate(strel(1), [ty, tx]);
```

- imdilate函数来对图像作变换：

```
i2 = imdilate(i1, se);
```

实验结果：（水平和垂直各自移动40个单位）



图像的旋转

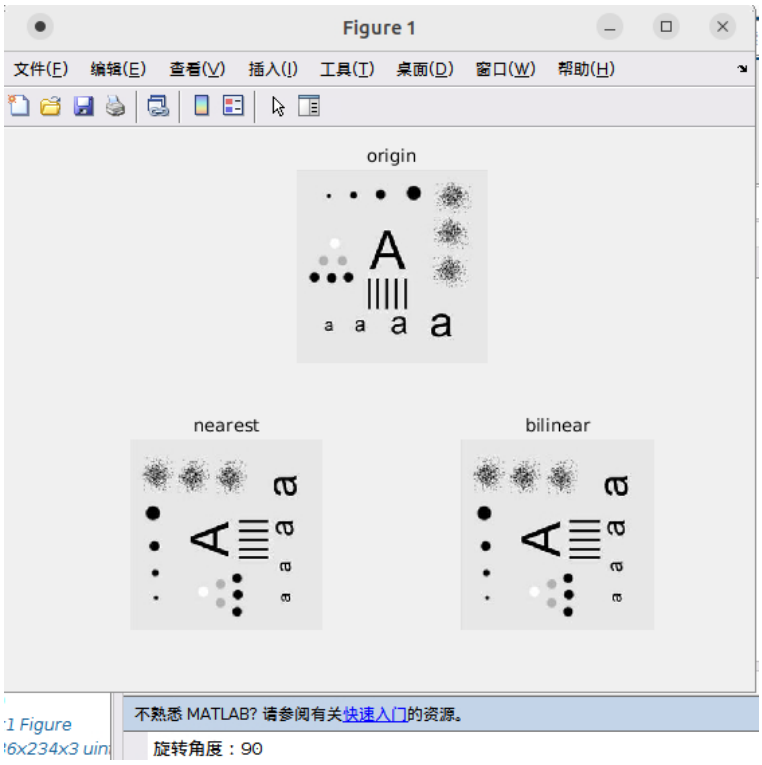
- imrotate函数实现旋转：

> 根据参数不同，分别采用最近邻插值和双线性插值方法

```
i2 = imrotate(i1, angle, 'nearest');
```

```
i3 = imrotate(i1, angle, 'bilinear')
```

实验结果：（旋转90度）



图像的缩放

先使用size获取行列像素个数

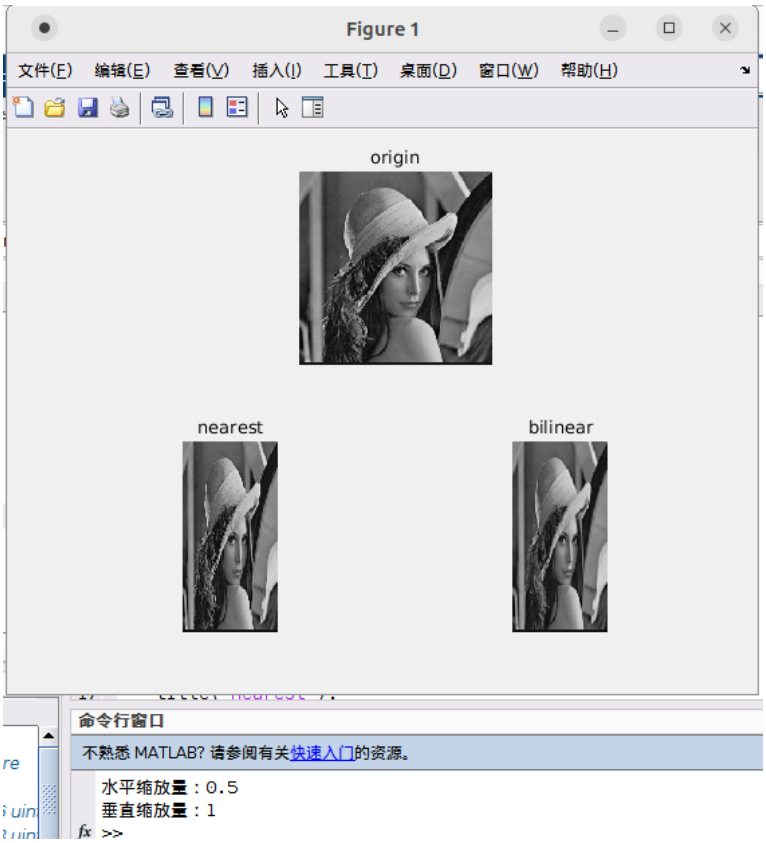
- imresize函数实现缩放

> 根据参数不同，分别采用最近邻插值和双线性插值方法

