# MATLAB

# 图像处理实例详解

——程序部分

# 目录

# 第 2 章　MATLAB 基础

```
close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=240;
B1=int8(A);                         %将 A 进行强制类型转换为 8 位有符号整数
B2=int16(A);                        %将 A 进行强制类型转换为 16 位有符号整数
B3=uint8(A);                        %将 A 进行强制类型转换为 8 位无符号整数
B4=uint16(A);                       %将 A 进行强制类型转换为 16 位无符号整数

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=123.567;
B=single(A);                        %将双精度浮点型转换为单精度浮点型
C=int16(A);                         %将双精度浮点型转换为 16 位有符号整型

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A1=round(-1.9);                     %应用 round( )函数对浮点数取整
A2=round(3.4);
B1=fix(-1.9);                       %应用 fix( )函数对浮点数取整
B2=fix(3.4);
C1=floor(-1.9);                     %应用 floor( )函数对浮点数取整
C2=floor(3.4);
D1=ceil(-1.9);                      %应用 ceil( )函数对浮点数取整
D2=ceil(3.4);

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
S='Please create a string!';        %创建字符串
[m,n]=size(S);                      %计算字符串大小
a=double(S);                        %计算字符串的 ASCII 码
S1=lower(S);                        %将所有字母转换成小写字母
S2=upper(S);                        %将所有字母转换成大写字母

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
S1='How are you!     ';             %创建 S1 字符串
S2='Fine, Thank you!';              %创建 S2 字符串
A=[S1,S2];                          %合并字符数组
B=char(S1,S2);                      %连接字符串 S1 和 S2
C=strcat(S1,S2);                    %横向连接字符串 S1 和 S2
D=strvcat(S1,S2);                   %纵向连接字符串 S1 和 S2
E=S2(7:16);                         %拆分截取字符串 S2

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
S1='My name is Tommy';
S2='Nice to meet you';
```

```
a=S1==S2;                    %判断两个字符串是否相等
b=S1>S2;                     %判断 S1 是否大于 S2
c=lt(S1,S2);                 %应用函数判断 S1 是否小于 S2
d=S1<S2;                     %判断 S1 是否小于 S2

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
S1='Good morning!';
S2='good morning, Sir.';
a=strcmp(S1,S2);             %比较两个字符串大小
b=strncmp(S1,S2,7);          %比较两个字符串前 7 个字符大小，区分大小写
c=strncmpi(S1,S2,7);         %比较两个字符串前 7 个字符大小，不区分大小写

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
num=rand(3,3);               %产生 3×3 随机矩阵
s1=num2str(num);             %将数值转换成字符串
s2=num2str(pi,10);           %将 pi 的前 10 位转换成字符串
int=12345;
s3=int2str(int);             %将整数转换成字符串
s4=mat2str(pascal(3));       %将矩阵转换成字符串
num1=str2num('123456');      %将字符串转换成数值
num2=str2double('1234.56');  %将字符串转换成双精度浮点数

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
a=bin2dec('1011001');        %将二进制数转换成十进制数
b=dec2bin(18);               %将十进制数转换成二进制数
c=hex2dec('9A2B');           %将十六进制数转换成十进制数
d=dec2hex(97);               %将十进制数转换成十六进制数
e=base2dec('212',3);         %将任意进制数转换成十进制数

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
s = 'Find the starting indices of the shorter string.';
a1=findstr(s, 'the');        %在长字符串中查找短字符串
a2=findstr('the', s);
a3=findstr(s,'a');
a4=findstr(s,' ');
a5=strfind(s, 'the');        %在前字符串中查找后字符串
a6=strfind(s, 'a');
a7=strfind('the',s);

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
s1 = 'This is a good example.';
s2=strrep(s1, 'good', 'great');   %在在符串中查找 good 用 great 替换
s3=strrep(s1,'Good','great');
```

```
close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=[0 0 1;2 0 0;0 3 0];
B=logical(A);                       %将矩阵 A 转换成逻辑矩阵 B
C=true(3);                          %生成 3 阶逻辑真矩阵
D=false(3);                         %生成 3 阶逻辑假矩阵

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
fhandle=@sin;                       %建立一个函数句柄
y1=fhandle(2*pi);                   %用函数句柄调用函数
y2=sin(2*pi);                       %直接调用函数

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
f1=@help;                           %创建函数句柄
s1=func2str(f1);                    %将函数句柄转换成字符串
f2=str2func('help');                %将字符串转换成函数句柄
a1=isa(f1,'function_handle');       %判断 f1 是否为函数句柄
a2=isequal(f1,f2);                  %判断 f1 和 f2 是否指向同一函数
a3=functions(f1);                   %获取 f1 信息

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
stu(1).name='LiMing';               %直接创建结构体 stu
stu(1).number='20120101';
stu(1).sex='f';
stu(1).age=20;
stu(2).name='WangHong';
stu(2).number='20120102';
stu(2).sex='m';
stu(2).age=19;
student=struct('name',{'LiMing','WangHong'},'number',{'20120101','20120102'},'sex',{'f','m' },'age
',{20,19});                         %应用 struct 函数创建结构体 student
stu;
stu(1);
stu(2);
student;
student(1);
student(2);

close all; clear all; clc;          %关闭所有图形窗口，清除工作空间所有变量，清空命令行
stu=struct('name',{'LiMing','WangHong'},'number',{'20120101','20120102'},'sex',{'f','m' },'age',{20
,19});
a=fieldnames(stu);                  %获取 stu 所有成员名
b=getfield(stu,{1,2},'name');       %获取指定成员内容
c=isfield(stu,'sex');               %判断 sex 是否为 stu 中成员
stunew=orderfields(stu);            %按结构体成员首字母重新排序
```

```
rmfield(stu,'sex');                %删除 sex
s1=setfield(stu(1,1),'sex','M');   %重新设置 stu 中 sex 内容
s2=setfield(stu{1,2},'sex','F');   %重新设置 stu 中 sex 内容
s2(1,2)

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
student{1,1}={'LiMing','WangHong'};%直接赋值法建立细胞数组
student{1,2}={'20120101','20120102'};
student{2,1}={'f','m'};
student{2,2}={20,19};
student;
student{1,1};
student{2,2};
cellplot(student);                 %显示细胞数组结构图
close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
stu=cell(2);                       %cell 函数建立 2×2 细胞数组
stu{1,1}={'LiMing','WangHong'};
stu{1,2}={'20120101','20120102'};
stu{2,1}={'f','m'};
stu{2,2}={20,19};
stu;
cellplot(stu)                      %显示细胞数组结构图

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
stu_cell={'LiMing','20120101','M','20'}; %建立细胞数组
celldisp(stu_cell)                 %显示细胞数组
fields={'name','number','sex','age'};
stu_struct=cell2struct(stu_cell,fields,2);%将细胞数组转换成结构体
stu_struct;
a=iscell(stu_cell);                %判断 stu_cell 是否为细胞数组
b=iscell(stu_struct);
stu_t=struct('name',{'LiMing','WangHong'},'number',{'20120101','20120102'},'sex',{'f','m'  },'age',{
20,19});
stu_c=struct2cell(stu_t);          %将结构体转换成细胞数组
c= {[1] [2 3 4]; [5; 9] [6 7 8; 10 11 12]};%建立细胞数组
m= cell2mat(c);                    %将细胞数组合并成矩阵
M = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20];
C1= mat2cell(M, [2 2], [3 2]);     %将 M 拆分成细胞数组
C2=num2cell(M);                    %将 M 转换成细胞数组
figure;
subplot(121);cellplot(C1);         %显示 C1 结构图
subplot(122);cellplot(C2);         %显示 C2 结构图

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
```

```
A=eye(3);                       %建立 3×3 对角矩阵 A
B=magic(3);                     %建立 3×3 魔方矩阵 B
C1=A.*B;                        %A 点乘 B
C2=A*B;                         %A 乘以 B
C3=A/B;                         %A 左除 B
C4=A./B;                        %A 左点除 B

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A = [2 7 6;9 0 5;3 0.5 6];
B = [8 7 0;3 2 5;4 -1 7];
C1=A<B;                         %小于运算符
C2=lt(A,B);                     %lt 函数求小于
C3=A==B;                        %恒等于
C4=~A;                          %不等于

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A = [0 1 1 0 1];
B = [1 1 0 0 1];
C=A&B;                          %与运算
D=A|B;                          %或运算
E=~A;                           %非运算
F=and(A,B);                     %and 函数与运算
G=xor(A,B);                     %异或运算

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
a=54;
b=12;
c1=(a<b)&&(a*4<b);              %快速与运算
c2=(b*4<a)||(b<a);              %快速或运算

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
a1=2+3*4>5&1;                   %运算符优先级
a2=2+(3*4>5&1);

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=[1,2,3,4,5];                  %建立行向量
B=[1 2 3 4 5];                  %建立行向量

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
C=[1,2,3;4,5,6;7,8,9;10,11,12]; %建立 4×3 矩阵

close all; clear all; clc;     %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=ones(3);                      %建立一个元素都为 1 的 3 阶方阵
B=ones(2,3);                    %建立一个元素都为 1 的 2×3 阶矩阵
```

```
C=zeros(2,3);                %建立一个元素都为 0 的 2×3 阶矩阵
D=eye(3);                    %建立一个对角元素为 1 其它元素为 0 的 3 阶方阵
v=[1,2,3,4,5];               %生成一个行向量
E=diag(v);                   %将一个向量变成一个对角矩阵
F=magic(3);                  %建立一个 3 阶魔方方阵
G=rand(2,3);                 %建立一个 2×3 阶随机矩阵

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=ones(2,4)*3;
B=rand(3,4);
C=[A;B];                     %纵向合并两矩阵
D=[A B];                     %横向合并两矩阵

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=eye(3);
B=rand(3);
C1=repmat(A,2,3);            %将矩阵复制合并成新矩阵
C2=blkdiag(A,B);             %将矩阵合并成对角矩阵

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=magic(5);
B=A(:,[2 4]);                %提取矩阵 A 中的第 2 列和第 4 列组成矩阵 B
C=A([1 3],[2 4]);            %提取矩阵 A 中的第 1 行和第 3 行，
                               第 2 列和第 4 列元素组成矩阵 C
D=A(1:3,3:4);                %提取矩阵 A 中的 1 至 3 行，3 至 4 列中元素组成新矩阵 D
E=A([1:3;4 5 7;10:12]);      %提取矩阵 A 中单下标为 1 至 3 的元素为第一行
                             %下标为 4,5,7 的元素为第二行，下标为 10 至 12 的为第三
                               行组成矩阵 E

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=[1,3,4;5,6,7;1,0,0];
a=det(A);                    %求行列式的值

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
B1= transpose(A);            %求转置矩阵
B2= A';

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=magic(3);
B=[1 2 3 4;5 6 7 8];
C=inv(A);                    %求逆矩阵
D=pinv(B);                   %求伪逆矩阵

close all; clear all; clc;   %关闭所有图形窗口，清除工作空间所有变量，清空命令行
```

```
rank(A)                          %求矩阵的秩

close all; clear all; clc;       %关闭所有图形窗口，清除工作空间所有变量，清空命令行
k = 5;
hilbert = zeros(k,k);            %产生一个 5×5 全 0 矩阵
for m = 1:k                      %应用 for 给 Hilbert 矩阵赋值
    for n = 1:k
        hilbert(m,n) = 1/(m+n -1);
    end
end
format rat

close all; clear all; clc;       %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=[1,2,3,4,5,6];                 %计算矩阵 A 所有元素和
sum=0;
k=0;
for n=A
n;
k=k+1;
sum=sum+n;
end

close all; clear all; clc;       %关闭所有图形窗口，清除工作空间所有变量，清空命令行
i=1;                             %计算 1 至 100 的和
sum=0;
while(i<=100)
    sum=sum+i;
    i=i+1;
end

close all; clear all; clc;       %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=magic(5);                      %产生 5 阶魔方矩阵 A
a=A(1);                          %查找 A 中最大元素，及元素下标
for i=2:25
    if A(i)>a
        a=A(i);
        n=i;
    end
end

close all; clear all; clc;       %关闭所有图形窗口，清除工作空间所有变量，清空命令行
k=5;
for m = 1:k                      %创建一个 5 阶方阵 A，当行标和列标相等的元素赋 2，
    for n = 1:k                  %行标和列标的差的绝对值为 2 的元素赋 1
```

```
            if m == n
                a(m,n) = 2;
            elseif abs(m-n) == 2
                a(m,n) = 1;
            else
                a(m,n) = 0;
            end
        end
end

close all; clear all; clc;              %关闭所有图形窗口，清除工作空间所有变量，清空命令行
try                                     %打开一个文件名为 girl.bmp 的文件，若文件不存在，则打
开
picture=imread('girl.bmp','bmp');       %一个文件名为 girl.jpg 的文件
filename='girl.bmp';
catch
picture=imread('girl.jpg','jpg');
filename='girl.jpg';
end
filename;
lasterror;

%求 1 至 100 的和
i=1;
s=0;
for i=1:100
s=s+i;
end

close all; clear all; clc;              %关闭所有图形窗口，清除工作空间所有变量，清空命令行
x=0:0.02:2*pi;                          %定义自变量 x 取值
y=sin(x);                               %定义函数 y 与变量 x 的关系，生成绘制图形的数据
plot(x,y);                              %将函数 y 与自变量取值的点连接起来

close all; clear all; clc;              %关闭所有图形窗口，清除工作空间所有变量，清空命令行
x=-20:20;
y=x.^2+2*x+1;                           %输入为一维向量
plot(x,y);
z=magic(4);                             %输入为 4 阶方阵
plot(z);
c=[1+2i,4+3i,7+11i];                    %输入为一维复向量
plot(c);
x1=0:0.01:10;
y1=exp(sin(x1));
```

```
y2=sin(2*x1+2.*pi./3);
y3=exp(-0.1.*x1).*sin(6*x1);
plot(x1,y1,x1,y2,x1,y3);          %在同一图形窗口中画出 y1，y2 和 y3 曲线

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
x=0:0.01:10;
y1=sin(2.*x);
y2=2.*sin(x);
figure(1);                        %打开一个图形窗口
subplot(121);plot(y1);            %将窗口分割成 1×2 两个区域，分别绘制 y1 和 y2
subplot(122);plot(y2);

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
x=0:0.1:10;
y1=sin(2.*x);
y2=2.*sin(x);
plot(x,y1,'b*:',x,y2,'r+-');      %设置颜色、顶点和线型
axis([0 pi 0 2 ]);                %设置坐标轴
title('正弦曲线');                %设置标题行
xlabel('时间/单位：秒');
ylabel('电压/单位：伏特');         %设置横坐标纵坐标
gtext('y=sin(2x)');               %在图中鼠标指定位置添加文字 y=sin(2x)
gtext(2.5,1.5,'y=2sin(x)');       %在图中(2.5, 1.5)处添加文字 y=2sin(x)
grid;                             %设置栅格

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
y=randn(1000,1);                  %建立正态分布的向量
figure;
subplot(121);hist(y);             %采用 hist 绘制默认直方图
subplot(122);hist(y,20)           %采用 hist 绘制指定直方图

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
A=magic(4);
B=[1,2,3;5,5,7;6,3,4;9,4,7];
figure;
subplot(121);bar(A);              %画出 A 的柱状图
subplot(122);barh(B);             %画出 B 的柱状图

close all; clear all; clc;        %关闭所有图形窗口，清除工作空间所有变量，清空命令行
[x,y,z]=peaks;
figure(1)
subplot(121);
contour(z);                       %绘制峰函数的等高线
subplot(122);
```

```
contour(z,16);                          %绘制等高线指定条数

function y = average(x)
%求向量元素平均值.
%函数名为 average，输入参数为一向量
%输入为非向量时报错
[m,n] = size(x);
if (~((m == 1) || (n == 1)) || (m == 1 && n == 1))
      error('Input must be a vector')
end
y = sum(x)/length(x);
end
```

# 第 3 章　MATLAB 图像处理基础

```
close all;                                  %关闭当前所有图形窗口
clear all;                                  %清空工作空间变量
clc;                                        %清屏
X=imread('football.jpg');                   %读取文件格式为.jpg,
                                            文件名为 football 的 RGB 图像的信息
I=rgb2gray(X);                              %将 RGB 图像转换为灰度图像
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);    %修改图形背景颜色的设置
subplot(121),imshow(X);                     %显示原 RGB 图像
subplot(122),imshow(I);                     %显示转换后灰度图像

close all;                                  %关闭当前所有图形窗口
clear all;                                  %清空工作空间变量
clc;                                        %清屏
[X,map] = imread('trees.tif');              %读取原图像信息
newmap = rgb2gray(map);                     %将彩色颜色映射表转换为灰度颜色映射表
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);       %修改图形背景颜色的设置
figure,imshow(X,map);                       %显示原图像
figure,imshow(X,newmap);                    %显示转换后灰度图像

close all;                                  %关闭当前所有图形窗口
clear all;                                  %清空工作空间变量
clc;                                        %清屏
RGB = imread('football.jpg');               %读取图像信息
[X1,map1]=rgb2ind(RGB,64);                  %将 RGB 图像转换成索引图像，颜色种数 N 是 64 种
[X2,map2]=rgb2ind(RGB,0.2);                 %将 RGB 图像转换成索引图像，颜色种数 N 是 216 种
map3= colorcube(128);                       %创建一个指定颜色数目的 RGB 颜色映射表
X3=rgb2ind(RGB,map3);
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);       %修改图形背景颜色的设置
figure;
subplot(131),imshow(X1,map1);               %显示用最小方差法转换后索引图像
subplot(132),imshow(X2,map2);               %显示用均匀量化法转换后索引图像
subplot(133),imshow(X3,map3);               %显示用颜色近似法转换后索引图像

close all                                   %关闭当前所有图形窗口
clear all;                                  %清空工作空间变量
clc                                         %清屏
I = imread('cameraman.tif')                 %读取灰度图像信息
```

```
[X,map]=gray2ind(I,8);                    %实现灰度图像向索引图像的转换,N 取 8
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure,imshow(I);                         %显示原灰度图像
figure, imshow(X, map);                   %显示 N=8 转换后索引图像

close all;                                %关闭当前所有图形窗口
clear all;                                %清空工作空间变量
clc;                                      %清屏
I = imread('coins.png');                  %读取图像信息
X = grayslice(I,32);                      %将灰度图像转换为索引图像
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure,imshow(I);                         %显示原图像
figure,imshow(X,jet(32));                 %jet(M)是相当于 colormap，是一个 M×3 的数组，

close all;                                %关闭当前所有图形窗口
clear all;                                %清空工作空间变量
clc;                                      %清屏
[X,map]=imread('forest.tif');%像信息
I = ind2gray(X,map);                      %再将索引图像转换为灰度图像
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure,imshow(X,map);                     %将索引图像显示
figure,imshow(I);                         %将灰度图像显示

close all;                                %关闭当前所有图形窗口
clear all                                 %清空工作空间变量
clc;                                      %清屏
[X,map]=imread('kids.tif');               %读取图像信息
RGB=ind2rgb(X,map);                       %将索引图像转换为真彩色图像
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure, imshow(X,map);                    %显示原图像
figure,imshow(RGB);                       %显示真彩色图像

close all;                                %关闭当前所有图形窗口
clear all;                                %清空工作空间变量
clc;                                      %清屏
I=imread('rice.png');                     %读取图像信息
BW1=im2bw(I,0.4);                         %将灰度图像转换为二值图像，level 值为 0.4

BW2=im2bw(I,0.6);                         %将灰度图像转换为二值图像，level 值为 0.6
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
```

```
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure;
subplot(131),imshow(I);                    %显示 level=0.4 转换后的二值图像
subplot(132),imshow(BW1);                  %显示 level=0.5 转换后的二值图像
subplot(133),imshow(BW2);                  %显示 level=0.6 转换后的二值图像

close all;                                 %关闭当前所有图形窗口
clear all;                                 %清空工作空间变量
clc;                                       %清屏
load trees;                                %从文件‘trees。mat’中载入数据到 workplace
BW = im2bw(X,map,0.4);                     %将缩=索引图像转换为二值图像
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure, imshow(X,map);                     %显示原索引图像
figure, imshow(BW);                        %显示转换后二值图像

close all;                                 %关闭当前所有图形窗口
clear all;                                 %清空工作空间变量
clc;                                       %清屏
I=imread('pears.png');                     %读取图像信息
BW=im2bw(I,0.5);                           %将 RGB 图像转换为二值图像
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
figure,
subplot(121),imshow(I);                    %显示原图像
subplot(122),imshow(BW);                   %显示转换后二值图像

close all;                                 %关闭当前所有图形窗口
clear all;                                 %清空工作空间变量
clc;                                       %清屏
X=magic(256);
I= mat2gray(X);                            %将矩阵 J 转换为灰度图像
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);      %修改图形背景颜色的设置
imshow(I);                                 %显示转换后灰度图像

close all;                                 %关闭当前所有图形窗口
clear all;                                 %清空工作空间变量
clc;                                       %清屏
I1=imread('football.jpg');                 %读取一幅 RGB 图像
I2=imread('cameraman','tif');              %读取一幅灰度图像
I3=imread('E:\onion.png');                 %读取非当前路径下的一幅 RGB 图像
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])       %修改图形背景颜色的设置
```

```
figure,
subplot(1,3,1),imshow(I1);                %显示该 RGB 图像
subplot(1,3,2),imshow(I2);                %显示该灰度图像
subplot(1,3,3),imshow(I3);                %显示该 RGB 图像

clear all;
close all;
clc;
[X,map]=imread('beach.gif',2);
[X1,map1]=imread('beach.gif',12);
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);       %修改图形背景颜色的设置
figure,
subplot(121),imshow(X,map);
subplot(122),imshow(X1,map1);
I1=imread('pillsetc.png','BackgroundColor',[1 0 0]);
I2=imread('rice.png','BackgroundColor',1);
I3=imread('forest.tif','BackgroundColor',64);

close all;                                %关闭当前所有图形窗口
clear all;                                %清空工作空间变量
clc;                                      %清屏
I=imread('lena.bmp');                     %读取图像信息
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])      %修改图形背景颜色的设置
subplot(121),imshow(I,128);               %以 128 灰度级显示该灰度图像
subplot(122),imshow(I,[60,120]);          %设置灰度上下为[60,120]显示该灰度图像

close all;                                %关闭当前所有图形窗口
clear all;                                %清空工作空间变量
clc;                                      %清屏
I=imread('lena.bmp');                     %读取图像信息
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])      %修改图形背景颜色的设置
figure,
subplot(221),imshow(I);
subplot(222),image(I);
subplot(223),image([50,200],[50,300],I);
subplot(224),imagesc(I,[60,150]);

close all;                                %关闭当前所有图形窗口
clear all;                                %清空工作空间变量
clc;                                      %清屏
I=imread('tire.tif');                     %读取图像信息
```

```
H=[1 2 1;0 0 0;-1 -2 -1];                    %设置 subol 算子
X=filter2(H,I);                              %对灰度图像 G 进行 2 次滤波，实现边缘检测
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1]);        %修改图形背景颜色的设置
figure,
subplot(131),imshow(I);
subplot(132),imshow(X,[]),colorbar();   %显示图像，并添加颜色条
subplot(133),imshow(X,[]),colorbar('east');

close all;                                   %关闭当前所有图形窗口
clear all;                                   %清空工作空间变量
clc;                                         %清屏
I=zeros(128,128,1,27);                       %建立四维数组 I
for i=1:27
[I(:,:,:,i),map]=imread('mri.tif',i);        %读取多帧图像序列，存放在数组 I 中
end
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])         %修改图形背景颜色的设置
montage(I,map);                              %将多帧图像同时显示

close all;                                   %关闭当前所有图形窗口
clear all;                                   %清空工作空间变量
clc;                                         %清屏
%I=imread('testpat.png');
I=imread('football.jpg');                    %读取图像信息
[x,y,z]=sphere;                              %创建三个（N+1）×（N+1）的矩阵，
                                               使得 surf(X,Y,Z)建立一个球体，缺省时 N 取 20
set(0,'defaultFigurePosition',[100,100,1000,400]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])         %修改图形背景颜色的设置
figure,
subplot(121),warp(I);                        %显示图像映射到矩形平面
subplot(122),warp(x,y,z,I);                  %将二维图像纹理映射三维球体表面
grid;                                        %建立网格

close all;                                   %关闭当前所有图形窗口
clear all;                                   %清空工作空间变量
clc;                                         %清屏
load trees;                                  %载入图像文件 trees.mat，
                                               将其中的变量载入 workspace 中
[X1,map1]=imread('forest.tif');              %读取图像信息
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])         %修改图形背景颜色的设置
figure,
subplot(1,2,1),subimage(X,map);              %将图像窗口分成 1×2 个子窗口，
```

```
                                        在左边子窗口中显示图像 X
subplot(1,2,2),subimage(X1,map1);      %在右边子窗口中显示图像 X1

close all;                             %关闭当前所有图形窗口
clear all;                             %清空工作空间变量
clc;                                   %清屏
RGB = imread('peppers.png');           %读取图像信息
c = [12 146 410];                      %新建一个向量 c，存放像素纵坐标
r = [104 156 129];                     %新建一个向量 r，存放像素横坐标
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])   %修改图形背景颜色的设置
pixels1=impixel(RGB)                   %交互式用鼠标选择像素
pixels2= impixel(RGB,c,r)              %将像素坐标作为输入参数，显示特定像素的颜色值

close all;                             %关闭当前所有图形窗口
clear all                              %清空工作空间变量
clc;                                   %清屏
set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
set(0,'defaultFigureColor', [1 1 1])   %修改图形背景颜色的设置
h = imshow('hestain.png');             %显示图像
hp = impixelinfo;                      %创建图像像素信息显示工具
set(hp,'Position',[150 290 300 20]);   %设置像素信息工具显示的位置
figure
imshow('trees.tif');
impixelinfo                            %创建图像像素信息显示工具

close all;                             %关闭当前所有图形窗口
clear all;                             %清空工作空间变量
clc;                                   %清屏
obj = mmreader('xylophone.mpg', 'tag', 'myreader1');
                                       %创建多媒体文件对象句柄，并设置标签
Frames = read(obj);                    %读取视频流，将每一帧图像存在数组 Frames 中
numFrames = get(obj, 'numberOfFrames');          %获取视频流中总帧数
for k = 1 : numFrames
mov(k).cdata = Frames(:,:,:,k);        %将每一图像帧中的数据矩阵
                                        读取出来存在 mov(k).cdata 中
mov(k).colormap = [];                  %将颜色表赋值为空
end
hf = figure;                           %创建一个图像窗口
set(hf, 'position', [150 150 obj.Width obj.Height]);     %根据视频帧的宽度和高度，重新设置图像窗口大小
movie(hf, mov, 1, obj.FrameRate);      %按照视频流原来的帧速率来播放该视频

close all;                             %关闭当前所有图形窗口
```

```
clear all;                          %清空工作空间变量
clc;                                %清屏
load mri
mov = immovie(D,map);
implay(mov)
```

