|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  | | | | |
| **《计算机图像处理》**  **实验报告** | | | | |
|  | | | | |
|  | **姓名：** | **：** | **邸相耀** |  |
| **学号：** | **：** | **225392** |
| **班级：** | **：** | **计221** |
| 2024年 11月 | | | | |

# 实验一 MATLAB数字图像处理基本操作

**一、实验目的与要求**

1. 熟悉MATLAB软件的开发环境、基本操作以及图像处理工具箱，为编写图像处理程序奠定基础。

2. 掌握二值、灰度和彩色图像的读、写和显示方法，以及图像的高、宽、颜色等参数的获取方法。

3. 根据实验内容进行问题的简单分析和初步编码。

**二、实验相关知识**

**1、Matlab软件Image Processing Toolbox简介**

MatLab的原文是Matrix Laboratory，它包括若干个工具箱，如Communications Toolbox、Control System Toolbox、Neural Network Toolbox、Wavelet Toolbox等等，其中Image Processing Toolbox图像处理工具箱可以完成Geometric Operations、Enhancement、Color Segmentation、Image Transformation、Image Analysis、Morphological Operations等操作。在MatLab中，图像就是一个矩阵，在进行处理时当作一个变量即可，因此运算的书写十分简洁，故MatLab有草稿纸式的算法语言之称。例如：

J=I+50; %为原始图像I加上一常数50，并将结果赋予变量J，其效果相当于得到一幅加亮的图像J

以此类推可以书写出减法J=I-0.5；乘法J=I\*2；除法J=I/3；等等。

利用MatLab提供的imread和imwrite函数可以完成对图像文件的读写操作，它们所支持的一些常用的图像文件格式见表1-1。

**表1-1 MATLAB支持的一些常用的图像文件格式**

|  |  |  |
| --- | --- | --- |
| 格式名称 | 描述 | 可识别的扩展符 |
| TIFF | 加标签的图像文件格式 | .tiff、.tif |
| JPEG | 联合图像专家组 | .jpg、.jpeg |
| GIF | 图像交换格式 | .gif |
| BMP | Windows位图 | .bmp |
| PNG | 可移植网络图形 | .png |
| XWD | X Window转储 | .xwd |

MatLab提供了两种运行方式，即命令行方式和M文件方式。打开MatLab界面后，在Command窗口的提示符号“>>”下直接键入命令即可运行，如键入：

>>clear %执行本命令将会清除内存中的全部变量

>> figure(1); %生成一个图像窗口1

>> I=imread('e:lena.bmp'); %将硬盘e:根目录上的图像文件lena.bmp的数据读入矩阵变量I中

>> imshow(I); %在当前的图像窗口中显示图像矩阵I

>> title('原始图像'); %在当前的图像窗口中加上标题

但为了能够对程序进行调试和重复应用，我们要求用M文件的方式完成实验中各个程序的编写。

**2、数字图像的表示和类别**

根据图像数据矩阵解释方法的不同，MATLAB把其处理为四类：

**(1) 亮度图像（Intensity images）**

一幅亮度图像是一个数据矩阵，其归一化的取值表示亮度。若亮度图像的像素都是uint8类或uint16类，则它们的整数值范围分别是[0，255]和[0，65536]。若图像是double类，则像素取值就是浮点数。规定双精度型归一化亮度图像的取值范围是[0，1]。

**(2) 二值图像（Binary images）**

一幅二值图像是一个取值只有0和1的逻辑数组。而一幅取值只包含0和1的uint8类数组，在MATLAB中并不认为是二值图像。使用logical函数可以把数值数组转化为逻辑数组，其语法为B=logical(A)；其中A是由0和1构成的数值数组。要测试一个数组是否为逻辑数组，可以使用函数islogical(C)；若C是逻辑数组，则该函数返回1；否则，返回0。

**(3) 索引图像（Indexed images）**

索引颜色通常也称为映射颜色，在这种模式下，颜色都是预先定义的，并且可供选用的一组颜色也很有限，索引颜色的图像最多只能显示256种颜色。一幅索引颜色图像在图像文件里定义，当打开该文件时，构成该图像具体颜色的索引值就被读入程序里，然后根据索引值找到最终的颜色。

**(4) RGB图像（RGB images）**

一幅RGB图像就是彩色像素的一个M×N×3数组，其中每一个彩色像素点都是在特定空间位置上相对应的红、绿、蓝三个分量。按照惯例，形成一幅RGB彩色图像的三个矩阵常称为红、绿或蓝分量图像。令fR，fG和fB分别代表三种RGB分量图像。一幅RGB图像就利用cat(级联)操作将这些分量图像组合成彩色图像：rgb\_image=cat(3,fR,fG,fB) 在操作中，图像按顺序放置。

**3、数据类和图像类型间的转化**

表1-2中列出了MATLAB和Image Processing Toolbox为表示图像所支持的各种数据类。表中的前8项称为数值数据类，第9项称为字符类，最后一项称为逻辑数据类。表1-3列出了工具箱中提供的一些必要函数，以便在图像类和数据类之间进行转化。

**表1-2 MATLAB和IPT支持数据类型**

|  |  |
| --- | --- |
| 名称 | 描述 |
| double | 双精度浮点数，范围为 |
| uint8 | 无符号8比特整数，范围为[0 255] |
| uint16 | 无符号16比特整数，范围为[0 65536] |
| uint32 | 无符号32比特整数，范围为[0 4294967295] |
| int8 | 有符号8比特整数，范围为[-128 127] |
| int16 | 有符号16比特整数，范围为[-32768 32767] |
| int32 | 有符号32比特整数，范围为[-2147483648 2147483647] |
| single | 单精度浮点数，范围为 |
| char | 字符 |
| logical | 值为0或1 |

**表1-3 格式转换函数**

|  |  |  |
| --- | --- | --- |
| 名称 | 将输入转化为 | 有效的输入图像数据类 |
| im2uint8 | uint8 | logical,uint8,uint16和doulbe |
| im2uint16 | uint16 | logical,uint8,uint16和doulbe |
| mat2gray | double,范围为[0 1] | double |
| im2double | double | logical,uint8,uint16和doulbe |
| im2bw | logical | uint8,uint16和double |

**4、常用函数和命令**

(1) 读写图像文件

imread：用于读入各种图像文件，返回一个矩阵，如：a=imread('d:\p1.jpg')

whos：用于读取图像的基本信息，如：whos a

imwrite：用于写入图像文件，如：imwrite(a,'d:\p2.tif',’tif’)

imfinfo：用于读取图像文件的有关信息，如：imfinfo('d:\p2.tif')

(2) 图像的显示

image：image函数是MATLAB提供的最原始的图像显示函数，如：

a=[1,2,3,4;4,5,6,7;8,9,10,11]; image(a);

imshow：用于图像文件的显示，如：

i=imread('d:\p1.jpg');imshow(i);

colorbar：用于显示图像的颜色条，如：i=imread('d:\p1.jpg');imshow(i);colorbar;

figure：用于设定图像显示窗口，如：figure(1);figure(2);% n为图形窗口号数

subplot：把图形窗口分成多个矩形部分，每个部分可以分别用来进行显示，如：subplot(m,n,p)

分成m×n个小窗口，在第p个窗口中创建坐标轴为当前坐标轴，用于显示图形

plot：绘制二维图形，如：plot(y)；plot(x，y)；%x、y可以是向量、矩阵

(3) 图像类型转换

rgb2gray：把真彩图像转换为灰度图像，如：i=rgb2gray(j)

im2bw：通过阈值化方法把图像转换为二值图像，如：I=im2bw(j，level) imbinarize

Level表示灰度阈值，取值范围0~1，即0.n表示阈值取自原图像灰度范围的n%

(4) 其它运算函数

zeros：生成全0数组或矩阵，如：B=zeros(m,n)或B=zeros([m n])，返回一个m×n的全0矩阵

取整函数：round四舍五入取整函数；floor最小取整函数；ceil最大取整函数，

如：a=[-1.9 -0.2 3.4 5.6 2.4+3.6i]; I=round(a)

输出I=[-2 0 3 6 2+4i]

**三、实验内容**

**1、读入一幅RGB图像（sunset.jpg），分别转换为灰度图像和二值图像，并在同一个窗口内分成三个子窗口来分别显示这三幅图像，注上文字标题，即“原图像”、“灰度图像”、“二值图像”。**

clear;

org = imread('sunset.jpg');

gray = rgb2gray(org);

bw = im2bw(org,0.5);

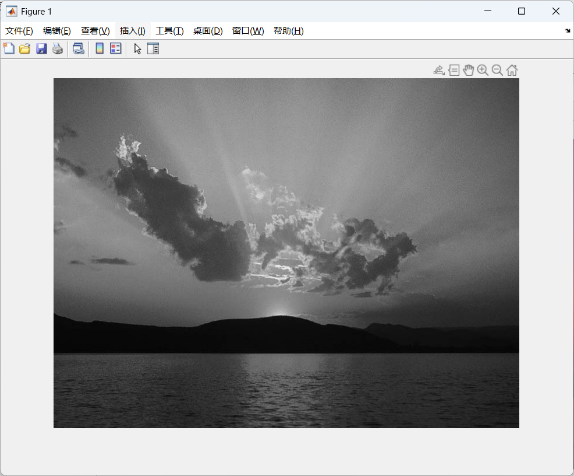
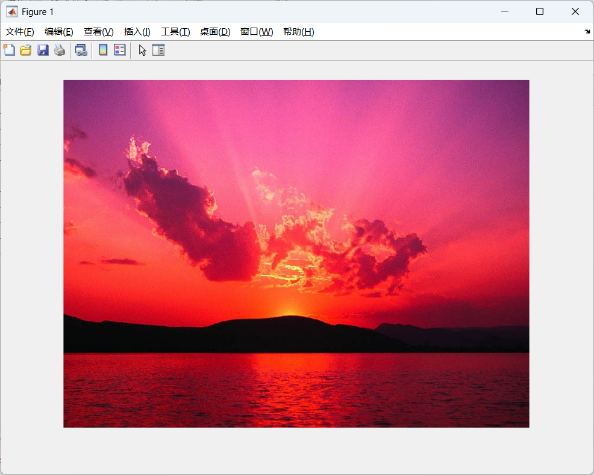
imshow(org);

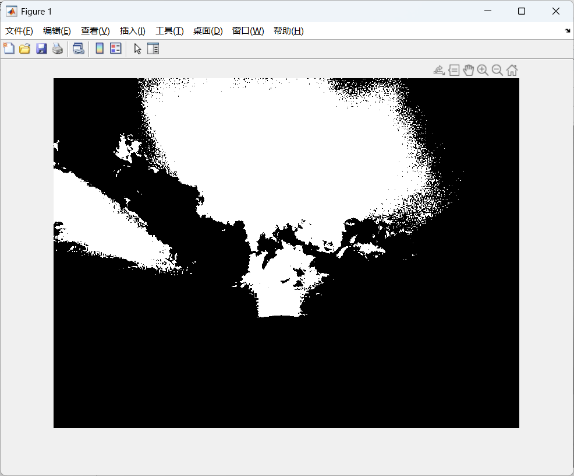
pause; % Be used to show picture.

imshow(gray);

pause;

imshow(bw);





**2、说明以下程序的显示结果为何是一幅几乎全黑的图像。**

clear;

close all;

myi=zeros(20,20); % Create a 20\*20 zero matrix.

myi(2:2:18,2:2:18)=1; % Change [2,4,6...18][2,4,6...18].

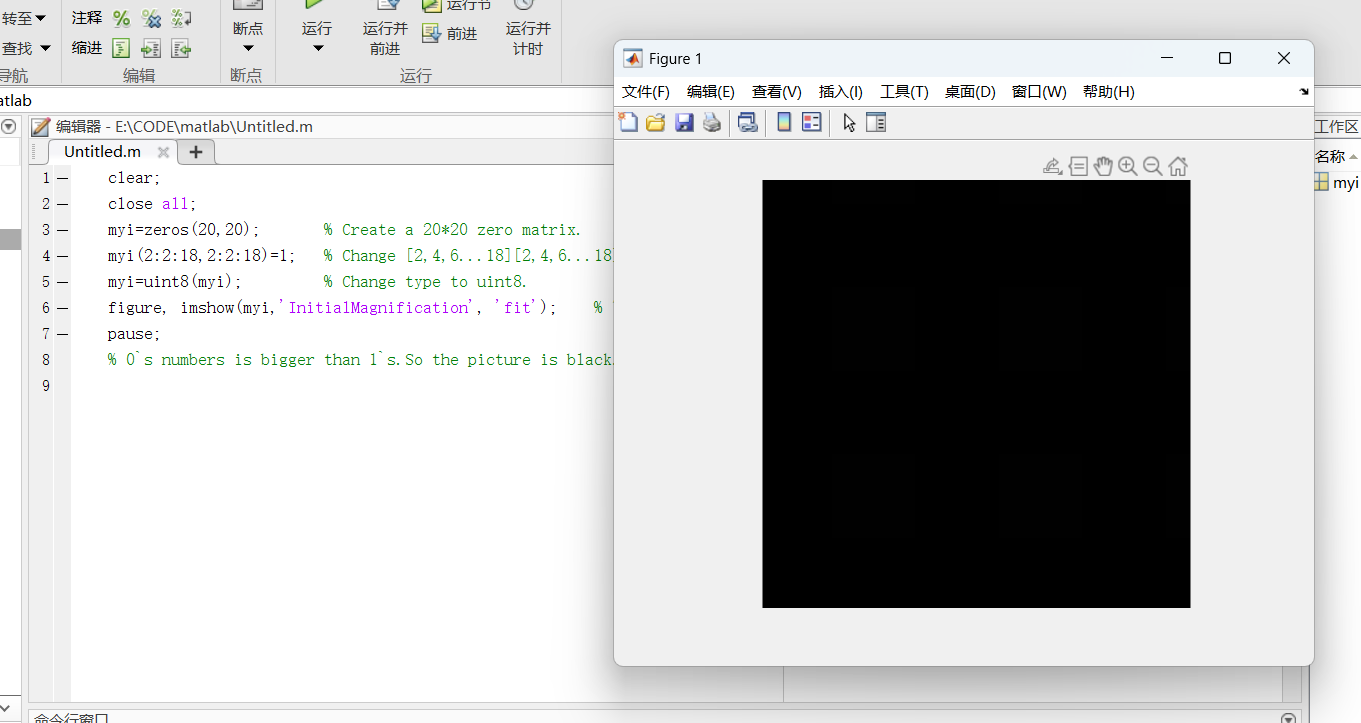
myi=uint8(myi); % Change type to uint8.

figure, imshow(myi,'InitialMagnification', 'fit');

% "figure, imshow(myi,'notruesize');" is wrong, be used to show the picture.

pause;

% 0`s numbers is bigger than 1`s.So the picture is black.