```
i2 = imresize(i1, [h*y,w*x], 'nearest'); % 最近邻插值
```

```
i3 = imresize(i1, [h*y,w*x], 'bilinear'); % 双线性插值
```

实验结果：（水平缩放0.5，垂直缩放1）

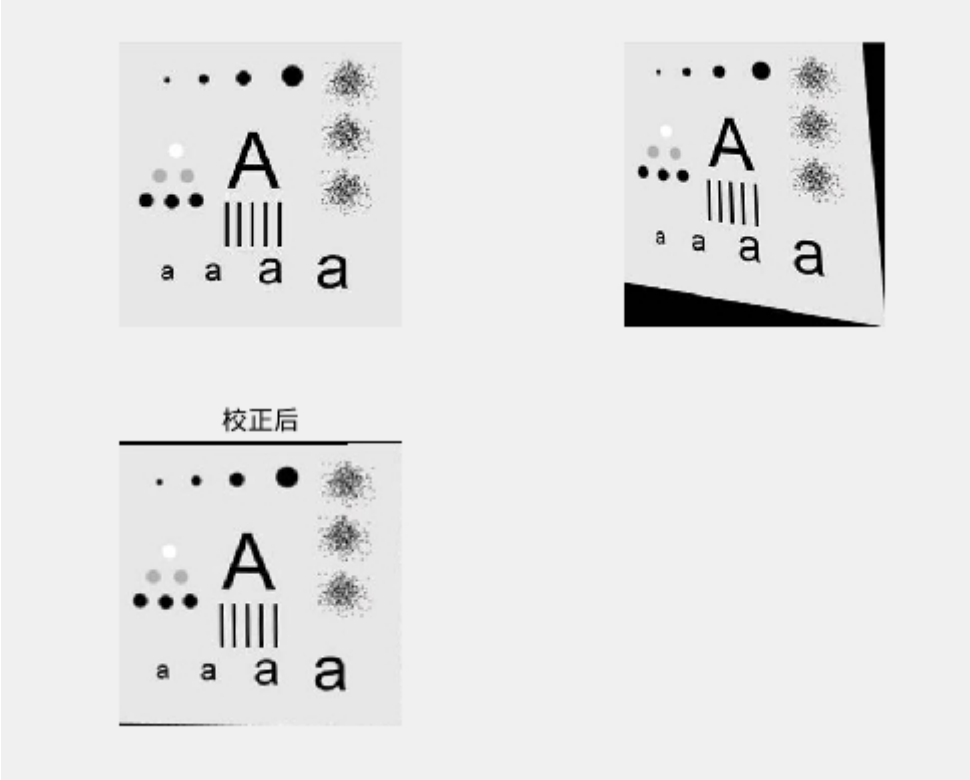


图像失真几何校正

- ginput函数生成8个点对
- 其中1，3，5，7设置为需要移动的点，2，4，6，8设置为修正后的点
- 使用fitgeotrans函数生成变换矩阵，并用imwarp进行修正

```
[x, y] = ginput(8);
moving = [x(1) y(1);x(3) y(3);x(5) y(5);x(7) y(7)];
fixed = [x(2) y(2);x(4) y(4);x(6) y(6);x(8) y(8)];
tform = fitgeotrans(moving,fixed,'projective');
res = imwarp(i2,tform,'OutputView',imref2d(size(i1)));
```

实验结果：（按照点对顺序点击两幅图对应位置）



实验二 图像点处理增强

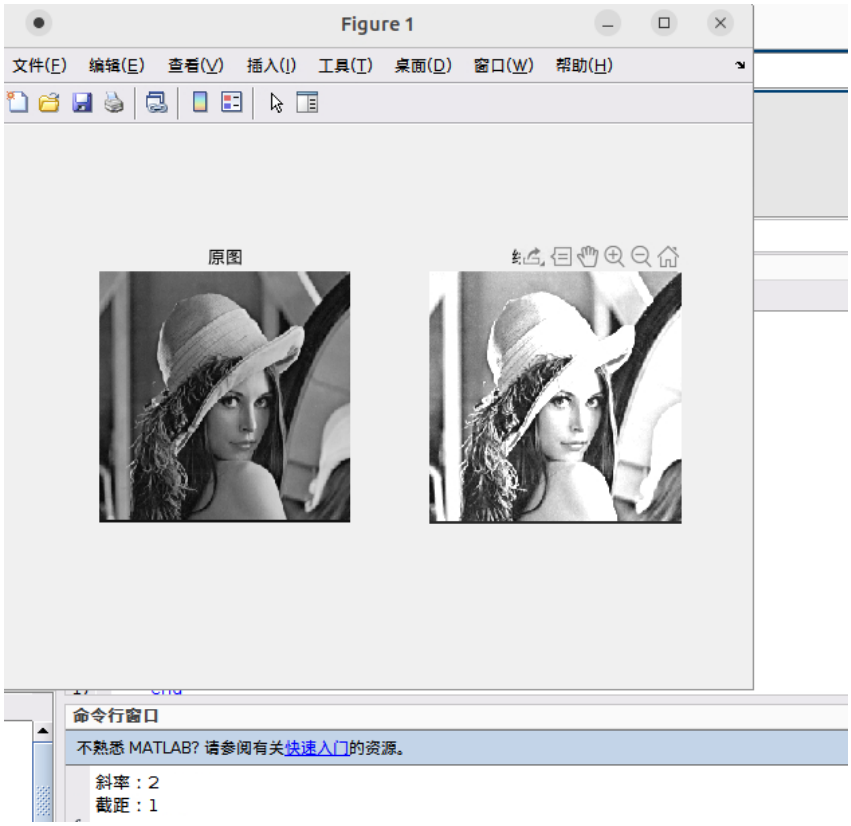
灰度线性变换

- $D_B = f(D_A) = f_A \cdot D_A + f_B$

  并对灰度范围做限制，控制在0-255之间