# 第 4 章　数字图像的运算

```
close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
gamma=0.5;                                  %设定调整线性度取值
I=imread('peppers.png');                    %读入要处理的图像，并赋值给 I
R=I;                                        %将图像数据赋值给 R
R (:,:,2)=0;                                %将原图像变成单色图像，保留红色
R(:,:,3)=0;
R1=imadjust(R,[0.5 0.8],[0 1],gamma);   %利用函数 imadjust 调整 R 的灰度，结果返回 R1
G=I;                                        %将图像数据赋值给 G
G(:,:,1)=0;                                 %将原图像变成单色图像，保留绿色
G(:,:,3)=0;
G1=imadjust(G,[0 0.3],[0 1],gamma);     %利用函数 imadjust 调整 G 的灰度，结果返回 G1
B=I;                                        %将图像数据赋值给 B
B(:,:,1)=0;                                 %将原图像变成单色图像，保留蓝色
B(:,:,2)=0;
B1=imadjust(B,[0 0.3],[0 1],gamma);     %利用函数 imadjust 调整 B 的灰度，结果返回 B1
I1=R1+G1+B1;                                %求变换后的 RGB 图像
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure(1)
subplot(121),imshow(R);                     %绘制 R、R1、G、G1、B、B1 图像，
                                              观察线性灰度变换结果
subplot(122),imshow(R1);
figure(2);
subplot(121),imshow(G);
subplot(122),imshow(G1);
figure(3);
subplot(121),imshow(B);
subplot(122),imshow(B1);
figure(4);
subplot(121),imshow(I);
subplot(122),imshow(I1);

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
R=imread('peppers.png');                    %读入原图像，赋值给 R
J=rgb2gray(R);                              %将彩色图像数据 R 转换为灰度图像数据 J
[M,N]=size(J);                              %获得灰度图像数据 J 的行列数 M，N
x=1;y=1;                                    %定义行索引变量 x、列索引变量 y
for x=1:M
```

```
    for y=1:N
        if (J(x,y)<=35);              %对灰度图像 J 进行分段处理，处理后的结果返回给矩阵 H
            H(x,y)=J(x,y)*10;
        elseif(J(x,y)>35&J(x,y)<=75);
            H(x,y)=(10/7)*[J(x,y)-5]+50;
        else(J(x,y)>75);
            H(x,y)=(105/180)*[J(x,y)-75]+150;
        end
    end
end
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])    %修改图形背景颜色的设置
subplot(121),imshow(J)                 %显示处理前后的图像
subplot(122),imshow(H);

close all;clear all;clc;               %关闭当前所有图形窗口，清空工作空间变量，
                                         清除工作空间所有变量
R=imread('peppers.png');               %读入图像，赋值给 R
G=rgb2gray(R);                         %转成灰度图像
J=double(G);                           %数据类型转换成双精度
H=(log(J+1))/10;                       %进行基于常用对数的非线性灰度变换
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])    %修改图形背景颜色的设置
subplot(121),imshow(G);                %显示图像
subplot(122),imshow(H);

close all;clear all;clc;               %关闭当前所有图形窗口，清空工作空间变量，
                                         清除工作空间所有变量
I=imread('rice.png');                  %读入图像 rice，赋值给 I
J=imread('cameraman.tif');             %读入图像 cameraman，赋值给 J
K=imadd(I,J);                          %进行两幅图像的加法运算
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])    %修改图形背景颜色的设置
subplot(131),imshow(I);                %显示 rice，cameraman 及相加以后的图像
subplot(132),imshow(J);
subplot(133),imshow(K);

close all;clear all;clc;               %关闭当前所有图形窗口，清空工作空间变量，
                                         清除工作空间所有变量
I=imread('flower.tif');                %读入 flower 图像
J=imadd(I,30);                         %每个像素值增加 30
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])    %修改图形背景颜色的设置
subplot(121),imshow(I);                %显示原图像和加常数后的图像
```

```
subplot(122),imshow(J);

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
RGB=imread('eight.tif');                    %读入 eight 图像，赋值给 RGB
A=imnoise(RGB,'gaussian',0,0.05);           %加入高斯白噪声
I=A;                                        %将 A 赋值给 I
M=3;                                        %设置叠叠加次数 M
I=im2double(I);                             %将 I 数据类型转换成双精度
RGB=im2double(RGB);
for i=1:M
    I=imadd(I,RGB);                         %对用原图像与带噪声图像进行多次叠加，结果返回给 I
end
avg_A=I/(M+1);                              %求叠加的平均图像
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
subplot(121); imshow(A);                    %显示加入椒盐噪声后的图像
subplot(122); imshow(avg_A);                %显示加入乘性噪声后的图像

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
RGB=imread('eight.tif');                    %读入 eight 图像，赋值给 RGB
M1=3;
[BW1,runningt1]=Denoise(RGB,M1);    % M=3 叠加
M2=9;
[BW2,runningt2]=Denoise(RGB,M2);    % M=9 叠加
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
subplot(121); imshow(BW1);                  %显示结果
subplot(122); imshow(BW2);
disp('叠加 4 次运行时间')
runningt1
disp('叠加 10 次运行时间')
runningt2

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
A=imread('cameraman.tif');%
B=imread('testpat1.png');
C=imsubtract(A,B);                          %进行图像减法
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure,                                     %显示原图像及差异图像
subplot(121),imshow(C);
```

```
subplot(122),imshow(255-C);

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
A=imread('tire.tif');                       %读取图像 tire，并赋值给 A
[m,n]=size(A);                               %获取图像矩阵 A 的行列数 m，n
B=imread('eight.tif');                      %读取图像 eight 的值，并赋值给 B
C=B;                                        %初始化矩阵 C
A=im2double(A);                             %定义 A\B\C 的数据类型为双精度
B=im2double(B);
C=im2double(C);
for i=1:m                                    %将图像 B 和 A 叠加，结果赋值给 C
    for j=1:n
    C(i,j)=B(i,j)+A(i,j);
    end
end
D=imabsdiff(C,B);                            %求叠加后图像 C 和 B 的差异，赋值给 D
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure(1),
subplot(121),imshow(A);                      %显示 tire、eight 图像，叠加图像及差异图像
subplot(122),imshow(B);
figure(2),
subplot(121),imshow(C);
subplot(122),imshow(D);

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
A=imread('ipexroundness_04.png');           %读入原始图像赋值给 A 和 B
B=imread('ipexroundness_01.png');
C=immultiply(A,B);                           %计算 A 和 B 的乘法，计算结果返回给 C
A1=im2double(A);                            %将 A 和 B 转换成双精度类型，存为 A1 和 B1
B1=im2double(B);
C1=immultiply(A1,B1);                        %重新计算 A1 和 B1 的乘积，结果返回给 C1
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])          %修改图形背景颜色的设置
figure(1),                                   % 显示原图像 A 和 B
subplot(121),imshow(A),axis on;
subplot(122),imshow(B),axis on;
figure(2),                                   % 显示 uint8 和 double 图像数据格式下，乘积 C 和 C1
subplot(121),imshow(C),axis on;;
subplot(122),imshow(C1),axis on;;

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
```

```
                                        清除工作空间所有变量
A=imread('house.jpg');                  %读入图像，赋值给 A
B=immultiply(A,1.5);                    %分别乘以缩放因子 1.5 和 0.5，结果返回给 B 和 C
C=immultiply(A,0.5);
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])     %修改图形背景颜色的设置
figure(1),
subplot(1,2,1),imshow(B),axis on;       %显示乘以缩放因子以后的图像
subplot(1,2,2),imshow(C),axis on;

close all;                              %关闭当前所有图形窗口，清空工作空间变量，清除工
作空间所有变量
clear all;
clc;
I=imread('office_1.jpg');               %读入图像 office_1 和 office_2，并赋值
J=imread('office_2.jpg');
Ip=imdivide(J,I);                       %两幅图像相除
K=imdivide(J,0.5);                      %图像跟一个常数相除
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])     %修改图形背景颜色的设置
figure(1);                              %依次显示四幅图像
subplot(121); imshow(I);
subplot(122); imshow(J);
figure(2)
subplot(121); imshow(Ip);
subplot(122); imshow(K);

close all;clear all;clc;                %关闭当前所有图形窗口，清空工作空间变量，
                                        清除工作空间所有变量
I = imread('cameraman.tif');            %读取图像,赋值给 I
J = filter2(fspecial('prewitt'), I);    %对图像矩阵 I 进行滤波
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])     %修改图形背景颜色的设置
K = imabsdiff(double(I),J);             %求滤波后的图像与原图像的绝对值差
figure,                                 %显示图像及结果
subplot(131),imshow(I);
subplot(132),imshow(J,[]);
subplot(133),imshow(K,[]);
X =[ 255 10 75; 44 225 100];            %输入矩阵,数据格式 double
Y =[ 50 50 50; 50 50 50 ];
X1 =uint8([ 255 10 75; 44 225 100]);    %输入矩阵，数据格式 uint8
Y1 =uint8([ 50 50 50; 50 50 50 ]);
Z=imabsdiff(X,Y)                        %求绝对值的差
Z1=abs(X-Y)                             %利用函数 abs 计算绝对值差
```

```
Z2=abs(X1-Y1)
disp('Z 与 Z1 比较结果：'),Z_Z1=(Z==Z1)%比较不同数据类型下两种指令结果。
disp('Z 与 Z2 比较结果：'),Z_Z2=(Z==Z2)

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
J=imread('rice.png');                       % 读取灰度图像，赋值给 J
J1=im2bw(J);                                %将灰度图像转换成二值图像，赋值给 J1
J2=imcomplement(J);                         %求灰度图像的补，赋值给 J2
J3=imcomplement(J1);                        %求二值图像的补，赋值给 J3
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure,                                     %显示运算结果
subplot(131),imshow(J1)                     %显示灰度图像及其补图像
subplot(132),imshow(J2)                     %显示二值图像及其补图像
subplot(133),imshow(J3)

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
I=imread('cameraman.tif');                  %读取图像
J=imread('rice.png');
K1=imlincomb(1.0,I,1.0,J);                  %两个图像叠加
K2=imlincomb(1.0,I,-1.0,J,'double');        %两个图像相减
K3=imlincomb(2,I);                          %图像的乘法
K4=imlincomb(0.5,I);                        %图像的除法
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure(1),                                  %显示结果
subplot(121),imshow(K1);
subplot(122),imshow(K2);
figure,
subplot(121),imshow(K3);
subplot(122),imshow(K4);

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
I=imread('ipexroundness_01.png');           %读入图像，赋值给 I 和 J
J=imread('ipexroundness_04.png');
I1=im2bw(I);                                %转化为二值图像
J1=im2bw(J);
K1=I1 & J1;                                 %实现图像的逻辑“与”运算
K2=I1 | J1;                                 %实现图像的逻辑“或”运算
K3=~I1;                                     %实现逻辑“非”运算
K4=xor(I1,J1);                              %实现“异或”运算
```

```
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])        %修改图形背景颜色的设置
figure,                                    %显示原图像及相应的二值图像
subplot(121);imshow(I1),axis on;
subplot(122);imshow(J1),axis on;
figure,                                    %显示逻辑运算图像
subplot(121);imshow(K1),axis on;
subplot(122);imshow(K2),axis on;
figure,
subplot(121);imshow(K3),axis on;
subplot(122);imshow(K4),axis on;

close all;clear all;clc;                   %关闭当前所有图形窗口，清空工作空间变量，
                                             清除工作空间所有变量
I=imread('girl.bmp');                      %读入图像，赋值给 I 和 J
J=imread('lenna.bmp');
I1=im2bw(I);                               %转化为二值图像
J1=im2bw(J);
H=~(I1|J1);
G=~(I1&J1);
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])        %修改图形背景颜色的设置
figure(1),                                 %显示原图像及相应的二值图像
subplot(121),imshow(I1),axis on;
subplot(122),imshow(J1),axis on;
figure(2),                                 %显示运算以后的图像
subplot(121),imshow(H),axis on;
subplot(122),imshow(G),axis on;

close all;clear all;clc;                   %关闭当前所有图形窗口，清空工作空间变量，
                                             清除工作空间所有变量
I=imread('lenna.bmp');                     %输入图像
a=50;b=50;                                 %设置平移坐标
J1=move(I,a,b);                            %移动原图像
a=-50;b=50;                                %设置平移坐标
J2=move(I,a,b);                            %移动原图像
a=50;b=-50;                                %设置平移坐标
J3=move(I,a,b);                            %移动原图像
a=-50;b=-50;                               %设置平移坐标
J4=move(I,a,b);                            %移动原图像
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])        %修改图形背景颜色的设置
figure,
subplot(1,2,1),imshow(J1),axis on;         %绘制移动后图像
```

```
subplot(1,2,2),imshow(J2),axis on;       %绘制移动后图像
figure,
subplot(1,2,1),imshow(J3),axis on;       %绘制移动后图像
subplot(1,2,2),imshow(J4),axis on;       %绘制移动后图像

close all;clear all;clc;                 %关闭当前所有图形窗口，清空工作空间变量，
                                           清除工作空间所有变量
I=imread('lenna.bmp');                   %输入图像
a=50;b=50;                               %设置平移坐标
J1=move1(I,a,b);                         %移动原图像
a=-50;b=50;                              %设置平移坐标
J2=move1(I,a,b);                         %移动原图像
a=50;b=-50;                              %设置平移坐标
J3=move1(I,a,b);                         %移动原图像
a=-50;b=-50;                             %设置平移坐标
J4=move1(I,a,b);                         %移动原图像
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])      %修改图形背景颜色的设置
figure,
subplot(1,2,1),imshow(J1),axis on;       %绘制移动后图像
subplot(1,2,2),imshow(J2),axis on;       %绘制移动后图像
figure,
subplot(1,2,1),imshow(J3),axis on;       %绘制移动后图像
subplot(1,2,2),imshow(J4),axis on;       %绘制移动后图像

close all;clear all;clc;                 %关闭当前所有图形窗口，清空工作空间变量，
                                           清除工作空间所有变量
I=imread('cameraman.tif');               %输入图像
J1=mirror(I,1);                          %原图像的水平镜像
J2=mirror(I,2);                          %原图像的垂直镜像
J3=mirror(I,3);                          %原图像的水平垂直镜像
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])      %修改图形背景颜色的设置
figure,
subplot(1,2,1),imshow(I) ;               %绘制原图像
subplot(1,2,2),imshow(J1);               %绘制水平镜像后图像
figure,
subplot(1,2,1),imshow(J2);               %绘制水平镜像后图像
subplot(1,2,2),imshow(J3);               %绘制垂直镜像后图像

close all;clear all;clc;                 %关闭当前所有图形窗口，清空工作空间变量，
                                           清除工作空间所有变量
 [X,map]=imread('trees.tif');            %读入图像
J1=imresize(X, 0.25);                    %设置缩放比例，实现缩放图像并显示
```

```
J2=imresize(X, 3.5);
J3=imresize(X, [64 40]);                    %设置缩放后图像行列，实现缩放图像并显示
J4=imresize(X, [64 NaN]);
J5=imresize(X, 1.6, 'bilinear');            %设置图像插值方法，实现缩放图像并显示
J6=imresize(X, 1.6, 'triangle');
[J7, newmap]=imresize(X,'Antialiasing',true,'Method','nearest',...
                              'Colormap','original','Scale', 0.15);
                                            %设置图像多个参数，实现缩放图像并显示
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure(1)                                   %显示各种缩放效果图
subplot(121),imshow(J1);
subplot(122),imshow(J2);
figure(2)
subplot(121),imshow(J3);
subplot(122),imshow(J4);
figure(3)
subplot(121),imshow(J5);
subplot(122),imshow(J6);
figure(4),
subplot(121),imshow(X);
subplot(122),imshow(J7);

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
I=imread('trees.tif');                      %输入图像
J1=transp(I);                               %对原图像的转置
I1=imread('lenna.bmp');                     %输入图像
J2=transp(I1);                              %对原图像的转置
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure,
subplot(1,2,1),imshow(J1);                  %绘制移动后图像
subplot(1,2,2),imshow(J2);                  %绘制移动后图像

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
A=imread('office_2.jpg');                   %读入图像
J1=imrotate(A, 30);                         %设置旋转角度，实现旋转并显示
J2=imrotate(A, -30);
J3=imrotate(A,30,'bicubic','crop');         %设置输出图像大小，实现旋转图像并显示
J4=imrotate(A,30, 'bicubic','loose');
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
```

```
figure(1)                                %显示旋转处理结果
subplot(121),imshow(J1);
subplot(122),imshow(J2);
figure(2)
subplot(121),imshow(J3);
subplot(122),imshow(J4);

close all;clear all;clc;                 %关闭当前所有图形窗口，清空工作空间变量，
                                           清除工作空间所有变量
  [A,map]=imread('peppers.png');         %读入图像
rect=[75 68 130 112];                    %定义剪切区域
X1=imcrop(A,rect);                       %进行图像剪切
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])      %修改图形背景颜色的设置
subplot(121),imshow(A);                  %显示原图像
rectangle('Position',rect,'LineWidth',2,'EdgeColor','r') %显示图像剪切区域
subplot(122),imshow(X1);                 %显示剪切的图像

close all;clear all;clc;                 %关闭当前所有图形窗口，清空工作空间变量，
                                           清除工作空间所有变量
  [A,map]=imread('peppers.png');         %读入图像
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])      %修改图形背景颜色的设置
[I2,rect]=imcrop(A);                     %进行图像剪切
subplot(121),imshow(A);                  %显示原图像
rectangle('Position',rect,'LineWidth',2,'EdgeColor','r') %显示图像剪切区域
subplot(122),imshow(I2);                 %显示剪切的图像

close all;clear all;clc;                 %关闭当前所有图形窗口，清空工作空间变量，
                                           清除工作空间所有变量
  [I,map]=imread('peppers.png');         %读入图像
Ta = maketform('affine', ...
[cosd(30) -sind(30) 0; sind(30) cosd(30) 0; 0 0 1]');%  创建旋转参数结构体
Ia = imtransform(I,Ta);                  %实现图像旋转
Tb = maketform('affine',[5 0 0; 0 10.5 0; 0 0 1]'); %创建缩放参数结构体
Ib = imtransform(I,Tb);                  %实现图像缩放
xform = [1 0 55; 0 1 115; 0 0 1]';       %创建图像平移参数结构体
Tc = maketform('affine',xform);
Ic = imtransform(I,Tc, 'XData', ...      %进行图像平移
[1 (size(I,2)+xform(3,1))], 'YData', ...
[1 (size(I,1)+xform(3,2))],'FillValues', 255 );
Td = maketform('affine',[1 4 0; 2 1 0; 0 0 1]');%  创建图像整体切变的参数结构体
Id = imtransform(I,Td,'FillValues', 255);   %实现图像整体切变
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
```

```
set(0,'defaultFigureColor',[1 1 1])        %修改图形背景颜色的设置
figure                                     %显示结果
subplot(121),imshow(Ia),axis on;
subplot(122),imshow(Ib),axis on;
figure
subplot(121),imshow(Ic),axis on;
subplot(122),imshow(Id),axis on;

close all;clear all;clc;                   %关闭当前所有图形窗口，清空工作空间变量，
                                             清除工作空间所有变量
A = imread('cameraman.tif');               %读取图像
A1=im2double(A);                           %数值类型转换
B1 = nlfilter(A1,[4 4],'std2');            %对图像 A 利用滑动邻域操作函数进行处理
fun = @(x) max(x(:));                      %对图像 A 利用滑动邻域操作函数进行处理
B2 = nlfilter(A1,[3 3],fun);
B3 = nlfilter(A1,[6 6],fun);
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])        %修改图形背景颜色的设置
subplot(131),imshow(B1);                   %显示处理后图像
subplot(132),imshow(B2);
subplot(133),imshow(B3);

close all;clear all;clc;                   %关闭当前所有图形窗口，清空工作空间变量，
                                             清除工作空间所有变量
I=imread('tire.tif');                      %输入图像
I1=im2double(I);                           %数值类型转换
f=@(x) min(x);
I2=colfilt(I1,[4 4],'sliding',f);          %按照滑动邻域方式 进行对图像进行最小值邻域操作
m=2;n=2;
f=@(x) ones(m*n,1)*min(x);                 %
I3=colfilt(I1,[m n],'distinct',f);
m=4;n=4;
f=@(x) ones(m*n,1)*min(x);
I4=colfilt(I1,[4 4],'distinct',f);
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])        %修改图形背景颜色的设置
figure
subplot(131),imshow(I2);
subplot(132),imshow(I3);                   %显示原图像和处理后图像
subplot(133),imshow(I4);

close all;clear all;clc;                   %关闭当前所有图形窗口，清空工作空间变量，
                                             清除工作空间所有变量
I = imread('peppers.png');                 %输入图像
```

```
fun = @(block_struct) imrotate(block_struct.data,30);%获取分离块操作的函数句柄
I1 = blockproc(I,[64 64],fun);                          %进行分离块操作
fun = @(block_struct) std2(block_struct.data) ;         %获取获取分离块操作的函数句柄
I2 = blockproc(I,[32 32],fun);                          %进行分离块操作
fun = @(block_struct) block_struct.data(:,:,[3 1 2]);   %获取分离块操作的函数句柄
blockproc(I,[100 100],fun,'Destination','brg_peppers.tif');%进行分离块操作
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                     %修改图形背景颜色的设置
figure                                                  %显示处理后结果
subplot(131),imshow(I1);
subplot(132),imshow(I2,[]);
subplot(133),imshow('brg_peppers.tif');

close all;clear all;clc;                    %关闭当前所有图形窗口，清空工作空间变量，
                                              清除工作空间所有变量
I=imread('pout.tif');                       %输入原图像
BW1=roicolor(I,55,100);                     %基于灰度图像 ROI 区域选取
c=[87 171 201 165 79 32 87];
r=[133 133 205 259 259 209 133];            %定义 ROI 顶点位置
BW=roipoly(I,c,r);                          %根据 c 和 r 选择 ROI 区域
I1=roifill(I,BW);                           %根据生成 BW 掩膜图像进行区域填充
h=fspecial('motion',20,45);                 %创建 motion 滤波器并说明参数
I2=roifilt2(h,I,BW);                        %进行区域滤波
set(0,'defaultFigurePosition',[100,100,1000,500]);%修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])         %修改图形背景颜色的设置
figure
subplot(121),imshow(BW1);                   %显示处理结果
subplot(122),imshow(BW);                    %显示 ROI 区域
figure
subplot(121),imshow(I1);                    %显示填充效果
subplot(122),imshow(I2);                    %显示区域滤波效果

function [BW,runningt]=Denoise(RGB,M)
                                            %RGB 原图像，M 表示叠加噪声的次数；
                                              BW 为消除噪声的图像，runningt 为函数运行时间
A=imnoise(RGB,'gaussian',0,0.05);           %加入高斯白噪声
I=A;                                        %将 A 赋值给 I
I=im2double(I);                             %将 I 数据类型转换成双精度
RGB=im2double(RGB);
tstart=tic;                                 %开始计时
for i=1:M
    I=imadd(I,RGB);                         %对用原图像与带噪声图像进行多次叠加，
                                              结果返回给 I
end
```

```
avg_A=I/(M+1);                          %求叠加的平均图像
runningt=toc(tstart);                   %计时结束
BW=avg_A;

function OutImage=mirror(InImage,n)
                                        %mirror 函数实现图像镜像变换功能
                                        %参数 n 为 1 时，实现水平镜像变换
                                        %参数 n 为 2 时，实现垂直镜像变换
                                        %参数 n 为 3 时，实现水平垂直镜像变换
I=InImage;
[M,N,G]=size(I);                        %获取输入图像 I 的大小
J=I;                                    %初始化新图像矩阵全为 1，大小与输入图像相
if (n==1)
    for i=1:M
        for j=1:N
            J(i,j,:)=I(M-i+1,j,:);      %n=1,水平镜像
        end
    end;
elseif (n==2)
    for i=1:M
        for j=1:N
            J(i,j,:)=I(i,N-j+1,:);      %n=2,垂直镜像
        end
    end
elseif (n==3)
    for i=1:M
        for j=1:N
            J(i,j,:)=I(M-i+1,N-j+1,:);  %n=3,水平垂直镜像
        end
    end
else
    error('参数 n 输入不正确，n 取值 1、2、3')      %n 输入错误时提示
end
    OutImage=J;

function J=move(I,a,b)
                                        %定义一个函数名字 move，I 表示输入图像，a 和 b
                                          描述 I 图像沿着 x 轴和 y 轴移动的距离
                                        %不考虑平移以后，图像溢出情况，找不到对应点的
                                          地方都赋值为 1
[M,N,G]=size(I);                        %获取输入图像 I 的大小
I=im2double(I);                         %将图像数据类型转换成双精度
J=ones(M,N,G);                          %初始化新图像矩阵全为 1，大小与输入图像相同
for i=1:M
```

```
    for j=1:N
        if((i+a)>=1&&(i+a<=M)&&(j+b>=1)&&(j+b<=N));
                                        %判断平移以后行列坐标是否超出范围
            J(i+a,j+b,:)=I(i,j,:);      %进行图像平移
        end
    end
end

function J=move1(I,a,b)
                                        %定义一个函数名字 move，I 表示输入图像，a 和 b
                                          描述 I 图像沿着 x 轴和 y 轴移动的距离
                                        %考虑平移以后，图像溢出情况，采用扩大显示区域
                                          的方法
[M,N,G]=size(I);                        %获取输入图像 I 的大小
I=im2double(I);                         %将图像数据类型转换成双精度
J=ones(M+abs(a),N+abs(b),G);            %初始化新图像矩阵全为 1，
                                          大小根据考虑 x 轴和 y 轴的平移范围
for i=1:M
 for j=1:N
  if(a<0 && b<0);                       %如果进行右下移动，对新图像矩阵进行赋值
      J(i,j,:)=I(i,j,:);
  else if(a>0 && b>0);
      J(i+a,j+b,:)=I(i,j,:);            %如果进行右上移动，对新图像矩阵进行赋值
  else if(a>0 && b<0);
      J(i+a,j,:)=I(i,j,:);              %如果进行左上移动，对新图像矩阵进行赋值
  else
      J(i,j+b,:)=I(i,j,:);              %如果进行右下移动，对新图像矩阵进行赋值
      end
    end
   end
  end
end

function J=transp(I)
                                        %I 表示输入的原始图像
                                        %J 表示经过转置以后的图像
[M,N,G]=size(I);                        %获取输入图像 I 的大小
I=im2double(I);                         %将图像数据类型转换成双精度
J=ones(N,M,G);                          %初始化新图像矩阵全为 1，大小与输入图像相同
for i=1:M
    for j=1:N
       J(j,i,:)=I(i,j,:);               %进行图像转置
    end
end
```

```
function J=transp(I)
                                    %I 表示输入的原始图像
                                    %J 表示经过转置以后的图像
[M,N,G]=size(I);                    %获取输入图像 I 的大小
I=im2double(I);                     %将图像数据类型转换成双精度
J=ones(N,M,G);                      %初始化新图像矩阵全为 1，大小与输入图像相同
for i=1:M
      for j=1:N
         J(j,i,:)=I(i,j,:);          %进行图像转置
      end
en
```

# 第 5 章　图像增强技术

```
clear all; close all;
I=imread('pout.tif');
row=size(I,1);
column=size(I,2);
N=zeros(1, 256);
for i=1:row
    for j=1:column
        k=I(i, j);
        N(k+1)=N(k+1)+1;
    end
end
figure;
subplot(121);
imshow(I);
subplot(122);
bar(N);
axis tight;
```

```
clear all; close all;
I=imread('pout.tif');
I=double(I);
J=(I-80)*255/70;
row=size(I,1);
column=size(I,2);
for i=1:row
    for j=1:column
        if J(i, j)<0
            J(i, j)=0;
        end
        if J(i, j)>255;
            J(i, j)=255;
        end
    end
end
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));
```

```
clear all; close all;
I=imread('pout.tif');
J=imadjust(I, [0.2 0.5], [0 1]);
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));
```

```
clear all; close all;
I=imread('pout.tif');
J=imadjust(I, [0.1 0.5], [0, 1], 0.4);
K=imadjust(I, [0.1, 0.5], [0, 1], 4);
figure;
subplot(121);
imshow(uint8(J));
subplot(122);
imshow(uint8(K));
```

```
clear all; close all;
I=imread('football.jpg');
J=imadjust(I, [0.2 0.3 0; 0.6 0.7 1], []);
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));
```

```
clear all; close all;
I=imread('cameraman.tif');
figure;
imshow(I);
brighten(0.6);
figure;
imshow(I);
brighten(-0.6);
```

```
clear all; close all;
I=imread('pout.tif');
M=stretchlim(I);
```

```
J=imadjust(I, M, [ ]);
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));

clear all; close all;
I=imread('glass.png');
J=imcomplement(I);
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));

clear all; close all;
I=imread('pout.tif');
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imhist(I);

clear all; close all;
I=imread('onion.png');
figure;
subplot(141);
imshow(uint8(I));
subplot(142);
imhist(I(:,:,1));
title('R');
subplot(143);
imhist(I(:,:,2));
title('G');
subplot(144);
imhist(I(:,:,3));
title('B');

clear all; close all;
I=imread('football.jpg');
J=rgb2hsv(I);
h=figure;
set(h, 'position', [200, 200, 1000, 400]);
subplot(141);
imshow(uint8(I));
subplot(142);
imhist(J(:,:,1));
title('H');
subplot(143);
imhist(J(:,:,2));
title('S');
subplot(144);
imhist(J(:,:,3));
title('V');

clear all; close all;
I=imread('tire.tif');
J=histeq(I);
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));
figure;
subplot(121);
imhist(I, 64);
subplot(122);
imhist(J, 64);

clear all; close all;
I=imread('tire.tif');
hgram=ones(1, 256);
J=histeq(I, hgram);
figure;
subplot(121);
imshow(uint8(J));
subplot(122);
imhist(J);

clear all; close all;
I=imread('onion.png');
J=rgb2gray(I);
gray=mean2(J)
rgb=mean2(I)
r=mean2(I(:, :, 1))
g=mean2(I(:, :, 2))
b=mean2(I(:, :, 3))
```

```
figure;
subplot(121);
imshow(uint8(I));
subplot(122);
imshow(uint8(J));

clear all; close all;
I=imread('pout.tif');
s1=std2(I)
J=histeq(I);
s2=std2(J)

clear all; close all;
I=imread('pout.tif');
J=medfilt2(I);
r=corr2(I, J)
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(J);

clear all; close all;
I=imread('peppers.png');
J=rgb2gray(I);
figure;
subplot(121);
imshow(J);
subplot(122);
imcontour(J,3);

clear all; close all;
I=imread('coins.png');
J=imnoise(I, 'salt & pepper', 0.02);
h=ones(3,3)/5;
h(1,1)=0;      h(1,3)=0;
h(3,1)=0;      h(1,3)=0;
K=imfilter(J, h);
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(J);
subplot(133);
imshow(K);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
J=imnoise(I, 'gaussian', 0, 0.01);
h=ones(3,3)/9;
K=conv2(J, h);
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(J);
subplot(133);
imshow(K);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
J=imnoise(I, 'salt & pepper', 0.02);
h1=fspecial('average', 3);
h2=fspecial('average', 5);
K1=filter2(h1, J);
K2=filter2(h2, J);
figure;
subplot(131);
imshow(J);
subplot(132);
imshow(K1);
subplot(133);
imshow(K2);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
J=imnoise(I, 'salt & pepper', 0.03);
K=medfilt2(J);
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(J);
subplot(133);
imshow(K);
```

```
clear all; close all;
I=imread('coins.png');
I=im2double(I);
J1=ordfilt2(I, 1, true(5));
J2=ordfilt2(I, 25, true(5));
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(J1);
subplot(133);
imshow(J2);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
J=imnoise(I, 'gaussian', 0, 0.01);
K=wiener2(J, [5 5]);
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(J);
subplot(133);
imshow(K);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
h=[0,1,0; 1, -4, 1; 0, 1, 0];
J=conv2(I, h, 'same');
K=I-J;
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(K);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
M=2*size(I,1);
N=2*size(I,2);
u=-M/2:(M/2-1);
v=-N/2:(N/2-1);
[U,V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=80;
H=double(D<=D0);
J=fftshift(fft2(I, size(H, 1), size(H, 2)));
K=J.*H;
L=ifft2(ifftshift(K));
L=L(1:size(I,1), 1:size(I, 2));
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(L);

clear all; close all;
I=imread('liftingbody.png');
I=im2double(I);
M=2*size(I,1);
N=2*size(I,2);
u=-M/2:(M/2-1);
v=-N/2:(N/2-1);
[U,V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=50;
n=6;
H=1./(1+(D./D0).^(2*n));
J=fftshift(fft2(I, size(H, 1), size(H, 2)));
K=J.*H;
L=ifft2(ifftshift(K));
L=L(1:size(I,1), 1:size(I, 2));
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(L);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
M=2*size(I,1);
N=2*size(I,2);
u=-M/2:(M/2-1);
```

```
v=-N/2:(N/2-1);
[U,V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=30;
n=6;
H=1./(1+(D0./D).^(2*n));
J=fftshift(fft2(I, size(H, 1), size(H, 2)));
K=J.*H;
L=ifft2(ifftshift(K));
L=L(1:size(I,1), 1:size(I, 2));
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(L);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
M=2*size(I,1);
N=2*size(I,2);
u=-M/2:(M/2-1);
v=-N/2:(N/2-1);
[U,V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=20;
H=1-exp(-(D.^2)./(2*(D0^2)));
J=fftshift(fft2(I, size(H, 1), size(H, 2)));
K=J.*H;
L=ifft2(ifftshift(K));
L=L(1:size(I,1), 1:size(I, 2));
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(L);

clear all; close all;
I=imread('coins.png');
I=imnoise(I, 'gaussian', 0, 0.01);
I=im2double(I);
M=2*size(I,1);
N=2*size(I,2);
u=-M/2:(M/2-1);
v=-N/2:(N/2-1);
[U,V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=50;
W=30;
H=double(or(D<(D0-W/2), D>D0+W/2));
J=fftshift(fft2(I, size(H, 1), size(H, 2)));
K=J.*H;
L=ifft2(ifftshift(K));
L=L(1:size(I,1), 1:size(I, 2));
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(L);

clear all; close all;
I=imread('pout.tif');
J=log(im2double(I)+1);
K=fft2(J);
n=5;
D0=0.1*pi;
rh=0.7;
rl=0.4;
[row, column]=size(J);
for i=1:row
    for j=1:column
        D1(i,j)=sqrt(i^2+j^2);
        H(i,j)=rl+(rh/(1+(D0/D1(i,j))^(2*n)));
    end
end
L=K.*H;
M=ifft2(L);
N=exp(M)-1;
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(real(N));

clear all; close all;
[x, y]=meshgrid(-128:2:127, -128:2:127);
n1=0;
n2=0;
```

```
z=sqrt((x-n1).^2+(y-n2).^2);
D1=40;    D2=40;
n=6;
H1=1./(1+(z/D1).^(2*n));
H2=1-H1;
figure;
mesh(H2);
axis tight;

clear all; close all;
[x, y]=meshgrid(-128:2:127, -128:2:127);
n1=0;
n2=0;
z=sqrt((x-n1).^2+(y-n2).^2);
D1=10;    D2=20;
n=6;
H1=z<30;
H2=1-H1;
figure;
mesh(double(H1));
axis tight;
figure;
mesh(double(H1));
axis tight;

clear all; close all;
[x, y]=meshgrid(-128:2:127, -128:2:127);
z=sqrt(x.^2+y.^2);
D0=30;
H1=(exp(1)).^(-(z.*z)/(2*D0*D0));
H2=1-H1;
figure;
mesh(x, y, H2);
axis tight;
```