**3、阅读分析下列程序，对每条语句给出注释，运行并解释最终执行结果；若将for循环改为方框中的程序，有何不同？**

clear all;close all;

I=imread('d:\lena512.jpg');

[height,width]=size(I);

for m=1:5

L=2\*L;

quartimage=I(1:L:height,1:L:width);

subplot(2,3,m+1);

imshow(uint8(quartimage));

end

figure;

subplot(2,3,1);

imshow(I);

title('原图像');

L=1;

for m=1:5

L=2\*L;

quartimage=zeros(ceil(height/L),ceil(width/L));

k=1;n=1;

for i=1:L:height

for j=1:L:width

quartimage(k,n)=I(i,j);

n=n+1;

end

k=k+1;n=1;

end

subplot(2,3,m+1);

imshow(uint8(quartimage));

end

clear; %"clear all" has a warning,so we delete "all" to improve it.

close all;

I=imread('lena512.jpg');

[height,width]=size(I); % This function is return two value "height and width".

figure;

subplot(2,3,1);

imshow(I);

title('origional');

L=1; % Reduce rate

for m=1:5 % a Cycle, which can reduce the picture`s pixel with rate "L".

L=2\*L;

quartimage=zeros(ceil(height/L),ceil(width/L)); % Create a new zero matrix.

k=1;n=1;

for i=1:L:height % reading values from original picture.

for j=1:L:width

quartimage(k,n)=I(i,j); %I[1,1+L,1+2L...][1,1+L,1+2L...] to quartimage[1,2,3...][1,2,3...]

n=n+1;

end

k=k+1;n=1;

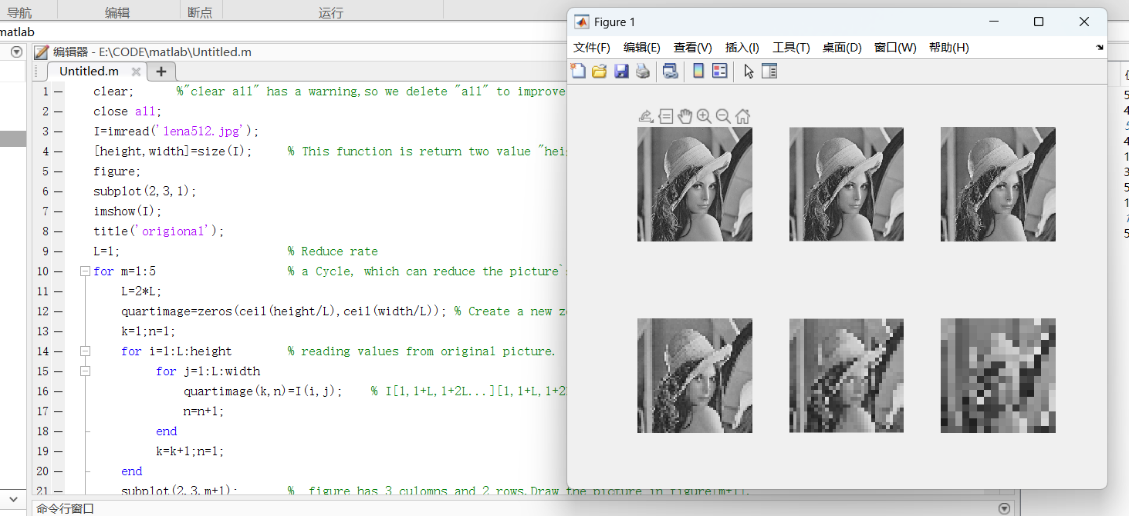
end

subplot(2,3,m+1); % figure has 3 culomns and 2 rows.Draw the picture in figure[m+1].

imshow(uint8(quartimage));

end

% Same function between two codes.



**4、熟悉数字图像处理常用函数的使用，调出帮助文档查看其各种不同用法。**

MatLab具有完善的帮助系统，包括命令行帮助、联机帮助和演示帮助等，应学会充分利用帮助系统来解决问题。获得帮助的途径有多种，如：

(1) 在MatLab界面中单击工具条上的问号，或单击Help菜单中的MATLAB Help选项；或按F1键打开Help窗口。

(2) 选择函数（函数所在区变暗），点右键弹出菜单，选择Help on Selection；或在command窗口键入help函数名（或工具箱名）。

I got it.

**5、改变灰度图像的灰度分辨率（256颜色-128颜色-64颜色-32颜色-16颜色-8颜色-4颜色-2颜色），给出实验结果；图像大小不变的情况下，仿真改变图像的空间分辨率（变为原来的1/4（例如原图像(1,1);(1,2);(2,1);(2,2)四个像素值设成相同的值）；1/16），给出实验结果。**

空间分辨率：决定图像的清晰度和细节，通过像素数量表示。

灰度分辨率：决定图像的灰度层次和细腻程度，通过灰度级别数量表示。

gray\_image = imread('orangutan.tif'); % Read original picture.

figure('Name','gray pixel');

subplot(3,3,1), imshow(gray\_image), title('256 Colors'); % Show original picture. "title" is used to show name.

gray\_levels = [2, 4, 8, 16, 32, 64, 128];

for i = 1:length(gray\_levels) % Show different pixel of this picture.

reduced\_gray\_image = gray\_image(1:gray\_levels(i):end, 1:gray\_levels(i):end);

subplot(3,3,i+1), imshow(reduced\_gray\_image), title([num2str(gray\_levels(i)) ' Colors']);

end

% Space pixel.We use "imresize" to change it.

figure('Name','space pixel');

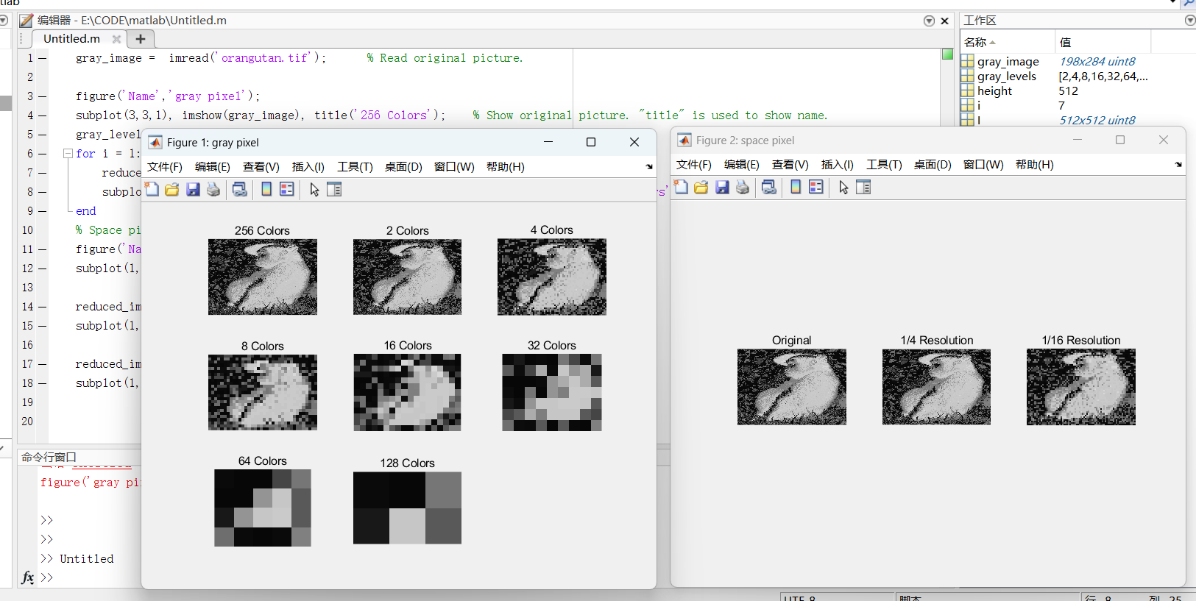
subplot(1,3,1), imshow(gray\_image), title('Original');

reduced\_image\_1\_4 = imresize(gray\_image, 0.5, 'nearest');

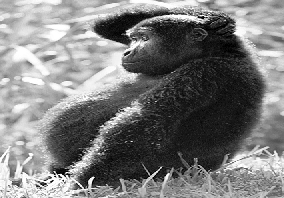
subplot(1,3,2), imshow(reduced\_image\_1\_4), title('1/4 Resolution');

reduced\_image\_1\_16 = imresize(gray\_image, 0.25, 'nearest');

subplot(1,3,3), imshow(reduced\_image\_1\_16), title('1/16 Resolution');

****

**四、实验图像**

sunset.jpg lena512.jpg orangutan.tif

# 实验二 图像直方图及灰度变换

**一、实验目的与要求**

1．掌握图像灰度直方图的概念及其计算方法，编写灰度直方图统计程序。

2．通过对图像直方图的分析，学习应用直方图法解决诸如图像二值化等具体问题。

3．熟悉直方图均衡化的计算过程及其应用。

4．掌握图像灰度变换技术，通过调整图像的对比度和亮度等参数，改善视觉效果。

**二、实验相关知识**

直方图是图像最基本的统计特征，是图像亮度分布的概率密度函数，反映了图像灰度值的分布情况。直方图是多种空间域处理技术的基础。直方图操作能有效地用于图像增强，如通过直方图均衡化处理，可使图像在整个灰度级范围内的分布均匀化，即在每个灰度级上都具有相同的像素点数，从而获得较好的视觉效果。另外，直方图固有的信息也可用在图像分割等其它图像处理的应用中。

灰度级变换技术可用g(x,y)=T[f(x,y)]的形式表示，其中f(x,y)为输入图像，g(x,y)为输出图像，T是对图像f进行某种处理的操作。由于(x,y)处的g值仅由f在该点处的亮度决定，T也称为一个亮度或灰度级变换函数，它与位置坐标(x,y)无关，所以通常写成如下的简化形式s=T(r)，其中r和s分别表示图像f和g在相应点(x,y)的亮度。利用灰度变换可调整一幅图像的明暗、对比度等。

**三、实验内容**

**1、编写一个图像灰度直方图统计函数my\_imhist，选择一幅图像利用my\_imhist显示其直方图，将结果与MATLAB图像处理工具箱中提供的灰度直方图函数imhist的处理结果进行比较，并在同一窗口中显示出来。**

**% main.m**

image = imread('iris.tif');

myhist = my\_imhist(image);

subplot(2,1,1);

my\_imhist(image);

bar(0:255, myhist);

title('Myimhist`s picture');

xlabel('Grayscale');

ylabel('Number of pixels');

matlab\_hist = imhist(image);

subplot(2, 1, 2);

imhist(image);

title('MATLAB\_imhist`s picture');

xlabel('Grayscale');

ylabel('Number of pixels');

**% my\_imhist.m**

function counts = my\_imhist(image) % return counts

counts = zeros(256, 1);