```
for i = 1 : h
    for j = 1 : w
        res(i,j) = src(i,j)*fa+fb;%线性映射
        if res(i,j) > 255
            res(i,j) = 255;
        elseif res(i,j) < 0
            res(i,j) = 0;
        end
    end
end
```

实验结果：（斜率2，截距2）



灰度拉伸

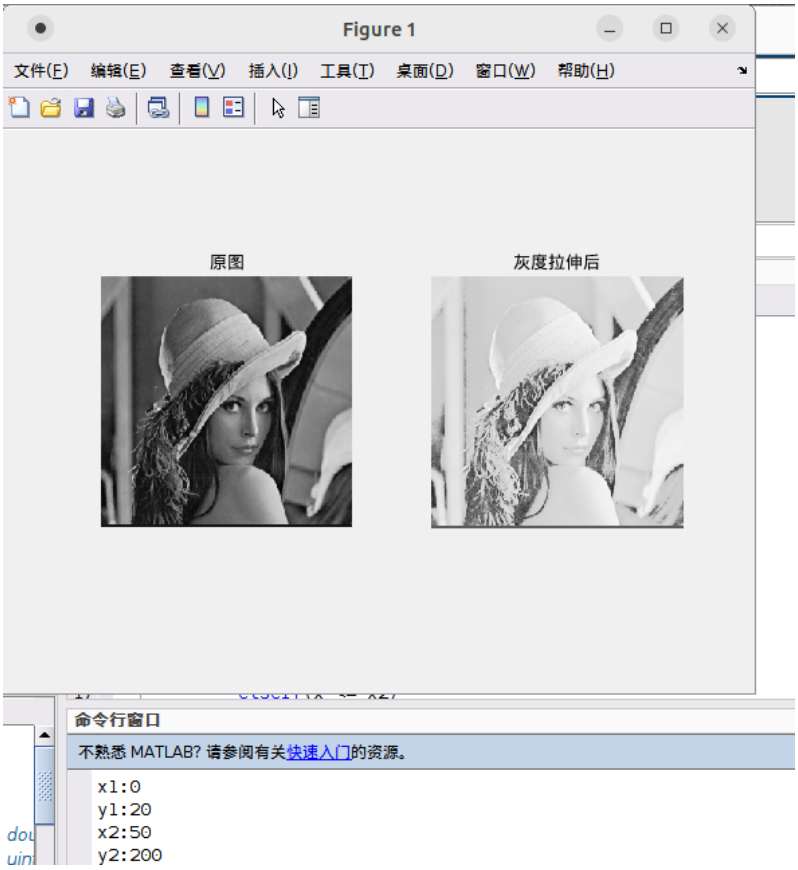
- 根据灰度拉伸的变换公式，完成灰度变换

  并且保证灰度范围在0-255

