拓展资源 1.2　MATLAB 与数字图像处理

1.2.1　MATLAB 简介

MATLAB 语言是由美国 MathWorks 公司推出的计算机软件，经过多年的逐步发展与不断完善，现已成为国际公认的优秀的科学计算与数学应用软件之一，其内容涉及矩阵代数、微积分、应用数学、有限元法、科学计算、信号与系统、神经网络、小波分析及其应用、数字图像处理、计算机图形学、电子线路、电机学、自动控制与通信技术、物理、力学和机械振动等方面。MATLAB 的特点是语法结构简单，数值计算高效，图形功能完备，特别受到以完成数据处理与图形图像生成为主要目的的技术研发人员的青睐。各国的学生（包括硕士生和博士生）也将 MATLAB 作为必须掌握的基本程序设计语言。

MATLAB 中的基本数据结构是由一组有序的实数或复数元素构成的数组，同样地，图像对象的表达采用的是一组有序的灰度或彩色数据元素构成的实值数组。MATLAB 中通常用二维数组来存储图像，数组的每个元素对应于图像的一个像素值。例如，由 200 行和 300 列的不同颜色点组成的一幅图像在 MATLAB 中采用 200×300 的矩阵存储。MATLAB 支持多种类型的图像，而不同类型的图像其存储结构通常是不同的，如 RGB 图像则需要一个三维数组，3 个数据维分别对应于某像素点的红色、绿色和蓝色强度值。由于对图像采用了通用的数据矩阵的表达方式，MATLAB 中原有的所有基本矩阵操作都可以应用于图像矩阵。

1.2.2　数字图像的表示和类别

一幅图像可以被定义为一个二维函数 f(x,y),其中 x 和 y 是空间（平面）坐标，f 在任何坐标处(x,y)处的振幅称为图像在该点的亮度。灰度是用来表示黑白图像亮度的一个术语，而彩色图像是由单个二维图像组合形成的。例如，在 RGB 彩色系统中，一幅彩色图像是由三幅独立的分量图像（红、绿、蓝）组成的。因此，许多为黑白图像处理开发的技术适用于彩色图像处理，方法是分别处理三幅独立的分量图像即可。

图像关于 x 和 y 坐标以及振幅连续。要将这样的一幅图像转化为数字形式，就要求数字化坐标和振幅。将坐标值数字化成为取样；将振幅数字化成为量化。采样和量化的过程如图 1.1 所示。因此，当 f 的 x、y 分量和振幅都是有限且离散的量时，称该图像为数字图像。

作为 MATLAB 基本数据类型的数值数组本身十分适于表达图像，矩阵的元素和图像的像素之间有着十分自然的对应关系。

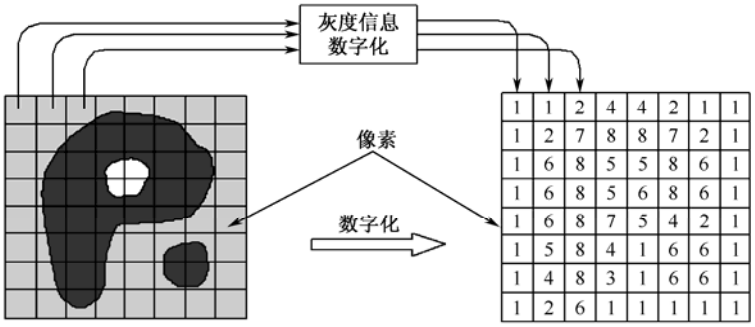


图 1.1　图像的采样和量化

根据图像数据矩阵解释方法的不同，MATLAB 把其处理为 4 类：亮度图像（Intensity images）、二值图像（Binary images）、索引图像（Indexed images）、RGB 图像（RGB images）。

（1）亮度图像

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

（2）二值图像

一幅二值图像是一个取值只有 0 和 1 的逻辑数组。而一幅取值只包含 0 和 1 的 uint8 类数组，在 MATLAB 中并不认为是二值图像。使用 logical 函数可以把数值数组转化为二值数组或逻辑数组。创建一个逻辑图像，其语法为：

B=logical(A)

其中，B 是由 0 和 1 构成的数值数组。

要测试一个数组是否为逻辑数组，可以使用函数：

islogical(c)

若 C 是逻辑数组，则该函数返回 1；否则，返回 0。

（3）索引图像

索引颜色通常也称为映射颜色，在这种模式下，颜色都是预先定义的，并且可供选用的一组颜色也很有限，索引颜色的图像最多只能显示 256 种颜色。

一幅索引颜色图像在图像文件里定义，当打开该文件时，构成该图像具体颜色的索引值就被读入程序里，然后根据索引值找到最终的颜色。

（4）RGB 图像

一幅 RGB 图像就是彩色像素的一个 M×N×3 数组，其中每一个彩色相似点都是在特定空间位置的彩色图像相对应的红、绿、蓝三个分量。按照惯例，形成一幅 RGB 彩色图像的三个图像常称为红、绿或蓝分量图像。

令 fR，fG 和 fB 分别代表三种 RGB 分量图像。一幅 RGB 图像就利用 cat（级联）操作将这些分量图像组合成彩色图像：

rgb_image=cat(3,fR,fG,fB)

在操作中，图像按顺序放置。

1.2.3 数据类和图像类型间的转化

表 1.1 中列出了 MATLAB 和 IPT 为表示像素所支持的各种数据类。表中的前 8 项称为数值数据类，第 9 项称为字符类，第 10 项称为逻辑数据类。

工具箱中提供了执行必要缩放的函数，如表 1.2 所示，以在图像类和类型间进行转化。

表 1.1 MATLAB 和 IPT 支持数据类型

序号	名称	描述
1	double	双精度浮点数，范围为-10^{308}–10^{308}
2	uint8	无符号 8bit 整数，范围为[0 255]
3	uint16	无符号 16bit 整数，范围为[0 65536]
4	uint32	无符号 32bit 整数，范围为[0 4294967295]
5	int8	有符号 8bit 整数，范围为[−128 127]
6	int16	有符号 16bit 整数，范围为[−32768 32767]