# 第 6 章　图像复原技术

```
clear all; close all;
I=uint8(100*ones(256, 256));
J=imnoise(I, 'gaussian', 0, 0.01);
K=imnoise(I, 'gaussian', 0, 0.03);
figure;
subplot(121);    imshow(J);
subplot(122);    imhist(J);
figure;
subplot(121);    imshow(K);
subplot(122);    imhist(K);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
V=zeros(size(I));
for i=1:size(V, 1)
      V(i,:)=0.02*i/size(V,1);
end
J=imnoise(I, 'localvar', V);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
h=0:0.1:1;
v=0.01:-0.001:0;
J=imnoise(I, 'localvar', h, v);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
J=imnoise(I, 'salt & pepper', 0.01);
K=imnoise(I, 'salt & pepper', 0.03);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
R=rand(size(I));
J=I;
J(R<=0.02)=0;
K=I;
K(R<=0.03)=1;
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
J=imnoise(I, 'poisson');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
J=imnoise(I, 'speckle');
K=imnoise(I, 'speckle', 0.2);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
m=256; n=256;
a=50;
b=180;
I=a+(b-a)*rand(m,n);
figure;
subplot(121);    imshow(uint8(I));
subplot(122);    imhist(uint8(I));

clear all; close all;
m=256; n=256;
```

```
a=0.04;
k=-1/a;
I=k*log(1-rand(m, n));
figure;
subplot(121);    imshow(uint8(I));
subplot(122);    imhist(uint8(I));

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
I=imnoise(I, 'gaussian', 0.05);
PSF=fspecial('average', 3);
J=imfilter(I, PSF);
K=exp(imfilter(log(I), PSF));
figure;
subplot(131);    imshow(I);
subplot(132);    imshow(J);
subplot(133);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
I=imnoise(I, 'salt & pepper', 0.01);
PSF=fspecial('average', 3);
Q1=1.6;
Q2=-1.6;
j1=imfilter(I.^(Q1+1), PSF);
j2=imfilter(I.^Q1, PSF);
J=j1./j2;
k1=imfilter(I.^(Q2+1), PSF);
k2=imfilter(I.^Q2, PSF);
K=k1./k2;
figure;
subplot(131);    imshow(I);
subplot(132);    imshow(J);
subplot(133);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
I=imnoise(I, 'salt & pepper', 0.05);
J=medfilt2(I, [3, 3]);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
I=imnoise(I, 'salt & pepper', 0.1);
domain=[0 1 1 0; 1 1 1 1; 1 1 1 1; 0 1 1 0];
J=ordfilt2(I, 6, domain);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
I=imnoise(I, 'salt & pepper', 0.01);
J=ordfilt2(I, 1, ones(4,4));
K=ordfilt2(I, 9, ones(3));
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
RGB=imread('saturn.png');
I=rgb2gray(RGB);
I=imcrop(I, [100, 100, 1024, 1024]);
J=imnoise(I, 'gaussian', 0, 0.03);
[K, noise]=wiener2(J, [5, 5]);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
[m, n]=size(I);
M=2*m; n=2*n;
u=-m/2:m/2-1;
v=-n/2:n/2-1;
[U, V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=130;
H=exp(-(D.^2)./(2*(D0^2)));
N=0.01*ones(size(I,1), size(I,2));
```

```
N=imnoise(N, 'gaussian', 0, 0.001);
J=fftfilter(I, H)+N;
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J, [ ]);
HC=zeros(m, n);
M1=H>0.1;
HC(M1)=1./H(M1);
K=fftfilter(J, HC);
HC=zeros(m, n);
M2=H>0.01;
HC(M2)=1./H(M2);
L=fftfilter(J, HC);
figure;
subplot(121);    imshow(K, [ ]);
subplot(122);    imshow(L, [ ]);

clear all; close all;
I=imread('onion.png');
I=rgb2gray(I);
I=im2double(I);
LEN=25;
THETA=20;
PSF=fspecial('motion', LEN, THETA);
J=imfilter(I, PSF, 'conv', 'circular');
NSR=0;
K=deconvwnr(J, PSF, NSR);
figure;
subplot(131);    imshow(I);
subplot(132);    imshow(J);
subplot(133);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
LEN=21;
THETA=11;
PSF=fspecial('motion', LEN, THETA);
J=imfilter(I, PSF, 'conv', 'circular');
noise_mean=0;
noise_var=0.0001;
K=imnoise(J,    'gaussian',    noise_mean,
noise_var);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(K);
NSR1=0;
L1=deconvwnr(K, PSF, NSR1);
NSR2=noise_var/var(I(:));
L2=deconvwnr(K, PSF, NSR2);
figure;
subplot(121);    imshow(L1);
subplot(122);    imshow(L2);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
LEN=20;
THETA=10;
PSF=fspecial('motion', LEN, THETA);
J=imfilter(I, PSF, 'conv', 'circular');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);
noise=0.03*randn(size(I));
K=imadd(J, noise);
NP=abs(fft2(noise)).^2;
NPower=sum(NP(:))/prod(size(noise));
NCORR=fftshift(real(ifft2(NP)));
IP=abs(fft2(I)).^2;
IPower=sum(IP(:))/prod(size(I));
ICORR=fftshift(real(ifft2(IP)));
L=deconvwnr(K, PSF, NCORR, ICORR);
figure;
subplot(121);    imshow(K);
subplot(122);    imshow(L);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
PSF=fspecial('gaussian', 8, 4);
J=imfilter(I, PSF, 'conv');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);
v=0.02;
K=imnoise(J, 'gaussian', 0, v);
NP=v*prod(size(I));
```

```
L=deconvreg(K, PSF, NP);
figure;
subplot(121);    imshow(K);
subplot(122);    imshow(L);


clear all; close all;
I=imread('rice.png');
I=im2double(I);
PSF=fspecial('gaussian', 10, 5);
J=imfilter(I, PSF, 'conv');
v=0.02;
K=imnoise(J, 'gaussian', 0, v);
NP=v*prod(size(I));
[L, LAGRA]=deconvreg(K, PSF, NP);
edged=edgetaper(K, PSF);
figure;
subplot(131);    imshow(I);
subplot(132);    imshow(K);
subplot(133);    imshow(edged);
M1=deconvreg(edged, PSF, [], LAGRA);
M2=deconvreg(edged, PSF, [], LAGRA*30);
M3=deconvreg(edged, PSF, [], LAGRA/30);
figure;
subplot(131);    imshow(M1);
subplot(132);    imshow(M2);
subplot(133);    imshow(M3);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
LEN=30;
THETA=20;
PSF=fspecial('motion', LEN, THETA);
J=imfilter(I, PSF, 'circular', 'conv');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);
K=deconvlucy(J, PSF, 5);
L=deconvlucy(J, PSF, 15);
figure;
subplot(121);    imshow(K);
subplot(122);    imshow(L);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
PSF=fspecial('gaussian', 7, 10);
v=0.0001;
J=imnoise(imfilter(I, PSF), 'gaussian',0, v);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);
WT=zeros(size(I));
WT(5:end-4, 5:end-4)=1;
K=deconvlucy(J, PSF, 20, sqrt(v));
L=deconvlucy(J, PSF, 20, sqrt(v), WT);
figure;
subplot(121);    imshow(K);
subplot(122);    imshow(L);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
LEN=20;
THETA=20;
PSF=fspecial('motion', LEN, THETA);
J=imfilter(I, PSF, 'circular', 'conv');
INITPSF=ones(size(PSF));
[K, PSF2]=deconvblind(J, INITPSF, 30);
figure;
subplot(121);    imshow(PSF, []);
subplot(122);    imshow(PSF2, []);
axis auto;
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=checkerboard(8);
PSF=fspecial('gaussian', 7, 10);
v=0.001;
J=imnoise(imfilter(I, PSF), 'gaussian', 0, v);
INITPSF=ones(size(PSF));
WT=zeros(size(I));
WT(5:end-4, 5:end-4)=1;
[K,    PSF2]=deconvblind(J,    INITPSF,    20,
10*sqrt(v), WT);
```

```
figure;
subplot(131);    imshow(I);
subplot(132);    imshow(J);
subplot(133);    imshow(K);

function Z=fftfilter(X, H)
F=fft2(X, size(H,1), size(H, 2));
Z=H.*F;
Z=ifftshift(Z);
Z=abs(ifft2(Z));
Z=Z(1:size(X, 1), 1:size(X, 2));
End
```

# 第 7 章　图像分割技术

```
clear all; close all;
I=imread('gantrycrane.png');
I=rgb2gray(I);
h1=[-1, -1. -1; 2, 2, 2; -1, -1, -1];
h2=[-1, -1, 2; -1, 2, -1; 2, -1, -1];
h3=[-1, 2, -1; -1, 2, -1; -1, 2, -1];
h4=[2, -1, -1; -1, 2, -1; -1, -1, 2];
J1=imfilter(I, h1);
J2=imfilter(I, h2);
J3=imfilter(I, h3);
J4=imfilter(I, h4);
J=J1+J2+J3+J4;
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
[J, thresh]=edge(I, 'roberts', 35/255);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
[J, thresh]=edge(I, 'prewitt', [], 'both');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('gantrycrane.png');
I=rgb2gray(I);
I=im2double(I);
[J, thresh]=edge(I, 'sobel', [], 'horizontal');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all;    close all;    clc;
format rat;
hsobel=fspecial('sobel')
hprewitt=fspecial('prewitt')
hlaplacian=fspecial('laplacian')
hlog=fspecial('log', 3)
format short;

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
h=fspecial('laplacian');
J=imfilter(I, h, 'replicate');
K=im2bw(J, 80/255);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('rice.png');
I=im2double(I);
J=imnoise(I, 'gaussian', 0, 0.01);
[K, thresh]=edge(J, 'canny');
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
J=imnoise(I, 'gaussian', 0, 0.005);
[K, thresh]=edge(J, 'log', [], 2.3);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('rice.png');
figure;
```

```
subplot(121);    imshow(I);
subplot(122);    imhist(I, 200);

clear all; close all;
I=imread('rice.png');
J=I>120;
[width, height]=size(I);
for i=1:width
      for j=1:height
            if (I(i, j)>130)
                  K(i, j)=1;
            else
                  K(i, j)=0;
            end
      end
end
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
[X, map]=imread('trees.tif');
J=ind2gray(X, map);
K=im2bw(X, map, 0.4);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('coins.png');
I=im2double(I);
T=graythresh(I);
J=im2bw(I, T);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
T0=0.01;
T1=(min(I(:))+max(I(:)))/2;
r1=find(I>T1);
r2=find(I<=T1);
T2=(mean(I(r1))+mean(I(r2)))/2;
while abs(T2-T1)<T0
      T1=T2;
      r1=find(I>T1);
      r2=find(I<=T1);
      T2=(mean(I(r1))+mean(I(r2)))/2;
end
J=im2bw(I, T2);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('circbw.tif');
J=watershed(I, 8);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

function th=thresh(X)
count=imhist(X);
[M, N]=size(X);
th=M/5;
end
```

# 第 8 章　图像变换技术

```
clear all; close all;
I=zeros(200, 200);
I(50:150, 50:150)=1;
[R, xp]=radon(I, [0, 45]);
figure;
subplot(131);
imshow(I);
subplot(132);
plot(xp, R(:, 1));
subplot(133);
plot(xp, R(:, 2));
```

```
clear all; close all;
I=zeros(200, 200);
I(50:150, 50:150)=1;
theta=0:10:180;
[R, xp]=radon(I, theta);
figure;
subplot(121);
imshow(I);
subplot(122);
imagesc(theta, xp, R);
colormap(hot);
colorbar;
```

```
clear all; close all;
I=fitsread('solarspectra.fts');
J=mat2gray(I);
BW=edge(J);
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(BW);
theta=0:179;
[R, xp]=radon(BW, theta);
figure;
imagesc(theta, xp, R);
colormap(hot);
colorbar;
Rmax=max(max(R))
[row, column]=find(R>=Rmax)
x=xp(row)
angel=theta(column)
```

```
clear all; close all;
I=imread('circuit.tif');
theta=0:2:179;
[R, xp]=radon(I, theta);
J=iradon(R, theta);
figure;
subplot(131);
imshow(uint8(I));
subplot(132);
imagesc(theta, xp, R);
axis normal;
subplot(133);
imshow(uint8(J));
```

```
clear all; close all;
I=phantom(256);
figure;
imshow(I);
theta1=0:10:170;
theta2=0:5:175;
theta3=0:2:178;
[R1, xp]=radon(I, theta1);
[R2, xp]=radon(I, theta2);
[R3, xp]=radon(I, theta3);
figure;
imagesc(theta3, xp, R3);
colormap hot;
colorbar;
J1=iradon(R1, 10);
J2=iradon(R2, 5);
J3=iradon(R3, 2);
figure;
subplot(131);
imshow(J1);
subplot(132);
```

```
imshow(J2);
subplot(133);
imshow(J3);

clear all; close all;
I1=ones(4)
I2=[2 2 2 2;1 1 1 1; 3 3 0 0; 0 0 0 0]
J1=fft2(I1)
J2=fft2(I2)

clear all; close all;
I=imread('cameraman.tif');
J=fft2(I);
K=abs(J/256);
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(uint8(K));

clear all; close all;
N=0:4
X=fftshift(N)
Y=fftshift(fftshift(N))
Z=ifftshift(fftshift(N))

clear all; close all;
I=imread('peppers.png');
J=rgb2gray(I);
K=fft2(J);
K=fftshift(K);
L=abs(K/256);
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(uint8(L));

clear all; close all;
I=imread('peppers.png');
J=rgb2gray(I);
J=J*exp(1);
J(find(J>255))=255;
K=fft2(J);
K=fftshift(K);
L=abs(K/256);
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(uint8(L));

clear all; close all;
I=imread('peppers.png');
J=rgb2gray(I);
J=imrotate(J,45,'bilinear');
K=fft2(J);
K=fftshift(K);
L=abs(K/256);
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(uint8(L));

clear all; close all;
I=imread('peppers.png');
J=rgb2gray(I);
J=imnoise(J, 'gaussian', 0, 0.01);
K=fft2(J);
K=fftshift(K);
L=abs(K/256);
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(uint8(L));

clear all; close all;
I=imread('onion.png');
J=rgb2gray(I);
K=fft2(J);
L=fftshift(K);
M=ifft2(K);
figure;
subplot(121);
imshow(uint8(abs(L)/198));
subplot(122);
```

```
imshow(uint8(M));

clear all; close all;
I=imread('peppers.png');
J=rgb2gray(I);
K=fft2(J);
L=fftshift(K);
fftr=real(L);
ffti=imag(L);
A=sqrt(fftr.^2+ffti.^2);
A=(A-min(min(A)))/(max(max(A))-min(min(A)))*255;
B=angle(K);
figure;
subplot(121);
imshow(A);
subplot(122);
imshow(real(B));

clear all; close all;
I=imread('onion.png');
J=rgb2gray(I);
J=double(J);
s=size(J);
M=s(1); N=s(2);
for u=0:M-1
    for v=0:N-1
        k=0;
        for x=0:M-1
            for y=0:N-1
                k=J(x+1, y+1)*exp(-j*2*pi*(u*x/M+v*y/N))+k;
            end
        end
        F(u+1, v+1)=k;
    end
end
K=fft2(J);
figure;
subplot(121);
imshow(K);
subplot(122);
imshow(F);

clear all; close all;
I=imread('text.png');
```

```
a=I(32:45, 88:98);
figure;
imshow(I);
figure;
imshow(a);
c=real(ifft2(fft2(I).*fft2(rot90(a, 2), 256, 256)));
figure;
imshow(c, []);
max(c(:))
thresh=60;
figure;
imshow(c>thresh)
```

```
clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
J=fftshift(fft2(I));
[x, y]=meshgrid(-128:127, -128:127);
z=sqrt(x.^2+y.^2);
D1=10;    D2=30;
n=6;
H1=1./(1+(z/D1).^(2*n));
H2=1./(1+(z/D2).^(2*n));
K1=J.*H1;
K2=J.*H2;
L1=ifft2(ifftshift(K1));
L2=ifft2(ifftshift(K2));
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(real(L1));
subplot(133);
imshow(real(L2))
```

```
clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
J=fftshift(fft2(I));
[x, y]=meshgrid(-128:127, -128:127);
z=sqrt(x.^2+y.^2);
D1=10;    D2=40;
n1=4;    n2=8;
H1=1./(1+(D1./z).^(2*n1));
H2=1./(1+(D2./z).^(2*n2));
K1=J.*H1;
K2=J.*H2;
L1=ifft2(ifftshift(K1));
L2=ifft2(ifftshift(K2));
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(real(L1));
subplot(133);
imshow(real(L2))
```

```
clear all; close all;
I=imread('coins.png');
I=im2double(I);
J=dct2(I);
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(log(abs(J)), []);
```

```
clear all; close all;
A=[1 1 1 1; 2 2 2 2; 3 3 3 3]
s=size(A);
M=s(1);
N=s(2);
P=dctmtx(M)
Q=dctmtx(N)
B=P*A*Q'
```

```
clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
s=size(I);
M=s(1);
N=s(2);
P=dctmtx(M);
Q=dctmtx(N);
J=P*I*Q';
K=dct2(I);
E=J-K;
find(abs(E)>0.000001)
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(K);

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
J=dct2(I);
J(abs(J)<0.1)=0;
K=idct2(J);
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(J);
subplot(133);
imshow(K);

clear all; close all;
I=imread('cameraman.tif');
fun1=@dct2;

clear all; close all;
I=imread('peppers.png');
I=rgb2gray(I);
I=im2double(I);
h1=size(I, 1);
h2=size(I, 2);
H1=hadamard(h1);
J1=blkproc(I, [8 8], fun1);
fun2=@(x) std2(x)*ones(size(x));
J2=blkproc(I, [8 8], fun2);
figure;
subplot(121);
imagesc(J1);
subplot(122);
imagesc(J2);
colormap gray;

clear all; close all;
I=imread('rice.png');
J=im2double(I);
T=dctmtx(8);
K=blkproc(J, [8 8], 'P1*x*P2', T, T');
mask=[ 1   1   1   1   0   0   0   0
       1   1   1   0   0   0   0   0
       1   1   0   0   0   0   0   0
       1   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0 ];
K2=blkproc(K, [8 8], 'P1.*x', mask);
L=blkproc(K2, [8 8], 'P1*x*P2', T', T);
figure;
subplot(121);
imshow(J);
subplot(122);
imshow(L);

clear all; close all;
H1=hadamard(2)
H2=hadamard(4)
H3=H2'*H2
```

```
H2=hadamard(h2);
J=H1*I*H2/sqrt(h1*h2);
figure;
set(0,'defaultFigurePosition',[100,100,1000,500]);
set(0,'defaultFigureColor',[1 1 1])
subplot(121);
imshow(I);
subplot(122);
imshow(J);

clear all; close all;
I=imread('circuit.tif');
I=im2double(I);
BW=edge(I, 'canny');
[H, Theta, Rho]=hough(BW, 'RhoResolution', 0.5, 'ThetaResolution', 0.5);
figure;
set(0,'defaultFigurePosition',[100,100,1000,500]);
set(0,'defaultFigureColor',[1 1 1])
subplot(121);
imshow(BW);
subplot(122);
imshow(imadjust(mat2gray(H)));
axis normal;
hold on;
colormap hot;

clear all; close all;
I=imread('gantrycrane.png');
I=rgb2gray(I);
BW=edge(I, 'canny');
[H, Theta, Rho]=hough(BW, 'RhoResolution', 0.5, 'Theta', -90:0.5:89.5);
P=houghpeaks(H, 5, 'threshold', ceil(0.3*max(H(:))));
x=Theta(P(:, 2));
y=Rho(P(:, 1));
figure;
set(0,'defaultFigurePosition',[100,100,1000,500]);
set(0,'defaultFigureColor',[1 1 1])
subplot(121);
imshow(imadjust(mat2gray(H)), 'XData', Theta, 'YData', Rho,...
      'InitialMagnification', 'fit');
axis on;
axis normal;
hold on;
plot(x, y, 's', 'color', 'white');
```

```
lines=houghlines(BW, Theta, Rho, P, 'FillGap', 5, 'MinLength', 7);
subplot(122);
imshow(I);
hold on;
maxlen=0;
for k=1:length(lines)
    xy=[lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:, 2), 'linewidth', 2, 'color', 'green');
    plot(xy(1,1), xy(1, 2), 'linewidth', 2,    'color', 'yellow');
    plot(xy(2,1), xy(2, 2), 'linewidth', 2,    'color', 'red');
    len=norm(lines(k).point1-lines(k).point2);
    if (len>maxlen)
        maxlen=len;
        xylong=xy;
    end
end
hold on;
plot(xylong(:, 1), xylong(:, 2), 'color', 'blue');
```

```
clear all; close all;
I=imread('coins.png');
I=im2double(I);
M=2*size(I,1);
N=2*size(I,2);
u=-M/2:(M/2-1);
v=-N/2:(N/2-1);
[U,V]=meshgrid(u, v);
D=sqrt(U.^2+V.^2);
D0=20;
H=1-exp(-(D.^2)./(2*(D0^2)));
J=fftshift(fft2(I, size(H, 1), size(H, 2)));
K=J.*H;
L=ifft2(ifftshift(K));
L=L(1:size(I,1), 1:size(I, 2));
figure;
subplot(121);
imshow(I);
subplot(122);
imshow(L);
```

```
clear all; close all;
[x, y]=meshgrid(-128:2:127, -128:2:127);
D=sqrt((x).^2+(y).^2);
D0=50;
W=30;
H=double(or(D<(D0-W/2), D>D0+W/2));
H2=1-H;
figure;
mesh(double(H));
axis tight;
figure;
mesh(double(H2));
axis tight;
```

```
clear all; close all;
[x, y]=meshgrid(-128:2:127, -128:2:127);
D=sqrt((x).^2+(y).^2);
D0=50;
W=30;
n=5;
H=1./(1+(D*W./(D.^2-D0^2)).^(2*n));
H2=1-H;
figure;
mesh(double(H));
% axis tight;
```

```
clear all; close all;
[x, y]=meshgrid(-128:2:127, -128:2:127);
D=sqrt((x).^2+(y).^2);
```

```
D0=50;
W=30;
H=1-exp(-((D.^2-D0^2)./(W*D)).^2/2);
H2=1-H;
figure;
mesh(double(H));

clear all; close all;
I=imread('cameraman.tif');
I=im2double(I);
J=fftshift(fft2(I));
[x, y]=meshgrid(-128:127, -128:127);
[M, N]=size(I);
n1=floor(M/2);
n2=floor(N/2);
z=sqrt((x-n1).^2+(y-n2).^2);
D1=10;    D2=30;
n=6;
H1=1./(1+(z/D1).^(2*n));
H2=1./(1+(z/D2).^(2*n));
K1=J.*H1;
K2=J.*H2;
L1=ifft2(ifftshift(K1));
L2=ifft2(ifftshift(K2));
figure;
subplot(131);
imshow(I);
subplot(132);
imshow(real(L1));
subplot(133);
imshow(real(L2))
```

# 第 9 章　彩色图像处理

```
close all; clear all; clc;
RGB=reshape(ones(64,1)*reshape(jet(64),1,192),[64,64,3]);
                                        %调整颜色条尺寸为正方形
HSV=rgb2hsv(RGB);                       %将 RGB 图像转换为 HSV 图像
H=HSV(:,:,1);                           %提取 H 矩阵
S=HSV(:,:,2);                           %提取 S 矩阵
V=HSV(:,:,3);                           %提取 V 矩阵
figure(1)
subplot(121), imshow(H)                 %显示 H 图像
subplot(122), imshow(S)                 %显示 S 图像
figure(2)
subplot(121), imshow(V)                 %显示 V 图像
subplot(122), imshow(RGB)               %显示 RGB 图像

close all; clear all; clc;
RGB = imread('board.tif');              %读入 RGB 图像
YCBCR = rgb2ycbcr(RGB);                 %将 RGB 图像转换为 YCBCR 图像
figure;
subplot(121), imshow(RGB)               %显示 RGB 图像
subplot(122), imshow(YCBCR)             %显示 YCBCR 图像

function SEASON(month)                  % 计算几月处于什么季节
                                        %函数 SEASON(month)的输入参数 month 为整数型
switch month
case {3,4,5}
season='spring';
case {6,7,8}
season='summer';
case {9,10,11}
season='autumn';
case{1,2,12}
season='winter';
otherwise
season= 'Wrong';
end
```