```
for i = 1 : h
    for j = 1 : w
        x=src(i,j);
```

```
        %灰度拉伸
        if(x < x1)
            fx = y1 / x1 * x;
        elseif(x <= x2)
            fx = (y2 - y1)/(x2 - x1)*(x - x1) + y1;
        else
            fx = (255 - y2)/(255 - x2)*(x - x2) + y2;
        end
        %保证范围
        if fx > 255
            fx = 255;
        elseif fx < 0
            fx = 0;
        end
        %赋值
        res(i,j)=fx;
    end
end
```

实验结果：（选取（0,20）和（50,200）两个点）

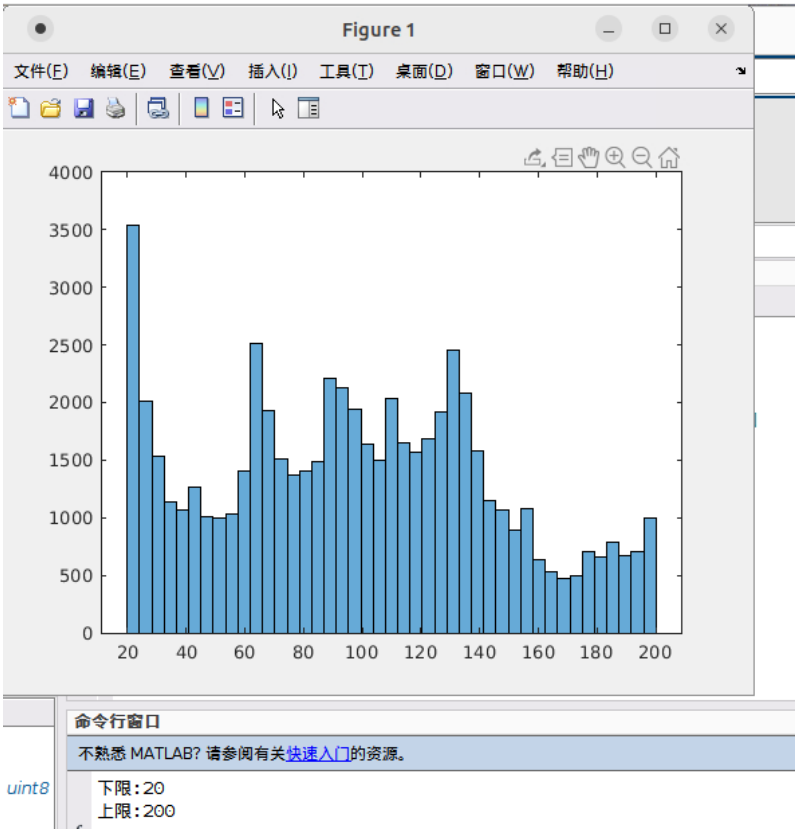


灰度直方图

- 输入上限和下限后，直接使用histogram函数显示灰度直方图

```
min = input("下限:");
max = input("上限:");
histogram(src,'BinLimits',[min,max])%显示给定范围的灰度直方图
```

实验结果：（范围在20-200）



直方图均衡

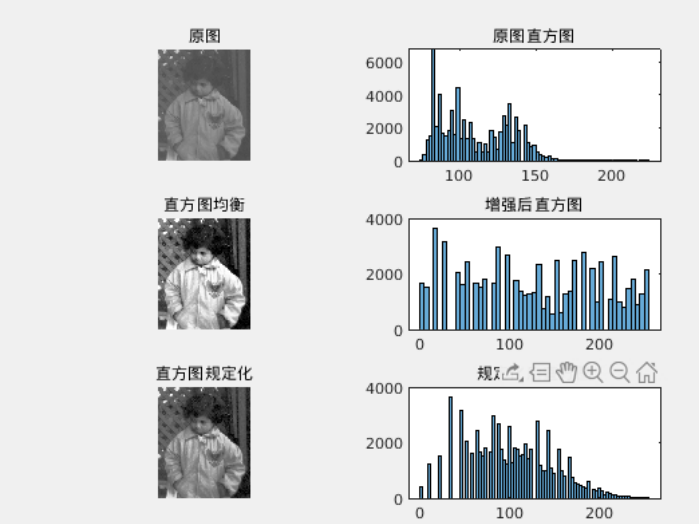
- histogram显示直方图
- histeq函数进行直方图均衡

```
res1 = histeq(src);%直方图均衡
```

- histeq传递更多参数，来进行规定化：

```
res2 = histeq(src,normpdf((0:1:255),100,50));
```

实验结果：



实验三 图像空间域滤波增强

均值滤波器

- imnoise函数为图像添加噪声

> 根据参数不同分别为3%椒盐噪声、高斯噪声、随机噪声

```
pepper = imnoise(src,'salt & pepper',0.03);
gaussian = imnoise(src,'gaussian');
random = imnoise(src,'speckle',0.05);
```

- imfilter函数实现均值滤波器去除图像中的噪声：

```
pepper_output = imfilter(pepper,fspecial('average',3));
gaussian_output = imfilter(gaussian,fspecial('average',3));
poisson_output = imfilter(random,fspecial('average',3));
```

实验结果：

超限邻域平均法

- 超限邻域平均法：

  使用mean求得某一个像素邻域的均值，如果该像素与均值的差值超过阈值，就会把这个点的像素设为邻域均值

```
function [output] = filter(input,T)
    output = input;
    [r,l] = size(output);
    for i = 2 : (r - 1)
        for j = 2 : (l - 1)
            mean_value = mean(mean(input(i - 1 : i + 1, j - 1 : j + 1)));
            if(abs(double(input(i,j)) - mean_value) > T)
                output(i,j) = mean_value;
            end
        end
    end
end
```

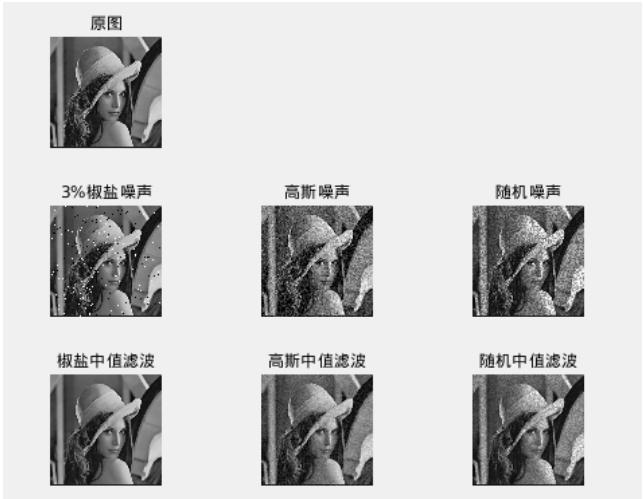
  实验结果：

中值滤波器

- medfilt2函数实现中值滤波，默认为3*3