[height,width]=size(image);

for i = 1:height

for j = 1:width % add 1 to improve x-axis

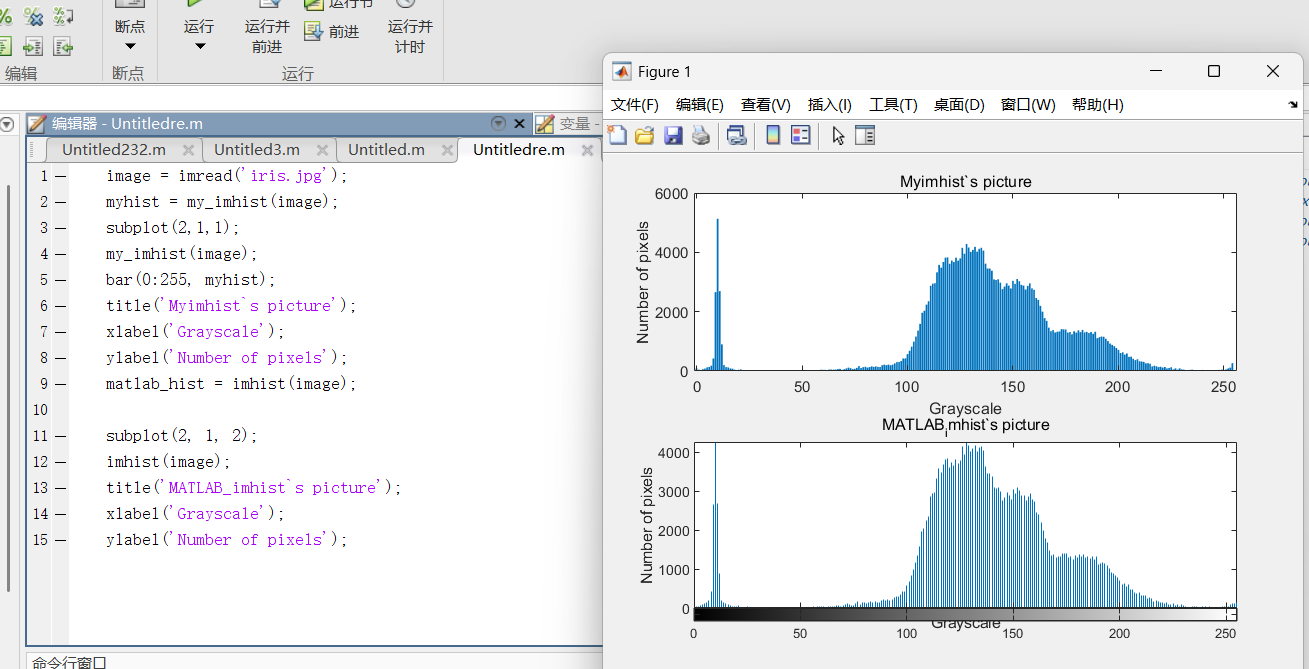
counts(image(i,j)+1) = counts(image(i,j)+1)+1;

end

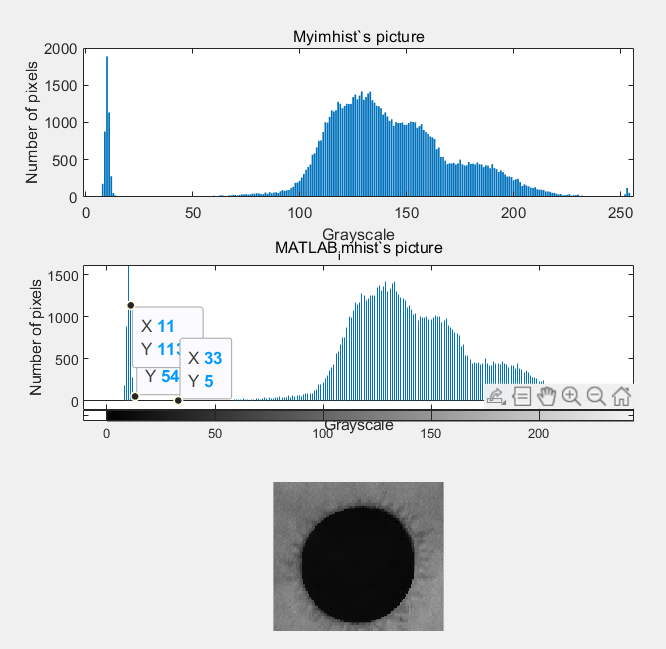
end

end

**% We use ‘iris.tif’ to run codes.**



**2、利用以上编写的函数my\_imhist或imhist，估算图像iris.tif中瞳孔的半径（以像素为单位）。**



counts\_my = my\_imhist(image);

numPixels0to50\_my = sum(counts\_my(1:51));

% sumBlack = 4561. S = PI \* r^2 ,so the iris`s r ≈ 38.

**3、按照教材68页上的公式（4.1.6）**

** (4.1.6)**

**编程实现图像的分段线性灰度变换.**

%org = imread('sunset.jpg'); % Change to gray picture.

%image = rgb2gray(org);

image = imread('pout.tif'); % Or input a gray picture.

M = 255; % Mf == Mg, MAX GRAY RATE

a = 255 \* 0.3; % a,Customize

b = 255 \* 0.6; % b,Customize

c = a + 50; % c, the offset is 50

d = b - 50; % d, with offset = 50

[height,width]=size(image); % Get the range

newImage = zeros(height, width); % Init

for i = 1:height % Recycling to deal this mission !!!

for j = 1:width

if(image(i,j) >= 0 && image(i,j)<a)

newImage(i,j) = (c/a) \* image(i,j);

elseif(image(i,j) >=a && image(i,j) <b)

newImage(i,j) = ((d-c)/(b-a))\*(image(i,j) - a) + c;

else

newImage(i,j) = ((M-d)/(M-b))\*(image(i,j)-b)+d;

end

end

end

figure(1); % Show

subplot(2,2,1);

imshow(image);

title("Origional");

subplot(2,2,2);

imshow(uint8(newImage));

title("New");

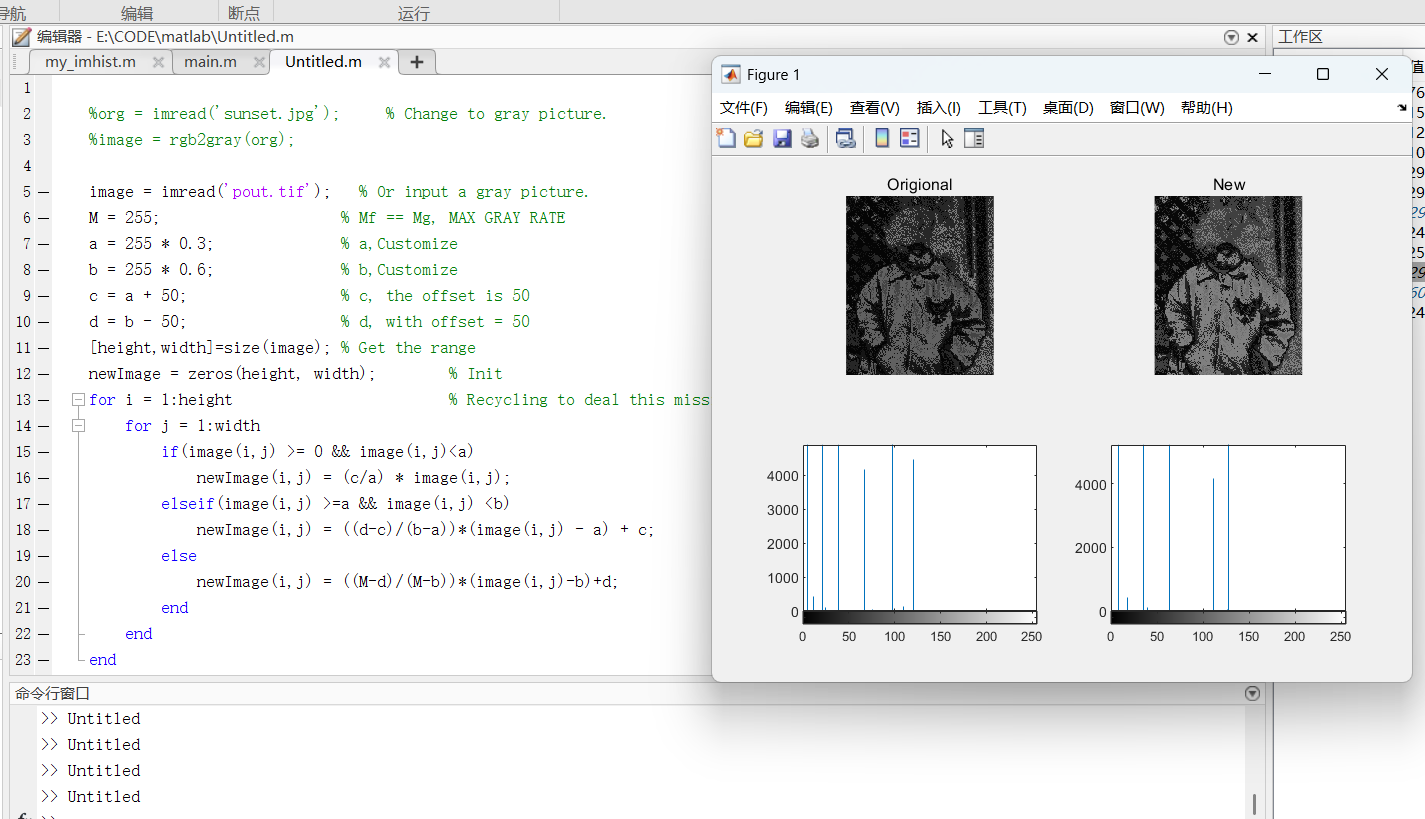
subplot(2,2,3);

imhist(image);

subplot(2,2,4);

imhist(uint8(newImage));

% It worked, but not as good as the histogram equalization.



**4、编写一个灰度图像的直方图均衡化函数（不可使用库函数）。（可使用的灰度等级数量不变即可），并对下图例题进行测试，给出测试结果。**

image = [[5,5,4,4,2,0,7],

[5,5,4,2,0,0,7],

[4,4,4,2,0,7,7],

[1,3,3,0,7,7,6],

[1,0,0,0,7,7,6],

[7,7,7,7,7,7,6],

[7,7,7,7,7,7,6]];

image = uint8(image);

[height, width] = size(image);

N = height \* width;

M = 7;

k = zeros(1,M+1);

for i = 1:height

for j = 1:width

k(image(i,j)+1) = k(image(i,j)+1) + 1;

end

end

p = zeros(1, M+1);

for i = 1:M+1

p(i) = k(i) / N;

end

s = zeros(1, M+1);

s(1) = p(1);

for i = 2:M+1

s(i) = s(i-1) + p(i);

end

nk = round(s \* 7);

newImage = uint8(zeros(height, width));

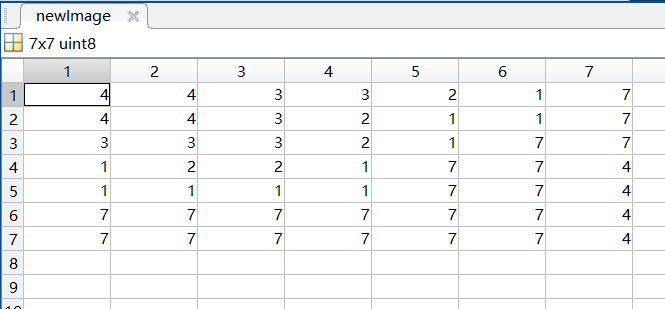
for i = 1:height

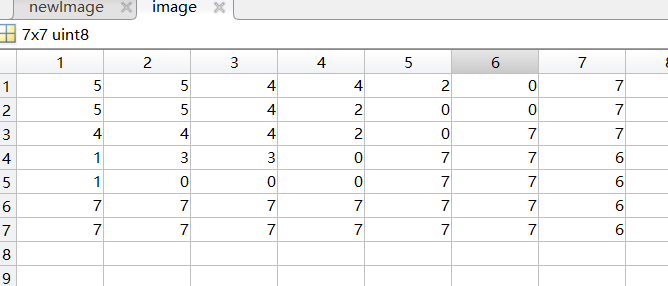
for j = 1:width

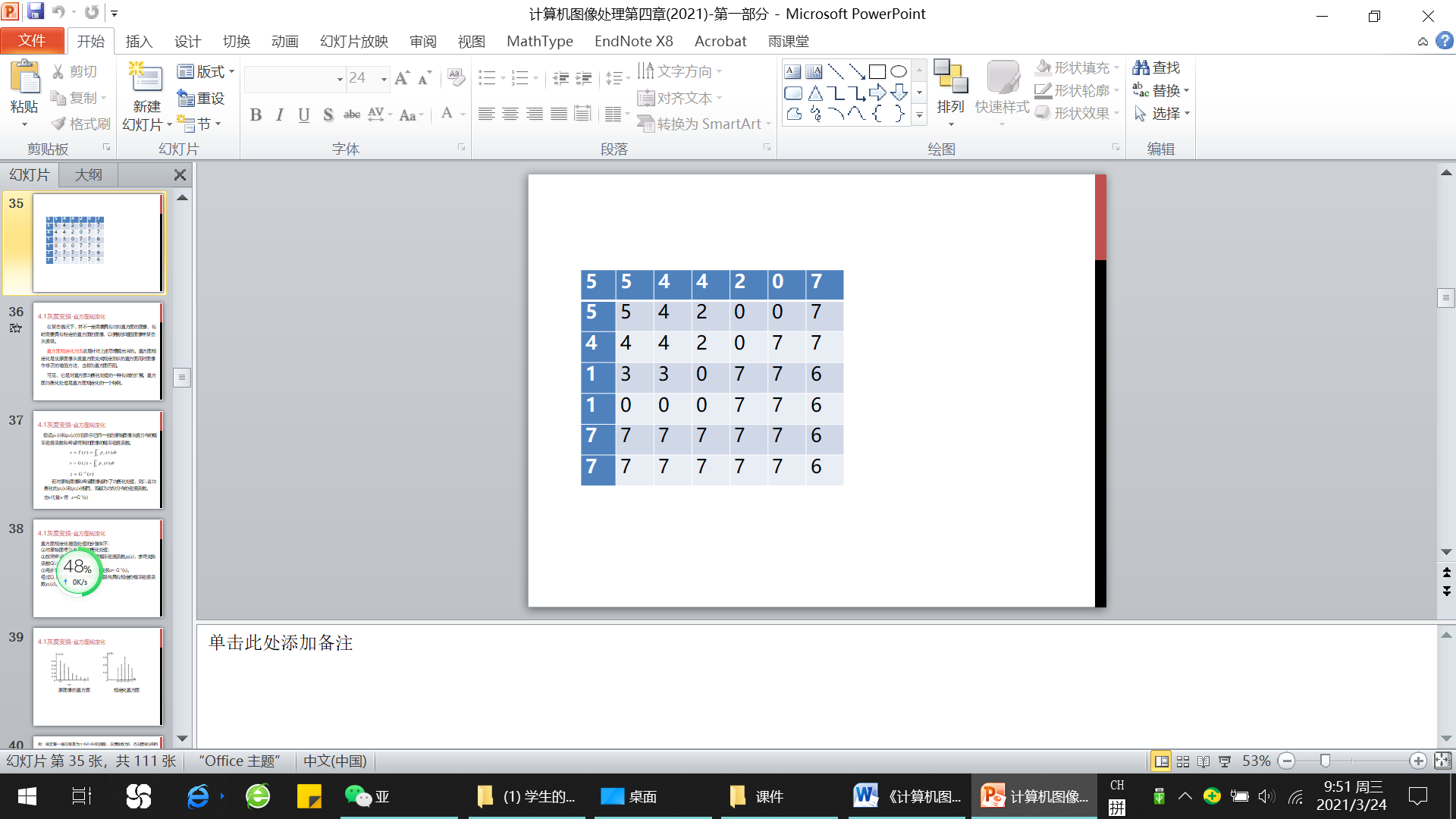
newImage(i,j) = nk(image(i,j) + 1);

end

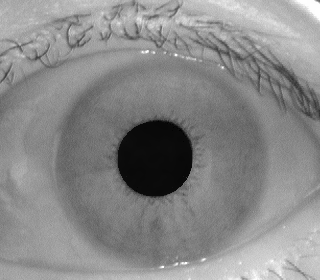
end

****

****



**四、实验图像(可使用其他自选图像)**

iris.tif pout.tif bottle.tif

# 实验三 图像空间域平滑

**一、实验目的与要求**

1．加深对图像增强及边缘检测技术的感性认识，应用MATLAB工具箱自带的处理函数或自己编程完成相关的工作，分析处理结果，巩固所学理论知识。

2．熟练掌握空域滤波中常用的平滑滤波器，针对不同类型和强度的噪声，进行滤波处理，体会并正确评价滤波效果，了解不同滤波方式的使用场合，能够从理论上作出合理的解释。

**二、实验相关知识**

图像增强是指按特定的需要突出一幅图像中的某些有用信息，同时消弱或去除某些不需要的信息的处理方法，其主要目的是使处理后的图像对某些特定的应用比原来的图像更加有效。图像平滑与锐化处理是图像增强的主要研究内容。