# 第 10 章　图像压缩编码

```
function s=binCode(a)                    %求任意整数的二进制码
if a>=0
    s=dec2bin(a);
else                                     %求 a 的反码，返回“01”字符串，按位取反
    s=dec2bin(abs(a));
    for t=1:numel(s)
        if s(t)=='0'
            s(t)='1';
        else s(t)='0';
        end
    end
end

%【例 10-2】霍夫曼编码
close all; clear all; clc;
A=[0.5,0.19,0.19,0.12];                  %信源消息的概率序列
A=fliplr(sort(A));                       %按降序排列
T=A;
[m,n]=size(A);
B=zeros(n,n-1);                          %空的编码表（矩阵）
for i=1:n
    B(i,1)=T(i);                         %生成编码表的第一列
end
r=B(i,1)+B(i-1,1);                       %最后两个元素相加
T(n-1)=r;
T(n)=0;
T=fliplr(sort(T));
t=n-1;
for j=2:n-1                              %生成编码表的其他各列
    for i=1:t
        B(i,j)=T(i);
    end
        K=find(T==r);
        B(n,j)=K(end);                   %从第二列开始，每列的最后一个元素记录特征元
                                           素在该列的位置
        r=(B(t-1,j)+B(t,j));             %最后两个元素相加
        T(t-1)=r;
        T(t)=0;
        T=fliplr(sort(T));
        t=t-1;
```

```
end
B;                                  %输出编码表
END1=sym('[0,1]');                  %给最后一列的元素编码
END=END1;
t=3;
d=1;
for j=n-2:-1:1                      %从倒数第二列开始依次对各列元素编码
    for i=1:t-2
        if i>1 & B(i,j)==B(i-1,j)
            d=d+1;
        else
            d=1;
        end
        B(B(n,j+1),j+1)=-1;
        temp=B(:,j+1);
        x=find(temp==B(i,j));
        END(i)=END1(x(d));
    end
    y=B(n,j+1);
    END(t-1)=[char(END1(y)),'0'];
    END(t)=[char(END1(y)),'1'];
    t=t+1;
    END1=END;
end
disp('排序后的原概率序列 A：');
disp(A)                             %排序后的原概率序列
disp('编码结果 END:')
disp(END)      ;                    %编码结果
for i=1:n
    [a,b]=size(char(END(i)));
    L(i)=b;
end
disp('平均码字长度')
avlen=sum(L.*A);disp(avlen);        %平均码长
H1=log2(A);
disp('信息熵')
H=-A*(H1');disp(H)                  %熵
disp('编码效率')
P=H/avlen;disp(P)                   %编码效率

%【例 10-3】
close all; clear all; clc;
I=imread('lena.bmp');
I=im2double(I)*255;
```

```
[height,width]=size(I);                    %求图像的大小
HWmatrix=zeros(height,width);
Mat=zeros(height,width);                   %建立大小与原图像大小相同的矩阵 HWmatrix 和
                                             Mat，矩阵元素为 0。
HWmatrix(1,1)=I(1,1);                      %图像第一个像素值 I(1,1)传给 HWmatrix(1,1)
for i=2:height                             %以下将图像像素值传递给矩阵 Mat
    Mat(i,1)=I(i-1,1);
end
for j=2:width
    Mat(1,j)=I(1,j-1);
end
for i=2:height                             %以下建立待编码的数组 symbols 和每个像素出现
                                             的概率矩阵 p
    for j=2:width
        Mat(i,j)=I(i,j-1)/2+I(i-1,j)/2;
    end
end
Mat=floor(Mat);HWmatrix=I-Mat;
SymPro=zeros(2,1); SymNum=1; SymPro(1,1)=HWmatrix(1,1); SymExist=0;
for i=1:height
    for j=1:width
        SymExist=0;
        for k=1:SymNum
            if SymPro(1,k)==HWmatrix(i,j)
                SymPro(2,k)=SymPro(2,k)+1;
                SymExist=1;
                break;
            end
        end
        if SymExist==0
           SymNum=SymNum+1;
           SymPro(1,SymNum)=HWmatrix(i,j);
           SymPro(2,SymNum)=1;
        end
    end
end
for i=1:SymNum
    SymPro(3,i)=SymPro(2,i)/(height*width);
end
symbols=SymPro(1,:);p=SymPro(3,:);
[dict,avglen] = huffmandict(symbols,p);    %产生霍夫曼编码词典，返回编码词典
                                             dict 和平均码长 avglen
actualsig=reshape(HWmatrix',1,[]);
compress=huffmanenco(actualsig,dict);      %利用 dict 对 actuals 来编码，其结果存
```

```
                                                    放在 compress 中
UnitNum=ceil(size(compress,2)/8);
Compressed=zeros(1,UnitNum,'uint8');
for i=1:UnitNum
    for j=1:8
        if ((i-1)*8+j)<=size(compress,2)
        Compressed(i)=bitset(Compressed(i),j,compress((i-1)*8+j));
        end
    end
end
NewHeight=ceil(UnitNum/512);Compressed(width*NewHeight)=0;
ReshapeCompressed=reshape(Compressed,NewHeight,width);
imwrite(ReshapeCompressed,'Compressed Image.bmp','bmp');
Restore=zeros(1,size(compress,2));
for i=1:UnitNum
    for j=1:8
        if ((i-1)*8+j)<=size(compress,2)
        Restore((i-1)*8+j)=bitget(Compressed(i),j);
        end
    end
end
decompress=huffmandeco(Restore,dict);          %利用 dict 对 Restore 来解码，其结果存
                                                    放在 decompress 中
RestoredImage=reshape(decompress,512,512);
RestoredImageGrayScale=uint8(RestoredImage'+Mat);
imwrite(RestoredImageGrayScale,'Restored Image.bmp','bmp');
figure;
subplot(1,3,1);imshow(I,[0,255]);              %显示原图
subplot(1,3,2);imshow(ReshapeCompressed);      %显示压缩后的图像
subplot(1,3,3);imshow('Restored Image.bmp');   %解压后的图像

%【10-5】
close all; clear all; clc;
p=[0.5 0.19 0.19 0.12]                         %输入信息符号对应的概率
n=length(p);                                   %输入概率的个数
y=fliplr(sort(p));                             %大到小排序
D=zeros(n,4);                                  %生成 n*4 的零矩阵
D(:,1)=y';                                     %把 y 赋给零矩阵的第一列
for i=2:n
D(1,2)=0;                                      %令第一行第二列的元素为 0
D(i,2)=D(i-1,1)+D(i-1,2);                      %求累加概率
 end
   for i=1:n
D(i,3)=-log2(D(i,1));                          %求第三列的元素
```

```
D(i,4)=ceil(D(i,3));                          %求第四列的元素，对 D(i,3)向无穷方向
                                                取最小正整数
    end
D
A=D(:,2)';                                    %取出 D 中第二列元素
B=D(:,4)';                                    %取出 D 中第四列元素
for j=1:n
C=binary(A(j),B(j))                           %生成码字
end
%建立 binary.m 文件，自定义求小数的二进制转换函数
function [C]=binary(A,B)                      %对累加概率求二进制的函数
C=zeros(1,B);                                 %生成零矩阵用于存储生成的二进制数，
                                                对二进制的每一位进行操作
temp=A;                                       %temp 赋初值
for i=1:B                                     %累加概率转化为二进制，循环求二进制
                                                的每一位，A 控制生成二进制的位数
   temp=temp*2;
if temp>=1
   temp=temp-1;
 C(1,i)=1;
   else
 C(1,i)=0;
   end
 end

%[10.7]
close all; clear all; clc;
I=[0 0 1 1 0 0 1 1;1 0 0 1 0 0 1 1;1 1 0 0 0 0 1 0];   %待编码的矩阵
[m,n]=size(I);                                %计算矩阵大小
I=double(I);
p_table=tabulate(I(:));                       %统计矩阵中元素出现的概率，第一列为
                                                矩阵元素，第二列为个数，第三列为概
                                                率百分数
color=p_table(:,1)';
p=p_table(:,3)'/100;                          %转换成小数表示的概率
psum=cumsum(p_table(:,3)');                   %计算数组各行的累加值
allLow=[0,psum(1:end-1)/100];                 %由于矩阵中元素只有两种，将[0,1）区
                                                间划分为两个区域 allLow 和 allHigh
allHigh=psum/100;
numberlow=0;                                  %定义算术编码的上下限 numberlow 和
                                                numberhigh
numberhigh=1;
for k=1:m                                     %以下计算算术编码的上下限，即编码结果
    for kk=1:n
```

```
        data=I(k,kk);
        low=allLow(data==color);
        high=allHigh(data==color);
        range=numberhigh-numberlow;
        tmp=numberlow;
        numberlow=tmp+range*low;
        numberhigh=tmp+range*high;
    end
end
fprintf('算术编码范围下限为%16.15f\n\n',numberlow);
fprintf('算术编码范围上限为%16.15f\n\n',numberhigh);
Mat=zeros(m,n);                                    %解码
for k=1:m
    for kk=1:n
        temp=numberlow<low;
        temp=[temp 1];
        indiff=diff(temp);
        indiff=logical(indiff);
        Mat(k,kk)=color(indiff);
        low=low(indiff);
        high=allHigh(indiff);
        range=high - low;
        numberlow=numberlow-low;
        numberlow=numberlow/range;
    end
end
fprintf('原矩阵为:\n')
disp(I);
fprintf('\n');
fprintf('解码矩阵:\n');
disp(Mat);

close all; clear all; clc;
I1=imread('lena.bmp');                             %读入图像
I2=I1(:);                                          %将原始图像写成一维的数据并设为 I2
I2length=length(I2);                               %计算 I2 的长度
I3=im2bw(I1,0.5);                                  %将原图转换为二值图像，阈值为 0.5
                                                   %以下程序为对原图像进行行程编码，压缩
X=I3(:);                                           %令 X 为新建的二值图像的一维数据组
L=length(X);
j=1;
I4(1)=1;
for z=1:1:(length(X)-1)                            %行程编码程序段
if    X(z)==X(z+1)
```

```
I4(j)=I4(j)+1;
else
data(j)=X(z);                                   % data(j)代表相应的像素数据
j=j+1;
I4(j)=1;
end
end
data(j)=X(length(X));                           %最后一个像素数据赋给 data
I4length=length(I4);                            %计算行程编码后的所占字节数，记为 I4length
CR=I2length/I4length;                           %比较压缩前于压缩后的大小
                                                %下面程序是行程编码解压
l=1;
for m=1:I4length
    for n=1:1:I4(m);
        decode_image1(l)=data(m);
        l=l+1;
    end
end
decode_image=reshape(decode_image1,512,512);    %重建二维图像数组
figure,
x=1:1:length(X);
subplot(131),plot(x,X(x));                      %显示行程编码之前的图像数据
y=1:1:I4length ;
subplot(132),plot(y,I4(y));                     %显示编码后数据信息
u=1:1:length(decode_image1);
subplot(133),plot(u,decode_image1(u));          %查看解压后的图像数据
subplot(121);imshow(I3);                        %显示原图的二值图像
subplot(122),imshow(decode_image);              %显示解压恢复后的图像
disp('压缩比: ')
disp(CR);
disp('原图像数据的长度：')
disp(L);
disp('压缩后图像数据的长度：')
disp(I4length);
disp('解压后图像数据的长度：')
disp(length(decode_image1));

%[10.9]
close all; clear all; clc;
J=imread('eye.bmp');                            %装入图像，用 Yucebianma 进行线性预测编码，
                                                用 Yucejiema 解码
X=double(J);
Y=Yucebianma(X);
```

```
XX=Yucejiema(Y);
e=double(X)-double(XX);[m,n]=size(e);
erm=sqrt(sum(e(:).^2)/(m*n));
set(0,'defaultFigurePosition',[100,100,1000,500]); %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure,
subplot(121);imshow(J);
subplot(122),imshow(mat2gray(255-Y));          %为方便显示，对预测误差图取反后再作显示
figure;
[h,x]=hist(X(:));                              %显示原图直方图
subplot(121);bar(x,h,'k');
[h,x]=hist(Y(:));
subplot(122);bar(x,h,'k');
                                               %Yucebianma 函数用一维预测编码压缩图像
                                                x,f 为预测系数，如果 f 去默认值，则默认 f=1,
                                                就是前值预测
function y=Yucebianma(x,f)
error(nargchk(1,2,nargin))
if nargin<2
   f=1;
end
x=double(x);
[m,n]=size(x);
p=zeros(m,n);                                  %存放预测值
xs=x;
zc=zeros(m,1);
for j=1:length(f)
      xs=[zc xs(:,1:end-1)];
      p=p+f(j)*xs;
end
y=x-round(p);
%Yucejiema 是解码程序，与编码程序用的是同一个预测器
function x=Yucejiema(y,f)
error(nargchk(1,2,nargin));
if nargin<2
   f=1;
end
f=f(end:-1:1);
[m,n]=size(y);
order=length(f);
f=repmat(f,m,1);
x=zeros(m,n+order);
for j=1:n
   jj=j+order;
```

```
   x(:,jj)=y(:,j)+round(sum(f(:,order:-1:1).*x(:,(jj-1):-1:(jj-order)),2));
end
x=x(:,order+1:end);

%[10.10]
close all; clear all; clc;
ORIGIN=imread('lena.bmp');                          %读入原始图像
%步骤 1：正向离散余弦变换(FDCT)
fun=@DCT_Measure;
%步骤 2：量化
B=blkproc(ORIGIN,[8,8],fun);                        %得到量化后的系数矩阵，与原始图像尺
                                                      寸相同，需要进一步处理
n=length(B)/8;                                      %对每个维度分成的块数
C=zeros(8);                                         %初始化为 8×8 的全 0 矩阵
for y=0:n-1
    for x=0:n-1
        T1=C(:,[end-7:end]);                        %取出上一组数据做差分,T1 的所有 8 行
                                                      和最后 8 列组成的 8*8
        T2=B(1+8*x:8+8*x,1+8*y:8+8*y);
        T2(1)=T2(1)-T1(1);                          %直流系数做差分
        C=[C,T2];                                   %将 C 和 T2 矩阵串联
    end
end
C=C(:,[9:end]);                                     %去除 C 的前 8 列，就是前面的全 0
%步骤 4：利用 Code_Huffman( )函数实现上述 JPEG 算法步骤中的步骤 3、4、5 和 6 步
JPGCode={''};                                       %存储编码的元胞初始化为空的字符串
for a=0:n^2-1
    T=Code_Huffman(C(:,[1+a*8:8+a*8]));
    JPGCode=strcat(JPGCode,T);
end
sCode=cell2mat(JPGCode);                            %将元胞转化为数组
Fid=fopen('JPGCode.txt','w');                       %用变量 fid 标记 I/O 流，打开文本文件
fprintf(Fid,'%s',sCode);                            %将压缩码 sCode 保存到文本文件中。添
                                                      加而不是覆盖
fclose(Fid);                                        %关闭 I/O 流
[x y]=size(A);
b=x*y*8/length(sCode);
v=8/b;                                              %计算压缩比和压缩效率
disp('JPEG 压缩数据已保存至 JPGCode.txt 中!');
disp(['压缩比为：',num2str(b),'；压缩效率：',num2str(v)]);

function B=Code_Huffman(A)
%根据 huffman 编码表对量化后的数据编码
%依次输入 DC 系数差值(A 中 DC 系数已做过差分)和 AC 系数的典型 Huffman 表
```

```
%只处理 8×8DCT 系数量化矩阵，每个 A 都是 8*8
DC_Huff={'00','010','011','100','101','110','1110','11110','111110','1111110','11111110','1111111
10'};
                                        %由于 AC 系数数据量较大，我们将它保存在
                                          AC_Huff.txt 文件中，将它读入元胞数组中
fid=fopen('AC_Huff.txt','r');
AC_Huff=cell(16,10);
for a=1:16
    for b=1:10
        temp=fscanf(fid,'%s',1);          %以行为单位读取，保存在 temp 中
        AC_Huff(a,b)={temp};              %代表每行的一组数据
    end
end
fclose(fid);
                                        %对 A 中的数据进行 Zig-Zag 扫描，保存在数组 Z
i=1;
for a=1:15
    if a<=8
        for b=1:a
            if mod(a,2)==1
                Z(i)=A(b,a+1-b);
                i=i+1;
            else
                Z(i)=A(a+1-b,b);
                i=i+1;
            end

        end
    else
        for b=1:16-a
            if mod(a,2)==0
                Z(i)=A(9-b,a+b-8);
                i=i+1;
            else
                Z(i)=A(a+b-8,9-b);
                i=i+1;
            end
        end
    end
end
                                          %先对 DC 差值系数编码：前缀码 SSSS+尾码
                                          %dc 为其 Huffman 编码
if Z(1)==0
    sa.s=DC_Huff(1);                      %size 分量存放前缀码
```

```
    sa.a='0';                                   %amp 分量存放尾码
    dc=strcat(sa.s,sa.a);
else
    n=fix(log2(abs(Z(1))))+1;
    sa.s=DC_Huff(n);
    sa.a=binCode(Z(1));
    dc=strcat(sa.s,sa.a);
end
                                                %再对 AC 系数进行行程编码,保存在
                                                  结构体数组 rsa 中
if isempty(find(Z(2:end)))                      %如果 63 个交流系数全部为 0，rsa 系
                                                  数全部为 0
    rsa(1).r=0;                                 %行程 runlength
    rsa(1).s=0;                                 %码长 size
    rsa(1).a=0;                                 %二进制编码
else
    T=find(Z);                                  %找出 Z 中非零元素的下标
    T=[0 T(2:end)];                             %为统一处理将第一个下标元素置为 0
    i=1;                                        % i 为 rsa 结构体的下标
                                                %从第二个元素即第一个交流元素开
                                                  始处理
    j=2;
    while j<=length(T)
        t=fix((T(j)-1-T(j-1))/16);              %判断下标间隔是否超过 16
        if t==0                                 %如果小于 16，较简单
            rsa(i).r=T(j)-T(j-1)-1;
            rsa(i).s=fix(log2(abs(Z(T(j)))))+1;
            rsa(i).a=Z(T(j));
            i=i+1;
        else                                    %如果超过 16，需要处理（15，0）的
                                                  特殊情况
            for n=1:t                           %可能出现 t 组（15，0）
                rsa(i)=struct('r',15,'s',0,'a',0);
                i=i+1;
            end
                                                %接着处理剩余的那部分
            rsa(i).r=T(j)-1-16*t;
            rsa(i).s=fix(log2(abs(Z(T(j)))))+1;
            rsa(i).a=Z(T(j));
            i=i+1;
        end
        j=j+1;
    end
                                                %判断最后一个非零元素是否为 Z 中最
```

```
                                                    后一个元素
    if T(end)<64
        rsa(i).r=0;
        rsa(i).s=0;
        rsa(i).a=0;
    end                                             %以 EOB 结束
end

                                                    %通过查表获取 AC 系数的 Huffman 编码

B=dc;                                               %B 初始化为直流系数编码
for n=1:length(rsa)
    if rsa(n).r==0&rsa(n).s==0&rsa(n).a==0
        ac(n)={'1010'};
    elseif rsa(n).r==15&rsa(n).s==0&rsa(n).a==0
        ac(n)={'11111111001'};
    else
        t1=AC_Huff(rsa(n).s+1,rsa(n).s);
        t2=binCode(rsa(n).a);
        ac(n)=strcat(t1,t2);

    end
    B=strcat(B,ac(n));
end

function B=DCT1D(A)                                 %一维离散余弦变换
n=length(A);

                                                    %对变换数组延拓
T=zeros(1,n);
C=[A,T];
C=FFT1D(C)*2*n;
T=C(1:n);
T(1)=T(1)/n^0.5;
for u=2:n
    T(u)=(2/n)^0.5*T(u)*exp(-i*(u-1)*pi/2/n);
end
B=real(T);

function C=DCT2D(B)                                 %将图像数据进行快速傅立叶变换，返
                                                      回幅度和相位信息
a=length(B);

                                                    %依次对每一行进行 FFT 操作
C=zeros(a);
for b=1:a
    C(b,:)=DCT1D(B(b,:));
```

```
end
                                        %依次对每一列进行 FFT 操作
for b=1:a
    T=C(:,b);
    T1=DCT1D(T');
    C(:,b)=T1';
end

function B=DCT_Measure(A)               % B 为返回值
                                        %对输入的 8×8 图像矩阵进行 DCT 变化和量化
                                        %定义 Y 分量系数量化矩阵
Y_Matrix=[16 11 10 16 24   40   51   61; 12 12 14 19 26   58   60 55;
          14 13 16 24 40   57   69   56; 14 17 22 29 51   87   80 62;
          18 22 37 56 68   109 103 77; 24 35 55 64 81   104 113 92;
          49 64 78 87 103 121 120 101;72 92 95 98 112 100 103 99];
                                        %图像为 8 位无符号数，将其减去 128 转化为
                                          有符号数
                                        %先化为 double 型，DCT 变换后化为 int8 型
C=double(A)-128;
B=round(DCT2D(C)./Y_Matrix);

function B1=FFT1D(A1)                   %对 A1 中的数据进行奇偶分解排序
B=SortOE(A1);
n=length(B);m=log2(n);
for s=1:n
    T(s)=double(B(s));                  %将图像数据转换为 double 型
end
for a=0:m-1
    M=2^a;nb=n/M/2;                     %每一块的半长度和分成的块数
    for j=0:nb-1                        %对每一块依次进行操作
        for k=0:M-1                     %对每一块中的一半的点依次操作
            t1=double(T(1+k+j*2*M));t2=double(T(1+k+j*2*M+M))*exp(-i*pi*k/M);
            T(1+k+j*2*M)=0.5*(t1+t2);
            T(1+k+j*2*M+M)=0.5*(t1-t2);
        end
    end
end
B1=T;

%奇偶分解排序函数
function   B=SortOE(T)
    n=length(T);m=log2(n/2);
for i=1:m
    nb=2^i;lb=n/nb;                     %分成的块数和每一块的长度
```

```
    lc=2*lb;                                    %操作间隔
        for j=0:nb/2-1                          %进行排序操作的次数
            t=T(2+j*lc:2:2*lb+j*lc);
            T(1+j*lc:lb+j*lc)=T(1+j*lc:2:(2*lb-1)+j*lc);
            T(lb+1+j*lc:2*lb+j*lc)=t;
        end
 end
B=T;
```

%Yucebianma 函数用一维预测编码压缩图像 x,f 为预测系数，建立 Yucebianma.m 文件

```
function y=Yucebianma(x,f)
error(nargchk(1,2,nargin))
if nargin<2
   f=1;
end
x=double(x);
[m,n]=size(x);
p=zeros(m,n);                                   %存放预测值
xs=x;
zc=zeros(m,1);
for j=1:length(f)
      xs=[zc xs(:,1:end-1)];
      p=p+f(j)*xs;
end
y=x-round(p);
```

%Yucejiema 是解码程序，与编码程序用的是同一个预测器。建立 Yucejiema.m 文件

```
function x=Yucejiema(y,f)
error(nargchk(1,2,nargin));
if nargin<2
   f=1;
end
f=f(end:-1:1);
[m,n]=size(y);
order=length(f);
f=repmat(f,m,1);
x=zeros(m,n+order);
for j=1:n
   jj=j+order;
   x(:,jj)=y(:,j)+round(sum(f(:,order:-1:1).*x(:,(jj-1):-1:(jj-order)),2));
end
x=x(:,order+1:end);
```