续表

序号	名称	描述
7	int32	有符号 32 比特整数，范围为[−2147483648 2147483647]
8	single	单精度浮点数，范围为-10^{308}–10^{308}
9	char	字符
10	logical	值为 0 或 1

表 1.2 格式转换函数

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

1.2.4 MATLAB7.0 图像处理工具箱

数字图像研究的领域非常广泛，从学科上可以分为图像的数字化、图像变换、图像增强、图像恢复、图像分割、图像分析和理解，以及图像的压缩等。

MATLAB 图像处理工具箱提供了非常丰富的图像处理函数，主要完成以下功能。

（1）图像的几何操作。

（2）图像的邻域和图像块操作。

（3）线性滤波和滤波器设计。

（4）图像变换。

（5）图像分析和增强。

（6）二值图像形态学操作。

（7）图像复原。

（8）图像编码。

（9）特定区域操作。

1.2.5 与图像处理相关的 MATLAB 函数的使用

1. 图像文件的读/写

（1）imread 函数用来实现图像文件的读取。

输入以下程序：

```
A=imread('drum.bmp'); %用 imread 函数来读入图像
imshow(A);            %用 imshow 函数来显示图像
```

得到的结果如图 1.2 所示。

图 1.2 drum.bmp 图像的显示

（2）imwrite 函数用来实现图像文件的写入。

输入以下程序：

```
imwrite(A,'drum.bmp');          %可把图像文件写入 MATLAB 的目录下
```

（3）imfinfo 函数用来查询图像文件信息。

输入以下程序：

```
info=imfinfo('drum.bmp');       %用 imfinfo 函数查询图像文件信息
```

得到：

```
info =

           Filename: 'drum.bmp'
        FileModDate: '05-Jan-2010 09:58:24'
           FileSize: 206694
             Format: 'bmp'
      FormatVersion: 'Version 3 (Microsoft Windows 3.x)'
              Width: 273
             Height: 252
           BitDepth: 24
          ColorType: 'truecolor'
    FormatSignature: 'BM'
 NumColormapEntries: 0
           Colormap: []
            RedMask: []
          GreenMask: []
           BlueMask: []
```

```
       ImageDataOffset: 54
      BitmapHeaderSize: 40
            NumPlanes: 1
       CompressionType: 'none'
           BitmapSize: 0
        HorzResolution: 3780
        VertResolution: 3780
         NumColorsUsed: 0
    NumImportantColors: 0
```

（4）imshow 函数用来显示图像。介绍 imread 函数时已使用此函数。

（5）colorbar 函数将颜色条添加到坐标轴对象中。

输入以下程序：

```
RGB=imread('drum.png');                %图像读入
I=rgb2gray(RGB);                       %把 RGB 图像转换成灰度图像
h=[1 2 1;0 0 0;-1 -2 -1];
I2=filter2(h,I);
imshow(I2,[]),colorbar('vert')         %将颜色条添加到坐标轴对象中
```

得到的结果如图 1.3 所示。

（6）warp 函数将图像作为纹理进行映射，使图像显示在一个特定的三维空间中。

输入以下程序：

```
A=imread('drum.bmp');
I=rgb2gray(A);
[x,y,z]=sphere;
warp(x,y,z,I);              %用 warp 函数将图像作为纹理进行映射
```

得到的结果如图 1.4 所示。

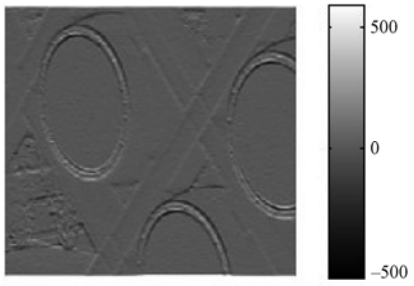


图 1.3　由 RGB 图像转换成的灰度图像

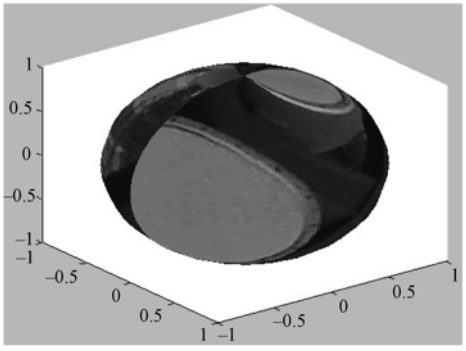


图 1.4　映射图

（7）subimage 函数实现在一个图形窗口中显示多幅图像。

输入以下程序：

```
RGB=imread('drum.bmp');
```

```
    I=rgb2gray(RGB);
    subplot(1,2,1),subimage(RGB) %subimage 函数实现在一个图形窗口中显示多幅图像
    subplot(1,2,2),subimage(I)
```

得到的结果如图 1.5 所示。

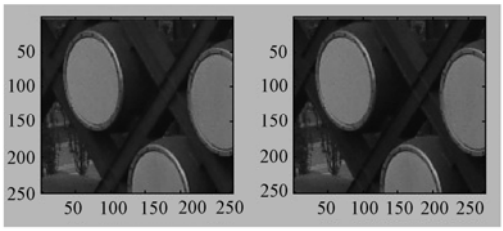


图 1.5　在一个窗口中显示多幅图像

2．图像处理的基本操作

1）图像代数运算

（1）imadd 函数实现两幅图像的相加或者给一幅图像加上一个常数。

给图像每个像素都增加亮度的程序如下：

```
    I=imread('drum.bmp');
    J=imadd(I,100);  %给图像增加亮度
    subplot(1,2,1),imshow(I)
    subplot(1,2,2),imshow(J)
```