图像分割是由图像处理过渡到图像分析的关键步骤，一般是按照一定的规则，把图像分成互不重叠的若干区域或子集，并提取出感兴趣目标的技术和过程。本实验主要研究利用边缘提取算子的分割方法。

和本实验有关的几个常用Matlab函数：

**(1) imnoise：**用于对图像生成模拟噪声，如：

j=imnoise(i,'gaussian',0,0.02) %在图像i上叠加均值为0、方差为0.02的高斯噪声，得到含噪图像j

j=imnoise(i,'salt & pepper',0.04) %在图像i上叠加密度为0.04的椒盐噪声，得到含噪图像j

**(2) fspecial：**用于产生预定义滤波器，如：

h=fspecial('average',3); %产生3×3模板的均值滤波器

h=fspecial('sobel'); %产生sobel水平边缘增强的滤波器

可选项还有：'gaussian'高斯低通滤波器、'laplacian'拉普拉斯滤波器、'log'高斯拉普拉斯滤波器等

**(3) imfilter、filter2、conv2：**均是基于卷积的图像滤波函数，都可用于图像滤波，用法类似，如：

i=imread('p1.tif');

h=fspecial('prewitt'); %产生prewitt算子的水平方向模板

j1=imfilter(i,h); %或者j2=filter2(h,i); 或者j3=conv2(i,h);

**(4) medfilt2：**用于图像的中值滤波，如：

j=medfilt2(i, [M N]); %对矩阵i进行二维中值滤波，邻域为M×N，缺省值为3×3

**三、实验内容**

**1.对实验图像加噪声；**

image = imread('rice.png');

image = im2gray(image);

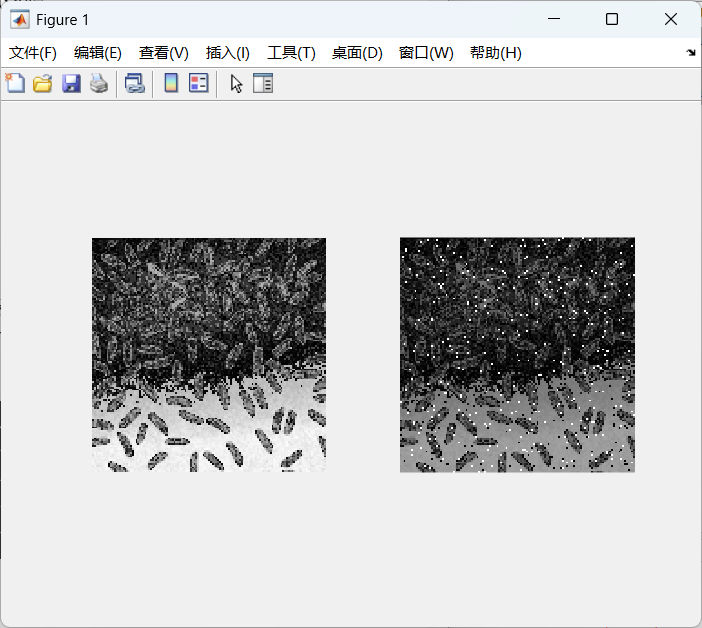
imageNoise = imnoise(image, 'salt & pepper', 0.04);

subplot(1,2,1);

imshow(image,[]);

subplot(1,2,2);

imshow(imageNoise,[]);



**2.编写局部平滑法、超限像素平滑法，灰度最相近的K个邻点平均法，中值滤波法（函数名称可以自定义），对加噪声的图像进行处理，显示处理前后的各个图像，分析不同方法对不同噪声的处理效果及其优缺点。**

**Main.m**

image = imread('rice.png');

image = im2gray(image);

imageNoise = imnoise(image, 'salt & pepper', 0.04);

subplot(3,2,1);

imshow(image,[]);

title('Origion');

subplot(3,2,2);

imshow(imageNoise,[]);

title('With Noise');

localSmooth = local\_smooth(image);

subplot(3,2,3);

imshow(localSmooth,[]);

title('Local Smooth');

beyondSmooth = beyond\_smooth(image);

subplot(3,2,4);

imshow(beyondSmooth,[]);

title('Beyond Smooth');

KNSmooth = k\_nearest\_smooth(image);

subplot(3,2,5);

imshow(KNSmooth,[]);

title('KN Smooth');

MidSmooth = mid\_number\_smooth(image);

subplot(3,2,6);

imshow(MidSmooth,[]);

title('Mid Pixel Smooth');

**local\_smooth.m**

function counts = local\_smooth(image) %four\_area

[height,width]=size(image);

counts = zeros(height,width);

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

counts(i,j) = (image(i+1,j)+image(i-1,j)+image(i,j-1)+image(i,j+1))/4;

end

end

end

counts = uint8(counts);

end

**beyond\_smooth.m**

function counts = beyond\_smooth(image)

[height,width]=size(image);

counts = zeros(height,width);

T = 30;

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

counts(i,j) = (image(i+1,j)+image(i-1,j)+image(i,j-1)+image(i,j+1))/4;

if abs(image(i,j) - counts(i,j)) <= T

counts(i,j) = image(i,j);

end

end

end

end

counts = uint8(counts);

end

**k\_nearest\_smooth.m**

function counts = k\_nearest\_smooth(image)

window\_size = 3;

K = 5;

[height,width]=size(image);

counts = image;

half\_window = floor(window\_size / 2);

neighbor =zeros(1, window\_size^2);

for i = 1:height

for j = 1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

neighbor = image(max(1, i-half\_window):min(height, i+half\_window), max(1, j-half\_window):min(width, j+half\_window));

center\_pixel = image(i, j);

differences = abs(neighbor - center\_pixel);

[~, sorted\_indices] = sort(differences); % Return the nearest pixels` indexs.

nearest\_elements = neighbor(sorted\_indices(1:K));

counts(i,j) = sum(nearest\_elements) / K;

end

end

end

end

**mid\_number\_smooth.m**

function counts = mid\_number\_smooth(image) % use 3\*3 square.

window\_size = 3;

half\_window = floor(window\_size / 2);

[height,width]=size(image);

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

neighbor = image(max(1, i-half\_window):min(height, i+half\_window), max(1, j-half\_window):min(width, j+half\_window));

sortArea = sort(neighbor);

counts(i,j) = sortArea(ceil(window\_size^2));

end

end

end

counts = uint8(counts);

end

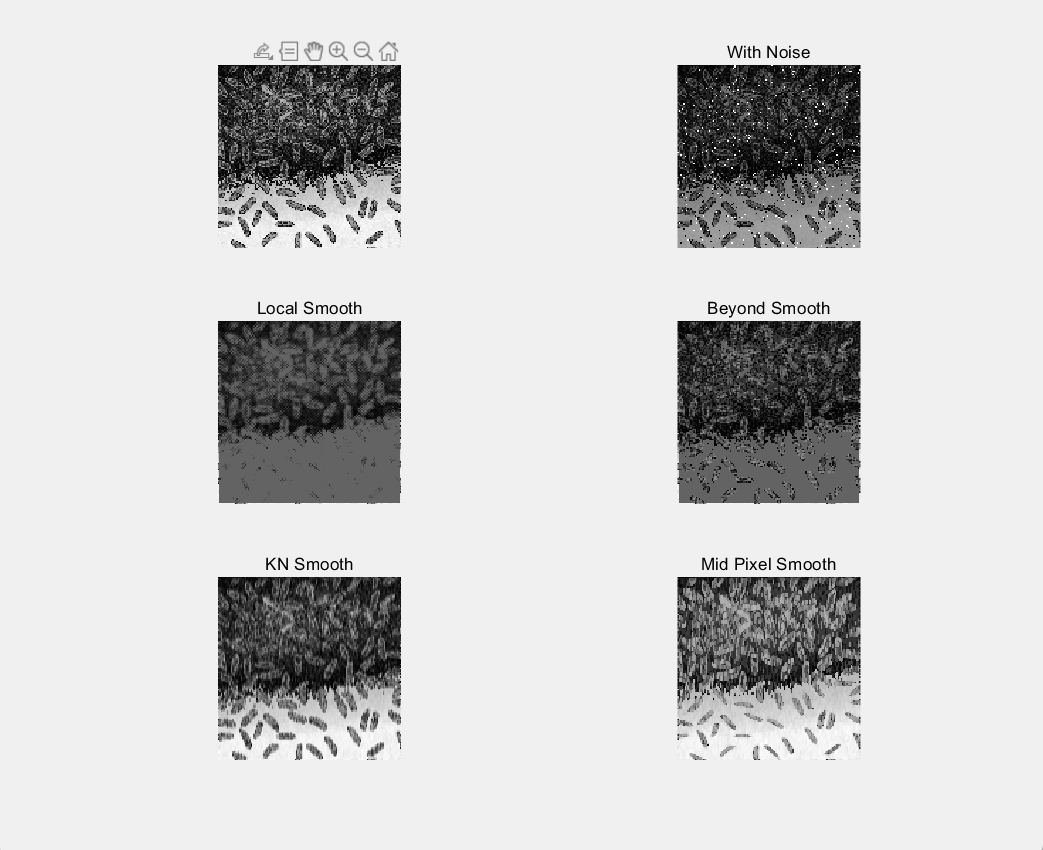
优缺点：

1．局部平滑法：适用于高斯噪声，简单高效，但对椒盐噪声效果差，容易模糊细节。

2．超限像素平滑法：适用于高斯噪声和椒盐噪声，能够保留细节，能比较有效的保护纹理和细节，但实现复杂，阈值选择关键。

3．灰度最相近的K个邻点平均法：适用于高斯噪声和椒盐噪声，能够保留细节，但实现复杂，K值选择关键，会使图像的边缘模糊。

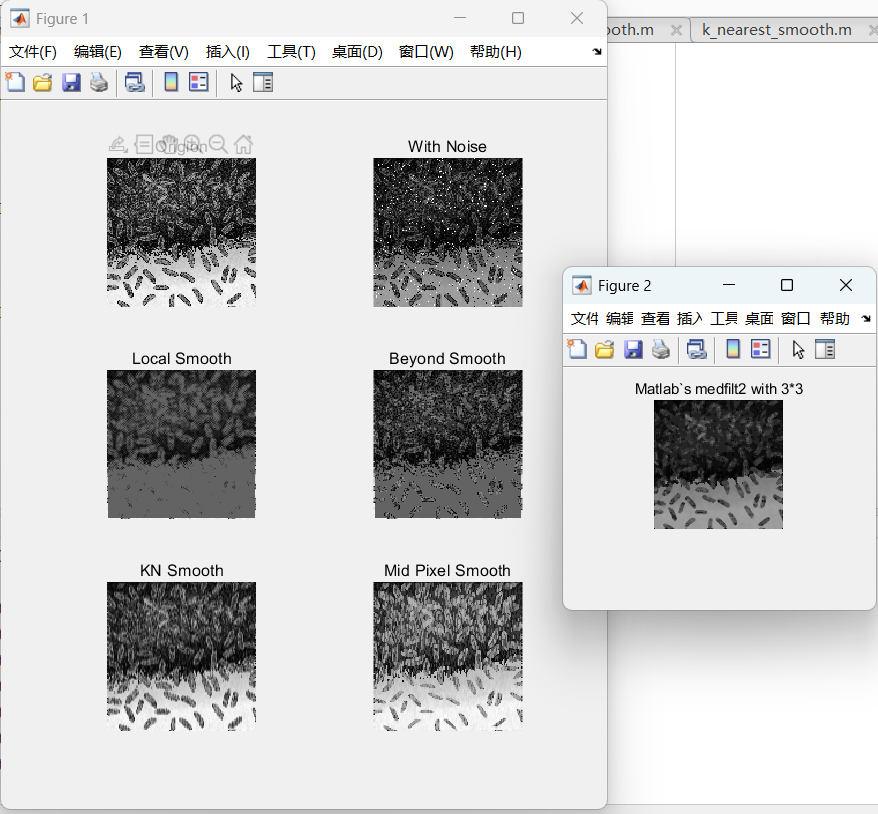
4．中值滤波法：对椒盐噪声效果最佳，能够保留边缘和细节，可根据不同的需求使用不同的中值滤波模板，但对高斯噪声效果一般，计算效率较低。

****

**3.与图像处理库函数处理结果进行对比。**

imshow(medfilt2(image, [3 3]));

title('Matlab`s medfilt2 with 3\*3');