# 第 11 章　图像特征分析

```
%【例 11-1】颜色矩求法
close all; clear all;clc;
I=imread('hua.jpg');                                    %I 为花的彩色图像，以下是求花的图像
                                                          的 RGB 分量均值
R=I(:,:,1);                                             %红色分量
G=I(:,:,2);                                             %绿色分量
B=I(:,:,3);                                             %蓝色分量
R=double(R);    G=double(G); B=double(B);       %利用 double()函数将变量类型转为 double 型
Ravg1=mean2(R);                                         %红色分量均值
Gavg1=mean2(G);                                         %绿色分量均值
Bavg1=mean2(B);                                         %蓝色分量均值
Rstd1=std(std(R));                                      %红色分量的方差
Gstd1= std(std(G));                                     %绿色分量的方差
Bstd1=std(std(B));                                      %蓝色分量的方差
J=imread('yezi.jpg');                                   %J 为叶子的彩色图像以下是求叶子的
                                                          图像的 RGB 分量均值
R=J(:,:,1);                                             %红色分量
G=J(:,:,2);                                             %绿色分量
B=J(:,:,3);                                             %蓝色分量
R=double(R);    G=double(G); B=double(B);       %利用 double()函数将变量类型转为 double 型
Ravg2=mean2(R);                                         %红色分量均值
Gavg2=mean2(G);                                         %绿色分量均值
Bavg2=mean2(B);                                         %蓝色分量均值
Rstd2=std(std(R));                                      %红色分量的方差
Gstd2= std(std(G));                                     %绿色分量的方差
Bstd2=std(std(B));                                      %蓝色分量的方差
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
K=imread('flower1.jpg');figure;subplot(131),imshow(K);  %显示原图像
subplot(132),imshow(I);                                 %显示花的图像
subplot(133),imshow(J);                                 %显示叶子的图像

close all; clear all;clc;
I=imread('hua.jpg');                                    %I 为花的彩色图像，以下是求花的图
                                                          像的 RGB 分量均值
R=I(:,:,1);                                             %红色分量
G=I(:,:,2);                                             %绿色分量
B=I(:,:,3);                                             %蓝色分量
R=double(R);    G=double(G); B=double(B);       %利用 double()函数将变量类型转为 double 型
Ravg1=mean2(R);                                         %红色分量均值
```

```
Gavg1=mean2(G);                                    %绿色分量均值
Bavg1=mean2(B);                                    %蓝色分量均值
Rstd1=std(std(R));                                 %红色分量的方差
Gstd1= std(std(G));                                %绿色分量的方差
Bstd1=std(std(B));                                 %蓝色分量的方差
J=imread('yezi.jpg');                              %J 为叶子的彩色图像以下是求叶子的
                                                    图像的 RGB 分量均值
R=J(:,:,1);                                        %红色分量
G=J(:,:,2);                                        %绿色分量
B=J(:,:,3);                                        %蓝色分量
R=double(R);   G=double(G); B=double(B);          %利用 double()函数将变量类型转为 double 型
Ravg2=mean2(R);                                    %红色分量均值
Gavg2=mean2(G);                                    %绿色分量均值
Bavg2=mean2(B);                                    %蓝色分量均值
Rstd2=std(std(R));                                 %红色分量的方差
Gstd2= std(std(G));                                %绿色分量的方差
Bstd2=std(std(B));                                 %蓝色分量的方差
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
K=imread('flower1.jpg');figure;subplot(131),imshow(K);   %显示原图像
subplot(132),imshow(I);                            %显示花的图像
subplot(133),imshow(J);                            %显示叶子的图像

%例【11-3】
close all; clear all;clc;
I=imread('huangguahua.jpg');                       %读入要处理的图像，并赋值给 I
R=I(:,:,1);                                        %图像的 R 分量
G=I(:,:,2);                                        %图像的 G 分量
B=I(:,:,3);                                        %图像的 B 分量
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(121);imshow(I);                     %显示彩色图像
subplot(122);imshow(R);                            %R 分量灰度图
figure;subplot(121);imshow(G);                     %G 分量灰度图
subplot(122);imshow(B);                            %B 分量灰度图
figure;subplot(131);
imhist(I(:,:,1))                                   %显示红色分辨率下的直方图
subplot(132);imhist(I(:,:,2))                      %显示绿色分辨率下的直方图
subplot(133);imhist(I(:,:,3))                      %显示蓝色分辨率下的直方图

%例【11-4】求 HSV 空间的直方图，未对 H,S,V 进行量化。
close all; clear all;clc;
J=imread('huangguahua.jpg');                       %读入要处理的图像，并赋值给 J
hsv = rgb2hsv(J);                                  %图像由 RGB 空间变换到 HSV 空间
```

```
h = hsv(:, :, 1);                                   %为色调 h 赋值
s = hsv(:, :, 2);                                   %为饱和度 s 赋值
v = hsv(:, :, 3);                                   %为亮度 v 赋值
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(121);imshow(J);                      %显示原图
subplot(122);imshow(h);                             %基于色调 h 的灰度图像
figure;subplot(121);imshow(s);                      %基于饱和度 s 的灰度图像
subplot(122);imshow(v);                             %基于亮度 v 的灰度图像
figure;subplot(131);imhist(h);                      %显示色调 h 的直方图
subplot(132);imhist(s);                             %显示饱和度 s 的直方图
subplot(133);imhist(v);                             %显示亮度 v 的图

%例【11-5】灰度差分统计特征计算，分析纹理图像
J=imread('wall.jpg');                               %读入纹理图像，分别输入 wall.jpg
                                                      和 stone.jpg 两幅图进行对比
A=double(J);
[m,n]=size(A);                                      %求 A 矩阵的大小，赋值给 m n
B=A;
C=zeros(m,n);                          %新建全零矩阵 C，以下求解归一化的灰度直方图
for i=1:m-1
    for j=1:n-1
        B(i,j)=A(i+1,j+1);
        C(i,j)=abs(round(A(i,j)-B(i,j)));
    end
end
h=imhist(mat2gray(C))/(m*n);
mean=0;con=0;ent=0;                    %均值 mean、对比度 con 和熵 ent 初始值赋零
for i=1:256                            %循环求解均值 mean、对比度 con 和熵 ent
    mean=mean+(i*h(i))/256;
    con=con+i*i*h(i);
    if(h(i)>0)
        ent=ent-h(i)*log2(h(i));
    end
end
    mean,con,ent

%【例 11-6】
%步骤 1：定义自相关函数 zxcor()，建立 zxcor.m 文件
function [epsilon,eta,C]=zxcor(f,D,m,n)
%自相关函数 zxcor(),f 为读入的图像数据，D 为偏移距离，【m，n】是图像的尺寸数据，返
回图像相关函数 C 的
%值，epsilon 和 eta 是自相关函数 C 的偏移变量。
for epsilon=1:D                                     %循环求解图像 f(x,y)与偏离值为 D
```

```
                                                    的像素之间的相关值
  for eta=1:D
      temp=0;
      fp=0;
      for x=1:m
          for y=1:n
              if(x+ epsilon -1)>m|(y+ eta -1)>n
                 f1=0;
              else
                f1=f(x,y)*f(x+ epsilon -1,y+ eta -1);
              end
              temp=f1+temp;
              fp=f(x,y)*f(x,y)+fp;
          end
      end
          f2(epsilon, eta)=temp;
          f3(epsilon, eta)=fp;
          C(epsilon, eta)= f2(epsilon, eta)/ f3(epsilon, eta);      %相关值 C
    end
end
epsilon =0:(D-1);                                                   % 方向的取值范围
eta =0:(D-1);                                                       % 方向的取值范围
%步骤 2：调用 zxcor()函数，分析不同图像的纹理特征。
f11=imread('wall.jpg');                                %读入砖墙面图像，图像数据赋值给 f
f1=rgb2gray(f11);                                                   %彩色图像转换成灰度图像
f1=double(f1);                                                      %图像数据变为 double 类型
[m,n]=size(f1);                                                     %图像大小赋值为[m,n]
D=20;                                                               %偏移量为 20
[epsilon1,eta1,C1]=zxcor1(f1,D,m,n);                                %调用自相关函数
f22=imread('stone.jpg');                               %读入大理石图像，图像数据赋值给 f
f2=rgb2gray(f22);
f2=double(f2);
[m,n]=size(f2);
[epsilon2,eta2,C2]=zxcor1(f2,20,m,n);                               %调用自相关函数
set(0,'defaultFigurePosition',[100,100,1000,500]);                  %修改图形图像位置默认设置
set(0,'defaultFigureColor',[1 1 1]);
figure;subplot(121);imshow(f11);
subplot(122);imshow(f22);
figure;subplot(121);mesh(epsilon1,eta1,C1);            %显示自相关函数与 x，y 的三维图像
xlabel(' epsilon ');ylabel(' eta ');                                %标示坐标轴变量
subplot(122);mesh(epsilon2,eta2,C2);
xlabel(' epsilon ');ylabel(' eta ');

%例【11.7】求图像的灰度共生矩阵
```

```
close all; clear all;clc;
I = imread('circuit.tif');                                    %读入图像 circuit.tif
glcm = graycomatrix(I,'Offset',[0 2]);      %图像 I 的灰度共生矩阵，2 表示当前像素与邻居的距
                                              离为 2，offset 为[0 2]表示角度为 0 为水平方向
set(0,'defaultFigurePosition',[100,100,1000,500]);            %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
imshow(I);
glcm

%【例 11-8】
I=imread('hill.jpg');
HSV=rgb2hsv(I);
Hgray=rgb2gray(HSV);
%计算 64 位灰度共生矩阵
glcms1=graycomatrix(Hgray,'numlevels',64,'offset',[0 1;-1 1;-1 0;-1 -1]);
%纹理特征统计值(包括对比度、相关性、熵、平稳度、二阶矩也叫能量)
stats=graycoprops(glcms1,{'contrast','correlation','energy','homogeneity'});
ga1=glcms1(:,:,1);                                            %0 度
ga2=glcms1(:,:,2);                                            %45 度
ga3=glcms1(:,:,3);                                            %90 度
ga4=glcms1(:,:,4);                                            %135 度
energya1=0;energya2=0;energya3=0;energya4=0;
for i=1:64
    for j=1:64
        energya1=energya1+sum(ga1(i,j)^2);
        energya2=energya2+sum(ga2(i,j)^2);
        energya3=energya3+sum(ga3(i,j)^2);
        energya4=energya4+sum(ga4(i,j)^2);
        j=j+1;
    end
    i=i+1;
end
s1=0;s2=0;s3=0;s4=0;s5=0;
for m=1:4
    s1=stats.Contrast(1,m)+s1;
    m=m+1;
end
for m=1:4
    s2=stats.Correlation(1,m)+s2;
    m=m+1;
end
for m=1:4
    s3=stats.Energy(1,m)+s3;
    m=m+1;
```

```
end
for m=1:4
    s4=stats.Homogeneity(1,m)+s4;
    m=m+1;
end
s5=0.000001*(energya1+energya2+energya3+energya4);
I=imread('hill.jpg');
J=imread('sea.jpg');
K=imread('house.jpg');
set(0,'defaultFigurePosition',[100,100,1000,500]);          %修改图形图像位置默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(131);imshow(I);
subplot(132);imshow(J);
subplot(133);imshow(K);

%【例 11-9】
close all; clear all;clc;
I = imread('wall.jpg');                                     %读入图像
I=rgb2gray(I);                                              %图像变为灰度图像
wall=fft2(I);                                               %对图像做快速傅里叶变换
s=fftshift(wall);                                           %将变换后的图象频谱中心从矩
                                                              阵的原点移到矩阵的中心
s=abs(s);
[nc,nr]=size(s);
x0=floor(nc/2+1);
y0=floor(nr/2+1);
rmax=floor(min(nc,nr)/2-1);
srad=zeros(1,rmax);
srad(1)=s(x0,y0);
thetha=91:270;                                              %thetha 取值 91-270
for r=2:rmax                                                %循环求解纹理频谱能量
    [x,y]=pol2cart(thetha,r);
    x=round(x)'+x0;
    y=round(y)'+y0;
    for j=1:length(x)
        srad(r)=sum(s(sub2ind(size(s),x,y)));
    end
end
[x,y]=pol2cart(thetha,rmax);
x=round(x)'+x0;
y=round(y)'+y0;
sang=zeros(1,length(x));
for th=1:length(x)
    vx=abs(x(th)-x0);
```

```
    vy=abs(y(th)-y0);
    if((vx==0)&(vy==0))
        xr=x0;
        yr=y0;
    else
        m=(y(th)-y0)/(x(th)-x0)
        xr=(x0:x(th)).'
        yr=round(y0+m*(xr-x0));
    end
    for j=1:length(xr)
        sang(th)=sum(s(sub2ind(size(s),xr,yr)));
    end
end
set(0,'defaultFigurePosition',[100,100,1000,500]);          %修改图形图像位置默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(121);
imshow('wall.jpg');subplot(122);                            %显示原图
imshow(log(abs(wall)),[]);                                  %显示频谱图
figure;subplot(121);plot(srad);                             %显示
subplot(122);plot(sang);                                    %显示

%【例 11-10】
close all; clear all;clc;
I = imread('wall.jpg');                                     %读取图像，并赋值给 I
I=rgb2gray(I);                                              %彩色图像变为灰度图像
[G,gabout] = gaborfilter(I,2,4,16,pi/10);                   %调用 garborfilter()函数对图像做小波变换
J=fft2(gabout);                                             %对滤波后的图像做 FFT 变换，变换到频域
A=double(J);
[m,n]=size(A);
B=A;
C=zeros(m,n);
for i=1:m-1
    for j=1:n-1
        B(i,j)=A(i+1,j+1);
        C(i,j)=abs(round(A(i,j)-B(i,j)));
    end
end
h=imhist(mat2gray(C))/(m*n);                                %对矩阵 C 归一化处理后求其灰
                                                            度直方图，得到归一化的直方图
mean=0;con=0;ent=0;
for i=1:256                                                 %求图像的均值、对比度和熵
    mean=mean+(i*h(i))/256;
    con=con+i*i*h(i);
    if(h(i)>0)
```

```
            ent=ent-h(i)*log2(h(i));
        end
end
set(0,'defaultFigurePosition',[100,100,1000,500]);                %修改图形图像位置默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(121);imshow(I);
subplot(122);imshow(uint8(gabout));
mean,con,ent

%【例 11-11】
I=[1 1 1 1;1 1 0 1;0 1 0 1;0 1 1 1];                    %图像数据赋值给 I，I 为 4 4 大小的矩阵
%跟踪目标的边界，返回值为一个 p  1 的数组单元，p 为目标的个数，其中每一个单元又是一个 Q 2 的矩阵，即 Q 个点的 x,y 坐标。
g=boundaries(I,4);                                      %追踪 4 连接的目标边界
c=fchcode(g{:},4);                                      %求 4 方向 freeman 链码
c.x0y0                                                  %显示代码开始处的坐标（1 2）
c.fcc                                                   %Freeman 链码（1 n），边界点集大小为 n 2
c.diff                                                  %代码 c.fcc 的一阶差分（1 n）
c.mm                                                    %最小幅度的整数（1 n）
c.diffmm                                                %代码 c.mm 的一阶差分（1 n）

%【例 11-14】
I=imread('leaf1.bmp');                                  %读入图像数据赋值给 I
I=rgb2gray(I);                                          %将彩色图像变为灰度图像
bwI=im2bw(I,graythresh(I));                %对图像进行二值化处理得到二值化图像赋值给 bwI
bwIsl=~bwI;                                             %对二值图像取反
h=fspecial('average');                                  %选择中值滤波
bwIfilt=imfilter(bwIsl,h);                              %对图像进行中值滤波
bwIfiltfh=imfill(bwIfilt,'holes');                      %填充二值图像的空洞区域
bdI=boundaries(bwIfiltfh,4,'cw');                       %追踪 4 连接目标边界
d=cellfun('length',bdI);                %求 bdI 中每一个目标边界的长度，返回值 d 是一个向量
[dmax,k]=max(d);                        %返回向量 d 中最大的值，存在 max_d 中，k 为其索引
B4=bdI{k(1)};                                           %若最大边界不止一条，则取出其中的一
                                                          条即可。B4 是一个坐标数组
[m,n]=size(bwIfiltfh);                                  %求二值图像的大小
xmin=min(B4(:,1));
ymin=min(B4(:,2)); %生成一幅二值图像,大小为 m n，xmin,ymin 是 B4 中最小的 x 和 y 轴坐标
bim=bound2im(B4,m,n,xmin,ymin);
[x,y]=minperpoly(bwIfiltfh,2);                          %使用大小为 2 的方形单元
b2=connectpoly(x,y);                                %按照坐标(X,Y)顺时针或者逆时针连接成多边形
B2=bound2im(b2,m,n,xmin,ymin);
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure,subplot(121);imshow(bim);                        %显示原图像边界
```

```
subplot(122),imshow(B2);                          %显示按大小为 2 的正方形单元近似的边界

%【例 11-13】
I= imread('leaf1.bmp');                           %读入图像
c= im2bw(I, graythresh(I));                       %I 转换为二值图像
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(131);imshow(I);                    %显示原图
c=flipud(c);                                      %实现矩阵 c 上下翻转
b=edge(c,'canny');                                %基于 canny 算子进行轮廓提取
[u,v]=find(b);                                    %返回边界矩阵 b 中非零元素的位置
xp=v;                                             %行值 v 赋给 xp
yp=u;                                             %列值 u 赋给 yp
x0=mean([min(xp),max(xp)]);                       %x0 为行值的均值
y0=mean([min(yp),max(yp)]);                       %y0 为列值的均值
xp1=xp-x0;
yp1=yp-y0;
[cita,r]=cart2pol(xp1,yp1);                       %直角坐标转换成极坐标
q=sortrows([cita,r]);                             %从 r 列开始比较数值并按升序排序
cita=q(:,1);                                      %赋角度值
r=q(:,2);                                         %赋半径模值
subplot(132);polar(cita,r);                       %画出极坐标下的轮廓图
[x,y]=pol2cart(cita,r);
x=x+x0;
y=y+y0;
subplot(133);plot(x,y);                           %画出直角坐标下的轮廓图

%【例 11-14】
I= imread('leaf1.bmp');                           %读入图像
I= im2bw(I);                                      %转换为二值图像
C=bwlabel(I,4);                                   %对二值图像进行 4 连通的标记
Ar=regionprops(C,'Area');                         %求 C 的面积
Ce=regionprops(C,'Centroid');                     %求 C 的重心
Ar
Ce

%【例 11-15】
close all; clear all;clc;
I = imread('liftingbody.png');                    %读入图像
S = qtdecomp(I,.27);                              %四叉树分解，阈值为 0.27
blocks = repmat(uint8(0),size(S));                %矩阵扩充为 S 的大小
for dim = [512 256 128 64 32 16 8 4 2 1];
   numblocks = length(find(S==dim));
   if (numblocks > 0)
```

```
        values = repmat(uint8(1),[dim dim numblocks]);      %左上角元素为 1
        values(2:dim,2:dim,:) = 0;                          %其他地方元素为 0
        blocks = qtsetblk(blocks,S,dim,values);
    end
end
blocks(end,1:end) = 1;    blocks(1:end,end) = 1;
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(121);imshow(I);
subplot(122), imshow(blocks,[])                         %显示四叉树分解的图像

%【例 11-16】
close all; clear all;clc;
I=imread('number.jpg');                                 %读入图像
K=im2bw(I);                                             %I 转换为二值图像
J=~K;                                                   %图像取反
EUL=bweuler(J)                                          %求图像的欧拉数
set(0,'defaultFigurePosition',[100,100,1000,500]);
set(0,'defaultFigureColor',[1 1 1])
figure;subplot(131);imshow(I);                          %绘出原图
subplot(132);imshow(K);                                 %二值图
subplot(133);imshow(J);                                 %取反后的图

%【例 11-17】
I=imread('cameraman.tif');                              %读入要处理的图像，并赋值给 I
%set(0,'defaultFigurePosition',[100,100,1000,500]);     %修改图形图像位置的默认设置
%set(0,'defaultFigureColor',[1 1 1])
Image=I;                                                %图像 I 数据赋给 Image
figure;subplot(121);imshow(Image);
Image1=imrotate(I,10,'bilinear');                       %图像顺时针旋转 10 度--旋转变换
subplot(122);imshow(Image1);
Image2=fliplr(I);                                       %对图像做镜像变换---镜像变换
figure;subplot(121);imshow(Image2);
Image3=imresize(I,0.3,'bilinear');                      %图像缩小 1/3---尺寸变换
subplot(122);imshow(Image3);
%调用自定义函数 Moment_Seven()求解图像七阶矩%
display('原图像');
Moment_Seven(Image);
display('旋转变化后的图像');
Moment_Seven(Image1);
display('镜像变化后的图像');
Moment_Seven(Image2);
display('尺度变化后的图像');
Moment_Seven(Image3);
```

%求 7 阶矩函数 Moment_Seven()的函数清单：

```
function Moment_Seven(J)                                %J 为要求解的图像
A=double(J);                                            %将图像数据转换为 double 类型
[m,n]=size(A);                                          %求矩阵 A 的大小
[x,y]=meshgrid(1:n,1:m);              %生成网格采样点的数据，x,y 的行数等于 m，列数等于 n
x=x(:);                                                 %矩阵赋值
y=y(:);
A=A(:);
m00=sum(A);                                             %求矩阵 A 中每列的和，得到 m00 是行向量
if m00==0                                               %如果 m00=0，则赋值 m00=eps，即 m00=0
    m00=eps;
end
m10=sum(x.*A);                                          %以下为 7 阶矩求解过程，参见 7 阶矩的公式
m01=sum(y.*A);
xmean=m10/m00;
ymean=m01/m00;
cm00=m00;
cm02=(sum((y-ymean).^2.*A))/(m00^2);
cm03=(sum((y-ymean).^3.*A))/(m00^2.5);
cm11=(sum((x-xmean).*(y-ymean).*A))/(m00^2);
cm12=(sum((x-xmean).*(y-ymean).^2.*A))/(m00^2.5);
cm20=(sum((x-xmean).^2.*A))/(m00^2);
cm21=(sum((x-xmean).^2.*(y-ymean).*A))/(m00^2.5);
cm30=(sum((x-xmean).^3.*A))/(m00^2.5);
Mon(1)=cm20+cm02;                                       %1 阶矩 Mon(1)
Mon(2)=(cm20-cm02)^2+4*cm11^2;                          %2 阶矩 Mon(2)
Mon(3)=(cm30-3*cm12)^2+(3*cm21-cm03)^2;                 %3 阶矩 Mon(3)
Mon(4)=(cm30+cm12)^2+(cm21+cm03)^2;                     %4 阶矩 Mon(4)
Mon(5)=(cm30-3*cm12)*(cm30+cm12)*((cm30+cm12)^2-3*(cm21+cm03)^2)+(3*(cm30+cm12)^
2-(cm21+cm03)^2);                                       %5 阶矩 Mon(5)
Mon(6)=(cm20-cm02)*((cm30+cm12)^2-(cm21+cm03)^2)+4*cm11*(cm30+cm12)*(cm21+cm03)
;                                                       %6 阶矩 Mon(6)
Mon(7)=(3*cm21-cm03)*(cm30+cm12)*((cm30+cm12)^2-3*(cm21+cm03)^2)+(3*cm12-cm30)*(
cm21+cm03)*(3*(cm30+cm12)^2-(cm21+cm03)^2);     %7 阶矩 Mon(7)
Moment=abs(log(Mon))                                    %采用 log 函数缩小不变矩的动态范围值
```

```
function B = bound2im(b, M, N, x0, y0)
%BOUND2IM Converts a boundary to an image.
%    B = BOUND2IM(b) converts b, an np-by-2 or 2-by-np array representing the integer coordinates of a boundary, into a binary image with 1s in the locations defined by the coordinates in b and 0s elsewhere.
%    B = BOUND2IM(b, M, N) places the boundary approximately centered in an M-by-N image.
```

If any part of the boundary is outside the M-by-N rectangle, an error is issued.

% B = BOUND2IM(b, M, N, X0, Y0) places the boundary in an image of size M-by-N, with the topmost boundary point located at X0 and the leftmost point located at Y0. If the shifted boundary is outside the M-by-N rectangle, an error is issued. XO and X0 must be positive integers.

```
[np, nc] = size(b);
if np < nc
    b = b'; % To convert to size np-by-2.
    [np, nc] = size(b);
end
% Make sure the coordinates are integers.
x = round(b(:, 1));
y = round(b(:, 2));
% Set up the default size parameters.
x = x - min(x) + 1;
y = y - min(y) + 1;
B = false(max(x), max(y));
C = max(x) - min(x) + 1;
D = max(y) - min(y) + 1;
if nargin == 1
    % Use the preceding default values.
elseif nargin == 3
    if C > M | D > N
        error('The boundary is outside the M-by-N region.')
    end
    % The image size will be M-by-N. Set up the parameters for this.
    B = false(M, N);
    % Distribute extra rows approx. even between top and bottom.
    NR = round((M - C)/2);
    NC = round((N - D)/2); % The same for columns.
    x = x + NR; % Offset the boundary to new position.
    y = y + NC;
elseif nargin == 5
    if x0 < 0 | y0 < 0
        error('x0 and y0 must be positive integers.')
    end
    x = x + round(x0) - 1;
    y = y + round(y0) - 1;
    C = C + x0 - 1;
    D = D + y0 - 1;
    if C > M | D > N
        error('The shifted boundary is outside the M-by-N region.')
    end
```

```
    B = false(M, N);
else
    error('Incorrect number of inputs.')
end

B(sub2ind(size(B), x, y)) = true;


function B = boundaries(BW, conn, dir)
%BOUNDARIES Trace object boundaries.
%     B = BOUNDARIES(BW) traces the exterior boundaries of objects in the binary image BW.  B is a P-by-1 cell array, where P is the number of objects in the image. Each cell contains a Q-by-2 matrix, each row of which contains the row and column coordinates of a boundary pixel. Q is the number of boundary pixels for the corresponding object. Object boundaries are traced in the clockwise direction.
%     B = BOUNDARIES(BW, CONN) specifies the connectivity to use when tracing boundaries. CONN may be either 8 or 4.  The default value for CONN is 8.
%     B = BOUNDARIES(BW, CONN, DIR) specifies the direction used for tracing boundaries.  DIR should be either 'cw' (trace boundaries clockwise) or 'ccw' (trace boundaries counterclockwise). If DIR is omitted BOUNDARIES traces in the clockwise direction.

if nargin < 3
    dir = 'cw';
end
if nargin < 2
    conn = 8;
end
L = bwlabel(BW, conn);
% The number of objects is the maximum value of L. Initialize the cell array B so that each cell initially contains a 0-by-2 matrix.
numObjects = max(L(:));
if numObjects > 0
    B = {zeros(0, 2)};
    B = repmat(B, numObjects, 1);
else
    B = {};
end

% Pad label matrix with zeros.  This lets us write the boundary-following loop without worrying about going off the edge of the image.
Lp = padarray(L, [1 1], 0, 'both');

% Compute the linear indexing offsets to take us from a pixel to its neighbors.
M = size(Lp, 1);
```

```
if conn == 8
    % Order is N NE E SE S SW W NW.
    offsets = [-1, M - 1, M, M + 1, 1, -M + 1, -M, -M-1];
else
    % Order is N E S W.
    offsets = [-1, M, 1, -M];
end

% next_search_direction_lut is a lookup table.  Given the direction from pixel k to pixel k+1, what is the direction to start with when examining the neighborhood of pixel k+1?
if conn == 8
    next_search_direction_lut = [8 8 2 2 4 4 6 6];
else
    next_search_direction_lut = [4 1 2 3];
end

% next_direction_lut is a lookup table.    Given that we just looked at
% neighbor in a given direction, which neighbor do we look at next?
if conn == 8
    next_direction_lut = [2 3 4 5 6 7 8 1];
else
    next_direction_lut = [2 3 4 1];
end

% Values used for marking the starting and boundary pixels.
START=-1;
BOUNDARY = -2;

% Initialize scratch space in which to record the boundary pixels as well as follow the boundary.
scratch = zeros(100, 1);

% Find candidate starting locations for boundaries.
[rr, cc] = find((Lp(2:end-1, :) > 0) & (Lp(1:end-2, :) == 0));
rr = rr + 1;

for k = 1:length(rr)
    r = rr(k);
    c = cc(k);
    if (Lp(r,c) > 0) & (Lp(r - 1, c) == 0) & isempty(B{Lp(r, c)})
        % We've found the start of the next boundary. Compute its linear offset, record which boundary it is, mark it, and initialize the counter for the number of boundary pixels.
        idx = (c-1)*size(Lp, 1) + r;
        which = Lp(idx);
```

```
scratch(1) = idx;
Lp(idx) = START;
numPixels = 1;
currentPixel = idx;
initial_departure_direction = [];
done = 0;
next_search_direction = 2;
while ~done
    % Find the next boundary pixel.
    direction = next_search_direction;
    found_next_pixel = 0;
    for k = 1:length(offsets)
        neighbor = currentPixel + offsets(direction);
        if Lp(neighbor) ~= 0
            % Found the next boundary pixel.
            if (Lp(currentPixel) == START) & ...
                    isempty(initial_departure_direction)
                % We are making the initial departure from the starting pixel.
                initial_departure_direction = direction;
            elseif (Lp(currentPixel) == START) & ...
                    (initial_departure_direction == direction)
                % We are about to retrace our path. That means we're done.
                done = 1;
                found_next_pixel = 1;
                break;
            end
            % Take the next step along the boundary.
            next_search_direction = ...
                 next_search_direction_lut(direction);
            found_next_pixel = 1;
            numPixels = numPixels + 1;
            if numPixels > size(scratch, 1)
                % Double the scratch space.
                scratch(2*size(scratch, 1)) = 0;
            end
            scratch(numPixels) = neighbor;

            if Lp(neighbor) ~= START
                Lp(neighbor) = BOUNDARY;
            end
            currentPixel = neighbor;
            break;
        end
```

```
            direction = next_direction_lut(direction);
        end

        if ~found_next_pixel
            % If there is no next neighbor, the object must just have a single pixel.
            numPixels = 2;
            scratch(2) = scratch(1);
            done = 1;
        end
    end
    % Convert linear indices to row-column coordinates and save in the output cell array.
    [row, col] = ind2sub(size(Lp), scratch(1:numPixels));
    B{which} = [row - 1, col - 1];
  end
end

if strcmp(dir, 'ccw')
  for k = 1:length(B)
    B{k} = B{k}(end:-1:1, :);
  end
end
```

```
function c = connectpoly(x, y)
%CONNECTPOLY Connects vertices of a polygon.
%     C = CONNECTPOLY(X, Y) connects the points with coordinates given in X and Y with straight lines. These points are assumed to be a sequence of polygon vertices organized in the clockwise or counterclockwise direction. The output, C, is the set of points along the boundary of the polygon in the form of an nr-by-2 coordinate sequence in the same direction as the input. The last point in the sequence is equal to the first.

v = [x(:), y(:)];
% Close polygon.
if ~isequal(v(end, :), v(1, :))
  v(end + 1, :) = v(1, :);
end
% Connect vertices.
segments = cell(1, length(v) - 1);
for I = 2:length(v)
  [x, y] = intline(v(I - 1, 1), v(I, 1), v(I - 1, 2), v(I, 2));
  segments{I - 1} = [x, y];
end
```

```
c = cat(1, segments{:});
```

```
function    c = fchcode(b, conn, dir)
%FCHCODE Computes the Freeman chain code of a boundary.
%    C = FCHCODE(B) computes the 8-connected Freeman chain code of set of 2-D coordinate pairs contained in B, an np-by-2 array. is a structure with the following fields:
%    c.fcc =Freeman chain code (1-by-np)
%    c.diff=First difference of code c.fcc (1-by-np)
%    c.mm= Integer of minimum magnitude from c.fcc (1-by-np)
%    c.diffmm = First difference of code c.mm (1-by-np)
%    c.x0y0 = Coordinates where the code starts (1-by-2)
%
%    C = FCHCODE(B, CONN) produces the same outputs as above, but with the code connectivity specified in CONN. CONN can be 8 for an 8-connected chain code, or CONN can be 4 for a 4-connected chain code. Specifying CONN=4 is valid only if the input sequence, B, contains transitions with values 0, 2, 4, and 6, exclusively.
%
%    C = FHCODE(B, CONN, DIR) produces the same outputs as above, but, in addition, the desired code direction is specified. Values for DIR can be:
%
%     'same'     Same as the order of the sequence of points in b.This is the default.
%     'reverse'   Outputs the code in the direction opposite to the direction of the points in B. The starting point for each DIR is the same.
%
%    The elements of B are assumed to correspond to a 1-pixel-thick, fully-connected, closed boundary. B cannot contain duplicate coordinate pairs, except in the first and last positions, which is a common feature of boundary tracing programs.
%
%    FREEMAN CHAIN CODE REPRESENTATION
%    The table on the left shows the 8-connected Freeman chain codes corresponding to allowed deltax, deltay pairs. An 8-chain is converted to a 4-chain if (1) if conn = 4; and (2) only transitions 0, 2, 4, and 6 occur in the 8-code.    Note that dividing 0, 2, 4, and 6 by 2 produce the 4-code.
%
%          ----------------------   ----------------
%          deltax | deltay | 8-code    corresp 4-code
%          ----------------------   ----------------
%             0          1          0                  0
%            -1          1          1
%            -1          0          2                  1
%            -1         -1          3
%             0         -1          4                  2
%             1         -1          5
```

```
%              1           0          6                 3
%              1           1          7
%           ----------------------    ----------------
%
```

% The formula z = 4*(deltax + 2) + (deltay + 2) gives the following sequence corresponding to rows 1-8 in the preceding table: z = 11,7,6,5,9,13,14,15. These values can be used as indices into the table, improving the speed of computing the chain code. The preceding formula is not unique, but it is based on the smallest integers (4 and 2) that are powers of 2.

```
% Preliminaries.
if nargin == 1
    dir = 'same';
    conn = 8;
elseif nargin == 2
    dir = 'same';
elseif nargin == 3
    % Nothing to do here.
else
    error('Incorrect number of inputs.')
end
[np, nc] = size(b);
if np < nc
    error('B must be of size np-by-2.');
end
```

% Some boundary tracing programs, such as boundaries.m, output a sequence in which the coordinates of the first and last points are the same. If this is the case, eliminate the last point.

```
if isequal(b(1, :), b(np, :))
    np = np - 1;
    b = b(1:np, :);
end
```

% Build the code table using the single indices from the formula for z given above:

```
C(11)=0; C(7)=1; C(6)=2; C(5)=3; C(9)=4;
C(13)=5; C(14)=6; C(15)=7;
```

```
% End of Preliminaries.
```

```
% Begin processing.
x0 = b(1, 1);
y0 = b(1, 2);
c.x0y0 = [x0, y0];
```