```
pepper_output = medfilt2(pepper);
gaussian_output = medfilt2(gaussian);
speckle_output = medfilt2(random);
```

实验结果：



超限中值滤波器

- 超限中值滤波器

  使用median函数求得某个像素邻域的中位数，若该像素与邻域中位数的差值超过阈值，则把该像素设为邻域中值

  实现方法如下：

```
function [output] = filter(input,T)
    output = input;
    [r,l] = size(output);
    for i = 2 : (r - 1)
        for j = 2 : (l - 1)
            temp = input(i - 1 : i + 1, j - 1 : j + 1);
            middle_value = median(temp(:));
            if(abs(double(input(i,j)) - double(middle_value)) > T)
                output(i,j) = middle_value;
            end
        end
    end
end

```

实验结果：

四种滤波的比较

中值滤波对椒盐噪声的边缘处理好

均值滤波对随机噪声的边缘处理好

所有滤波法对高斯噪声的处理效果一般

边缘检测

以lena图为例

- edge函数进行边缘检测

> 根据参数不同设置不同算子

其中Roberts算子、Sobel算子、Prewitt算子、Canny 算子直接使用对应参数

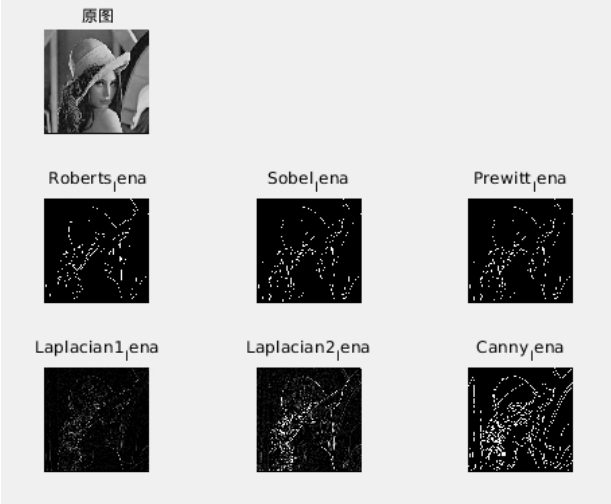
```
%Roberts算子
Roberts_lena = edge(src1,'Roberts');
%Sobel算子
Sobel_lena = edge(src1,'Sobel');
%Prewitt算子
Prewitt_lena = edge(src1,'Prewitt');
%Canny算子
Canny_lena = edge(src1,'Canny');

```

拉普拉斯算子使用给定的4邻域和8邻域参数：

```
%拉普拉斯算子
Laplacian1_lena = imfilter(src1,[0  1  0; 1  -4  1; 0  1  0]);
Laplacian2_lena = imfilter(src1,[-1 -1 -1; -1 8 -1; -1 -1 -1]);
```

实验结果：



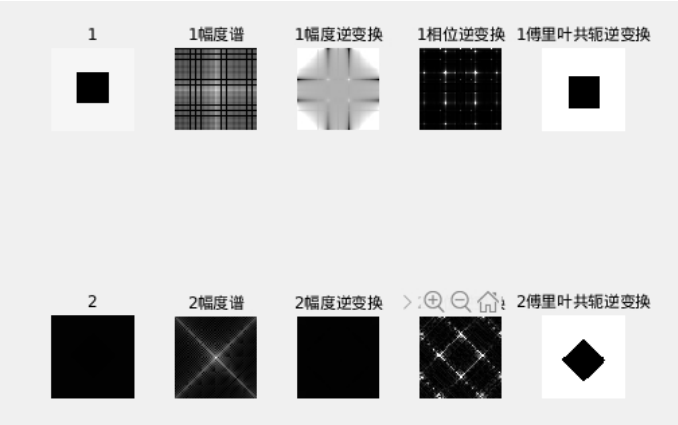
实验四 图像变换及频域滤波增强

傅里叶变换

- fft2函数进行傅里叶变换：
- fftshift将低频移到中心点，log进行低频增强
- ifft2函数和传入幅度进行幅度逆变换
- ifft2函数传入相位进行相位逆变换
- ifft2函数传入共轭进行共轭逆变换