****

rice.png wire.bmp

# 实验四 图像空间域锐化

**一、实验目的与要求**

1．加深对图像增强及边缘检测技术的感性认识，应用MATLAB工具箱自带的处理函数或自己编程完成相关的工作，分析处理结果，巩固所学理论知识。

2．熟练掌握空域滤波中常用的锐化滤波器，针对不同类型和强度的噪声，进行滤波处理，体会并正确评价滤波效果，了解不同滤波方式的使用场合，能够从理论上作出合理的解释。

**二、实验相关知识**

图像增强是指按特定的需要突出一幅图像中的某些有用信息，同时消弱或去除某些不需要的信息的处理方法，其主要目的是使处理后的图像对某些特定的应用比原来的图像更加有效。图像平滑与锐化处理是图像增强的主要研究内容。

图像分割是由图像处理过渡到图像分析的关键步骤，一般是按照一定的规则，把图像分成互不重叠的若干区域或子集，并提取出感兴趣目标的技术和过程。本实验主要研究利用边缘提取算子的分割方法。

和本实验有关的几个常用Matlab函数：

**(1) imnoise：**用于对图像生成模拟噪声，如：

j=imnoise(i,'gaussian',0,0.02) %在图像i上叠加均值为0、方差为0.02的高斯噪声，得到含噪图像j

j=imnoise(i,'salt & pepper',0.04) %在图像i上叠加密度为0.04的椒盐噪声，得到含噪图像j

**(2) fspecial：**用于产生预定义滤波器，如：

h=fspecial('average',3); %产生3×3模板的均值滤波器

h=fspecial('sobel'); %产生sobel水平边缘增强的滤波器

可选项还有：'gaussian'高斯低通滤波器、'laplacian'拉普拉斯滤波器、'log'高斯拉普拉斯滤波器等

**(3) imfilter、filter2、conv2：**均是基于卷积的图像滤波函数，都可用于图像滤波，用法类似，如：

i=imread('p1.tif');

h=fspecial('prewitt'); %产生prewitt算子的水平方向模板

j1=imfilter(i,h); %或者j2=filter2(h,i); 或者j3=conv2(i,h);

**(4) medfilt2：**用于图像的中值滤波，如：

j=medfilt2(i, [M N]); %对矩阵i进行二维中值滤波，邻域为M×N，缺省值为3×3

**三、实验内容**

**编写梯度锐化算法函数my\_grad(method，T)；参数method可以是梯度算子、Roberts算子、Prewitt和Sobel算子，T是梯度阈值，根据参数method和T选用不同的算子和阈值进行锐化处理，用二值图像表示处理结果，小于T的用黑色表示，大于等于T的用白色表示。显示处理前后的各个图像，分析不同方法对锐化效果及其优缺点。**

**My\_grad.m**

% At first, [Gx,Gy] = imgradientxy(I,method) returns the directional gradients using the specified method.

% But we dont use imgradientxy :)

function my\_grad(image, method, T)

[height,width]=size(image);

counts = zeros(height,width);

binary\_image = zeros(height,width);

switch method

case 'roberts'

% [Gx, Gy] = imgradientxy(image, 'roberts');

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

Gx = abs(image(i+1,j+1) - image(i,j));

Gy = abs(image(i+1,j) - image(i,j+1));

G = sqrt(Gx^2 + Gy^2);

if(G >= T)

counts(i,j) = G;

binary\_image(i,j) = 1;

else

counts(i,j) = image(i,j);

binary\_image(i,j) = 0;

end

end

end

end

case 'prewitt'

% [Gx, Gy] = imgradientxy(image, 'prewitt');

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

Gx = abs(image(i+1,j+1) + image(i+1,j) + image(i+1,j-1) - image(i-1,j) - image(i-1,j-1)-image(i-1,j+1));

Gy = abs(image(i+1,j+1) +image(i,j+1) + image(i-1,j+1) - image(i,j-1) + image(i+1,j-1) - image(i-1,j-1));

G = sqrt(Gx^2 + Gy^2);

if(G >= T)

counts(i,j) = G;

binary\_image(i,j) = 1;

else

counts(i,j) = image(i,j);

binary\_image(i,j) = 0;

end

end

end

end

case 'sobel'

% [Gx, Gy] = imgradientxy(image, 'sobel');

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

Gx = abs(image(i+1,j+1) + 2\*image(i+1,j) + image(i+1,j-1) - 2\*image(i-1,j) - image(i-1,j-1)-image(i-1,j+1));

Gy = abs(image(i+1,j+1) + 2\*image(i,j+1) + image(i-1,j+1) - 2\*image(i,j-1) + image(i+1,j-1) - image(i-1,j-1));

G = sqrt(Gx^2 + Gy^2);

if(G >= T)

counts(i,j) = G;

binary\_image(i,j) = 1;

else

counts(i,j) = image(i,j);

binary\_image(i,j) = 0;

end

end

end

end

case 'grad'

for i = 1:height

for j =1:width

if (i == 1) || (j == 1) || (i == height) || (j == width)

counts(i,j) = image(i,j);

else

Gx = abs(image(i,j+1)-image(i,j));

Gy = abs(image(i+1,j)-image(i,j));

G = sqrt(Gx^2 + Gy^2);

if(G >= T)

counts(i,j) = G;

binary\_image(i,j) = 1;

else

counts(i,j) = image(i,j);

binary\_image(i,j) = 0;

end

end

end

end

otherwise

error('Unsupported method. Choose from ''roberts'', ''prewitt'', or ''sobel''.');

end

figure;

subplot(1, 3, 1);

imshow(image,[]);

title('Origional');

subplot(1,3,2);

imshow(counts,[]);

title('Improved');

subplot(1,3,3);

imshow(binary\_image);

title('Binary',[]);

end

**main.m**

image = imread('rice.png');

image = double(image);

%{

figure;

subplot(1, 3, 1);

imshow(image);

title('Origional');

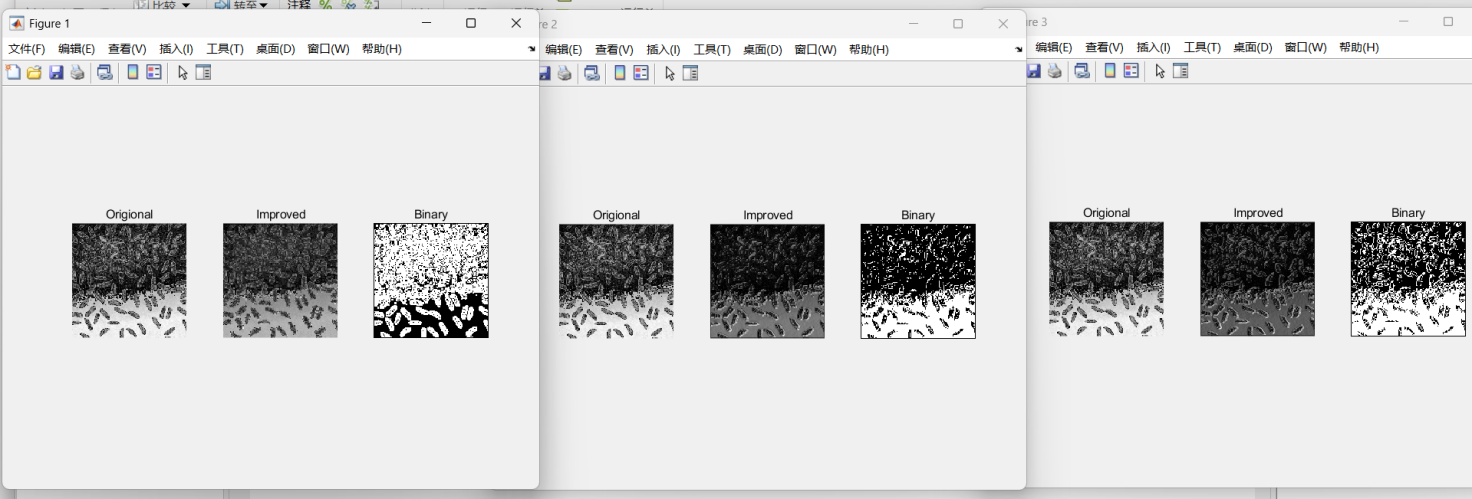
%}

my\_grad(image, 'roberts', 20);

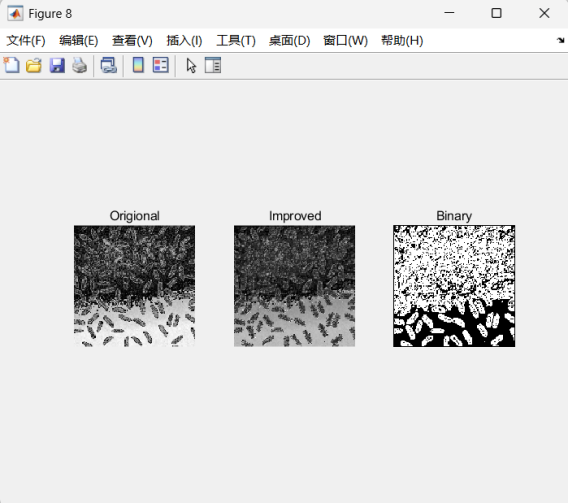
my\_grad(image, 'prewitt', 200);

my\_grad(image, 'sobel', 200);

my\_grad(image, 'grad', 20);

****

**From left to right, it's ‘roberts’,‘prewitt’,‘sobel’.**

****

**Basic grad**

基本梯度算子：基于梯度算子计算图像在水平方向和垂直方向上的差分来检测边缘，适用于简单的边缘检测，计算简单，但对噪声敏感。

Roberts 算子：计算对角方向上的差分检测边缘，适用于检测细小边缘，计算简单，但对噪声敏感，只能检测 45 度和 135 度方向的边缘。

Prewitt 算子：计算水平和垂直方向上的加权差分检测边缘，适用于检测粗大边缘，计算简单，但对噪声敏感，边缘检测效果不如 Sobel 算子。

Sobel 算子：计算水平和垂直方向上的加权差分检测边缘，适用于检测粗大边缘，对噪声有一定的抑制作用，边缘检测效果较好，但计算复杂度较高。

**四、实验图像**

rice.png wire.bmp

# 实验五 图像频域平滑与锐化

**一、实验目的与要求**

1．了解频域变换过程，掌握频域变换特点

2．熟练掌握频域滤波中常用的平滑和锐化滤波器，能够对不同要求的图像进行滤波处理，体会并正确评价滤波效果，了解不同滤波方式的使用场合，能够从理论上作出合理的解释。

**二、实验相关知识**

图像增强是指按特定的需要突出一幅图像中的某些有用信息，同时消弱或去除某些不需要的信息的处理方法，其主要目的是使处理后的图像对某些特定的应用比原来的图像更加有效。图像平滑与锐化处理是图像增强的主要研究内容。

和本实验有关的几个常用Matlab函数：

**(1) imnoise：**用于对图像生成模拟噪声，如：

j=imnoise(i,'gaussian',0,0.02) %在图像i上叠加均值为0、方差为0.02的高斯噪声，得到含噪图像j

j=imnoise(i,'salt & pepper',0.04) %在图像i上叠加密度为0.04的椒盐噪声，得到含噪图像j

**(2) fspecial：**用于产生预定义滤波器，如：

h=fspecial('average',3); %产生3×3模板的均值滤波器

h=fspecial('sobel'); %产生sobel水平边缘增强的滤波器

可选项还有：'gaussian'高斯低通滤波器、'laplacian'拉普拉斯滤波器、'log'高斯拉普拉斯滤波器等

**(3) imfilter、filter2、conv2：**均是基于卷积的图像滤波函数，都可用于图像滤波，用法类似，如：

i=imread('p1.tif');

h=fspecial('prewitt'); %产生prewitt算子的水平方向模板

j1=imfilter(i,h); %或者j2=filter2(h,i); 或者j3=conv2(i,h);

**(4) fft2:**二维快速傅里叶变换函数

**(5) fftshift:**中心变换函数

**(6) abs:**取绝对值或复数取幅值

**三、实验内容**

**1、图像频域平滑（去噪）：使用自生成图像（包含白色区域，黑色区域，并且部分区域添加椒盐噪声），然后进行傅里叶变换，并且分别使用理想低通滤波器、巴特沃斯低通滤波器、指数低通滤波器和梯形低通滤波器（至少使用两种低通滤波器），显示滤波前后的频域能量分布图，空间图像。分析不同滤波器对噪声、边缘的处理效果及其优缺点。**

image\_size = 256;

image = zeros(image\_size, image\_size); % Summon.

image(64:192, 64:192) = 1;

noisy\_image = imnoise(image, 'salt & pepper', 0.05);% Add noisy

% Show origional image and its noisy.

subplot(5, 3, 1);

imshow(image,[]);

title('Origion');

subplot(5, 3, 2);

imshow(noisy\_image,[]);

title('With noisy');

F = fftshift(fft2(noisy\_image)); % F

% Show Orgional energy

subplot(5, 3, 3);

imshow(log(1 + abs(F)), []);

title('Orgional energy');

**%OPT**

D0 = 30;

H\_ideal = zeros(image\_size, image\_size);

for u = 1:image\_size

for v = 1:image\_size

D = sqrt((u - image\_size/2)^2 + (v - image\_size/2)^2); % "image/2" is center of F

if D <= D0

H\_ideal(u, v) = 1;

end

end

end

F\_ideal\_filtered = F .\* H\_ideal; % C = A.\*B multiplies arrays A and B by multiplying corresponding elements.

ideal\_filtered\_image = ifft2(ifftshift(F\_ideal\_filtered));

subplot(5, 3, 4); % Show picture and energy`s image.

imshow(log(1 + abs(F\_ideal\_filtered)), []);

title('OPT energy');

subplot(5, 3, 5);

imshow(real(ideal\_filtered\_image), []);

title('OPT picture');

**%Butterworth**

n = 2;

H\_Butter = zeros(image\_size, image\_size);

for u = 1:image\_size

for v = 1:image\_size

D = sqrt((u - image\_size/2)^2 + (v - image\_size/2)^2); % "image/2" is center of F

H\_Butter(u, v) = 1 / (1 + (D / D0)^(2 \* n));

end

end

F\_butter\_filtered = F .\* H\_Butter; % C = A.\*B multiplies arrays A and B by multiplying corresponding elements.