得到的结果如图 1.6 所示。

（a）原始图像　　（b）增加亮度的图像

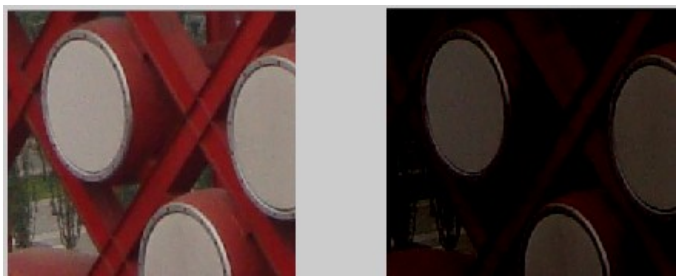
图 1.6　增加图像的亮度

（2）imsubtract 函数实现将一幅图像从另一幅图像中减去，或者从一幅图像中减去一个常数。

实现从一幅图像中减去一个常数，输入以下程序：

```
    I=imread('drum.bmp');
    J=imsubtract(I,100);        %给图像减少亮度
    subplot(1,2,1),imshow(I)
    subplot(1,2,2),imshow(J)
```

得到的结果如图 1.7 所示。



（a）原始图像　　（b）减少亮度的图像

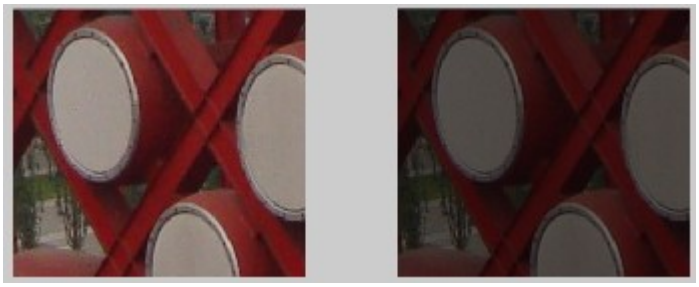
图 1.7　减少图像的亮度

（3）immultiply 实现两幅图像的相乘或者一幅图像的亮度缩放。

输入以下程序：

```
I=imread('drum.bmp');
J=immultiply(I,0.5);        %对图像进行亮度缩放
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(J)
```

得到的结果如图 1.8 所示。



（a）原始图像　　（b）亮度缩放的图像

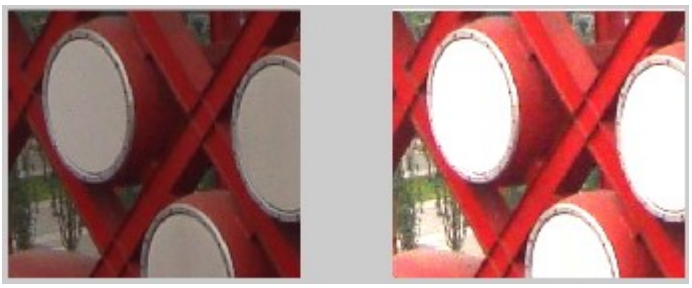
图 1.8　缩放亮度图像（变暗）

（4）imdivide 函数实现两幅图像的除法或一幅图像的亮度缩放。

输入以下程序：

```
I=imread('drum.bmp');
J=imdivide(I,0.5);                %图像的亮度缩放
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(J)
```

得到的结果如图 1.9 所示。



（a）原始图像　　（b）亮度缩放的图像

图 1.9　缩放亮度图像（变亮）

2）图像的空间域操作

（1）imresize 函数实现图像的缩放。

输入以下程序：

```
J=imread('drum.bmp');
X1=imresize(J,2);                    %对图像进行缩放
figure,imshow(J)
```

得到的结果如图 1.10 所示。

（2）imrotate 函数实现图像的旋转。

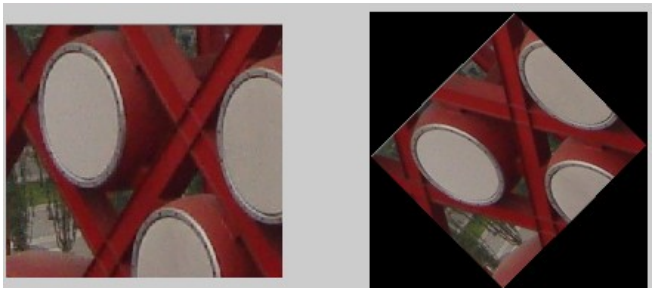
输入以下程序：

```
I=imread('drum.bmp');
J=imrotate(I,45,'bilinear');         %对图像进行旋转
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(J);
```

（a）原始图像　　（b）最近邻插值法

图 1.10　插值法变换图像（变小）

得到的结果如图 1.11 所示。



（a）原始图像　　（b）采用双线性插值法的图像旋转（45°）

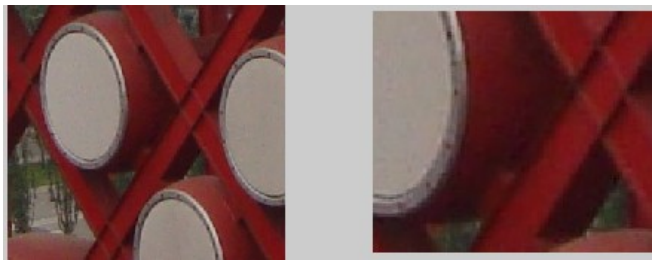
图 1.11　插值法变换图像（旋转）

（3）imcrop 函数实现图像的剪切。

输入以下程序：

```
I=imread('drum.bmp');
I2=imcrop(I,[75 68 130 112]);     %对图像进行剪切
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(I2);
```

得到的结果如图 1.12 所示。



（a）原始图像　　（b）剪切得到的图像

图 1.12　剪切变换图像

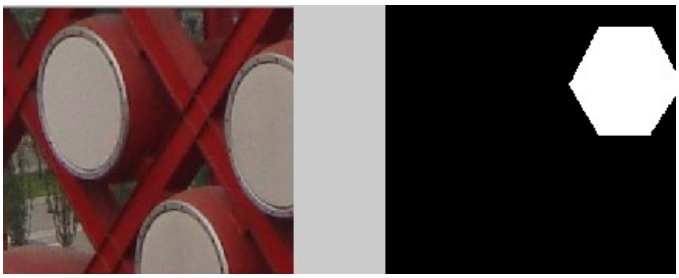
3）特定区域处理

（1）roipoly 函数用于选择图像中的多边形区域。

输入以下程序：

```
I=imread('drum.bmp');
c=[200 250 278 248 199 172];
r=[21 21 75 121 121 75];
BW=roipoly(I,c,r);          %roipoly 函数选择图像中的多边形区域
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(BW);
```