```
%傅里叶变换
f1 = fft2(src1);
%移动低频至中心并增强
f1_s=fftshift(f1);%平移
f1_f = log(abs(f1_s)+1);%尺度变化,abs获取幅度，log增强低频
%幅度逆变换
ifabs1 = uint8(ifft2(abs(f1)));
%相位逆变换
ifangle1 = uint8(abs(ifft2(4999*exp(1i*angle(f1)))));
%共轭逆变换
conj1 = conj(f1);% 获取 F1 和 F2 的共轭
ifconj1 = ifft2(conj1);% 对新的复数数组进行反变换得到图像
```

实验结果：



> 评价人眼对图像幅频特性和相频特性的敏感度

幅频特性包含了图像亮度的分布，而相频特性刻画了图像的边界轮廓信息，人眼对相频特性比幅频特性敏感，可以直接看出图像的大致内容

低通滤波器

实现三种低通滤波器，所以函数传递参数需要选择滤波器类型type，通过switch来选择不同的处理方式

- 先根据参数个数区分出只需要三个参数的理想低通滤波器
- meshgrid函数构建坐标生成矩阵
- hypot函数计算坐标矩阵中各点到中心点的距离
- 根据传入的type使用switch语句进行不同的处理，生成算子
- 傅里叶变换并移位后将算子应用，然后归位并逆变换

```
function output = applyFilter(img, type, D0, n)
    if nargin < 4
        n = 1; % 默认阶数为1
    end
    [M, N] = size(img);
    % 构建坐标
    [U, V] = meshgrid(-N/2:N/2-1, -M/2:M/2-1);
    D = hypot(U, V); % 计算中心偏移距离, 生成矩阵

    % 生成滤波器掩膜
    switch type
        case 'ideal'
            H = D <= D0;
        case 'blpf'
            H = 1.0 ./ (1.0 + (D ./ D0) .^ (2 * n));
        case 'gaussian'
            H = exp(-(D ./ D0) .^ n);
        otherwise
            error('error type.');
    end

    % 傅里叶变换并移位
    F = fft2(img);
    F = fftshift(F);
```

```
    % 应用频域滤波器
    G = F .* H;
    % 移回并反变换
    G = ifftshift(G);
    output = abs(ifft2(G));
end
```

实验结果：

（截止频率3）

（截止频率25）

（截止频率40）

> 观察不同截止频率下采用不同低通 滤波器得到的图像与原图像的区别

截止频率较低时，图像最平滑，但细节保留很少，理想低通滤波的振铃效应明显

截止频率适中时，图像平滑并且细节保留较多，振铃效应适中

截止频率较高时，图像变化不大，振铃效应均不明显

低通滤波器去噪

和前一问的区别是先添加了噪声再进行低通滤波

```
% 加椒盐噪声和高斯噪声
sp = imnoise(girl, 'salt & pepper', 0.03);
gauss = imnoise(girl, 'gaussian');
% 理想低通滤波器
D0 = 50;
sp_ideal = applyFilter(sp, 'ideal', D0);
gauss_ideal = applyFilter(gauss, 'ideal', D0);

% 巴特沃斯低通滤波器
D0 = 50;
n = 1; % 阶数
sp_blpf = applyFilter(sp, 'blpf', D0, n);
gauss_blpf = applyFilter(gauss, 'blpf', D0, n);