% Make sure the coordinates are organized sequentially:

```
% Get the deltax and deltay between successive points in b. The last row of a is the first row of b.
a = circshift(b, [-1, 0]);

% DEL = a - b is an nr-by-2 matrix in which the rows contain the deltax and deltay between
successive points in b. The two components in the kth row of matrix DEL are deltax and deltay
between point (xk, yk) and (xk+1, yk+1).  The last row of DEL contains the deltax and deltay
between (xnr, ynr) and (x1, y1), (i.e., between the last and first points in b).
DEL = a - b;

% If the abs value of either (or both) components of a pair (deltax, deltay) is greater than 1, then
by definition the curve is broken (or the points are out of order), and the program terminates.
if any(abs(DEL(:, 1)) > 1) | any(abs(DEL(:, 2)) > 1);
    error('The input curve is broken or points are out of order.')
end

% Create a single index vector using the formula described above.
z = 4*(DEL(:, 1) + 2) + (DEL(:, 2) + 2);

% Use the index to map into the table. The following are the Freeman 8-chain codes, organized in
a 1-by-np array.
fcc = C(z);

% Check if direction of code sequence needs to be reversed.
if strcmp(dir, 'reverse')
    fcc = coderev(fcc); % See below for function coderev.
end

% If 4-connectivity is specified, check that all components of fcc are 0, 2, 4, or 6.
if conn == 4
    val = find(fcc == 1 | fcc == 3 | fcc == 5 | fcc ==7 );
    if isempty(val)
        fcc = fcc./2;
    else
        warning('The specified 4-connected code cannot be satisfied.')
    end
end

% Freeman chain code for structure output.
c.fcc = fcc;

% Obtain the first difference of fcc.
c.diff = codediff(fcc,conn);                    % See below for function codediff.

% Obtain code of the integer of minimum magnitude.
```

```
c.mm = minmag(fcc); % See below for function minmag.

% Obtain the first difference of fcc
c.diffmm = codediff(c.mm, conn);

%-------------------------------------------------------------------%
function cr = coderev(fcc)
%     Traverses the sequence of 8-connected Freeman chain code fcc in the opposite direction, changing the values of each code segment. The starting point is not changed. fcc is a 1-by-np array.

% Flip the array left to right. This redefines the starting point as the last point and reverses the order of "travel" through the code.
cr = fliplr(fcc);

% Next, obtain the new code values by traversing the code in the opposite direction. (0 becomes 4, 1 becomes 5, ... , 5 becomes 1, 6 becomes 2, and 7 becomes 3).
ind1 = find(0 <= cr & cr <= 3);
ind2 = find(4 <= cr & cr <= 7);
cr(ind1) = cr(ind1) + 4;
cr(ind2) = cr(ind2) - 4;

%-------------------------------------------------------------------%
function z = minmag(c)
%MINMAG Finds the integer of minimum magnitude in a chain code.
%     Z = MINMAG(C) finds the integer of minimum magnitude in a given 4- or 8-connected Freeman chain code, C. The code is assumed to be a 1-by-np array.

% The integer of minimum magnitude starts with min(c), but there may be more than one such value. Find them all,
I = find(c == min(c));
% and shift each one left so that it starts with min(c).
J = 0;
A = zeros(length(I), length(c));
for k = I;
    J = J + 1;
    A(J, :) = circshift(c,[0 -(k-1)]);
end

% Matrix A contains all the possible candidates for the integer of minimum magnitude. Starting with the 2nd column, succesively find the minima in each column of A. The number of candidates decreases as the seach moves to the right on A.  This is reflected in the elements of J.  When length(J)=1, one candidate remains. This is the integer of minimum magnitude.
[M, N] = size(A);
```

```
J = (1:M)';
for k = 2:N
    D(1:M, 1) = Inf;
    D(J, 1) = A(J, k);
    amin = min(A(J, k));
    J = find(D(:, 1) == amin);
    if length(J)==1
        z = A(J, :);
        return
    end
end

%-------------------------------------------------------------------%
function d = codediff(fcc, conn)
%CODEDIFF Computes the first difference of a chain code.
%    D=CODEDIFF(FCC) computes the first difference of code, FCC. The code FCC is treated as a circular sequence, so the last element of D is the difference between the last and first elements of FCC.The input code is a 1-by-np vector.
%
%    The first difference is found by counting the number of direction changes (in a counter-clockwise direction) that separate two adjacent elements of the code.

sr = circshift(fcc, [0, -1]);                    % Shift input left by 1 location.
delta = sr - fcc;
d = delta;
I = find(delta < 0);

type = conn;
switch type
case 4                                           % Code is 4-connected
    d(I) = d(I) + 4;
case 8                                           % Code is 8-connected
    d(I) = d(I) + 8;
end
```

%garborfilter()

定义，I 为输入图像；Sx、Sy 是变量在 x、y 轴变化的范围，即选定的 gabor 小波窗口的大小；f 为正弦函数的频率；theta 为 gabor 滤波器的方向。G 为 gabor 滤波函数 g(x,y)；gabout 为 gabor 滤波后的图像。

```
%二维 gabor 滤波函数:
%                                    -1        xp ^        yp    ^
% G(x,y,theta,f) =    exp ([----{(----) 2+(----) 2}])*cos(2*pi*f*xp);
```

```
%                                    2      Sx          Sy
% xp = x*cos(theta)+y*sin(theta);
% p = y*cos(theta)-x*sin(theta);

function [G,gabout] = gaborfilter(I,Sx,Sy,f,theta);
if isa(I,'double')~=1                                    %判断输入图像数据是否为double类型
    I = double(I);                                       %若不是将 I 变为 double 类型
end
for x = -fix(Sx):fix(Sx)                                 %选定窗口大小
    for y = -fix(Sy):fix(Sy)                             %求 G
        xp = x * cos(theta) + y * sin(theta);
        yp = y * cos(theta) - x * sin(theta);
        G(fix(Sx)+x+1,fix(Sy)+y+1) = exp(-.5*((xp/Sx)^2+(yp/Sy)^2))*cos(2*pi*f*xp);
    end
end
Imgabout = conv2(I,double(imag(G)),'same');              %对图像虚部做二维卷积
Regabout = conv2(I,double(real(G)),'same');              %对图像数据实部做二维卷积
gabout = sqrt(Imgabout.*Imgabout + Regabout.*Regabout);   %gabor 小波变换后的图像 gabout



function [x, y] = intline(x1, x2, y1, y2)
%INTLINE Integer-coordinate line drawing algorithm.
%   [X, Y] = INTLINE(X1, X2, Y1, Y2) computes an approximation to the line segment joining (X1, Y1) and (X2, Y2) with integer coordinates.  X1, X2, Y1, and Y2 should be integers.INTLINE is reversible; that is, INTLINE(X1, X2, Y1, Y2) produces the same results as FLIPUD(INTLINE(X2, X1, Y2, Y1)).

dx = abs(x2 - x1);
dy = abs(y2 - y1);

% Check for degenerate case.
if ((dx == 0) & (dy == 0))
    x = x1;
    y = y1;
  return;
end
flip = 0;
if (dx >= dy)
    if (x1 > x2)
        % Always "draw" from left to right.
        t = x1; x1 = x2; x2 = t;
        t = y1; y1 = y2; y2 = t;
        flip = 1;
```

```
    end
    m = (y2 - y1)/(x2 - x1);
    x = (x1:x2).';
    y = round(y1 + m*(x - x1));
else
    if (y1 > y2)
        % Always "draw" from bottom to top.
        t = x1; x1 = x2; x2 = t;
        t = y1; y1 = y2; y2 = t;
        flip = 1;
    end
    m = (x2 - x1)/(y2 - y1);
    y = (y1:y2).';
    x = round(x1 + m*(y - y1));
end

if (flip)
    x = flipud(x);
    y = flipud(y);
end
```

```
function [x, y] = minperpoly(B, cellsize)
%MINPERPOLY Computes the minimum perimeter polygon.
%     [X, Y] = MINPERPOLY(F, CELLSIZE) computes the vertices in [X, Y] of the minimum perimeter polygon of a single binary region or boundary in image B. The procedure is based on Slansky's shrinking rubber band approach. Parameter CELLSIZE determines the size of the square cells that enclose the boundary of the region in B. CELLSIZE must be a nonzero integer greater than 1.
%     The algorithm is applicable only to boundaries that are not self-intersecting and that do not have one-pixel-thic    protrusions.

if cellsize <= 1
    error('CELLSIZE must be an integer > 1.');
end

% Fill B in case the input was provided as a boundary. Later the boundary will be extracted with 4-connectivity, which is required by the algorithm. The use of bwperim assures that 4-connectivity is preserved at this point.
B = imfill(B, 'holes');
B = bwperim(B);
[M, N] = size(B);

% Increase image size so that the image is of size K-by-K with (a) K >= max(M,N) and (b) K/cellsize
```

```
= a power of 2.

K = nextpow2(max(M, N)/cellsize);
K = (2^K)*cellsize;

% Increase image size to nearest integer power of 2, by appending zeros to the end of the image. This will allow quadtree  decompositions as small as cells of size 2-by-2,which is the smallest allowed value of cellsize.
M = K - M;
N = K - N;
B = padarray(B, [M N], 'post');                    % f is now of size K-by-K

% Quadtree decomposition.
Q = qtdecomp(B, 0, cellsize);

% Get all the subimages of size cellsize-by-cellsize.
[vals, r, c] = qtgetblk(B, Q, cellsize);

% Get all the subimages that contain at least one black pixel. These are the cells of the wall enclosing the boundary.
I = find(sum(sum(vals(:, :, :)) >= 1));
x = r(I);
y = c(I);

% [x', y'] is a length(I)-by-2 array.  Each member of this array is the left, top corner of a black cell of size cellsize-by-cellsize. Fill the cells with black to form a closed border of black cells around interior points. These cells are the cellular complex.
for k = 1:length(I)
    B(x(k):x(k) + cellsize-1, y(k):y(k) + cellsize-1) = 1;
end

BF = imfill(B, 'holes');

% Extract the points interior to the black border. This is the region of interest around which the MPP will be found.
B = BF & (~B);

% Extract the 4-connected boundary.
B = boundaries(B, 4, 'cw');
% Find the largest one in case of parasitic regions.
J = cellfun('length', B);
I = find(J == max(J));
B = B{I(1)};
```

```
% Function boundaries outputs the last coordinate pair equal to the first. Delete it.
B = B(1:end-1,:);

% Obtain the xy coordinates of the boundary.
x = B(:, 1);
y = B(:, 2);

% Find the smallest x-coordinate and corresponding smallest y-coordinate.
cx = find(x == min(x));
cy = find(y == min(y(cx)));

% The cell with top leftmost corner at (x1, y1) below is the first point considered by the algorithm.
The remaining points are visited in the clockwise direction starting at (x1, y1).
x1 = x(cx(1));
y1 = y(cy(1));

% Scroll data so that the first point is (x1, y1).
I = find(x == x1 & y == y1);
x = circshift(x, [-(I - 1), 0]);
y = circshift(y, [-(I - 1), 0]);

% The same shift applies to B.
B = circshift(B, [-(I - 1), 0]);

% Get the Freeman chain code.  The first row of B is the required starting point. The first
element of the code is the transition between the 1st and 2nd element of B, the second element
of  the code is the transition between the 2nd and 3rd elements of B, and so on.  The last
element of the code is the transition between the last and 1st elements of B. The elements of B
form a cw sequence (see above), so we use 'same' for the direction in function fchcode.

code = fchcode(B, 4, 'same');
code = code.fcc;

% Follow the code sequence to extract the Black Dots, BD, (convex corners) and White Dots, WD,
(concave corners). The transitions are as follows: 0-to-1=WD; 0-to-3=BD; 1-to-0=BD; 1-to-2=WD;
2-to-1=BD;  2-to-3=WD;  3-to-0=WD;  3-to-2=dot.  The formula t = 2*first - second gives the
following unique values for these transitions: -1, -3, 2, 0, 3, 1, 6, 4.  These are applicable to
travel in the cw direction. The WD's are displaced one-half a diagonal from the BD's to form the
half-cell expansion required in the algorithm.

% Vertices will be computed as array "vertices" of dimension nv-by-3, where nv is the number of
vertices. The first two elements of any row of array vertices are the (x,y) coordinates of the vertex
% corresponding to that row, and the third element is 1 if the vertex is convex (BD) or 2 if it is
concave (WD). The first vertex is known to be convex, so it is black.
```

```
vertices = [x1, y1, 1];
n = 1;
k = 1;
for k = 2:length(code)
    if code(k - 1) ~= code(k)
        n = n + 1;
        t = 2*code(k-1) - code(k);                          % t = value of formula.
        if t == -3 | t == 2 | t == 3 | t == 4               % Convex: Black Dots.
            vertices(n, 1:3) = [x(k), y(k), 1];
        elseif t == -1 | t == 0 | t == 1 | t == 6           % Concave: White Dots.
            if t == -1
                vertices(n, 1:3) = [x(k) - cellsize, y(k) - cellsize,2];
            elseif t==0
                vertices(n, 1:3) = [x(k) + cellsize, y(k) - cellsize,2];
            elseif t==1
                vertices(n, 1:3) = [x(k) + cellsize, y(k) + cellsize,2];
            else
                vertices(n, 1:3) = [x(k) - cellsize, y(k) + cellsize,2];
            end
        else
            % Nothing to do here.
        end
    end
end

% The rest of minperpoly.m processes the vertices to arrive at the MPP.

flag = 1;
while flag
    % Determine which vertices lie on or inside the polygon whose vertices are the Black Dots.
Delete all other points.
    I = find(vertices(:, 3) == 1);
    xv = vertices(I, 1);                                    % Coordinates of the Black Dots.
    yv = vertices(I, 2);
    X = vertices(:, 1);                                     % Coordinates of all vertices.
    Y = vertices(:, 2);
    IN = inpolygon(X, Y, xv, yv);
    I = find(IN ~= 0);
    vertices = vertices(I, :);

    % Now check for any Black Dots that may have been turned into concave vertices after the
previous deletion step. Delete any such Black Dots and recompute the polygon as in the previous
section of code. When no more changes occur, set flag to 0, which causes the loop to terminate.
    x = vertices(:, 1);
```

```
    y = vertices(:, 2);
    angles = polyangles(x, y);                          % Find all the interior angles.
    I = find(angles > 180 & vertices(:, 3) == 1);
    if isempty(I)
        flag = 0;
    else
        J = 1:length(vertices);
        for k = 1:length(I)
            K = find(J ~= I(k));
            J = J(K);
        end
        vertices = vertices(J, :);
    end
end

% Final pass to delete the vertices with angles of 180 degrees.
x = vertices(:, 1);
y = vertices(:, 2);
angles = polyangles(x, y);
I = find(angles ~= 180);

% Vertices of the MPP:
x = vertices(I, 1);
y = vertices(I, 2);



function angles = polyangles(x, y)
%POLYANGLES Computes internal polygon angles.
%    ANGLES = POLYANGLES(X, Y) computes the interior angles (in degrees) of an arbitrary
polygon whose vertices are given in [X, Y], ordered in a clockwise manner.  The program
eliminates duplicate adjacent rows in [X Y], except that the first row may equal the last, so that
the polygon is closed.

% Preliminaries.
[x y] = dupgone(x, y);                                  % Eliminate duplicate vertices.
xy = [x(:) y(:)];
if isempty(xy)
    % No vertices!
    angles = zeros(0, 1);
    return;
end
if size(xy, 1) == 1 | ~isequal(xy(1, :), xy(end, :))
    % Close the polygon
```

```
    xy(end + 1, :) = xy(1, :);
end

% Precompute some quantities.
d = diff(xy, 1);
v1 = -d(1:end, :);
v2 = [d(2:end, :); d(1, :)];
v1_dot_v2 = sum(v1 .* v2, 2);
mag_v1 = sqrt(sum(v1.^2, 2));
mag_v2 = sqrt(sum(v2.^2, 2));

% Protect against nearly duplicate vertices; output angle will be 90 degrees for such cases. The "real" further protects against possible small imaginary angle components in those cases.
mag_v1(~mag_v1) = eps;
mag_v2(~mag_v2) = eps;
angles = real(acos(v1_dot_v2 ./ mag_v1 ./ mag_v2) * 180 / pi);

% The first angle computed was for the second vertex, and the last was for the first vertex. Scroll one position down to make the last vertex be the first.
angles = circshift(angles, [1, 0]);

% Now determine if any vertices are concave and adjust the angles accordingly.
sgn = convex_angle_test(xy);

% Any element of sgn that's -1 indicates that the angle is concave. The corresponding angles have to be subtracted from 360.
I = find(sgn == -1);
angles(I) = 360 - angles(I);
%-------------------------------------------------------------------%
function sgn = convex_angle_test(xy)
%     The rows of array xy are ordered vertices of a polygon. If the kth angle is convex (>0 and <= 180 degress) then sgn(k) = 1. Otherwise sgn(k) = -1. This function assumes that the first vertex in the list is convex, and that no other vertex has a smaller value of x-coordinate. These two conditions are true in the first vertex generated by the MPP algorithm. Also the vertices are assumed to be ordered in a clockwise sequence, and there can be no duplicate vertices.
%     The test is based on the fact that every convex vertex is on the positive side of the line passing through the two vertices immediately following each vertex being considered.  If a vertex is concave then it lies on the negative side of the line joining the next two vertices. This property is true also if positive and negative are interchanged in the preceding two sentences.

% It is assumed that the polygon is closed.    If not, close it.
if size(xy, 1) == 1 | ~isequal(xy(1, :), xy(end, :))
    xy(end + 1, :) = xy(1, :);
end
```

% Sign convention: sgn = 1 for convex vertices (i.e, interior angle > 0 and <= 180 degrees), sgn = -1 for concave vertices.

% Extreme points to be used in the following loop. A 1 is appended to perform the inner (dot) product with w, which is 1-by-3 (see below).

```
L = 10^25;
top_left = [-L, -L, 1];
top_right = [-L, L, 1];
bottom_left = [L, -L, 1];
bottom_right = [L, L, 1];

sgn = 1;                                    % The first vertex is known to be convex.

% Start following the vertices.
for k = 2:length(xy) - 1
    pfirst= xy(k - 1, :);
    psecond = xy(k, :);                     % This is the point tested for convexity.
    pthird = xy(k + 1, :);
    % Get the coefficients of the line (polygon edge) passing through pfirst and psecond.
    w = polyedge(pfirst, psecond);
```

% Establish the positive side of the line w1x + w2y + w3 = 0.The positive side of the line should be in the right side of the vector (psecond - pfirst). deltax and deltay of this vector give the direction of travel. This establishes which of the extreme points (see above) should be on the + side. If that point is on the negative side of the line, then w is replaced by -w.

```
    deltax = psecond(:, 1) - pfirst(:, 1);
    deltay = psecond(:, 2) - pfirst(:, 2);
    if deltax == 0 & deltay == 0
        error('Data into convexity test is 0 or duplicated.')
    end
    if deltax <= 0    & deltay >= 0          % Bottom_right should be on + side.
        vector_product = dot(w, bottom_right);  % Inner product.
        w = sign(vector_product)*w;
    elseif deltax <= 0 & deltay <= 0         % Top_right should be on + side.
        vector_product = dot(w, top_right);
        w = sign(vector_product)*w;
    elseif deltax >= 0 & deltay <= 0         % Top_left should be on + side.
        vector_product = dot(w, top_left);
        w = sign(vector_product)*w;
    else                                % deltax >= 0 & deltay >= 0, so bottom_left should be on + side.
        vector_product = dot(w, bottom_left);
        w = sign(vector_product)*w;
```

```
    end
    % For the vertex at psecond to be convex, pthird has to be on the positive side of the line.
    sgn(k) = 1;
    if (w(1)*pthird(:, 1) + w(2)*pthird(:, 2) + w(3)) < 0
        sgn(k) = -1;
    end
end
%--------------------------------------------------------------------%
function w = polyedge(p1, p2)
%     Outputs the coefficients of the line passing through p1 and p2. The line is of the form w1x +
w2y + w3 = 0.

x1 = p1(:, 1);    y1 = p1(:, 2);
x2 = p2(:, 1);    y2 = p2(:, 2);
if x1==x2
    w2 = 0;
    w1 = -1/x1;
    w3 = 1;
elseif y1==y2
    w1 = 0;
    w2 = -1/y1;
    w3 = 1;
elseif x1 == y1 & x2 == y2
    w1 = 1;
    w2 = 1;
    w3 = 0;
else
    w1 = (y1 - y2)/(x1*(y2 - y1) - y1*(x2 - x1) + eps);
    w2 = -w1*(x2 - x1)/(y2 - y1);
    w3 = 1;
end
w = [w1, w2, w3];
%--------------------------------------------------------------------%
function [xg, yg] = dupgone(x, y)
% Eliminates duplicate, adjacent rows in [x y], except that the first and last rows can be equal so
that the polygon is closed.
xg = x;
yg = y;
if size(xg, 1) > 2
    I = find((x(1:end-1, :) == x(2:end, :)) & ...
                (y(1:end-1, :) == y(2:end, :)));
    xg(I) = [];
    yg(I) = [];
end
```

```
function angles = polyangles(x, y)
%POLYANGLES Computes internal polygon angles.
%    ANGLES = POLYANGLES(X, Y) computes the interior angles (in degrees) of an arbitrary polygon whose vertices are given in [X, Y], ordered in a clockwise manner.  The program eliminates duplicate adjacent rows in [X Y], except that the first row may equal the last, so that the polygon is closed.

% Preliminaries.
[x y] = dupgone(x, y);                                    % Eliminate duplicate vertices.
xy = [x(:) y(:)];
if isempty(xy)
    % No vertices!
    angles = zeros(0, 1);
    return;
end
if size(xy, 1) == 1 | ~isequal(xy(1, :), xy(end, :))
    % Close the polygon
    xy(end + 1, :) = xy(1, :);
end

% Precompute some quantities.
d = diff(xy, 1);
v1 = -d(1:end, :);
v2 = [d(2:end, :); d(1, :)];
v1_dot_v2 = sum(v1 .* v2, 2);
mag_v1 = sqrt(sum(v1.^2, 2));
mag_v2 = sqrt(sum(v2.^2, 2));

% Protect against nearly duplicate vertices; output angle will be 90 degrees for such cases. The "real" further protects against possible small imaginary angle components in those cases.
mag_v1(~mag_v1) = eps;
mag_v2(~mag_v2) = eps;
angles = real(acos(v1_dot_v2 ./ mag_v1 ./ mag_v2) * 180 / pi);

% The first angle computed was for the second vertex, and the last was for the first vertex. Scroll one position down to make the last vertex be the first.
angles = circshift(angles, [1, 0]);

% Now determine if any vertices are concave and adjust the angles accordingly.
sgn = convex_angle_test(xy);
```

```
% Any element of sgn that's -1 indicates that the angle is concave. The corresponding angles have to be subtracted  from 360.
I = find(sgn == -1);
angles(I) = 360 - angles(I);
%-----------------------------------------------------------------%
function sgn = convex_angle_test(xy)
%    The rows of array xy are ordered vertices of a polygon. If the kth angle is convex (>0 and <= 180 degress) then sgn(k) = 1. Otherwise sgn(k) = -1. This function assumes that the first vertex in the list is convex, and that no other vertex has a smaller value of x-coordinate. These two conditions are true in the first vertex generated by the MPP algorithm. Also the vertices are assumed to be ordered in a clockwise sequence, and there can be no duplicate vertices.
%    The test is based on the fact that every convex vertex is on the positive side of the line passing through the two vertices immediately following each vertex being considered.  If a vertex is concave then it lies on the negative side of the line joining the next two vertices. This property is true also if positive and negative are interchanged in the preceding two sentences.

% It is assumed that the polygon is closed.  If not, close it.
if size(xy, 1) == 1 | ~isequal(xy(1, :), xy(end, :))
    xy(end + 1, :) = xy(1, :);
end

% Sign convention: sgn = 1 for convex vertices (i.e, interior angle > 0 and <= 180 degrees), sgn = -1 for concave vertices.

% Extreme points to be used in the following loop. A 1 is appended to perform the inner (dot) product with w, which is 1-by-3 (see below).
L = 10^25;
top_left = [-L, -L, 1];
top_right = [-L, L, 1];
bottom_left = [L, -L, 1];
bottom_right = [L, L, 1];

sgn = 1;                                                    % The first vertex is known to be convex.

% Start following the vertices.
for k = 2:length(xy) - 1
    pfirst= xy(k - 1, :);
    psecond = xy(k, :);                                     % This is the point tested for convexity.
    pthird = xy(k + 1, :);
    % Get the coefficients of the line (polygon edge) passing
    % through pfirst and psecond.
    w = polyedge(pfirst, psecond);

    % Establish the positive side of the line w1x + w2y + w3 = 0. The positive side of the line
```

should be in the right side of the vector (psecond - pfirst). deltax and deltay of this vector give the direction of travel. This establishes which of the extreme points (see above) should be on the + side. If that point is on the negative side of the line, then w is replaced by -w.

```
    deltax = psecond(:, 1) - pfirst(:, 1);
    deltay = psecond(:, 2) - pfirst(:, 2);
    if deltax == 0 & deltay == 0
        error('Data into convexity test is 0 or duplicated.')
    end
    if deltax <= 0    & deltay >= 0                    % Bottom_right should be on + side.
        vector_product = dot(w, bottom_right);         % Inner product.
        w = sign(vector_product)*w;
    elseif deltax <= 0 & deltay <= 0                   % Top_right should be on + side.
        vector_product = dot(w, top_right);
        w = sign(vector_product)*w;
    elseif deltax >= 0 & deltay <= 0                   % Top_left should be on + side.
        vector_product = dot(w, top_left);
        w = sign(vector_product)*w;
    else % deltax >= 0 & deltay >= 0, so bottom_left should be on + side.
        vector_product = dot(w, bottom_left);
        w = sign(vector_product)*w;
    end
    % For the vertex at psecond to be convex, pthird has to be on the
    % positive side of the line.
    sgn(k) = 1;
    if (w(1)*pthird(:, 1) + w(2)*pthird(:, 2) + w(3)) < 0
        sgn(k) = -1;
    end
end
%-------------------------------------------------------------------%
function w = polyedge(p1, p2)
%     Outputs the coefficients of the line passing through p1 and p2. The line is of the form w1x +
w2y + w3 = 0.

x1 = p1(:, 1);    y1 = p1(:, 2);
x2 = p2(:, 1);    y2 = p2(:, 2);
if x1==x2
    w2 = 0;
    w1 = -1/x1;
    w3 = 1;
elseif y1==y2
    w1 = 0;
    w2 = -1/y1;
    w3 = 1;
```

```
elseif x1 == y1 & x2 == y2
    w1 = 1;
    w2 = 1;
    w3 = 0;
else
    w1 = (y1 - y2)/(x1*(y2 - y1) - y1*(x2 - x1) + eps);
    w2 = -w1*(x2 - x1)/(y2 - y1);
    w3 = 1;
end
w = [w1, w2, w3];
%--------------------------------------------------------------------%
function [xg, yg] = dupgone(x, y)
% Eliminates duplicate, adjacent rows in [x y], except that the first and last rows can be equal so that the polygon is closed.
xg = x;
yg = y;
if size(xg, 1) > 2
    I = find((x(1:end-1, :) == x(2:end, :)) & ...
                (y(1:end-1, :) == y(2:end, :)));
    xg(I) = [];
    yg(I) = [];
end
```

# 第 12 章　形态学图像处理

```
clear all; close all; clc;
se1=strel('square', 3)
se2=strel('line', 10, 45)
```

```
clear all; close all; clc;
se=strel('diamond', 3)
seq=getsequence(se)
seq(1)
seq(2)
seq(3)
```

```
clear all; close all;
bw=zeros(9,9);
bw(3:5, 4:6)=1
se=strel('square', 3)
bw2=imdilate(bw, se)
figure;
subplot(121);    imshow(bw);
subplot(122);    imshow(bw2);
```

```
clear all; close all;
bw=imread('text.png');
se=strel('line', 11, 90);
bw2=imdilate(bw, se);
figure;
subplot(121);    imshow(bw);
subplot(122);    imshow(bw2);
```

```
clear all; close all;
bw=imread('cameraman.tif');
se=strel('ball', 5, 5);
bw2=imdilate(bw, se);
figure;
subplot(121);    imshow(bw);
subplot(122);    imshow(bw2);
```

```
clear all; close all;
bw=imread('circles.png');
se=strel('disk', 11)
bw2=imerode(bw, se);
figure;
subplot(121);    imshow(bw);
subplot(122);    imshow(bw2);
```

```
clear all; close all;
se=strel('rectangle', [40, 30]);
bw1=imread('circbw.tif');
bw2=imerode(bw1, se);
bw3=imdilate(bw2, se);
figure;
subplot(131);    imshow(bw1);
subplot(132);    imshow(bw2);
subplot(133);    imshow(bw3);
```

```
clear all; close all;
I=imread('snowflakes.png');
se=strel('disk', 5);
J=imopen(I, se);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J, []);
```

```
clear all; close all;
I=imread('circles.png');
se=strel('disk', 10);
J=imclose(I, se);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J, []);
```

```
clear all; close all;
I=imread('rice.png');
se=strel('disk', 11);
J=imtophat(I, se);
K=imadjust(J);
figure;
subplot(131);    imshow(I);
subplot(132);    imshow(J);
subplot(133);    imshow(K);
```

```
clear all; close all;
I=imread('pout.tif');
se=strel('disk', 3);
J=imtophat(I, se);
K=imbothat(I, se);
L=imsubtract(imadd(I, J), K);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(L);

clear all; close all;
I=imread('coins.png');
J=im2bw(I);
K=imfill(J, 'holes');
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(K);

clear all; close all;
I=imread('tire.tif');
J=imfill(I, 'holes');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all; clc;
I=10*ones(6, 10);
I(3:4, 3:4)=13;
I(4:5, 7:8)=18;
I(2,8)=15
bw=imregionalmax(I)

clear all; close all;
I=imread('glass.png');
J=imextendedmax(I, 80);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all; clc;
I=2*ones(5, 10);
I(2:4, 2:4)=3;
I(4:5, 6:7)=9;
I(2,8)=5
J=imregionalmax(I)
K=imhmax(I, 4)

clear all; close all;
I=imread('rice.png');
se=strel('disk', 2);
J=imdilate(I, se);
K=imerode(I, se);
L=J-K;
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(L);

clear all; close all;
I=imread('circbw.tif');
J=bwperim(I, 8);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('text.png');
J=bwmorph(I, 'thin', Inf);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('circbw.tif');
J=bwmorph(I, 'skel', Inf);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('circles.png');
J=bwmorph(I, 'remove');
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('circles.png');
```

```
J=bwulterode(I);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
fun=@(x) (sum(x(:))==4);
lut=makelut(fun, 2);
I=imread('text.png');
J=applylut(I, lut);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all; clc;
BW=zeros(4, 8);
BW(2:3, 2:3)=1;
BW(2, 5)=1;
BW(3, 7)=1
[L, num]=bwlabel(BW, 8)

clear all; close all;
I=imread('rice.png');
J=im2bw(I, graythresh(I));
K=bwlabel(J);
RGB=label2rgb(K);
figure;
subplot(121);    imshow(J);
subplot(122);    imshow(RGB);

clear all; close all;
I=imread('text.png');
c=[43, 185, 212];
r=[38, 68, 181];
J=bwselect(I, c, r, 4);
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all;
I=imread('circbw.tif');
se=strel('disk', 3);
J=imdilate(I, se);
a1=bwarea(I)
a2=bwarea(J)
(a2-a1)/a1
figure;
subplot(121);    imshow(I);
subplot(122);    imshow(J);

clear all; close all; clc;
I=imread('circbw.tif');
J=imread('circles.png');
e1=bweuler(I, 8)
e2=bweuler(J, 8)
```