得到的结果如图 1.13 所示。



（a）原始图像　　（b）图像中的多边形区域

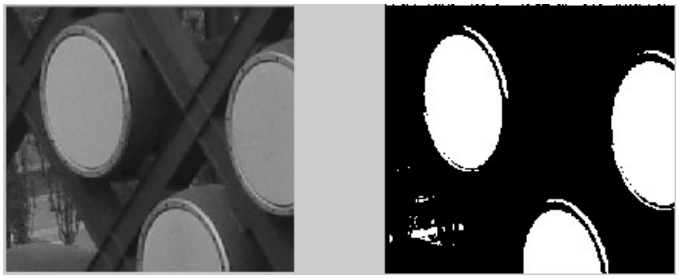
图 1.13　选择图像中的多边形区域

（2）roicolor 函数是对 RGB 图像和灰度图像实现按灰度或亮度值选择区域进行处理。

输入以下程序：

```
a=imread('drum.bmp');
I=rgb2gray(a);
BW=roicolor(I,128,225);     %按灰度值选择的区域
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(BW)
```

得到的结果如图 1.14 所示。



（a）原始图像　　（b）按灰度值选择的区域

图 1.14　按灰度值选择区域做二值化处理

（3）ploy2mask 函数转化指定的多边形区域为二值掩模。

输入以下程序：

```
x=[63 186 54 190 63];
y=[60 60 209 204 60];
bw=poly2mask(x,y,256,256);     %转化指定的多边形区域为二值掩模
imshow(bw)
hold on
plot(x,y,'b','LineWidth',2)
hold off
```

得到的结果如图 1.15 所示。

（4）roifilt2 函数实现区域滤波。

输入以下程序：

```
a=imread('drum.bmp');
I=rgb2gray(a);
c=[200 250 278 248 199 172];
r=[21 21 75 121 121 75];
BW=roipoly(I,c,r);          %roipoly 函数选择图像中的多边形区域
h=fspecial('unsharp');
J=roifilt2(h,I,BW);         %区域滤波
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(J)
```

得到的结果如图 1.16 所示。

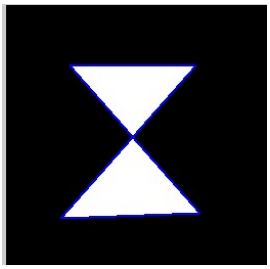
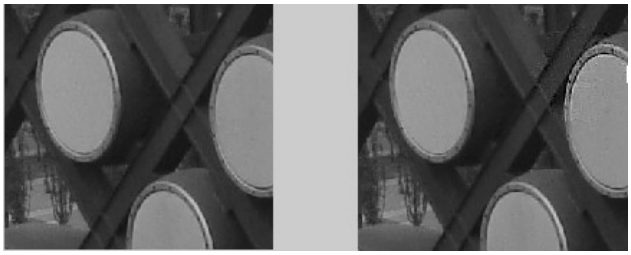


图 1.15　指定区域的二值掩模



（a）原始图像　　（b）roifilt2 滤波后结果

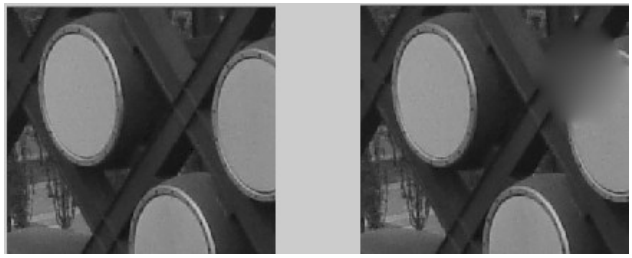
图 1.16　区域滤波

（5）roifill 函数实现对特定区域进行填充。

输入以下程序：

```
a=imread('drum.bmp');
I=rgb2gray(a);
c=[200 250 278 248 199 172];
r=[21 21 75 121 121 75];
J=roifill(I,c,r);           %对特定区域进行填充
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(J)
```

得到的结果如图 1.17 所示。



（a）原始图像　　（b）roifill 对特定区域进行填充

图 1.17　区域填充处理

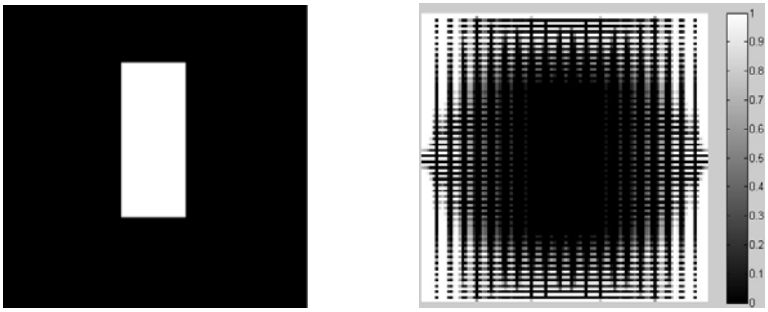
4）图像变换

（1）fft2 函数和 ifft2 函数分别是计算二维的快速傅里叶变换和反变换。

输入以下程序：

```
f=zeros(100,100);
f(20:70,40:60)=1;
imshow(f);
F=fft2(f);            %计算二维的快速傅里叶变换
F2=log(abs(F));       %对幅值取对数
imshow(F2),colorbar
```

得到的结果如图 1.18 所示。



（a）矩形图像　　（b）傅里叶变换幅值

图 1.18　图像的傅里叶变换幅值谱

（2）fftshift 函数实现了补零操作和改变图像显示象限。

输入以下程序：

```
f=zeros(100,100);
f(20:70,40:60)=1;
imshow(f);
F=fft2(f,256,256);
F2=fftshift(F);          %实现补零操作
imshow(log(abs(F2)));
```