Butter\_filtered\_image = ifft2(ifftshift(F\_butter\_filtered));

subplot(5, 3, 7);

imshow(log(1 + abs(F\_butter\_filtered)), []);

title('Butterworth energy');

subplot(5, 3, 8);

imshow(real(Butter\_filtered\_image), []);

title('Butterworth picture');

**% Exp**

n = 1;

H\_Exp = zeros(image\_size, image\_size);

for u = 1:image\_size

for v = 1:image\_size

D = sqrt((u - image\_size/2)^2 + (v - image\_size/2)^2); % "image/2" is center of F

H\_Exp(u, v) = exp((-D / D0)^n);

end

end

F\_exp\_filtered = F .\* H\_Exp; % C = A.\*B multiplies arrays A and B by multiplying corresponding elements.

Exp\_filtered\_image = ifft2(ifftshift(F\_exp\_filtered));

subplot(5, 3, 10);

imshow(log(1 + abs(F\_exp\_filtered)), []);

title('Exp energy');

subplot(5, 3, 11);

imshow(real(Exp\_filtered\_image), []);

title('Exp picture');

**% Trapezium**

D1 = 15;

H\_Trap = zeros(image\_size, image\_size);

for u = 1:image\_size

for v = 1:image\_size

D = sqrt((u - image\_size/2)^2 + (v - image\_size/2)^2); % "image/2" is center of F

if D < D1

H\_Trap(u,v) = 1;

else if (D1<= D) && (D<= D0)

H\_Trap(u,v) = (D-D0)/(D1-D0);

else

H\_Trap(u,v) = 0;

end

end

end

end

F\_trap\_filtered = F .\* H\_Trap; % C = A.\*B multiplies arrays A and B by multiplying corresponding elements.

Trap\_filtered\_image = ifft2(ifftshift(F\_trap\_filtered));

subplot(5, 3, 13);

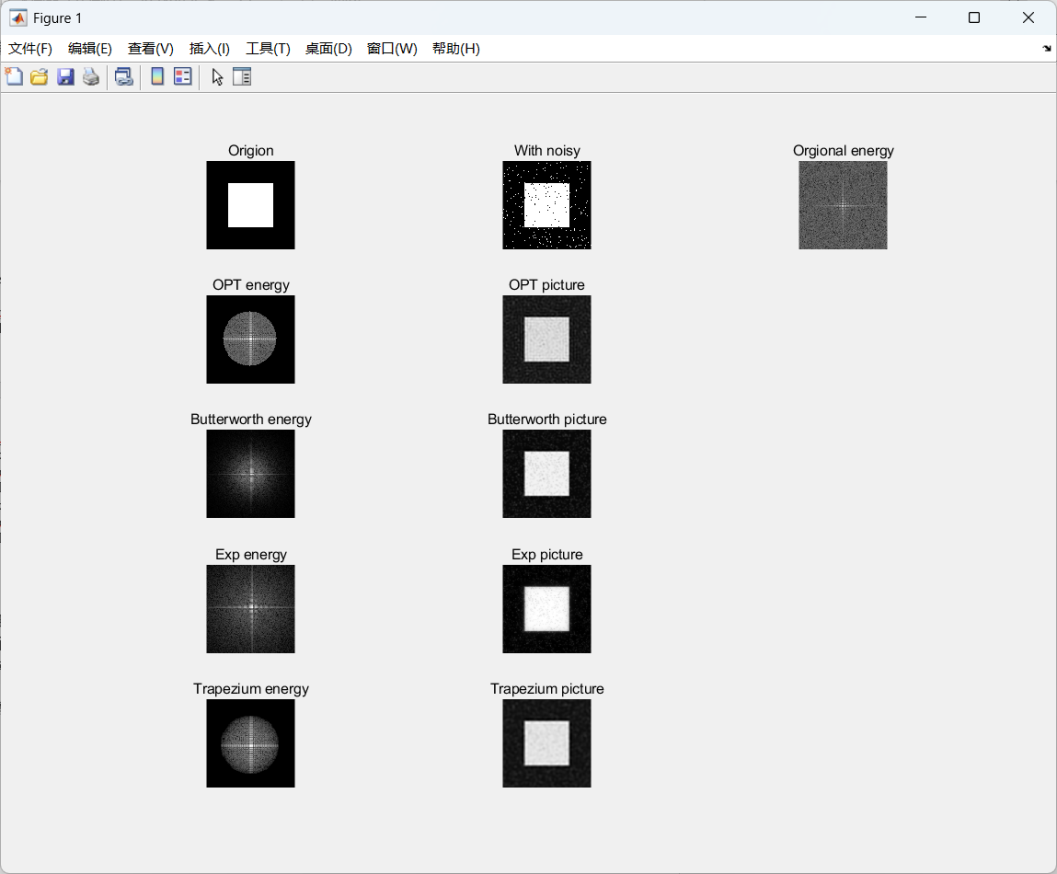
imshow(log(1 + abs(F\_trap\_filtered)), []);

title('Trapezium energy');

subplot(5, 3, 14);

imshow(real(Trap\_filtered\_image), []);

title('Trapezium picture');

****

理想低通滤波器：适用于需要完全去除高频噪声的场景，代码实现简单，但会引入明显的振铃效应，导致边缘模糊。

巴特沃斯低通滤波器：适用于需要平滑过渡的场景，可以通过调节参数达到需要的结果，相对代码计算量大，减少振铃效应，但高频去除效果不如理想低通滤波器。

指数低通滤波器：适用于需要平滑过渡且实现简单的场景，可以通过调参得到需要的结果，但高频去除效果和边缘保留效果一般。

梯形低通滤波器：适用于需要灵活控制滤波效果的场景，过渡平滑，减少了振铃效应，但实现复杂度较高，高频去除效果一般。

**2、图像频域平滑（锐化）：选择一幅图像，例如rice.png，分别使用理想高通滤波器、巴特沃斯高通滤波器、指数高通滤波器和梯形高通滤波器（至少使用两种高通滤波器），显示滤波前后的频域能量分布图，空间图像。分析不同滤波器处理效果及其优缺点。**

**四、实验图像**

rice.png wire.bmp

# 实验六 图像几何变换

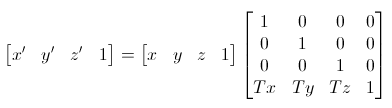
**一、实验目的与要求**

1．了解图像放大、缩小和旋转的操作

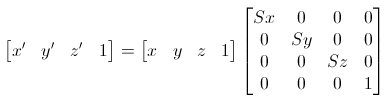
2．掌握常用的图像插值技术

**二、实验相关知识**

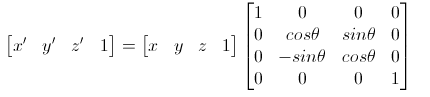
**几何变换就是在一个参考坐标系下将物体从一个位置移动到另一个位置的变换。几何变换大致分为：平移变换（Translation），缩放变换（Scaling），旋转变换（Rotation），仿射变换（Affine），投影变换（Projective）。**

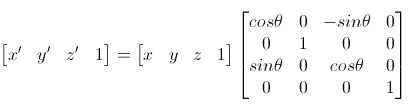
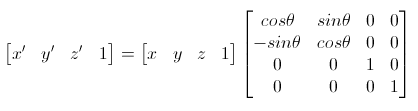
**平移变换： 将三维空间中的一个点[x, y, z, 1]移动到另外一个点[x’, y’, z’, 1]，三个坐标轴的移动分量分别为dx=Tx, dy=Ty, dz=Tz, 即  
**

缩放变换：**将模型放大或者缩小，本质也是对模型上每个顶点进行放大和缩小（顶点坐标值变大或变小），假设变换前的点是[x, y, z, 1]，变换后的点是[x’, y’, z’, 1]，**

****

**旋转变换：某个点，在坐标系中的变换为旋转角度为θ，并且逆时针旋转为正；又可以分为绕X轴、Y轴、Z轴或者绕某一个点、某条线旋转，后者需要具体问题具体分析了，并不能给出统一的表达式。  
其矩阵表达式分别为：**

**（绕X轴）  
**

**（绕Y轴）  
  
（绕Z轴）  
**

和图像几何变换有关的几个常用Matlab函数：

（1）[imcrop](https://www.mathworks.com/help/releases/R2021a/images/ref/imcrop.html?searchHighlight=imcrop&s_tid=doc_srchtitle)-裁剪图像

此 MATLAB 函数 创建与当前图窗中显示的灰度、真彩色或二值图像相关联的交互式裁剪图像工具。imcrop返回裁剪的图像 Icropped。

（2）[imrotate](https://www.mathworks.com/help/releases/R2021a/images/ref/imrotate.html?searchHighlight=imrotate&s_tid=doc_srchtitle) 旋转图像

此 MATLAB 函数 将图像 I 围绕其中心点逆时针方向旋转 angle 度。要顺时针旋转图像，请为 angle 指定负值。imrotate 使输出图像 J 足够大，可以包含整个旋转图像。默认情况下，imrotate 使用最近邻点插值，将 J 中位于旋转后的图像外的像素的值设置为 0。

（3）[imresize](https://www.mathworks.com/help/releases/R2021a/matlab/ref/imresize.html?searchHighlight=imrotate&s_tid=doc_srchtitle)-调整图像大小

此 MATLAB 函数 返回图像 B，它是将 A 的长宽大小缩放 scale 倍之后的图像。输入图像 A 可以是灰度、RGB 或二值图像。如果 A 有两个以上维度，则 imresize 只调整前两个维度的大小。如果 scale 在 [0, 1] 范围内，则 B 比 A 小。如果 scale 大于 1，则 B 比 A 大。默认情况下，imresize 使用双三次插值

（4）imtranslate -平移图像

这个MATLAB函数通过平移中指定的二维或三维平移向量平移图像A。

（5）[impyramid](https://www.mathworks.com/help/releases/R2021a/images/ref/impyramid.html?searchHighlight=impyramid&s_tid=doc_srchtitle)-图像金字塔减少和扩大

这个MATLAB函数计算高斯金字塔将a缩小或扩展一级。

**三、实验内容**

**1、编写函数，实现图像裁剪；**

**My\_cut.m**

function counts = my\_cut(image, startx,starty,endx,endy)

[height,width,channels] = size(image);

counts = zeros(endy-starty+1, endx-startx+1, channels, 'uint8');

for i = startx:endx

for j =starty:endy

counts(i-startx+1,j-starty+1,:) = image(i,j,:);

end

end

if channels == 3

counts = counts(:, :, [1, 2, 3]);

end

counts = uint8(counts);

end

**main.m**

image = imread('colorPic.png');

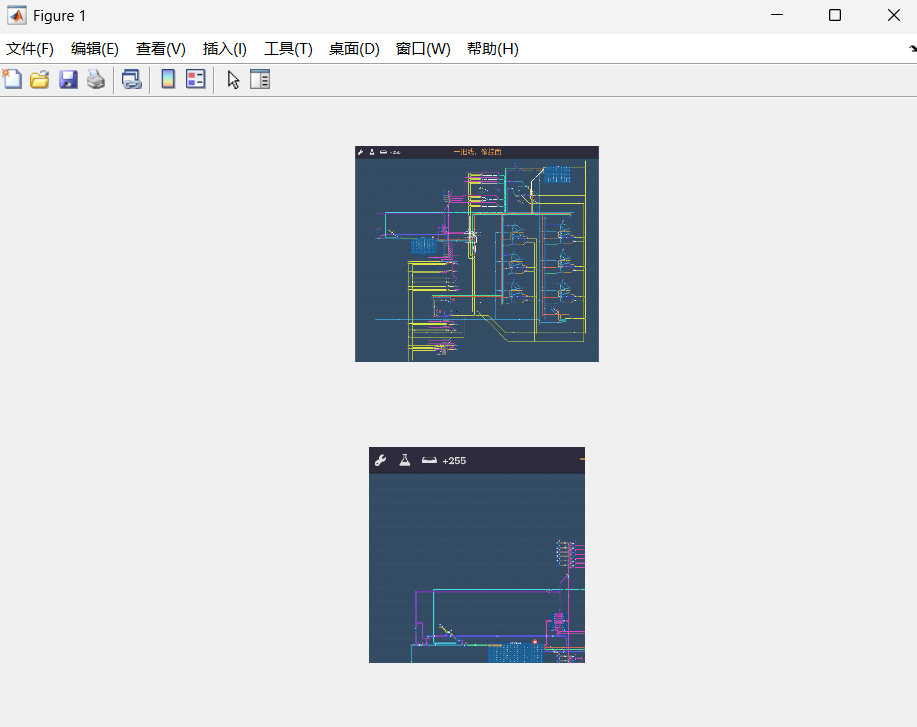
subplot(2,1,1);

imshow(image,[]);

newPic = my\_cut(image,1,1,500,500);

subplot(2,1,2);

imshow(newPic,[]);



**2、编写函数，实现图像缩小：行列等比例缩小，及行列不等比例缩小；**

**my\_Smallzoom.m**

function counts = my\_Smallzoom(image, height\_zoom,width\_zoom)