% 高斯低通滤波器
D0 = 50;
n = 2;
sp_gauss = applyFilter(sp, 'gaussian', D0, n);
```

```
gauss_gauss = applyFilter(gauss, 'gaussian', D0, n);

```

实验结果：

> 对比去噪效果

理想低通滤波器的振铃效果较为明显，降噪效果较差

而巴特沃斯低通滤波和高斯低通滤波几乎没有振铃效应，并且降噪效果更好

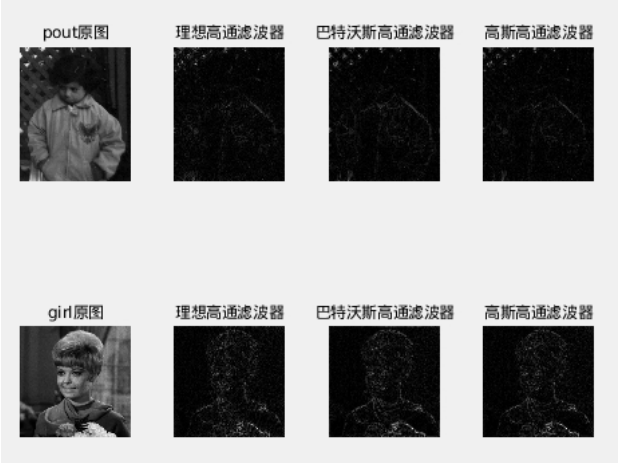
高通滤波器

对比低通滤波器，区别在于生成算子时公式不同：

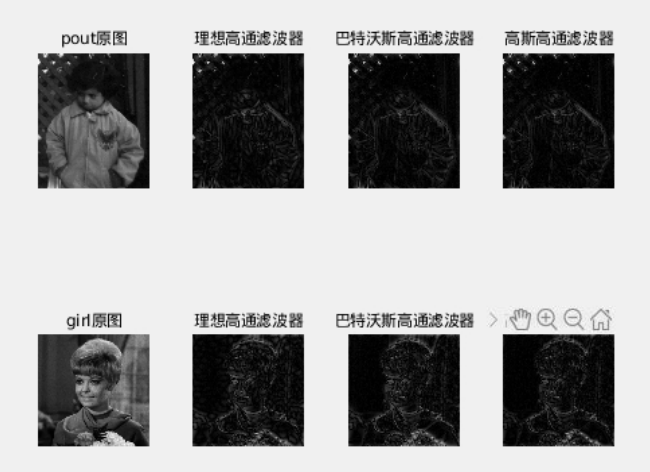
```
% 生成滤波器掩膜
  switch type
      case 'ideal'
          H = D > D0;
      case 'blpf'
          H = 1.0 ./ (1.0 + (D0./ D) .^ (2 * n));
      case 'gaussian'
          H = exp(-(D0./ D) .^ n);
      otherwise
          error('error type.');
  end
```

实验结果：

（截止频率40）



（截止频率10）



（截止频率1）

> 观察不同截止频率下采用不同高通滤波器得到的图像与原图像的区别

截止频率较高时，图像保留的低频信息较少，图像最暗，部分轮廓丢失，理想高通滤波有振铃效应

截止频率适中时，提取出高频的图像轮廓，有一定的振铃效应

截止频率较低时，大部分的细节被保留下来，振铃效应不明显

高频滤波增强与直方图均衡化

- 同样采用上一题的函数，改变直方图均衡化和应用高通滤波的顺序，如下：

```
% 理想高通滤波器
D0 = 15;
pout_ideal_histeq_after = histeq(uint8(applyFilter(pout, 'ideal', D0)));
pout_ideal_histeq_first = applyFilter(histeq(uint8(pout)), 'ideal', D0);

% 巴特沃斯高通滤波器
D0 = 15;
n = 1;
pout_blpf_histeq_after = histeq(uint8(applyFilter(pout, 'blpf', D0,n)));
pout_blpf_histeq_first = applyFilter(histeq(uint8(pout)), 'blpf', D0,n);

% 高斯高通滤波器
D0 = 15;
n = 2;
pout_gauss_histeq_after = histeq(uint8(applyFilter(pout, 'gaussian',
D0,n)));
pout_gauss_histeq_first = applyFilter(histeq(uint8(pout)), 'gaussian', D0,n
```

实验结果：

> 观察 对比不同处理顺序对结果图像的影响

- 先高频增强滤波再直方图均衡：

得到的图像较亮，因为先高频增强后能量普遍在高频，然后在使用直方图均衡使得灰度分布均匀就会使得整体变亮

- 先直方图均衡再高频增强滤波

得到的图像较暗，先直方图均衡会由于能量集中在低频，在高频增强滤波后会失去大部分能量，因此比较暗

实验五 图像恢复与图像分割

运动模糊与滤波恢复

- fspecial函数生成运动模糊滤波器，然后imfliter函数进行运动模糊

```
psf = fspecial('motion',30,45);
motion = imfilter(source,psf,'conv','circular');
```

- imnoise函数产生高斯噪声

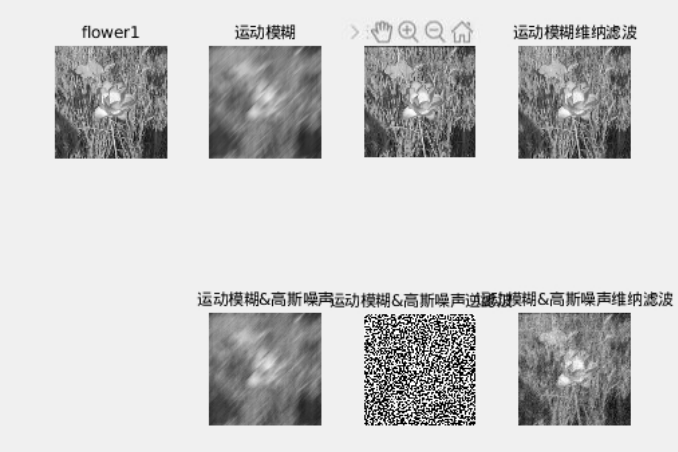
```
noisy = imnoise(motion,'gauss',0,0.0001);
```

- deconvwnr实现滤波

  根据参数不同分别实现逆滤波和维纳滤波