得到的结果如图 1.19 所示。

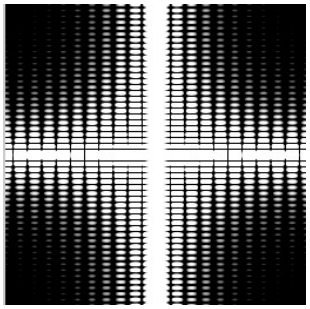


图 1.19　傅里叶变换移位后的结果（零频率分量在中心）

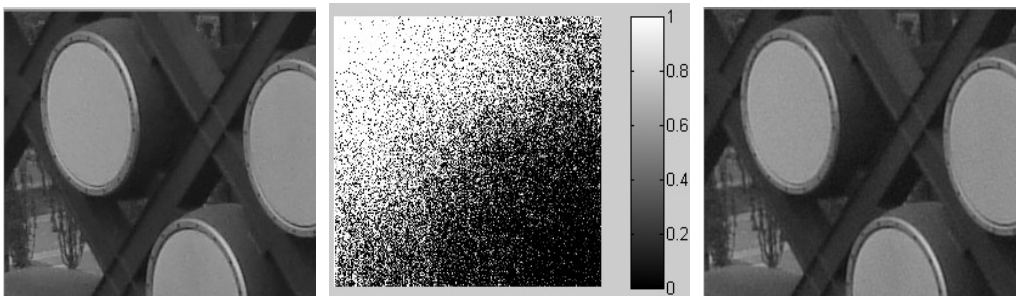
（3）dct2 函数采用基于快速傅里叶变换的算法，用于实现较大输入矩阵的离散余弦变换。

与之对应，idct2 函数实现图像的二维逆离散余弦变换。

输入以下程序：

```
RGB=imread('drum.bmp');
I=rgb2gray(RGB);
J=dct2(I);        %对 I 进行离散余弦变换
imshow(log(abs(J))),colorbar
J(abs(J)<10)=0;
K=idct2(J);       %图像的二维逆离散余弦变换
figure,imshow(I);
figure,imshow(K,[0,255])
```

得到的结果如图 1.20 所示。



（a）原始图像　　（b）执行离散余弦变换的结果　　（c）对压缩后的图像重构

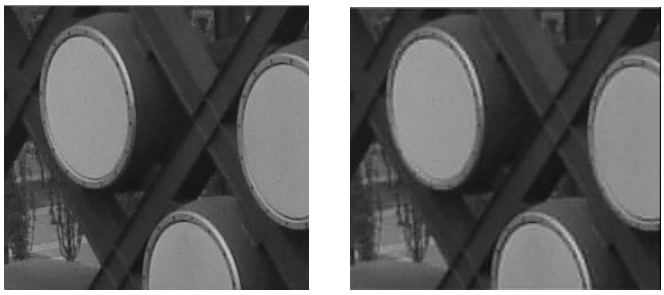
图 1.20　dct2 函数对图像压缩处理

（4）dctmtx 函数用于实现较小输入矩阵的离散余弦变换。

输入以下程序：

```
RGB=imread('drum.bmp');
I=rgb2gray(RGB);
I=im2double(I);
T=dctmtx(8);          %离散余弦变换
B=blkproc(I,[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];
B2=blkproc(B,[8,8],'P1.*x',mask);
I2=blkproc(B2,[8,8],'P1*x*P2',T',T);
imshow(I),figure,imshow(I2);
```

得到的结果如图 1.21 所示。



（a）原始图像　　（b）压缩后重构图像

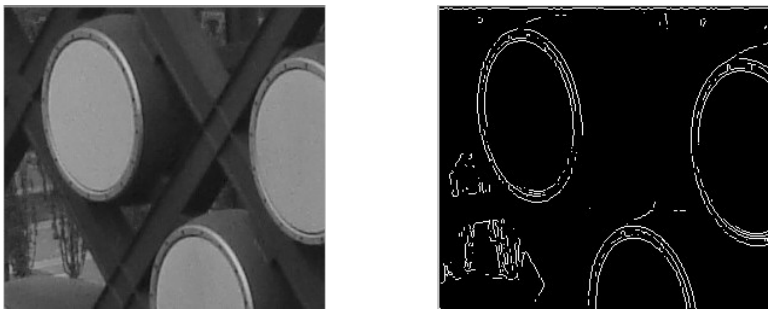
图 1.21　分块压缩图像

（5）edge 函数用于提取图像的边缘。

输入以下程序：

```
RGB=imread('drum.bmp');
I=rgb2gray(RGB);
BW=edge(I);          %提取图像的边缘
imshow(I),figure,imshow(BW);
```

得到的结果如图 1.22 所示。



（a）原始图像　　（b）边缘图像

图 1.22　提取图像的边缘

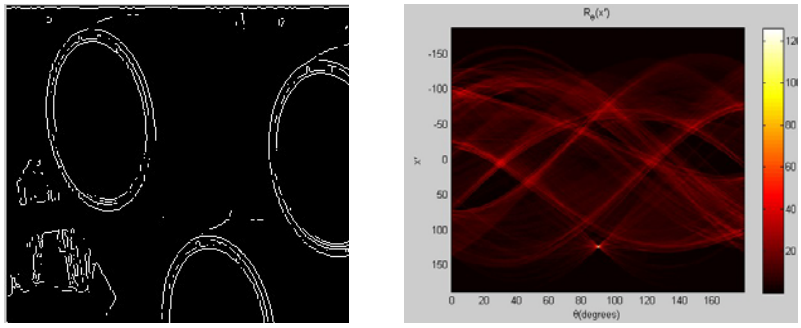
（6）radon 函数用来计算指定方向上图像矩阵的投影。

输入以下程序：

```
RGB=imread('drum.bmp');
I=rgb2gray(RGB);
BW=edge(I);
theta=0:179;
[R,xp]=radon(BW,theta);        %图像矩阵的投影
figure,imagesc(theta,xp,R);colormap(hot);
xlabel('\theta(degrees)');     %对 x 轴加标题
ylabel('x\prime');             %对 y 轴加标题
title('R_{\theta}(x\prime)'); %对图像加标题
```

```
colorbar
```