[height,width,channels] = size(image);

new\_height = round(height \* height\_zoom);

new\_width = round(width \* width\_zoom);

counts = zeros(new\_height, new\_width, channels, 'uint8');

for c = 1:channels

for i = 1:new\_height

for j = 1:new\_width

orig\_i = round(i / height\_zoom);

orig\_j = round(j / width\_zoom);

orig\_i = min(max(orig\_i, 1), height);

orig\_j = min(max(orig\_j, 1), width);

counts(i, j, c) = image(orig\_i, orig\_j, c);

end

end

end

end

**main.m**

image = imread('colorPic.png');

subplot(2,1,1);

imshow(image,[]);

newPic = my\_Smallzoom(image,0.1,0.1);

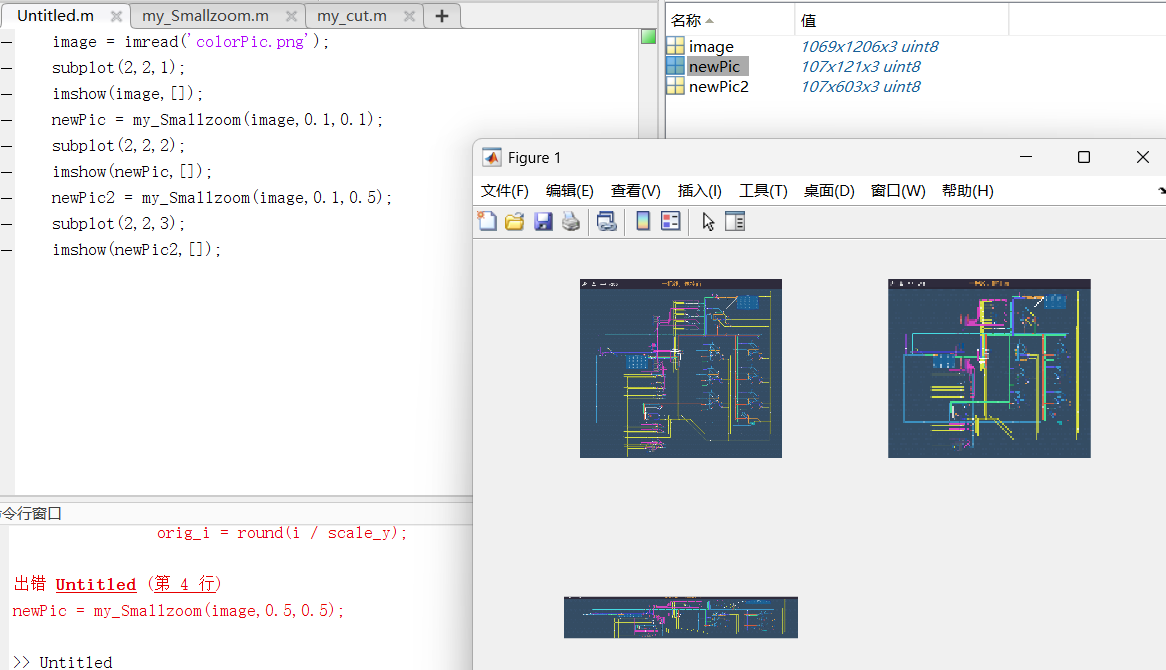
subplot(2,1,2);

imshow(newPic,[]);

newPic2 = my\_Smallzoom(image,0.1,0.5);

subplot(2,2,3);

imshow(newPic2,[]);

****

**3、编写函数，实现图像放大，并使用最近邻插值法进行插值；**

**my\_BigNearestzoom.m**

function counts = my\_BigNearestzoom(image,height\_zoom,width\_zoom)

[height, width, channels] = size(image);

new\_height = round(height \* height\_zoom);

new\_width = round(width \* width\_zoom);

counts = zeros(new\_height, new\_width, channels, 'uint8');

for c = 1:channels

for i = 1:new\_height

for j = 1:new\_width

% Select Nearest Situation

orig\_i = round((i - 0.5) / height\_zoom + 0.5);

orig\_j = round((j - 0.5) / width\_zoom + 0.5);

orig\_i = min(max(orig\_i, 1), height);

orig\_j = min(max(orig\_j, 1), width);

counts(i, j, c) = image(orig\_i, orig\_j, c);

end

end

end

end

**main.m**

image = imread('colorPic.png');

subplot(2,2,1);

imshow(image,[]);

newPic = my\_BigNearestzoom(image,1.5,1.5);

subplot(2,2,2);

imshow(newPic,[]);

newPic2 = my\_BigNearestzoom(image,1,1.5);

subplot(2,2,3);

imshow(newPic2,[]);

****

**4、编写函数，实现图像放大，并使用双线性插值法进行插值；**

**my\_BigBInsertzoom.m**

function counts = my\_BigBInsertzoom(image, height\_zoom, weight\_zoom)

[height, width, channels] = size(image);

new\_height = round(height \* height\_zoom);

new\_width = round(width \* weight\_zoom);

counts = zeros(new\_height, new\_width, channels, 'uint8');

for c = 1:channels

for i = 1:new\_height

for j = 1:new\_width

orig\_i = (i - 0.5) / height\_zoom + 0.5;

orig\_j = (j - 0.5) / weight\_zoom + 0.5;

i1 = max(floor(orig\_i),1);

i2 = min(i1 + 1, height);

j1 = max(floor(orig\_j),1);

j2 = min(j1 + 1, width);

di = orig\_i - i1; % u

dj = orig\_j - j1; % v

Q11 = double(image(i1, j1, c));

Q12 = double(image(i1, j2, c));

Q21 = double(image(i2, j1, c));

Q22 = double(image(i2, j2, c));

R1 = (1 - dj) \* Q11 + dj \* Q12;

R2 = (1 - dj) \* Q21 + dj \* Q22;

P = (1 - di) \* R1 + di \* R2;

counts(i, j, c) = uint8(P);

end

end

end

end

**main.m**

image = imread('colorPic.png');

subplot(2,2,1);

imshow(image,[]);

newPic = my\_BigNearestzoom(image,1.5,1.5);

subplot(2,2,2);

imshow(newPic,[]);

newPic2 = my\_BigNearestzoom(image,1,1.5);

subplot(2,2,3);

imshow(newPic2,[]);

newPic3 = my\_BigBInsertzoom(image,1,1.5);

subplot(2,2,4);

imshow(newPic3,[]);

*% 可发现，双线性插值法得到的放大图片更加的平滑。*

****

1. **编写函数，实现图像旋转和插值。**

**my\_rotate.m**

function counts = my\_rotate(image, radi)

radi = radi \* pi / 180;

[height, width, channels] = size(image);

center\_x = (height + 1) / 2;

center\_y = (width + 1) / 2;

new\_height = ceil(abs(height \* cos(radi)) + abs(width \* sin(radi)));

new\_width = ceil(abs(height \* sin(radi)) + abs(width \* cos(radi)));

counts = zeros(new\_height, new\_width, channels, 'uint8');

new\_center\_x = (new\_height + 1) / 2;

new\_center\_y = (new\_width + 1) / 2;

for c = 1:channels

for i = 1:new\_height

for j = 1:new\_width

x = (i - new\_center\_x) \* cos(radi) + (j - new\_center\_y) \* sin(radi) + center\_x;

y = -(i - new\_center\_x) \* sin(radi) + (j - new\_center\_y) \* cos(radi) + center\_y;

i1 = floor(x);

i2 = min(i1 + 1, height);

j1 = floor(y);

j2 = min(j1 + 1, width);

if i1 >= 1 && i1 <= height && j1 >= 1 && j1 <= width && i2 >= 1 && i2 <= height && j2 >= 1 && j2 <= width

di = x - i1;

dj = y - j1;

Q11 = double(image(i1, j1, c));

Q12 = double(image(i1, j2, c));

Q21 = double(image(i2, j1, c));

Q22 = double(image(i2, j2, c));

R1 = (1 - dj) \* Q11 + dj \* Q12;

R2 = (1 - dj) \* Q21 + dj \* Q22;

P = (1 - di) \* R1 + di \* R2;

counts(i, j, c) = uint8(P);

end

end

end

end

end

**main.m**

image = imread('colorPic.png');

new = my\_rotate(image,45);

imshow(counts, []);



**四、实验图像**

**自选彩色图片即可**

# 实验七 图像编码

**一、实验目的与要求**

1．了解图像编码常用方法

2．掌握图像编码和解码技术

**二、实验相关知识**

[哈夫曼](https://baike.so.com/doc/6525310.html" \t "_blank)编码(Huffman Coding)，又称霍夫曼编码，是一种编码方式，哈夫曼编码是可变[字长](https://baike.so.com/doc/6261553-6474973.html" \t "_blank)编码(VLC)的一种。Huffman于1952年提出一种编码方法，该方法完全依据[字符](https://baike.so.com/doc/2529629-2672477.html" \t "_blank)出现概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做Huffman编码(有时也称为霍夫曼编码)。

费诺-香农编码,将信源符号以概率递减的次序排列进来，将排列好的信源符号划分为两大组，使两组的概率和近于相同,并各赋于一个二元码符号"0"和"1".然后，将每一大组的信源符号再分成两组，使同一组的两个小组的概率和近于相同，并又分别赋予一个二元码符号。依次下去，直至每一个小组只剩下一个信源符号为止。这样，信源符号所对应的码符号序列则为编得的码字。译码原理，按照编码的二叉树从树根开始，按译码序列进行逐个的向其叶子结点走，直到找到相应的信源符号为止。之后再把指示标记回调到树根，按照同样的方式进行下一序列的译码到序列结束。如果整个译码序列能够完整的译出则返回成功，否则则返回译码失败。

**三、实验内容**

1. 通过Matlab编程，对7\*7图像子块实施huffman编码和费诺-香农编码；

% We used huffmanenco &huffmandict to success huffman`s code

**main.m**

image\_block = [0,0,2,2,2,2,7;

0,0,0,2,2,2,7;

1,1,2,6,6,6,7;

4,3,3,5,5,5,7;

4,3,3,5,5,5,5;

5,4,5,5,5,5,5;

5,5,5,5,5,5,5];

unique\_vals = unique(image\_block);

counts = histcounts(image\_block(:), [unique\_vals; max(unique\_vals)+1]);

prob = counts / numel(image\_block);

[dict, avglen\_huffman] = huffmandict(unique\_vals, prob);

huffman\_encoded = huffmanenco(image\_block(:), dict);

fprintf('Huffman`s Code:\n');

for i = 1:length(dict)

symbol = dict{i, 1};

code = dict{i, 2};

fprintf('Val %d: %s\n', symbol, num2str(code));

end

[shannon\_dict, avglen\_shannon] = shannonfano(unique\_vals, prob);

fprintf('Shannon`s Code:\n');

for i = 1:length(shannon\_dict)

symbol = shannon\_dict{i, 1};

code = shannon\_dict{i, 2};

fprintf('Val %d: %s\n', symbol, code);

end

shannon\_encoded = shannonenco(image\_block(:), shannon\_dict);

original\_size = numel(image\_block) \* 3;

huffman\_size = numel(huffman\_encoded);

shannon\_size = length(shannon\_encoded);

huffman\_compression\_ratio = original\_size / huffman\_size;

shannon\_compression\_ratio = original\_size / shannon\_size;

huffman\_efficiency = avglen\_huffman / entropy(prob);

shannon\_efficiency = avglen\_shannon / entropy(prob);

% Out put

fprintf('Huffman`s Efficiency: %.2f\n', huffman\_efficiency);

fprintf('Huffman`s Compression Ratio: %.2f\n', huffman\_compression\_ratio);

fprintf('Huffman`s Total BITS: %d\n', huffman\_size);

fprintf('Shannon`s Efficiency: %.2f\n', shannon\_efficiency);

fprintf('Shannon`s Compression Ratio: %.2f\n', shannon\_compression\_ratio);

fprintf('Shannon`s Total BITS: %d\n', shannon\_size);

**Shannonfano.m % Shannon encode**

function [dict, avglen] = shannonfano(symbols, prob)

function codes = shannonfano\_recursive(symbols, prob)

if length(symbols) == 1

codes = {''};

return;

end

[~, idx] = sort(prob, 'descend');

symbols = symbols(idx);

prob = prob(idx);

total = sum(prob);

cum\_prob = cumsum(prob);

% find Split Index

diff = abs(cum\_prob - total / 2);

[~, split\_idx] = min(diff);

left\_symbols = symbols(1:split\_idx);

right\_symbols = symbols(split\_idx+1:end);

left\_codes = shannonfano\_recursive(left\_symbols, prob(1:split\_idx));

right\_codes = shannonfano\_recursive(right\_symbols, prob(split\_idx+1:end));

codes = cell(size(symbols));

for i = 1:split\_idx

codes{idx(i)} = ['0', left\_codes{i}];

end

for i = split\_idx+1:length(symbols)

codes{idx(i)} = ['1', right\_codes{i-split\_idx}];

end

end

codes = shannonfano\_recursive(symbols, prob);

dict = [num2cell(symbols(:)), codes(:)];

a = cellfun('length', codes);

b = prob';

avglen = sum(a .\* b);

end

**Shannonenco.m % decode**

function encoded = shannonenco(data, dict)

encoded = '';

for i = 1:length(data)

idx = find(cell2mat(dict(:, 1)) == data(i));

encoded = [encoded, dict{idx, 2}];

end

end

2.计算上述编码的效率，压缩比并对相应数据做出解释；

I.编码策略：

哈夫曼编码通过构建哈夫曼树来生成编码，能够确保每个符号的编码长度尽可能接近其概率的负对数。费诺-香农编码通过递归分割符号集合来生成编码，分割策略可能导致某些符号的编码长度略大于其概率的负对数。