```
%运动模糊逆滤波和维纳滤波
deconvwnr(motion,psf)
deconvwnr(motion,psf,0.0001)
%运动模糊+高斯噪声逆滤波和维纳滤波
deconvwnr(noisy,psf)
deconvwnr(noisy,psf,0.0001/var(motion(:)))
```

实验结果：



> 对比分析恢复结果图像

- 只添加运动模糊：

  逆滤波和维纳滤波本质上等价，都可以恢复得到原始图像

- 同时添加运动模糊和高斯噪声

  逆滤波时高频的高斯噪声会被放大，得到的结果效果很差，在进行维纳滤波时可以抑制噪声得到较好的结果，但仍有部分噪声，是因为高斯噪声在图像上的分布是随机的，无法完全恢复

OTSU阈值分割

- OSTU函数实现如下：

  主要思路是按照公式 $g=\omega 0*(\mu 0-\mu)^2+\omega 1*(\mu 1-\mu)^2$

  不断用不同的阈值t去尝试，使得g最大t就是最佳阈值

  因此需要分别统计被t划分为两部分的像素比例和灰度均值，以及整个图片的灰度均值，代入公式计算即可

```
function k = OSTU(I)
    I = im2double(I);
    [M,N]=size(I);
    pix=M*N;
    gray=0;
    ICV_t=0;
    temp = 0;
    %得到图像总灰度值
    for i=1:M
        for j=1:N
            gray=gray+I(i,j);
        end
    end
    gray_aver=gray*255/pix;   %图像灰度值的总平均值


    %按照t分割，并分别计算前景和背景的像素比例和平均灰度
    for t=0:255
        gray_A=0;       %总灰度值
        gray_B=0;
        pix_A=0;        %总像素
        pix_B=0;

        for i=1:M
            for j=1:N
                if (I(i,j)*255>=t)
                    pix_A=pix_A+1;  %A总像素和总灰度
                    gray_A=gray_A+I(i,j);

                elseif (I(i,j)*255<t)
                    pix_B=pix_B+1;  %B总像素和总灰度
                    gray_B=gray_B+I(i,j);

                end
            end
        end

        PA=pix_A/pix; %A像素比例和平均灰度
        A_ave=gray_A*255/pix_A;
        PB=pix_B/pix; %B像素比例和平均灰度
        B_ave=gray_B*255/pix_B;

```

```
        %Otsu算法
        ICV=PA*((A_ave-gray_aver)^2)+PB*((B_ave-gray_aver)^2);
        if (ICV>ICV_t) %得到最大方差
            ICV_t=ICV;
            temp=t;
        end
    end

    k =temp;
end
```

实验结果：

用四叉树表达的迭代区域分裂合并算法

- qtdecomp() 函数对图像进行四叉树分割

  value_range 设置为 0.5

```
S = qtdecomp(source,range_value,2);
```

- 为了显示出分割块，将其边界设置为白色

  循环不同的分块大小，对每个分块将其边界像素设置为白色，其余像素设置为黑色

```
for dim = [64 32 16 8 4 2]
    blk_cnt = length(find(S==dim));
    if (blk_cnt > 0)
        values = repmat(uint8(1),[dim dim blk_cnt]);%整体置1
        values(2:dim,2:dim,:) = 0;% 非边界置0
    blocks = qtsetblk(blocks,S,dim,values);%保存分块在blocks中
  end
end

output1 = source;
output1(blocks==1) = 255;%置1的为边界，设置为白色
subplot(1,3,2); imshow(output1); title('分裂');
```

- 合并准备：

  qtgetblk从分块中提取出块，然后对每个块给定一个互异的标记

```
i = 0;
for dim = [64 32 16 8 4 2]
    % 从分块中提取块
    [vals,r,c] = qtgetblk(source,S,dim);
    % 如果块不为空，则对块进行标记
    if ~isempty(vals)
        for j = 1:length(r)
            i = i + 1;
            blocks(r(j):r(j)+ dim - 1,c(j):c(j)+ dim - 1) = i;
        end
    end
end
```

- 合并过程

  对于每个分块，计算其与周围分块合并后分块的极差，如果小于阈值则说明应该归于同一块，将它们的标记统一

  然后根据新的标记，仍然不同标记的才是不同的分块，并标记边界为白色

```
for j = 1 : i
    bound = boundarymask(blocks == j, 4) & (~(blocks == j));
    % 找到边界像素的位置
    [r,l] = find(bound == 1);
    for k = 1 : size(r,1)
```

```
        % 合并
        merge = source((blocks==j) | (blocks==blocks(r(k),l(k))));
        % 计算极差
        if(range(merge(:)) < value_range * 256)
            % 标记
            blocks(blocks == blocks(r(k), l(k))) = j;
        end
    end
end

%根据标记重新分割，合并相邻的分块
output2 = source;
for i = 2 : 255
    for j = 2 : 255
    %标记合并后，原图中还是不相同的块填充为白色区分边界
        if(blocks(i,j)~=blocks(i,j+1) || blocks(i,j)~=blocks(i+1,j))
            output2(i,j) = 255;
        end
    end
end
```

实验结果：