得到的结果如图 1.23 所示。



（a）边缘图像　　（b）边缘图像的 radon 变换

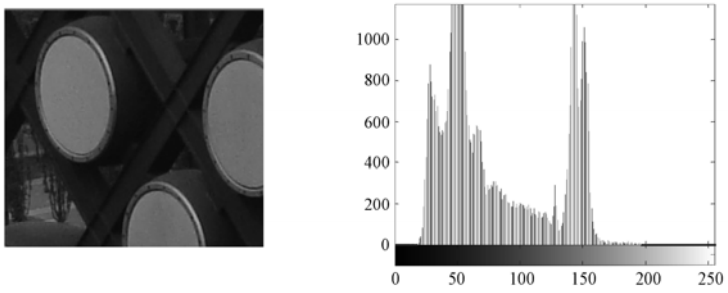
图 1.23　radon 变换

5）图像增强、分割和编码

（1）imhist 函数产生图像的直方图。

```
A=imread('drum.bmp');          %读入图像
B=rgb2gray(A);                 %把 RGB 图像转化成灰度图像
imshow(B);                     %显示灰度图像
imhist(B);                     %显示灰度图像的直方图
```

得到的结果如图 1.24 所示。



（a）原始图像　　（b）灰度图像的直方图

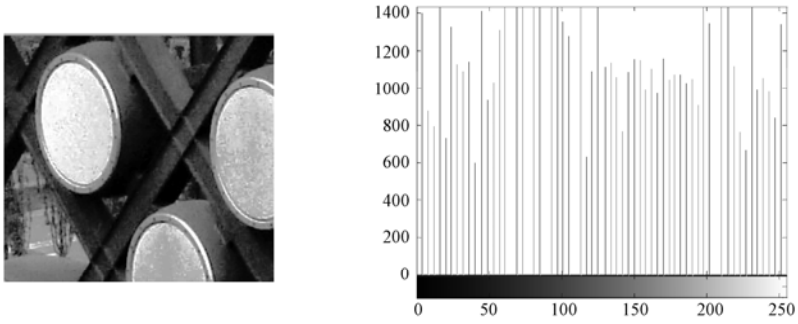
图 1.24　图像的直方图显示

（2）histeq 函数用于对图像的直方图均衡化。

接上面程序输入以下程序：

```
C=histeq(B);       %对图像 B 进行均衡化
imshow(C);         %显示图像
imhist(C);         %得到均衡化后的灰度直方图
```

得到的结果如图 1.25 所示。



（a）原始灰度图像　　（b）均衡化后的灰度直方图

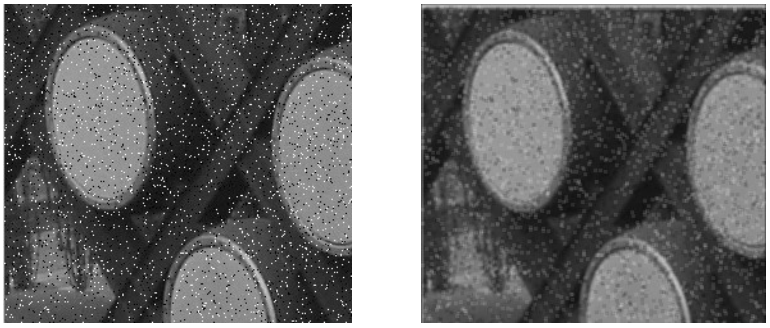
图 1.25　直方图均衡化处理

（3）filter2 函数实现均值滤波。

输入以下程序：

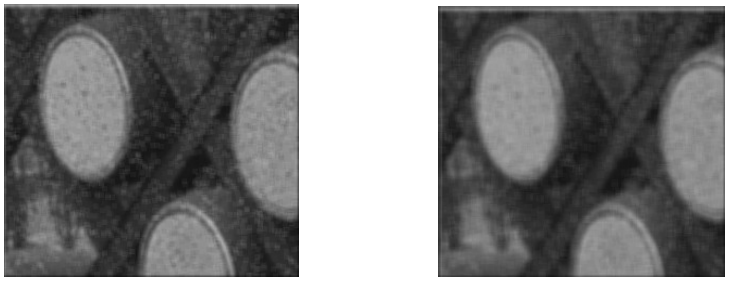
```
a=imread('noise.drum.jpg');
I=rgb2gray(a);
imshow(I);
K1=filter2(fspecial('average',3),I)/255;    %3×3 的均值滤波
K2=filter2(fspecial('average',5),I)/255;    %5×5 的均值滤波
K3=filter2(fspecial('average',7),I)/255;    %7×7 的均值滤波
figure,imshow(K1);
figure,imshow(K2);
figure,imshow(K3);
```

得到的结果如图 1.26 所示。



（a）原始图像　　（b）3×3 的均值滤波

图 1.26　均值滤波



（c）5×5 的均值滤波　　（d）7×7 的均值滤波

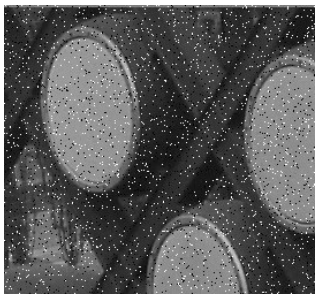
图 1.26　均值滤波（续）

（4）wiener2 函数实现 Wiener（维纳）滤波。

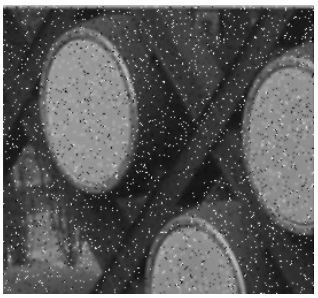
输入以下程序：

```
a=imread('noise.drum.jpg');
I=rgb2gray(a);
imshow(I);
K1=wiener2(I,[3,3]);       %3×3Wiener 滤波
K2=wiener2(I,[5,5]);       %5×5Wiener 滤波
K3=wiener2(I,[7,7]);       %7×7Wiener 滤波
figure,imshow(K1);
figure,imshow(K2);
figure,imshow(K3);
```