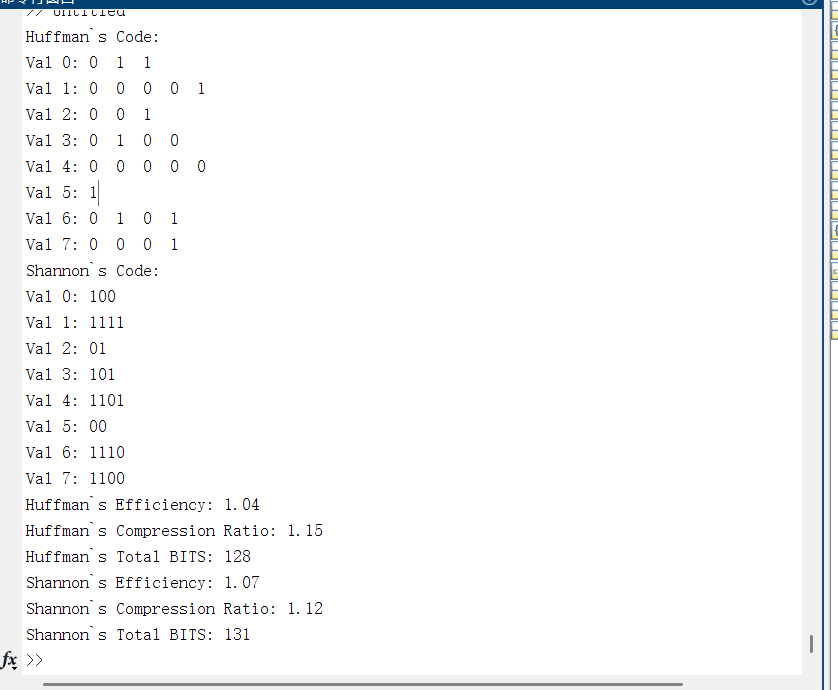
II.编码长度：

哈夫曼编码生成的编码长度是整数位长，且每个符号的编码长度与其概率的负对数接近。费诺-香农编码生成的编码长度也是整数位长，但由于分割策略的原因，某些符号的编码长度可能略大于哈夫曼编码。

III.实现复杂度：

哈夫曼编码的实现复杂度较高，需要构建哈夫曼树。费诺-香农编码的实现复杂度较低，通过递归分割符号集合即可生成编码。

% after execute：



**四、实验图像**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **2** | **2** | **2** | **2** | **7** |
| **0** | **0** | **0** | **2** | **2** | **2** | **7** |
| **1** | **1** | **2** | **6** | **6** | **6** | **7** |
| **4** | **3** | **3** | **5** | **5** | **5** | **7** |
| **4** | **3** | **3** | **5** | **5** | **5** | **5** |
| **5** | **4** | **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** | **5** | **5** |

# 实验八 图像分割

**一、实验目的与要求**

1．了解图像分割的意义和常用方法

2．掌握常用的图像分割方法

**二、实验相关知识**

图像分割就是把图像分成互不重叠的区域并提取出感兴趣目标的技术，是由图像处理到图像分析的关键步骤。

和本实验有关的常用Matlab函数：

**edge：**检测灰度或二值图像的边缘，返回一个二值图像，1像素是检测到的边缘，0像素是非边缘

用法：BW=edge(I,'sobel',thresh,direction); %I为检测对象；

边缘检测算子可用sobel、roberts、prewitt、zerocross、log、canny；

thresh指定阈值，检测时忽略所有小于阈值的边缘，默认自动选择阈值；

direction指定方向，可选项有horizontal、vertical或both，在指定的方向上用算子进行边缘检测

**三、实验内容**

**1、分别用Roberts、Prewitt、Sobel三种边缘检测算子，对图像wire.bmp进行水平、垂直及各个方向的边界检测，并将检测结果转化为白底黑线条的方式显示出来；**

I = imread('cell.png');

I\_gray = im2gray(I);

% Roberts

BW\_roberts\_h = ~edge(I\_gray, 'roberts', [], 'horizontal');

BW\_roberts\_v = ~edge(I\_gray, 'roberts', [], 'vertical');

BW\_roberts\_both = ~edge(I\_gray, 'roberts', [], 'both'); % this threshold can be improved.(the 2rd Val).

% Prewitt

BW\_prewitt\_h = ~edge(I\_gray, 'prewitt', [], 'horizontal');

BW\_prewitt\_v = ~edge(I\_gray, 'prewitt', [], 'vertical');

BW\_prewitt\_both = ~edge(I\_gray, 'prewitt', [], 'both');

% Sobel

BW\_sobel\_h = ~edge(I\_gray, 'sobel', [], 'horizontal');

BW\_sobel\_v = ~edge(I\_gray, 'sobel', [], 'vertical');

BW\_sobel\_both = ~edge(I\_gray, 'sobel', [], 'both');

figure;

subplot(3, 3, 1), imshow(BW\_roberts\_h,[]), title('Roberts Horizontal');

subplot(3, 3, 2), imshow(BW\_roberts\_v,[]), title('Roberts Vertical');

subplot(3, 3, 3), imshow(BW\_roberts\_both,[]), title('Roberts Both');

subplot(3, 3, 4), imshow(BW\_prewitt\_h,[]), title('Prewitt Horizontal');

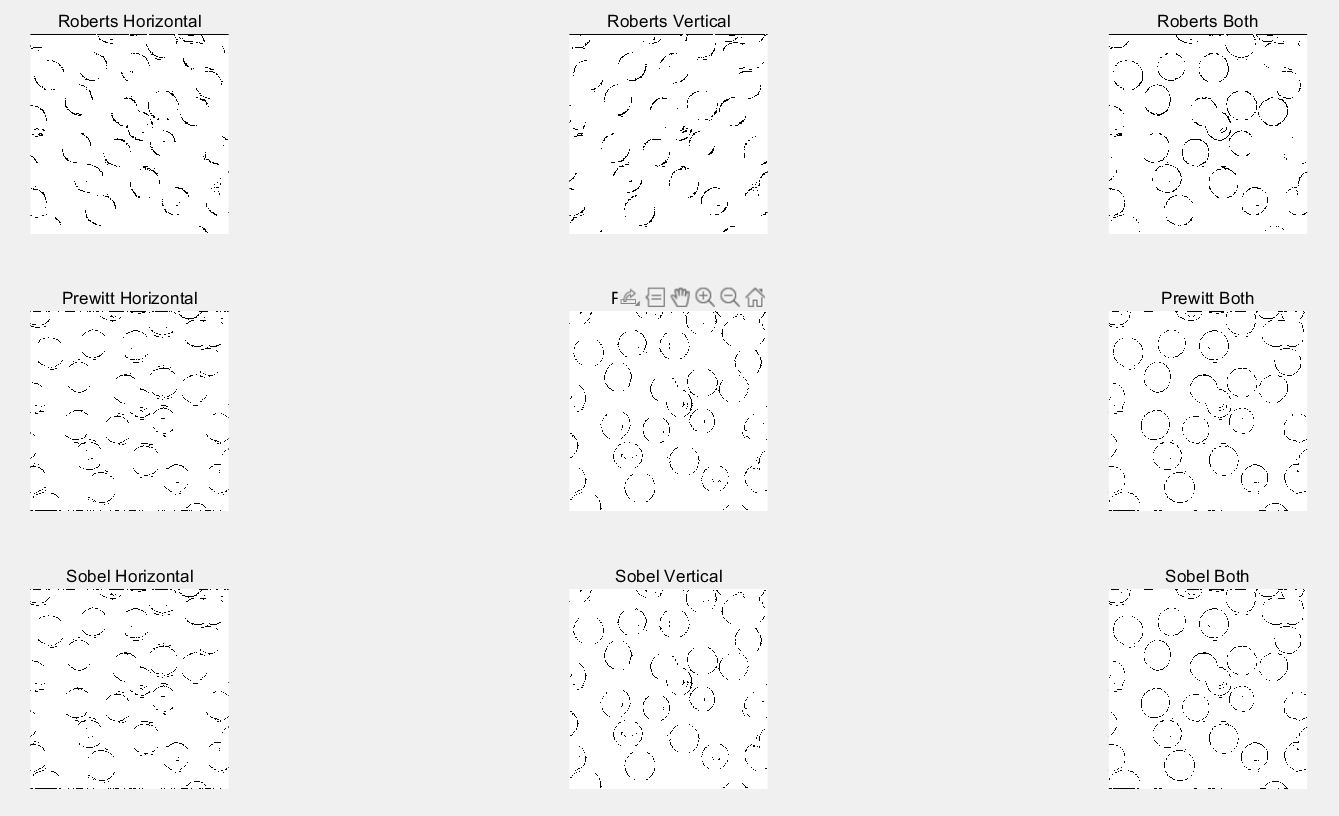
subplot(3, 3, 5), imshow(BW\_prewitt\_v,[]), title('Prewitt Vertical');

subplot(3, 3, 6), imshow(BW\_prewitt\_both,[]), title('Prewitt Both');

subplot(3, 3, 7), imshow(BW\_sobel\_h,[]), title('Sobel Horizontal');

subplot(3, 3, 8), imshow(BW\_sobel\_v,[]), title('Sobel Vertical');

subplot(3, 3, 9), imshow(BW\_sobel\_both,[]), title('Sobel Both');



1. **使用手动阈值分割法对bottle图像进行分割，显示分割结果，并分析其优缺点。**

image = imread('bottle.jpg');

gray\_image = rgb2gray(image);

threshold\_value = 127; % can be improved

binary\_image = gray\_image > threshold\_value;

figure;

subplot(1, 2, 1);

imshow(gray\_image);

title('Original Image');

subplot(1, 2, 2);

imshow(binary\_image);

title('Threshold Segmentation');



手动阈值分割法直接将大于等于阈值的设置为白色，小于的设置为背景。优点是简单易行，计算速度快，适用于灰度分布明显的图像。然而，它也有一些缺点，如对噪声敏感，阈值选择困难，适用性差，对灰度分布不明显的图像效果不好。

1. **对下图A进行处理，得到类似图B的样子；**

**% First using Manual threshold segmentation (100), Then Using Roberts(0.05).**

close all;

image = imread('cell.png');

gray\_image = im2gray(image);

threshold\_value = 100;

binary\_image = gray\_image > threshold\_value;

BW\_roberts\_both = ~edge(~binary\_image, 'roberts',0.05 , 'both');

figure;

subplot(1, 3, 1);

imshow(gray\_image,[]);

title('Original Image');

subplot(1, 3, 2);

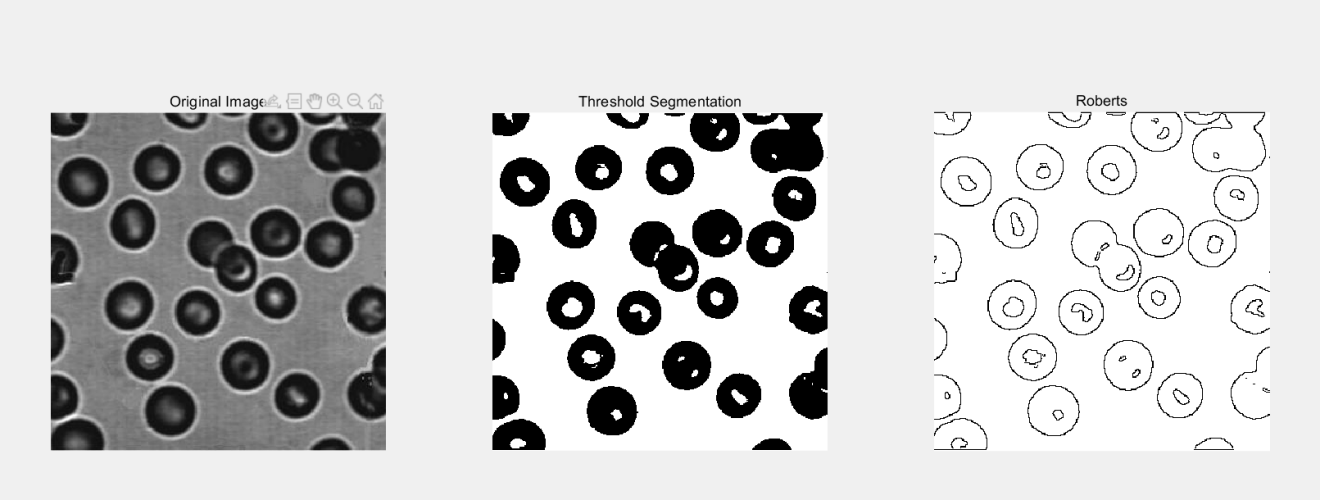
imshow(binary\_image,[]);

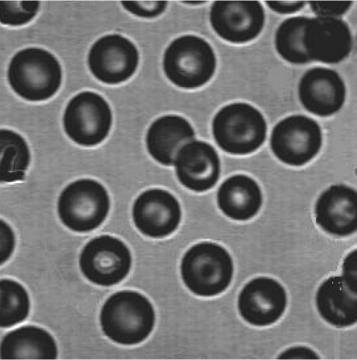
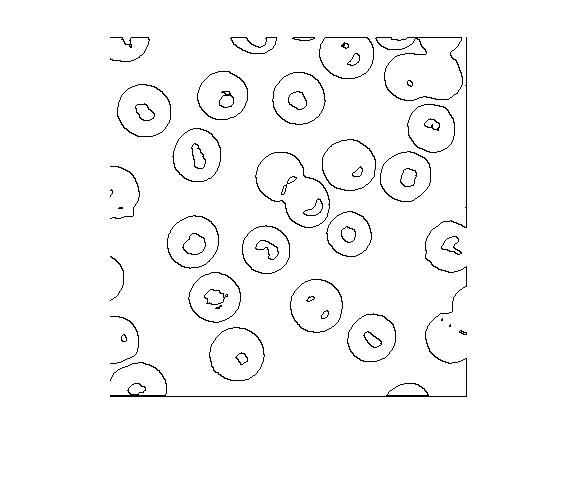
title('Threshold Segmentation');

subplot(1,3,3)

imshow(BW\_roberts\_both,[]);

title('Roberts');



图A 图B

**四、实验图像**

rice.png wire.bmp bottle