# 第 13 章　小波在图像处理中的应用

```
close all; clear all;clc;
[X,map]=imread('trees.tif');                          %读入图像
R=map(X+1,1);R=reshape(R,size(X));                    %获取图像 R 信息
G=map(X+1,2);G=reshape(G,size(X));                    %获取图像 G 信息
B=map(X+1,3);B=reshape(B,size(X));                    %获取图像 B 信息
Xrgb=0.2990*R+0.5870*G+0.1140*B;                      %将 RGB 混合成单通道
n=64                                                  %设置灰度级
X1=round(Xrgb*(n-1))+1;                               %将单通道颜色信息，转换成 64 灰度级
map2=gray(n);
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                   %修改图形背景颜色的设置
figure(1),image(X1);
colormap(map2);

close all; clear all;clc;
[phi,psi,xval]=wavefun('sym4',10);                    %得到 sym4 的尺度函数和小波函数
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
subplot(121);plot(xval,phi,'k');                      %显示尺度函数
axis([0 8 -0.5 1.3]);
axis square;
subplot(122); plot(xval,psi,'k');                     %显示小波函数
axis([0 8 -1.5 1.6]);
axis square;
[lo_d,hi_d,lo_r,hi_r]=wfilters('sym4');               %得到 sym4 的相关滤波器
figure                                                %显示相关滤波器
subplot(121);stem(lo_d,'.k');
subplot(122);stem(hi_d,'.k');
figure
subplot(121);stem(lo_r,'.k');
subplot(122);stem(hi_r,'.k');

close all; clear all;clc;
set(0,'defaultFigurePosition',[100,100,1000,500]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
iter = 4;                                             %设置采样点数
wav1 = 'db4';                                         %设置小波
wav2 = 'bior1.3';
[s,w1,w2,w3,xyval] = wavefun2(wav1,iter,'plot');      %计算二维小波并显示
[s1,w11,w21,w31,xyval1] = wavefun2(wav2,iter,'plot');
```

```
close all; clear all;clc;
s1=2^8;                                          %设置分解信号、数值向量、数值矩阵
s2=[2^8 2^7];
s3=[2^9 2^7;2^9 2^7];
w1='db1';                                        %设置分解采用的小波
w2='db2';
disp('一维信号 s1 采用 db1 的最大分解层数 L1')      %计算并显示最大分解层数
L1=wmaxlev(s1,w1)
disp('数值向量 s2 采用 db1 的最大分解层数 L2')
L2=wmaxlev(s2,w1)
disp('数值矩阵 s3 采用 db1 的最大分解层数 L3')
L3=wmaxlev(s3,w1)
disp('数值矩阵 s3 采用 db7 的最大分解层数 L4')
L4=wmaxlev(s3,w2)

close all; clear all;clc;
X=imread('girl.bmp');                            %读取图像
X=rgb2gray(X);                                   %转换图像数据类型
[ca1,chd1,cvd1,cdd1] = dwt2(X,'bior3.7');
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])
figure                                           %显示小波变换各个分量
subplot(141);
imshow(uint8(ca1));
subplot(1,4,2);
imshow(chd1);
subplot(1,4,3);
imshow(cvd1);
subplot(1,4,4);
imshow(cdd1);                                    %显示原图和小波变换分量组合图像
figure
subplot(121),imshow(X);
subplot(122),imshow([ca1,chd1;cvd1,cdd1]);

close all; clear all;clc;
load woman;                                      %读取待处理图像数据.
nbcol = size(map,1);                             %获取颜色映射表的列数
[cA1,cH1,cV1,cD1] = dwt2(X,'db1');          %对图像数据 X 利用 db1 小波，进行单层图像分解
sX = size(X);                                    %获取原图像大小
A0 = idwt2(cA1,cH1,cV1,cD1,'db4',sX);           % 用小波分解的第一层系数进行重构
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])              %修改图形背景颜色的设置
subplot(131),imshow(uint8(X));                   %显示处理结果
```

```
subplot(132),imshow(uint8(A0));
subplot(133),imshow(uint8(X-A0));

close all; clear all;clc;
load woman;                                          %读取图像数据
nbcol=size(map,1);
[c,s]=wavedec2(X,2,'db2');                           %采用 db4 小波进行 2 层图像分解
siz=s(size(s,1),:);                                  %获取原图像矩阵 X 的大小
ca2=appcoef2(c,s,'db2',2);                           %提取多层小波分解结构 C 和 S 的第 1 层小波
                                                       变换的近似系数
chd2=detcoef2('h',c,s,2);                            %利用的多层小波分解结构 C 和 S 来提取图像
                                                       第 1 层的细节系数的水平分量
cvd2=detcoef2('v',c,s,2);                            %利用的多层小波分解结构 C 和 S 来提取图像
                                                       第 1 层的细节系数的垂直分量
cdd2=detcoef2('d',c,s,2);                            %利用的多层小波分解结构 C 和 S 来提取图像
                                                       第 1 层的细节系数的对角分量
chd1=detcoef2('h',c,s,1);                            %利用的多层小波分解结构 C 和 S 来提取图像
                                                       第 1 层的细节系数的水平分量
cvd1=detcoef2('v',c,s,1);                            %利用的多层小波分解结构 C 和 S 来提取图像
                                                       第 1 层的细节系数的垂直分量
cdd1=detcoef2('d',c,s,1);                            %利用的多层小波分解结构 C 和 S 来提取图像
                                                       第 1 层的细节系数的对角分量
ca11=ca2+chd2+cvd2+cdd2;                             %叠加重构近似图像
ca1 = appcoef2(c,s,'db4',1);                         %提取多层小波分解结构 C 和 S 的第 1 层小波
                                                       变换的近似系数
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                  %修改图形背景颜色的设置
figure                                               %显示图像结果
subplot(1,4,1); imshow(uint8(wcodemat(ca2,nbcol)));
subplot(1,4,2); imshow(uint8(wcodemat(chd2,nbcol)));
subplot(1,4,3); imshow(uint8(wcodemat(cvd2,nbcol)));
subplot(1,4,4); imshow(uint8(wcodemat(cdd2,nbcol)));
figure
subplot(1,4,1); imshow(uint8(wcodemat(ca11,nbcol)));
subplot(1,4,2); imshow(uint8(wcodemat(chd1,nbcol)));
subplot(1,4,3); imshow(uint8(wcodemat(cvd1,nbcol)));
subplot(1,4,4); imshow(uint8(wcodemat(cdd1,nbcol)));
disp('小波二层分解的近似系数矩阵 ca2 的大小：')      %显示小波分解系数矩阵的大小
ca2_size=s(1,:)
disp('小波二层分解的细节系数矩阵 cd2 的大小:')
cd2_size=s(2,:)
disp('小波一层分解的细节系数矩阵 cd1 的大小:')
cd1_size=s(3,:)
disp('原图像大小:')
```

```
X_size=s(4,:)
disp('小波分解系数分量矩阵 c 的长度:')
c_size=length(c)

close all; clear all;clc;
load woman;                                          %读取图像数据
[c,s]=wavedec2(X,2,'db2');                           %采用 db4 小波进行 2 层图像分解
nbcol=size(map,1);
s1=s(1,:);                                           %获取小波分解系数矩阵大小
s2=s(3,:);
ca2=zeros(s1);                                       %初始化分解系数矩阵
chd2=zeros(s1);
cvd2=zeros(s1);
cdd2=zeros(s1);
chd1=zeros(s2);
cvd1=zeros(s2);
cdd1=zeros(s2);
l1=s1(1)*s1(1);
l2=s2(1)*s2(1);
%从分解系数矩阵 C 和长度矩阵 S 中提取细节
ca2=reshape(c(1:l1),s1(1),s1(1));                    %提取第 2 层小波变换的近似系数
chd2=reshape(c(l1+1:2*l1),s1(1),s1(1));              %提取图像第 2 层的细节系数的水平分量
cvd2=reshape(c(2*l1+1:3*l1),s1(1),s1(1));            %提取图像第 2 层的细节系数的垂直分量
cdd2=reshape(c(3*l1+1:4*l1),s1(1),s1(1));            %提取图像第 2 层的细节系数的对角分量
chd1=reshape(c(4*l1+1:4*l1+l2),s2(1),s2(1));         %提取图像第 1 层的细节系数的水平分量
cvd1=reshape(c(4*l1+l2+1:4*l1+2*l2),s2(1),s2(1));    %提取图像第 1 层的细节系数的垂直分量
cdd1=reshape(c(4*l1+2*l2+1:4*l1+3*l2),s2(1),s2(1)); %提取图像第 1 层的细节系数的对角分量
%利用函数 appcoef2()和 detcoef2()提取小波分解系数
ca2_1=appcoef2(c,s,'db2',2);                         %提取第 2 层小波变换的近似系数
chd2_1=detcoef2('h',c,s,2);                          %提取图像第 2 层的细节系数的水平分量
cvd2_1=detcoef2('v',c,s,2);                          %提取图像第 2 层的细节系数的垂直分量
cdd2_1=detcoef2('d',c,s,2);                          %提取图像第 2 层的细节系数的对角分量
chd1_1=detcoef2('h',c,s,1);                          %提取图像第 1 层的细节系数的水平分量
cvd1_1=detcoef2('v',c,s,1);                          %提取图像第 1 层的细节系数的垂直分量
cdd1_1=detcoef2('d',c,s,1);                          %提取图像第 1 层的细节系数的对角分量
disp('比较两种方法获取小波分解系数是否相同：')
disp(' ')
if isequal(ca2,ca2_1)
    disp('        ca2 和 ca2_1 相同')
    disp(' ')
end
if isequal(chd2,chd2_1)
    disp('        chd2 和 chd2_1 相同')
    disp(' ')
```

```
end
if isequal(cvd2,cvd2_1)
    disp('        cvd2 和 cvd2_1 相同')
    disp(' ')
end
if isequal(cdd2,cdd2_1)
    disp('        cdd2 和 cdd2_1 相同')
    disp(' ')
end
if isequal(chd1,chd1_1)
    disp('        chd1 和 chd1_1 相同')
    disp(' ')
end
if isequal(cvd1,cvd1_1)
    disp('        cvd1 和 cvd1_1 相同')
    disp(' ')
end
if isequal(cdd1,cdd1_1)
    disp('        cdd1 和 cdd1_1 相同')
    disp(' ')
end

close all; clear all;clc;
X=imread('flower.tif');                          %读取图像进行 灰度转换
X=rgb2gray(X);
[c,s] = wavedec2(X,2,'db4');                     %对图像进行小波 2 层分解
siz = s(size(s,1),:);                            %提取第 2 层小波分解系数矩阵大小
ca2 = appcoef2(c,s,'db4',2);                     %提取第 1 层小波分解的近似系数
chd2 = detcoef2('h',c,s,2);                      %提取第 1 层小波分解的细节系数水平分量
cvd2 = detcoef2('v',c,s,2);                      %提取第 1 层小波分解的细节系数垂直分量
cdd2 = detcoef2('d',c,s,2);                      %提取第 1 层小波分解的细节系数对角分量
a2 = upcoef2('a',ca2,'db4',2,siz);               %利用函数 upcoef2 对提取 2 层小波系数进行重构
hd2 = upcoef2('h',chd2,'db4',2,siz);
vd2 = upcoef2('v',cvd2,'db4',2,siz);
dd2 = upcoef2('d',cdd2,'db4',2,siz);
A1=a2+hd2+vd2+dd2;
[ca1,ch1,cv1,cd1] = dwt2(X,'db4');               %对图像进行小波单层分解
a1 = upcoef2('a',ca1,'db4',1,siz);               %利用函数 upcoef2 对提取 1 层小波分解系数进行重构
hd1 = upcoef2('h',cd1,'db4',1,siz);
vd1 = upcoef2('v',cv1,'db4',1,siz);
dd1 = upcoef2('d',cd1,'db4',1,siz);
A0=a1+hd1+vd1+dd1;
set(0,'defaultFigurePosition',[100,100,1000,500]);   %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                  %修改图形背景颜色的设置
```

```
figure                                          %显示相关滤波器
subplot(141);imshow(uint8(a2));
subplot(142);imshow(hd2);
subplot(143);imshow(vd2);
subplot(144);imshow(dd2);
figure
subplot(141);imshow(uint8(a1));
subplot(142);imshow(hd1);
subplot(143);imshow(vd1);
subplot(144);imshow(dd1);
figure
subplot(131);imshow(X);
subplot(132);imshow(uint8(A1));
subplot(133);imshow(uint8(A0));

close all; clear all;clc;
load gatlin2;                                   %装载并显示原始图像
init=2055615866;                                %生成含噪图象并显示
randn('seed',init)
XX=X+2*randn(size(X));
[c,l]=wavedec2(XX,2,'sym4');        %对图像进行消噪处理,用 sym4 小波函数对 x 进行两层分解
a2=wrcoef2('a',c,l,'sym4',2);                   %重构第二层图像的近似系数
n=[1,2];                                        %设置尺度向量
p=[10.28,24.08];                                %设置阈值向量
nc=wthcoef2('t',c,l,n,p,'s');                   %对高频小波系数进行阈值处理
mc=wthcoef2('t',nc,l,n,p,'s');                  %再次对高频小波系数进行阈值处理
X2=waverec2(mc,l,'sym4');                       %图像的二维小波重构
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])             %修改图形背景颜色的设置
figure                                          %  显示原图像及处理以后结果
colormap(map)
subplot(131),image(XX),axis square;
subplot(132),image(a2),axis square;
subplot(133),image(X2),axis square;
Ps=sum(sum((X-mean(mean(X))).^2));              %计算信噪比
Pn=sum(sum((a2-X).^2));
disp('利用小波 2 层分解去噪的信噪比')
snr1=10*log10(Ps/Pn)
disp('利用小波阈值去噪的信噪比')
Pn1=sum(sum((X2-X).^2));
snr2=10*log10(Ps/Pn1)

close all; clear all;clc;
load flujet;                                    %装载并显示原始图像
```

```
init=2055615866;                                    %生成含噪声图像并显示
XX=X+8*randn(size(X));
n=[1,2];                                            %设置尺度向量
p=[10.28,24.08];                                    %设置阈值向量
[c,l]=wavedec2(XX,2,'db2');                         %用小波函数 db2 对图像 XX 进行 2 层分解
nc=wthcoef2('t',c,l,n,p,'s');                       %对高频小波系数进行阈值处理
mc=wthcoef2('t',nc,l,n,p,'s');                      %再次对高频小波系数进行阈值处理
X2=waverec2(mc,l,'db2');                            %图像的二维小波重构

[c1,l1]=wavedec2(XX,2,'sym4');                      %首先用小波函数 sym4 对图像 XX 进行 2 层分解
nc1=wthcoef2('t',c1,l1,n,p,'s');                    %对高频小波系数进行阈值处理
mc1=wthcoef2('t',nc1,l1,n,p,'s');                   %再次对高频小波系数进行阈值处理
X3=waverec2(mc1,l1,'sym4');                         %图像的二维小波重构
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                 %修改图形背景颜色的设置
figure                                              %显示原图像及处理以后结果
colormap(map)
subplot(121);image(X);axis square;
subplot(122);image(XX);axis square;
figure
colormap(map)
subplot(121);image(X2);axis square;
subplot(122);image(X3);axis square;
Ps=sum(sum((X-mean(mean(X))).^2));                  %计算信噪比
Pn=sum(sum((XX-X).^2));
Pn1=sum(sum((X2-X).^2));
Pn2=sum(sum((X3-X).^2));
disp('未处理的含噪声图像信噪比')
snr=10*log10(Ps/Pn)
disp('采用 db2 进行小波去噪的图像信噪比')
snr1=10*log10(Ps/Pn1)
disp('采用 sym4 进行小波去噪的图像信噪比')
snr2=10*log10(Ps/Pn2)

X=imread('6.bmp');                                  %把原图象转化为灰度图像，装载并显示
X=double(rgb2gray(X));
init=2055615866;                                    %生成含噪图象并显示
randn('seed',init)
X1=X+25*randn(size(X));                             %生成含噪图像并显示
[thr,sorh,keepapp]=ddencmp('den','wv',X1);          %消噪处理：设置函数 wpdencmp 的消噪参数
X2=wdencmp('gbl',X1,'sym4',2,thr,sorh,keepapp);
X3=X;                                               %保存纯净的原图像
for i=2:577;
        for j=2:579
```

```
            Xtemp=0;
             for m=1:3
                   for n=1:3
                         Xtemp=Xtemp+X1((i+m)-2,(j+n)-2);%对图像进行平滑处理以增强消
                                                          噪效果（中值滤波）
                   end
             end
             Xtemp=Xtemp/9;
             X3(i-1,j-1)=Xtemp;
       end
end
set(0,'defaultFigurePosition',[100,100,1000,500]);          %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                          %修改图形背景颜色的设置
figure
subplot(121);imshow(uint8(X)); axis square;                  %画出原图象
subplot(122);imshow(uint8(X1));axis square;                  %画出含噪声图象
figure
subplot(121),imshow(uint8(X2)),axis square;                  %画出消噪后的图像
subplot(122),imshow(uint8(X3)),axis square;                  %显示结果
Ps=sum(sum((X-mean(mean(X))).^2));                           %计算信噪比
Pn=sum(sum((X1-X).^2));
Pn1=sum(sum((X2-X).^2));
Pn2=sum(sum((X3-X).^2));
disp('未处理的含噪声图像信噪比')
snr=10*log10(Ps/Pn)
disp('采用小波全局阈值滤波的去噪图像信噪比')
snr1=10*log10(Ps/Pn1)
disp('采用中值滤波的去噪图像信噪比')
snr2=10*log10(Ps/Pn2)

close all; clear all;clc;
load wmandril;                                               %导入图像数据
nbc=size(map,1);                                             %获取颜色映射阶数
Y=wcodemat(X,nbc);                                           %对图像的数值矩阵进行伪彩色编码
[C,S]=wavedec2(X,2,'db4');                                   %对图像小波分解
thr=20;                                                      %设置阈值
[Xcompress1,cxd,lxd,perf0,perfl2]=wdencmp('gbl',C,S,'db4',2,thr,'h',1);%对图像进行全局压缩
Y1=wcodemat(Xcompress1,nbc);                                 %对图像数据进行伪彩色编码
set(0,'defaultFigurePosition',[100,100,1000,500]);           %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                          %修改图形背景颜色的设置
figure                                                       %创建图形显示窗口
colormap(gray(nbc));                                         %设置映射谱图等级
subplot(121),image(Y),axis square                            %显示
subplot(122);image(Y1),axis square
```

```
disp('小波系数中置 0 的系数个数百分比：')          %输出压缩比率变量
perfl2
disp('压缩后图像剩余能量百分比：')
perf0

load detfingr;                                    %导入图像数据
nbc=size(map,1);
[C,S]=wavedec2(X,2,'db4');                        %图像小波分解
thr_h=[21 46];                                    %设置水平分量阈值
thr_d=[21 46];                                    %设置对角分量阈值
thr_v=[21 46];                                    %设置垂直分量阈值
thr=[thr_h;thr_d;thr_v];
[Xcompress2,cxd,lxd,perf0,perfl2]=wdencmp('lvd',X,'db3',2,thr,'h');%进行分层压缩
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])               %修改图形背景颜色的设置
Y=wcodemat(X,nbc);
Y1=wcodemat(Xcompress2,nbc);
figure                                            %显示原图像和压缩图像
colormap(map)
subplot(121),image(Y),axis square
subplot(122),image(Y1),axis square
figure
%colormap(map)
subplot(121),image(Y),axis square
subplot(122),image(Y1),axis square
disp('小波系数中置 0 的系数个数百分比：')          %显示压缩能量
perfl2
disp('压缩后图像剩余能量百分比：')
perf0

close all; clear all;clc;
load detfingr;                                    %导入图像数据
nbc=size(map,1);                                  %获取颜色映射阶数
[c,s]=wavedec2(X,3,'sym4');              %对图像数据 X 进行 3 层小波分解  采用小波函数 sym4
alpha=1.5;                        %设置参数 alpha 和 m，利用 wdcbm2 设置图像压缩的分层阈值
m=2.7*prod(s(1,:));
[thr,nkeep]=wdcbm2(c,s,alpha,m)
[xd,cxd,sxd,perf0,perfl2] =wdencmp('lvd',c,s,'sym4',3,thr,'h');
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])               %修改图形背景颜色的设置
figure                                            %创建图形显示窗口
colormap(pink(nbc));
subplot(121), image(wcodemat(X,nbc)),
subplot(122), image(wcodemat(xd,nbc)),
```

```
disp('小波系数中置 0 的系数个数百分比：')          %输出压缩比率变量
perfl2
disp('压缩后图像剩余能量百分比：')
perf0

close all; clear all;clc;
load wbarb;                                        %导入图像数据
[C,S] = wavedec2(X,3,'db4');                       %进行小波分解
[thr,sorh,keepapp] = ddencmp('cmp','wv',X)         %返回图像压缩所需要的一些参数
[Xcomp,CXC,LXC,PERF0,PERFL2] =wdencmp('gbl',C,S,'db4',3,thr,sorh,keepapp);
                                                   %按照参数压缩图像，并返回结果
disp('小波系数中置 0 的系数个数百分比：')          %返回压缩比率
PERFL2
disp('压缩后图像剩余能量百分比：')
PERF0
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                 %修改图形背景颜色的设置
figure;                                            %创建图像
colormap(map);
subplot(121); image(X); axis square                %显示压缩结果
subplot(122); image(Xcomp); axis square

close all; clear all;clc;
load woman;                                        %导入图像
X1=X;map1=map;                                     %保存图像数据和映射
load wbarb;                                        %导入图像
X2=X;map2=map;                                     %保存图像数据和映射
[C1,S1]=wavedec2(X1,2,'sym4');                     %图像的小波分解
[C2,S2]=wavedec2(X2,2,'sym4');
C=1.2*C1+0.5*C2;                                   %对图像的小波分解结果进行融合方案 1
C=0.4*C;
C0=0.2*C1+1.5*C2;                                  %对图像的小波分解结果进行融合方案 2
C0=0.5*C;
xx1=waverec2(C,S1,'sym4');                         %对小波分解的结果进行融合处理
xx2=waverec2(C0,S2,'sym4');
set(0,'defaultFigurePosition',[100,100,1000,500]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                 %修改图形背景颜色的设置
figure                                             %创建图形显示窗口
colormap(map2),
subplot(121),image(X1)                             %显示原图像和融合结果
subplot(122),image(X2)
figure                                             %创建图形显示窗口
colormap(map2),
subplot(121),image(xx1);
```

```
subplot(122),image(xx2);

close all; clear all;clc;
X1 = imread('girl.bmp');                                    % 载入原始两幅图像
X2 = imread('lenna.bmp');
FUSmean = wfusimg(X1,X2,'db2',5,'mean','mean');      %通过函数 wfusing 实现两种图像融合
FUSmaxmin = wfusimg(X1,X2,'db2',5,'max','min');
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                      %修改图形背景颜色的设置
figure                                                   %创建图形显示窗口
subplot(121), imshow(uint8(FUSmean))
subplot(122), imshow(uint8(FUSmaxmin))

close all; clear all;clc;
load mask;                                               %载入图像和数据
A=X;
load bust;
B=X;
Fus_Method = struct('name','userDEF','param','myfus_FUN');      %定义融合规则和调用函数名
C=wfusmat(A,B,Fus_Method);                               %设置图像融合方法
set(0,'defaultFigurePosition',[100,100,1000,500]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                      %修改图形背景颜色的设置
figure                                                   %创建图形显示窗口
subplot(1,3,1), imshow(uint8(A)),                        %显示结果
subplot(1,3,2), imshow(uint8(B)),
subplot(1,3,3), imshow(uint8(C)),

function C = myfus_FUN(A,B)
%定义融合规则
D = logical(tril(ones(size(A))));                        %提取矩阵的下三角部分
t = 0.8;                                                  %设置融合比例
C = B;                                                    %设置融合图像初值为 B
C(D)    = t*A(D)+(1-t)*B(D);                              %融合后的图像 C 的下三角融合规则
C(~D) = t*B(~D)+(1-t)*A(~D);                              %融合后的图像 Dd 的上三角融合规则
```

# 第 14 章　基于 SIMULINK 的视频和图像处理

```
I=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\pout.tif ');
                                                    %读入并显示原始图像
I=double(I);                                        %图像数据类型转换
[M,N]=size(I);
for i=1:M                                           %进行现行灰度变换
for j=1:N
            if I(i,j)<=30
I(i,j)=I(i,j);
              elseif I(i,j)<=150
                 I(i,j)=(200-30)/(150-30)*(I(i,j)-30)+30;
              else
                 I(i,j)=(255-200)/(255-150)*(I(i,j)-150)+200;
            end
end
end
figure,imshow(uint8(I));                            %显示变换后的结果


A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\eight.tif');
                                                    %读取图像
B=imnoise(A,'salt & pepper',0.02);                  %添加椒盐噪声
K=medfilt2(B);                                      %中值滤波
figure %显示
subplot(121),imshow(B);                             %显示添加椒盐噪声后的图像
subplot(122),imshow(K);                             %显示平滑处理后图像


A=imread('D:\ProgramFiles\MATLAB\R2010a\toolbox\images\imdemos\fuwa.jpg');
                                                    %读入并显示图像
B=fspecial('Sobel');                                %用 Sobel 算子进行边缘锐化
fspecial('Sobel');
B=B';                                               %Sobel 垂直模板
C=filter2(B,A);
figure,imshow(C);                                   %显示图像


A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\round.jpg');
                                                    %读入并显示原始图像
B=im2bw(A);                                         %转换为二值图像
figure,imshow(B);                                   %显示变换后的结果
```

```
A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\pears.png');
                                                    %读入并显示原始图像
figure(1),imshow(A);
HSV=rgb2hsv(A);
figure(2),imshow(HSV);                              %显示变换后的结果
```

```
A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\liftingbody.png');
                                                    %读入并显示原始图像
B=double(A);
B=256-1-B;
B=uint8(B);                                         %图像数据类型转换
figure,imshow(B);                                   %显示变换后的结果
```

```
A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\autumn.tif');
                                                    %读取并显示图像
B=imrotate(A,90,'nearest');                         %将图像旋转九十度
figure,imshow(B)                                    %显示旋转后图像
```

```
A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\kids.tif');
                                                    %读取并显示图像
B=imresize(A,0.5,'nearest');                        %缩小图像至原始图像的 50%
figure(1)
imshow(A);                                          %显示原始图像
figure(2)
imshow(B);                                          %显示缩小后的图像
```

```
A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\office_4.jpg');
                                                    %读取并显示图像
B=size(A);                                          %图像切变
C=zeros(B(1)+round(B(2)*tan(pi/6)),B(2),B(3));
for m=1:B(1)
        for n=1:B(2)
            C(m+round(n*tan(pi/6)),n,1:B(3))=A(m,n,1:B(3));
end
end
figure,imshow(uint8(C));                            %显示切变后图像
```

```
A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\cameraman.tif');
                                                    %读取并显示图像
SE=strel('disk',4,4);                               %定义模板
B=imdilate(A,SE);                                   %按模板膨胀
C=imerode(A,SE);                                    %按模板腐蚀
figure
subplot(121),imshow(B);
subplot(122),imshow(C);


A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\tire.tif');
                                                    %读取图像
B=imnoise(A,'salt & pepper');                       %添加椒盐噪声
SE = strel('disk',2);
C= imopen(B,SE);                                    %对图像进行开启操作
D= imclose(C,SE);                                   %对图像进行闭合操作
figure
subplot(131),imshow(B);                             %显示添加椒盐噪声后图像
subplot(132),imshow(C);                             %显示开运算后图像
subplot(133),imshow(D);                             %显示闭运算后图像


A=imread('D:\Program Files\MATLAB\R2010a\toolbox\images\imdemos\coins.png');
                                                    %读取图像
B=im2bw(A);                                         %转换成二值图像
SE=strel('disk',5);
C=imopen(B,SE);                                     %对图像进行开启操作
figure
subplot(121),imshow(B);                             %显示二值图像
subplot(122),imshow(C);                             %显示开运算后图像
```