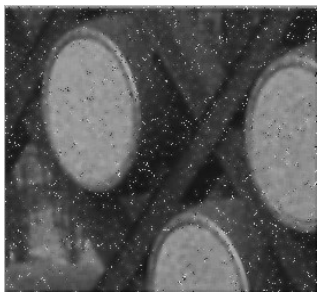
得到的结果如图 1.27 所示。



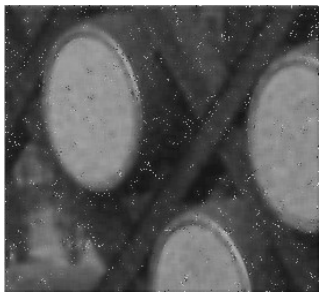
（a）原始图像



（b）3×3 Wiener 滤波



（c）5×5 Wiener 滤波



（d）7×7 Wiener 滤波

图 1.27　Wiener 滤波

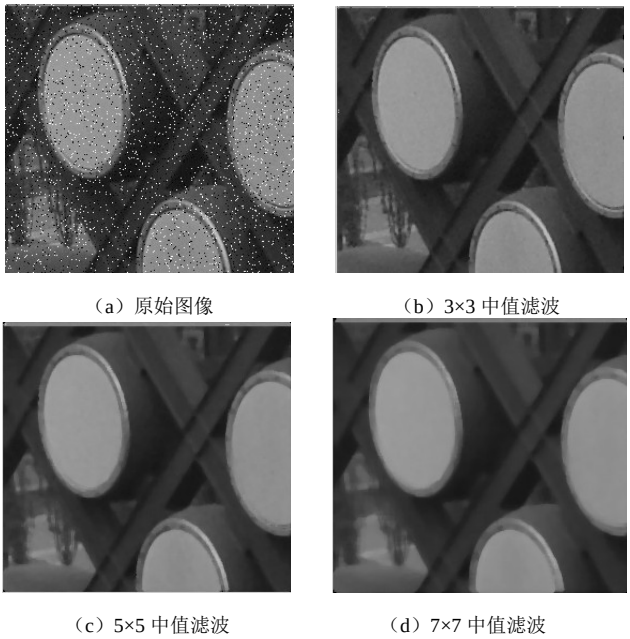
（5）medfilt2 函数实现中值滤波。

输入以下程序：

```
a=imread('noise.drum.jpg');
I=rgb2gray(a);
imshow(I);
K1=medfilt2(I,[3,3]);      %3×3 中值滤波
```

```
K2=medfilt2(I,[5,5]);       %5×5 中值滤波
K3=medfilt2(I,[7,7]);       %7×7 中值滤波
figure,imshow(K1);
figure,imshow(K2);
figure,imshow(K3);
```

得到的结果如图 1.28 所示。



（a）原始图像　（b）3×3 中值滤波

（c）5×5 中值滤波　（d）7×7 中值滤波

图 1.28　中值滤波

6）图像模糊及复原

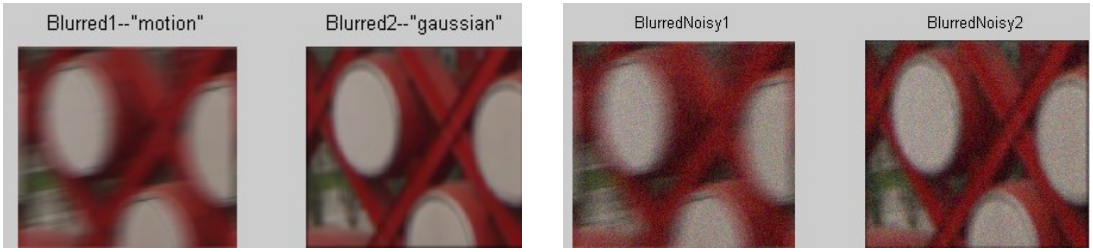
（1）deconvwnr 函数：使用维纳滤波器。

输入以下程序：

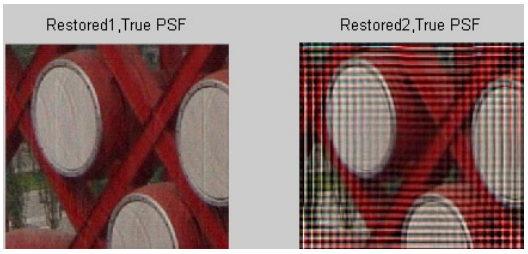
```
%读入图像
I=imread('drum.bmp');
imshow(I);
%对图像进行模糊处理
LEN=31;
THETA=11;
PSF1=fspecial('motion',LEN,THETA);          %运动模糊
PSF2=fspecial('gaussian',10,5);             %高斯噪声模糊
Blurred1=imfilter(I,PSF1,'circular','conv'); %得到运动模糊图像
Blurred2=imfilter(I,PSF2,'conv');           %得到高斯噪声模糊图像
subplot(1,2,1);imshow(Blurred1);title('Blurred1--"motion"');
subplot(1,2,2);imshow(Blurred2);title('Blurred2--"gaussian"');
```

```
%对模糊图像加噪声
V=0.002;
BlurredNoisy1=imnoise(Blurred1,'gaussian',0,V);    %加高斯噪声
BlurredNoisy2=imnoise(Blurred2,'gaussian',0,V);    %加高斯噪声
figure;
subplot(1,2,1);imshow(BlurredNoisy1);title('BlurredNoisy1');
subplot(1,2,2);imshow(BlurredNoisy2);title('BlurredNoisy2');
%进行维纳滤波
wnr1=deconvwnr(Blurred1,PSF1);    %维纳滤波
wnr2=deconvwnr(Blurred2,PSF2);    %维纳滤波
figure;
subplot(1,2,1);imshow(wnr1);title('Restored1,True PSF');
subplot(1,2,2);imshow(wnr2);title('Restored2,True PSF');
```

得到的结果如图 1.29 所示。



（a）运动模糊图像　（b）高斯噪声模糊图像　（c）模糊化图像后加高斯噪声



（d）维纳滤波复原

图 1.29　图像模糊及复原

（2）deconvreg 函数：使用约束最小二乘滤波器。

接以上维纳滤波中的变量定义，输入以下程序：

```
NP=V*prod(size(I));
reg1=deconvreg(BlurredNoisy1,PSF1,NP);  %约束最小二乘滤波
reg2=deconvreg(BlurredNoisy2,PSF2,NP);  %约束最小二乘滤波
figure;
subplot(1,2,1);imshow(reg1);
title('Restored1 with NP');
```

```
subplot(1,2,2);imshow(reg2);
title('Restored2 with NP');
```

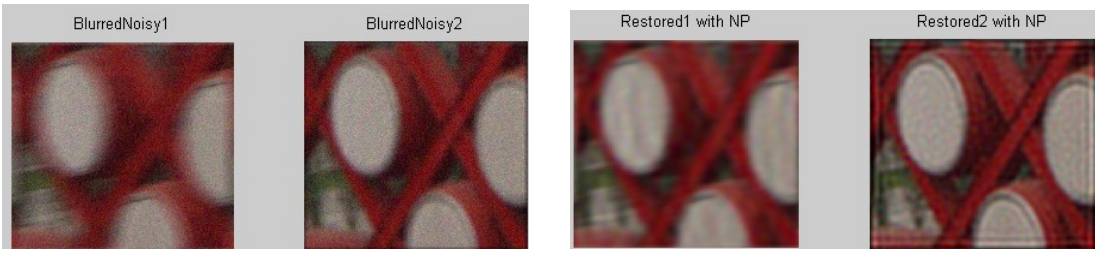
得到的结果如图 1.30 所示。

（3）deconvlucy 函数：使用 Lucy-Richardson 滤波器。

接以上维纳滤波中的变量定义，输入以下程序：

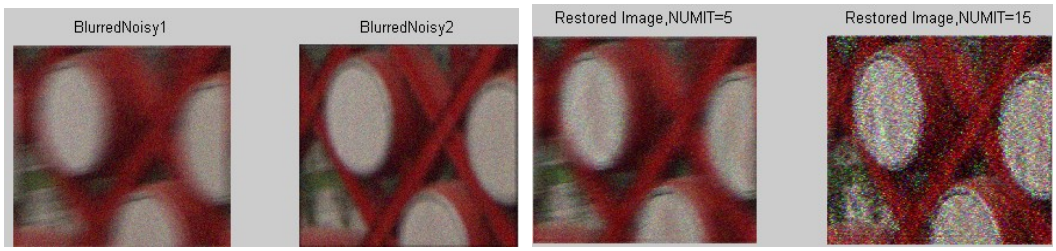
```
luc1=deconvlucy(BlurredNoisy1,PSF1,5);  %使用 Lucy-Richardson 滤波
luc2=deconvlucy(BlurredNoisy1,PSF1,15); %使用 Lucy-Richardson 滤波
figure;
subplot(1,2,1);imshow(luc1);
title('Restored Image,NUMIT=5');
subplot(1,2,2);imshow(luc2);
title('Restored Image,NUMIT=15');
```

得到的结果如图 1.31 所示。



（a）运动模糊图像　（b）高斯噪声模糊图像　（c）用 PSF 函数和噪声强度作为参数的约束最小二乘滤波复原

图 1.30　约束最小二乘滤波复原



（a）运动模糊图像　（b）高斯噪声模糊图像　（c）5 次迭代结果　（d）15 次迭代结果

图 1.31　使用 Lucy-Richardson 滤波器滤波复原结果

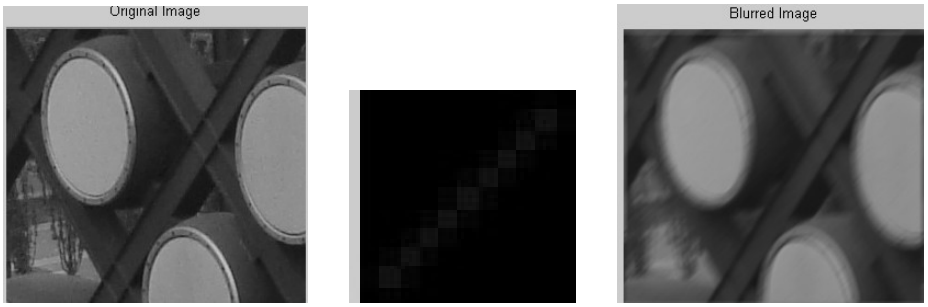
（4）deconvblind 函数：使用盲卷积算法。

输入以下程序：

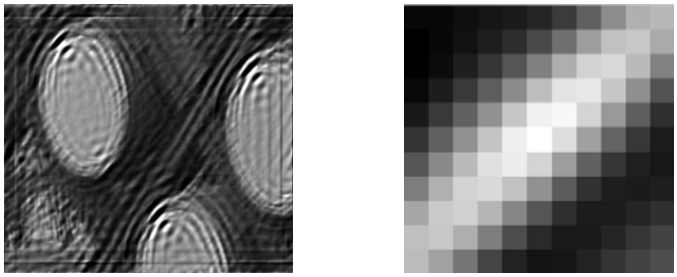
```
a=imread('drum.bmp');
I=rgb2gray(a);
figure;imshow(I);
title('Original Image');
PSF=fspecial('motion',13,45);          %运动模糊
figure;imshow(PSF);
```

```
Blurred=imfilter(I,PSF,'circ','conv');      %得到运动模糊图像
figure;imshow(Blurred);title('Blurred Image');
INITPSF=ones(size(PSF));
[J,P]=deconvblind(Blurred,INITPSF,30);      %使用盲卷积
figure;imshow(J);
figure;imshow(P,[],'notruesize');
```

得到的结果如图 1.32 所示。



（a）原始图像　　（b）真实的 PSF　　（c）模糊化的图像



（d）用盲卷积算法初步复原的图像　　（e）初步重建使用的 PSF

图 1.32　盲卷积算法所得图像