# 第 15 章　图像处理的 MATLAB 实例

```
close all; clear all;clc;
N1=256;                                          %输入头模型大小
[fp1,axes_x1,axes_y1,pixel1]=headata(N1);        %调用函数 headata 产生头模型数据
set(0,'defaultFigurePosition',[100,100,1200,450]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])              %修改图形背景颜色的设置
figure,                                          %显示 256*256 头模型
subplot(121),
 for i=1:N1,
     for j=1:N1,
            a=fscanf(fp1,'%d    %d    %f\n',[1 3]);
            plot(axes_x1(i,j),axes_y1(i,j),'color',[0.5*a(3) 0.5*a(3) 0.5*a(3)],...
                                              'MarkerSize',20);
            hold on;
     end
 end
fclose(fp1);
N2=512;                                          %输入头模型大小
[fp2,axes_x2,axes_y2,pixel2]=headata(N2);        %调函数 headata 产生头模型数据
 subplot(122),                                   %显示 512*512 头模型
 for i=1:N2,
     for j=1:N2,
            a=fscanf(fp2,'%d    %d    %f\n',[1 3]);
            plot(axes_x2(i,j),axes_y2(i,j),'color',[0.5*a(3) 0.5*a(3) 0.5*a(3)],...
                                              'MarkerSize',20);
            hold on;
     end
 end
 fclose(fp2);

close all; clear all;clc;
proj1=90,N1=128;                                 %输入投影数据大小
degree1=projdata(proj1,N1);                      %调用函数 projdata 产生头模型的投影数据
proj2=180,N2=256;                                %输入投影数据大小
degree2=projdata(proj2,N2);                      %调用函数 projdata 产生头模型的投影数据
set(0,'defaultFigurePosition',[100,100,1200,450]);  %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])              %修改图形背景颜色的设置
figure,
subplot(121),pcolor(degree1)                     %显示 180*128 头模型
subplot(122),pcolor(degree2)                     %显示 180*256 头模型
```

```
close all; clear all;clc;
N=64;                                          %定义量化值 N
m=15;
L=2.0;
[x,h]=RLfilter(N,L)
x1=x(N-m:N+m);
h1=h(N-m:N+m);
set(0,'defaultFigurePosition',[100,100,1200,450]);    %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                   %修改图形背景颜色的设置
figure,
subplot(121),
plot(x,h),axis tight,grid on                          %显示波形
subplot(122),
plot(x1,h1),axis tight,grid on                        %显示波形

close all; clear all;clc;
N=128;                                         %定义图像大小，椭圆个数，投影数
NUM=10;
PROJ=180;
[sp,axes_x,axes_y,pixel]=headata(N)            %创建仿真头模型数据
degree=projdata(PROJ,N);                       %从头模型产生投影函数
[axes_x,h]=RLfilter(N);                        %创建滤波函数
 m=N;
 n=NUM;
 k=PROJ;
 F=zeros(m,m);                                  %参数初始化
 for k=1:PROJ
     for j=1:N-1
        sn(j)=0;
        for i=1:N-1
           sn(j)=sn(j)+h(j+N-i)*degree(k,i);    %计算 Qtheta，投影数据与滤波函数卷积
        end
     end

 for i=1:N
      for j=1:N
         cq=N/2-(N-1)*(cos(k*pi/PROJ)+sin(k*pi/PROJ))/2;        %从投影数据重建图像
         s2=((i-1)*cos(k*pi/PROJ)+(j-1)*sin(k*pi/PROJ)+cq);
         n0=fix(s2);                                %整数部分
         s4=s2-n0;                                  %小数部分
         if((n0>=1) && ((n0+1)<N))
            F(j,i)=F(j,i)+(1.0-s4)*sn(n0)+s4*sn(n0+1);
         end
      end
```

```
    end
  end
set(0,'defaultFigurePosition',[100,100,1200,450]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                     %修改图形背景颜色的设置
figure,
subplot(121),pcolor(pixel);
subplot(122),pcolor(F)                                  %显示重构图像

close all; clear all;clc;
I=imread('车牌 1.jpg');                                 %图像输入
I1=rgb2gray(I);                                         %转换成灰度图像 I1
I2=wiener2(I1,[5,5]);                                   %对图像进行维纳滤波 I2
I3=edge(I2,'sobel', 'horizontal');                      %用 Sobel 水平算子对图像边缘化
theta=0:179;                                            %设置选择角度
r=radon(I3,theta);                                      %对图像进行 Radon 变换
[m,n]=size(r);
c=1;
for i=1:m
    for j=1:n
        if   r(1,1)<r(i,j)
            r(1,1)=r(i,j);
              c=j;
        end
    end
end                                                     %检测 Radon 变换矩阵中的峰值所对应的列坐标
rot=90-c;                                               %确定旋转角度
I4=imrotate(I2,rot,'crop');                             %对图像进行旋转矫正
set(0,'defaultFigurePosition',[100,100,1200,450]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                     %修改图形背景颜色的设置
figure,                                                 %显示处理结果
subplot(121),imshow(I)
subplot(122),imshow(I2)
figure,
subplot(121),imshow(I3)
subplot(122),imshow(I4)

close all; clear all;clc;
B=imread('girl2.bmp');                                  %读入图像
C=imread('boy1.bmp');
BW1=face_detection(B);                                  %调用函数 face_detection 进行人脸检测
BW2=face_detection(C);
set(0,'defaultFigurePosition',[100,100,1200,450]);      %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                     %修改图形背景颜色的设置
figure,
```

```
subplot(121),imshow(B);                                  %显示原图及结果
subplot(122),imshow(BW1);
figure,
subplot(121),imshow(C);
subplot(122),imshow(BW2);

close all; clear all;clc;
B=imread('girl2.bmp');                                   %读入图像
C=imread('boy1.bmp');
BW1=refine_face_detection(B);                %调用函数 refine_face_detection 进行人脸检测
BW2=refine_face_detection(C);
set(0,'defaultFigurePosition',[100,100,1200,450]);       %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                      %修改图形背景颜色的设置
figure,
subplot(121),imshow(BW1);                                %显示原图及结果
subplot(122),imshow(BW2);

close all; clear all;clc;
I=imread('girl1.bmp');
I1=refine_face_detection(I);                             %人脸分割
[m,n]=size(I1);
theta1=0;                                                %方向
theta2=pi/2;
f = 0.88;                                                %中心频率
sigma = 2.6;                                             %方差
Sx = 5;
Sy = 5;                                                  %窗宽度和长度
Gabor1=Gabor_hy(Sx,Sy,f,theta1,sigma);                   %产生 Gabor 变换的窗口函数
Gabor2=Gabor_hy(Sx,Sy,f,theta2,sigma);                   %产生 Gabor 变换的窗口函数
Regabout1=conv2(I1,double(real(Gabor1)),'same');
Regabout2=conv2(I1,double(real(Gabor2)),'same');
set(0,'defaultFigurePosition',[100,100,1200,450]);       %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                      %修改图形背景颜色的设置
figure,
subplot(131),imshow(I);
subplot(132),imshow(Regabout1);
subplot(133),imshow(Regabout2);

close all; clear all;clc;
I=imread('girl1.bmp');
I1=refine_face_detection(I);                             %人脸分割
I1=double(I1);
[m,n]=size(I1);
theta1=0;                                                %方向
```

```
theta2=pi/2;
f = 0.88;                                   %中心频率
sigma = 2.6;                                %方差
Sx = 5;
Sy = 5;                                     %窗宽度和长度
Gabor1=Gabor_hy(Sx,Sy,f,theta1,sigma);      %产生 Gabor 变换的窗口函数
Gabor2=Gabor_hy(Sx,Sy,f,theta2,sigma);      %产生 Gabor 变换的窗口函数
Regabout1=conv2(I1,double(real(Gabor1)),'same');
Regabout2=conv2(I1,double(real(Gabor2)),'same');
Regabout=(Regabout1+Regabout2)/2;
%% 第一次膨胀
J1 = im2bw(Regabout,0.2);
SE1 = strel('square',2);BW = imdilate(J1,SE1);
[B,L,N] = bwboundaries(BW,'noholes');       %边界跟踪
a = zeros(1,N);
for i1 = 1:N
    a(i1) = length(find(L == i1));
end
a1 = find(a > 300);
for i1 = 1:size(a1,2)
L(find(L == a1(i1))) = 0;
end
L1 = double(uint8(L*255))/255;
a = 0;
BW = I1 .* L1;
%% 第二此膨胀
for i2 = 1:m
    for j2 = 1:n
        if BW(i2,j2) > 0 && BW(i2,j2) < 50
            BW(i2,j2) = 255;
        end
    end
end
BW = uint8(BW);
J2 = im2bw(BW,0.8);
SE1 = strel('rectangle',[2 5]);BW = imdilate(J2,SE1);
[B,L,N] = bwboundaries(BW,'noholes');       %边界跟踪
a = zeros(1,N);
for i1 = 1:N
    a(i1) = length(find(L == i1));
end
a1 = find(a > 300);
for i1 = 1:size(a1,2)
L(find(L == a1(i1))) = 0;
```

```
end
L1 = double(uint8(L*255))/255;
a =0;
SE1 = strel('rectangle',[10 10]);BW = imdilate(L1,SE1);
BW = uint8(I1 .* double(BW));
set(0,'defaultFigurePosition',[100,100,1200,450]);        %修改图形图像位置的默认设置
set(0,'defaultFigureColor',[1 1 1])                       %修改图形背景颜色的设置
figure,
imshow(BW);

close all; clear all;clc;
X1=[1 1 1 1, 1 0 0 1, 1 1 1 1, 1 0 0 1];                  %识别模式
X2=[0 1 0 0, 0 1 0 0, 0 1 0 0, 0 1 0 0];
X3=[1 1 1 1, 1 0 0 1, 1 0 0 1, 1 1 1 1];
X=[X1;X2;X3];
Y1=[1 0 0];                                               %输出模式
Y2=[0 1 0];
Y3=[0 0 1];
Yo=[Y1;Y2;Y3];
n=16;                                                     %输入层神经元个数
p=8;                                                      %中间层神经元个数
q=3;                                                      %输出神经元个数
k=3 ;                                                     %训练模式个数
a1=0.2; b1=0.2;                                           %学习系数，
%rou=0.5;                                                 %动量系数，
emax=0.01; cntmax=100;                                    %最大误差，训练次数
[w,v,theta,r,t,mse]=bptrain(n,p,q,X,Yo,k,emax,cntmax,a1,b1);          %调用函数 bptrain 训练网络
X4=[1 1 1 1, 1 0 0 1, 1 1 1 1, 1 0 1 1 ];
disp('模式 X1 的识别结果：')                               %测试并显示对图形的识别结果
c1=bptest(p,q,n,w,v,theta,r,X1)
disp('模式 X2 的识别结果：')
c2=bptest(p,q,n,w,v,theta,r,X2)
disp('模式 X3 的识别结果：')
c3=bptest(p,q,n,w,v,theta,r,X3)
disp('模式 X4 的识别结果：')
c4=bptest(p,q,n,w,v,theta,r,X4)
c=[c1;c2;c3;c4];
for i=1:4
      for j=1:3
          if c(i,j)>0.5
              c(i,j)=1;
         elseif c(i,j)<0.2
           c(i,j)=0;
           end
```

```
    end
end
disp('模式 X1~X4 的识别结果：')
c

close all; clear all;clc;
%源程序：
%输入向量:16 种输入向量
P=[1 1 1,1 0 1,1 0 1,1 0 1,1 1 1;                    %0
   0 1 0,0 1 0,0 1 0,0 1 0,0 1 0;                    %1
   1 1 1,0 0 1,0 1 0,1 0 0,1 1 1;                    %2
   1 1 1,0 0 1,0 1 0,0 0 1,1 1 1;                    %3
   1 0 1,1 0 1,1 1 1,0 0 1,0 0 1;                    %4
   1 1 1,1 0 0,1 1 1,0 0 1,1 1 1;                    %5
   1 1 1,1 0 0,1 1 1,1 0 1,1 1 1;                    %6
   1 1 1,0 0 1,0 0 1,0 0 1,0 0 1;                    %7
   1 1 1,1 0 1,1 1 1,1 0 1,1 1 1;                    %8
   1 1 1,1 0 1,1 1 1,0 0 1,1 1 1;                    %9
   0 1 0,1 0 1,1 0 1,1 1 1,1 0 1;                    %A
   1 1 1,1 0 1,1 1 0,1 0 1,1 1 1;                    %B
   1 1 1,1 0 0,1 0 0,1 0 0,1 1 1;                    %C
   1 1 0,1 0 1,1 0 1,1 0 1,1 1 0;                    %D
   1 1 1,1 0 0,1 1 0,1 0 0,1 1 1;                    %E
   1 1 1,1 0 0,1 1 0,1 0 0,1 0 0]';                  %F
%目标向量
T=[0 0 0 0;0 0 0 1;0 0 1 0;0 0 1 1;
   0 1 0 0;0 1 0 1;0 1 1 0;0 1 1 1;
   1 0 0 0;1 0 0 1;1 0 1 0;1 0 1 1;
   1 1 0 0;1 1 0 1;1 1 1 0;1 1 1 1]';
%输入向量的最大值和最小值
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
%创建一个 BP 网络, []内分别为中间层和隐含层
net=newff(threshold,[13,4],{'tansig','logsig'},'trainlm');
%最大训练次数
net.trainParam.epochs=100;
%训练目标(最大误差)
net.trainParam.goal=0.0005;
%学习速率(学习系数)
LP.lr=0.01;
%训练网络
net=train(net,P,T);
%测试数据(和训练数据不一致)
P_test=[ 1 1 1,1 0 0,1 1 1,1 0 0,1 0 0;
         1 0 1,1 0 1,1 1 1,0 0 1,0 1 1]';
```

```
%对测试数据进行仿真,验证训练的网络是否完善
y=sim(net,P_test)'


function [w,v]=bptrain(n,p,q,X,Y,k,emax,cntmax)
close all; clear all;clc;
tic
n='输入神经元个数:16';                          %输入层神经元
disp(n)
p='中间层神经元个数:8';                          %中间层神经元
disp(p)
q='输出层神经元个数:3';                          %输出层神经元
disp(q)
X1=[1 1 1 1, 1 0 0 1, 1 1 1 1, 1 0 0 1];         %识别模式
X2=[0 1 0 0, 0 1 0 0, 0 1 0 0, 0 1 0 0];
X3=[1 1 1 1, 1 0 0 1, 1 0 0 1, 1 1 1 1];
X=[X1;X2;X3];
Y1=[1 0 0];                                      %输出模式
Y2=[0 1 0];
Y3=[0 0 1];
Yo=[Y1;Y2;Y3];
n=16;                                            %输入层神经元个数
p=8;                                             %中间层神经元个数
q=3;                                             %输出神经元个数
k=3 ;                                            %训练模式个数
w=rands(n,p);                                    %输入层与隐含层连接权
v=rands(p,q);                                    %隐含层与输出层连接权
theta=rands(1,p);                                %中间层的阈值
r=rands(1,q);                                    %输出层的阈值
a1=0.1, b1=0.1,                                  %学习系数，
E=zeros(1,q);
emax=0.01, cntmax=100;                           %最大误差，训练次数
cnt=1;
er=0;                                            %全局误差为零
while ((er>emax)|(cnt<=cntmax))
 %循环识别模式
 for cp=1:k
      X0=X(cp,:);
      Y0=Yo(cp,:);
      %计算中间层的输入 Y(j)
      Y=X0*w;
      %计算中间层的输出 b
      Y=Y-theta;                                 %中间层阈值
      for j=1:p
```

```
            b(j)=1/(1+exp(-Y(j)));                    %中间层输出 f(sj)
        end
      %计算输出层输出 c
                Y=b*v;
                Y=Y-r;                                % 输出层阈值
          for t=1:3
              c(t)=1/(1+exp(-Y(t)));                  %输出层输出
          end
      %计算输出层校正误差 d
          for t=1:3
             d(t)=(Y0(t)-c(t))*c(t)*(1-c(t));
          end
    %计算中间层校正误差 e
           xy=d*v';
           for t=1:8
             e(t)=xy(t)*b(t)*(1-b(t));
           end
    %计算下一次的中间层和输出层之间新的连接权 v(i,j),阈值 t2(j)
            for t=1:q
                for j=1:p
                    v(j,t)=v(j,t)+a1*d(t)*b(j);
                end
                r(t)=r(t)+a1*d(t);
            end
        %计算下一次的输入层和中间层之间新的连接权 w(i,j),阈值 t1(j)
             for j=1:p
                for i=1:n
                    w(i,j)=w(i,j)+b1*e(j)*X0(i)
                end
                theta(j)=theta(j)+b1*e(j)
             end
             for t=1:q
                  E(cp)=(Y0(t)-c(t))*(Y0(t)-c(t))+E(cp);   %求当前学习模式的全局误差
             end
              E(cp)=E(cp)*0.5;
      %输入下一模式
  end
   er=max(E);                                         %计算全局误差
   cnt=cnt+1;                                         %更新学习次数
end
  time=toc;
  %重新训练
  %S='训练次数:';
  %disp(S);
```

```
 %disp(cnt);
 %S='输出层权值:';
 %disp(S);
 %disp(v);
 %S='中间层权值:';
 %disp(S);
 %disp(w);


function c=bptest(p,q,n,w,v,theta,r,X)
%p,q,n 表示输入层、中间层和输出层神经元个数
%w，v 表示训练好的神经网络输入层到中间层、中间层到输出层权值。
%X 为输入测试的模式
%c 表示模式 X 送入神经网络的识别结果。
%计算中间层的输入 Y(j)
Y=X*w;
%计算中间层的输出 b
Y=Y-theta;                                          %中间层阈值
for j=1:p
      b(j)=1/(1+exp(-Y(j)));                        %中间层输出 f(sj)
end
%计算输出层输出 c
Y=b*v;
Y=Y-r;                                              %输出层阈值
thr1=0.01;thr2=0.5;
for t=1:q
    c(t)=1/(1+exp(-Y(t)));                          %输出层输出
end


function [w,v,theta,r,t,mse]=bptrain(n,p,q,X,Yo,k,emax,cntmax,a1,b1)
%n 表示输入神经元个数,p 表示中间层神经元个数，q 表示输出神经元个数,
%X 表示输入训练模式，Yo 表示标准输出，k 表示训练模式的个数
%emax 表示最大误差，cntmax 表示最大训练次数，a1，b1 表示学习系数，rou 表示动量系
数
%w、theta 表示训练结束后输入层与中间层连接权系数和阈值，
%v、r 表示训练结束后中间层与输出层连接权和阈值
%t 表示训练时间
tic
w=rands(n,p);                                       %输入层与隐含层连接权
v=rands(p,q);                                       %隐含层与输出层连接权
theta=rands(1,p);                                   %中间层的阈值
r=rands(1,q);                                       %输出层的阈值
cnt=1;
```

```
mse=zeros(1,cntmax);                                %全局误差为零
er=0;
while ((er>emax)|(cnt<=cntmax))
  E=zeros(1,q);
  %循环识别模式
  for cp=1:k
        X0=X(cp,:);
        Y0=Yo(cp,:);
        %计算中间层的输入 Y(j)
        Y=X0*w;
        %计算中间层的输出 b
        Y=Y-theta;                                  %中间层阈值
        for j=1:p
             b(j)=1/(1+exp(-Y(j)));                 %中间层输出 f(sj)
        end
      %计算输出层输出 c
                 Y=b*v;
                 Y=Y-r;                             % 输出层阈值
            for t=1:q
                c(t)=1/(1+exp(-Y(t)));              %输出层输出
            end
       %计算输出层校正误差 d
            for t=1:q
              d(t)=(Y0(t)-c(t))*c(t)*(1-c(t));
            end
     %计算中间层校正误差 e
             xy=d*v';
             for t=1:p
               e(t)=xy(t)*b(t)*(1-b(t));
             end
     %计算下一次的中间层和输出层之间新的连接权 v(i,j),阈值 t2(j)
              for t=1:q
                   for j=1:p
                        v(j,t)=v(j,t)+a1*d(t)*b(j);
                   end
                   r(t)=r(t)+a1*d(t);
              end
        %计算下一次的输入层和中间层之间新的连接权 w(i,j),阈值 t1(j)
               for j=1:p
                   for i=1:n
                        w(i,j)=w(i,j)+b1*e(j)*X0(i);
                   end
                   theta(j)=theta(j)+b1*e(j);
               end
```

```
            for t=1:q
                E(cp)=(Y0(t)-c(t))*(Y0(t)-c(t))+E(cp);   %求当前学习模式的全局误差
            end
            E(cp)=E(cp)*0.5;
        %输入下一模式
    end
    er=sum(E);                                          %计算全局误差
    mse(cnt)=er;
    cnt=cnt+1;                                          %更新学习次数
  end
  t=toc;


function BW= face_detection(I)
% I 是待识别的彩色图像，BW 是检测到二值人脸图像
I1=I;                                                   %输入图像矩阵 I
R=I1(:,:,1);                                            %获取 RGB 图像矩阵 I 的 R、G、B 取值
G=I1(:,:,2);
B=I1(:,:,3);
Y=0.299*R+0.587*G+0.114*B;                              %进行颜色空间转换  计算 Y  和 Cb
Cb=-0.1687*R-0.3313*G+0.5000*B+128;
for Cb=133:165
    r=(Cb-128)*1.402+Y;                                 %将 YCrCb 空间中 Cb=133:165 中的区域确定
    r1=find(R==r);                                      %产生肤色聚类的二值矩阵
    R(r1)=255;                                          %对肤色聚类的区域
    G(r1)=255;
    B(r1)=255;
end
I1(:,:,1)=R;                                            %生成肤色聚类后的图像
I1(:,:,2)=G;
I1(:,:,3)=B;
J=im2bw(I1,0.99);                                       %转换成灰度图像
BW=J;                                                   %返回结果


function Gabor= Gabor_hy(Sx,Sy,f,theta,sigma)
% Sx,Sy 是滤波器窗口长度，f 是中心频率，theta 是滤波器的方向，sigma 是高斯窗的方差
x = -fix(Sx):fix(Sx);                                   %Gabor 滤波器的窗口长度
y = -fix(Sy):fix(Sy);
[x y]=meshgrid(x,y);
xPrime = x*cos(theta) + y*sin(theta);
yPrime = y*cos(theta) - x*sin(theta);
Gabor = (1/(2*pi*sigma.^2)) .* exp(-.5*(xPrime.^2+yPrime.^2)/sigma.^2).*...     %Gabor 滤波器
                    (exp(j*f*xPrime)-exp(-(f*sigma)^2/2));
```

```
function [fp,axes_x,axes_y,pixel]=headata(N)
%N 表示创建头文件的大小；fp 表示存储头模型数据文件的指针头；
%axes_x 和 axes_y 表示的文件绘图坐标范围；
%pixel 为头模型数据矩阵；
%具体调用形式[fp,axes_x,axes_y,pixel]=headata(N)。
 lenth=N*N;
 pixel=zeros(N,N);                                    %生成图像的密度矩阵，初值为零
 coordx=[0,0,0.22,-0.22,0,0,0,-0.08,0,0.06]; %每个椭圆中心的 x 坐标，各个椭圆代表不同组织
 coordy=[0,-0.0184,0,0,0.35,0.1,-0.1,-0.605,-0.605,-0.605];          %每个椭圆中心的 y 坐标
 laxes=[0.92,0.874,0.31,0.41,0.25,0.046,0.046,0.046,0.023,0.046];    %每个椭圆长轴的大小
 saxes=[0.69,0.6624,0.11,0.16,0.21,0.046,0.046,0.023,0.023,0.023];   %每个椭圆短轴的大小
 angle=[90,90,72,108,90,0,0,0,0,90];                                 %每个椭圆旋转的角度
 density=[2.0,-0.98,-0.4,-0.4,0.2,0.2,0.2,0.2,0.2,0.3];             %每个椭圆的灰度值
    for i=1:N,
          for j=1:N,
               for k=1:10,
                    axes_x(i,j)=(-1+j*2/N-0.5*2/N);                  %画图像时的 x 坐标
                    x=(-1+j*2/N)-coordx(k);
                    axes_y(i,j)=(-1+i*2/N-0.5*2/N);                  %画图像时的 y 坐标
                    y=(-1+i*2/N)-coordy(k);
                    alpha=pi*angle(k)/180;
                    a=(x*cos(alpha)+y*sin(alpha))/laxes(k);   %判断像素点是否在第 k 个椭圆里
                    b=(-x*sin(alpha)+y*cos(alpha))/saxes(k);
                    if((a*a+b*b)<=1)
                    pixel(i,j)=density(k)+pixel(i,j);
                    end
               end
          end
    end
 fp=fopen('datafile_name.txt','w');                  %创建头模型数据文件，对其进行写操作
 for i=1:N,
     for j=1:N,
               a=[i j pixel(i,j)];
               fprintf(fp,'%d    %d    %f\n',a);
     end
 end
fclose(fp);                                          %关闭文件指针
fp=fopen('datafile_name.txt','r');                   %保存 datafile_name.txt 文件头


function degree=projdata(proj,N)
%proj 表示投影数;
```

```
%N 表示投影轴 R 的采样点数;
%具体的调用格式 degree=projdata(proj,N)
NUM=10;                                                    %椭圆的个数
%ellipse 定义十个椭圆，其中一个[]描述一个椭圆相关参数，每个椭圆对应不同组织
%[]椭圆的定义依次 X0，Y0，长轴，短轴，旋转角度，灰度值
%定义一个椭圆矩阵，将十个椭圆描述清楚，采用的是 cell 格式
ellipse={[0,0,0.92,0.69,90,2.0],
         [0,-0.0184,0.874,0.6624,90,0.98],
         [0.22,0,0.31,0.11,72,-0.4],
         [-0.22,0,0.41,0.16,108,-0.4],
         [0,0.35,0.25,0.21,90,0.4],
         [0,0.1,0.046,0.046,0,0.4],
         [0,-0.1,0.046,0.046,0,0.4],
         [-0.08,-0.605,0.046,0.023,0,0.4],
         [0,-0.605,0.023,0.023,0,0.4],
         [0.06,-0.605,0.046,0.023,90,0.4]};
  step=180/proj;                                           %投影角旋转的增量
  for i=1:NUM
    a(i)=ellipse{i,1}(3);                                  %第 i 个椭圆的长轴
    b(i)=ellipse{i,1}(4);                                  %第 i 个椭圆的短轴
    c(i)=2*a(i)*b(i);                                      % 2*长轴*短轴
    a2(i)=a(i)*a(i);                                       %长轴平方，矩阵 a2 1*10
    b2(i)=b(i)*b(i);                                       %短轴平方，矩阵 b2 1*10
    alpha(i)=ellipse{i,1}(5)*pi/180;                       %第 i 个椭圆旋转的角度转化成弧度
    sina(i)=sin(alpha(i));%sin(alpha)
    cosa(i)=cos(alpha(i));%cos(alpha)
  end
 for j=1:proj
    for i=1:NUM
      theta(j)=step*j*pi/180;                              %theta 投影线的与 x 轴夹角
      angle(i,j)=alpha(i)-theta(j);                        %alpha 表示椭圆的中心线与 x 轴夹角;
      zx2(i,j)=sin(angle(i,j))*sin(angle(i,j));            %zx2=sin 平方
      yx2(i,j)=cos(angle(i,j))*cos(angle(i,j));            %yx2=cos 平方
    end
 end
 length=2/N;
for i=1:proj
  R=-(N/2)*length;
  for j=1:N
    R=R+length;
    degree(i,j)=0;
      for m=1:10
        A=a2(m)*yx2(m,i)+b2(m)*zx2(m,i); %a2(m)相应椭圆长轴 a 的平方，yx2(m,i)余旋平方
        x0=ellipse{m,1}(1);
```

```
            y0=ellipse{m,1}(2);
            B=R-x0*cos(theta(i))-y0*sin(theta(i));          %计算投影值
            B=B*B;
            E=A-B;
            if (E>0)
              midu=ellipse{m,1}(6)*c(m)*sqrt(E)/A;
              degree(i,j)=degree(i,j)+midu;
            end
          end
    end
end


function BW=refine_face_detection(I)
% I 是待识别的彩色图像，BW 是检测到二值人脸图像
%%肤色聚类
I1=I;                                                   %输入图像矩阵 I
R=I1(:,:,1);                                            %获取 RGB 图像矩阵 I 的 R、G、B 取值
G=I1(:,:,2);
B=I1(:,:,3);
Y=0.299*R+0.587*G+0.114*B;                              %进行颜色空间转换  计算 Y  和 Cb
Cb=-0.1687*R-0.3313*G+0.5000*B+128;
for Cb=133:165
      r=(Cb-128)*1.402+Y;                               %将 YCrCb 空间中 Cb=133:165 中的区域确定
      r1=find(R==r);                                    %产生肤色聚类的二值矩阵
      R(r1)=255;                                        %对肤色聚类的区域
      G(r1)=255;
      B(r1)=255;
end
I1(:,:,1)=R;                                            %生成肤色聚类后的图像
I1(:,:,2)=G;
I1(:,:,3)=B;
J=im2bw(I1,0.99);                                       %转换成灰度图像
%%  膨胀和腐蚀
SE1=strel('square',8);
BW1=imdilate(J,SE1);                                    %先小面积膨胀
BW1=imfill(BW1,'holes');                                %填充区域里的洞
SE1=strel('square',20);
BW1=imerode(BW1,SE1);                                   %大面积的腐蚀
SE1=strel('square',12);
BW1=imdilate(BW1,SE1);                                  %膨胀，恢复人脸区域
%%  定位人脸的大致区域
[B,L,N]=bwboundaries(BW1,'noholes');                    %边界跟踪
a=zeros(1,N);
```

```
for i1=1:N
      a(i1)=length(find(L==i1));                    %获取斑点位置
end
a1=find(a==max(a));
L1=(abs(L-a1))*255;
I2=double(rgb2gray(I));                             %原彩色图像转灰度图像
I3=uint8(I2-L1);                                    %消除斑点
BW=I3;                                              %返回结果


function [axes_x,h]=RLfilter(N,L)
%定义量化值 N
delta=L/N;                                          %对滤波函数进行离散化单位量
for i=2:2:2*N                                       %偶数项=0
      h(i)=0;
end
k=1/delta/delta;                                    %原点项=0
h(N)=k/4;
for i=1:2:N-1
   down=-k/(i*i*pi*pi);                             %奇数项=-1/(n^2 π^2 d^2 )
   h(N+i)=down;
   h(N-i)=down;
end
for i=1:2*N
axes_x(i)=(-1+(i-1)/N);                             %画图像时的 x 坐标
end
```