第3章 基于系统对象的编程

3.1 基于系统对象编程的介绍

MATLAB的计算机视觉系统工具箱中的一大特点就是采用系统对象(System Object)进行编程,其提供了涉及视频显示、视频读写、特征检测、提取与匹配、目标检测、运动分析与跟踪、分析与增强、图像转换、滤波、几何变换、数学形态学操作、统计、添加文字和绘图、图像变换等方面的功能。

采用系统对象进行编程的主要步骤包括:

- 创建系统对象;
- 设置系统对象属性;
- 运行系统对象。

现通过下面实例说明采用系统对象进行编程的步骤,下面程序的功能是采用系统对象编程的形式实现快速傅里叶变换。

步骤1:创建系统对象。

```
H = vision.FFT      % 创建一个默认的系统 FFT 对象 H,H 实现的功能
                    % 与 vision.FFT 相同
```

输入上述指令后,命令行窗口中会显示:

```
H =

   System: vision.FFT

   Properties:
     FFTImplementation: 'Auto'
     BitReversedOutput: false
             Normalize: false

   Show fixed-point properties
```

因此，其可以设置的属性包括 FFTImplementation、BitReversedOutput、Normalize。
接着，在命令行窗口中输入：

```
%创建输入数据
Fs = 1000;                  %采样频率
T = 1/Fs;                   %采样时间
L = 1024;                   %信号程度
t = (0:L-1)*T;              %时间向量

%生成待处理的数据向量
X = 0.7*sin(2*pi*50*t.') + sin(2*pi*120*t.');
```

步骤 2：设置系统对象属性。

```
H.Normalize = true %将 Normalize 的属性设置成 true
```

修改后，命令行窗口会显示：

```
H =

  System: vision.FFT

  Properties:
    FFTImplementation: 'Auto'
    BitReversedOutput: false
            Normalize: true
```

由此可见，Normalize 的属性已经被设置成 true。
步骤 3：运行系统对象。

```
Y = step(H,X);                  %运行系统对象
```

注意：在运行“Y = step(A,B);”时，A 为系统对象，B 为待处理的数据。
在赋值时，也可以采用以下格式：

```
系统对象名(属性名,值的形式)
```

因此，上述程序又可以写成

```
H=vision.FFT('Normalize',true);
Fs = 1000;
T = 1/Fs;
L = 1024;
t=(0:L-1)*T;
X = 0.7*sin(2*pi*50*t.') + sin(2*pi*120*t.');
Y=step(H,X);
```

此外，还可以不创建 H，直接调用系统对象 vision.FFT 进行处理。
因此，程序还可以写为

```
Fs = 1000;
T = 1/Fs;
L = 1024;
t = (0:L-1)*T;
X = 0.7*sin(2*pi*50*t.') + sin(2*pi*120*t.');
Y = step(vision.FFT('Normalize',true),X);
```

下面通过例程 3.1.1、例程 3.1.2 来进一步体会基于系统对象 vision.X 的图像处理。

例程 3.1.1 的功能是对输入图像进行二值化处理，并对二值化的图像取反，其运行结果如图 3.1.1 所示。

例程 3.1.1

```
**********************************************************************
%定义系统对象
himgcomp = vision.ImageComplementer;
hautoth = vision.Autothresholder;
%读入图像
I = imread('coins.png');
%将读入的图像转换为二值图像
bw = step(hautoth,I);
%对转换后的二值图像取反
Ic = step(himgcomp,bw);
%显示结果
figure;
subplot(2,1,1),imshow(bw),title('Original Binary image')
subplot(2,1,2),imshow(Ic),title('Complemented image')
**********************************************************************
```

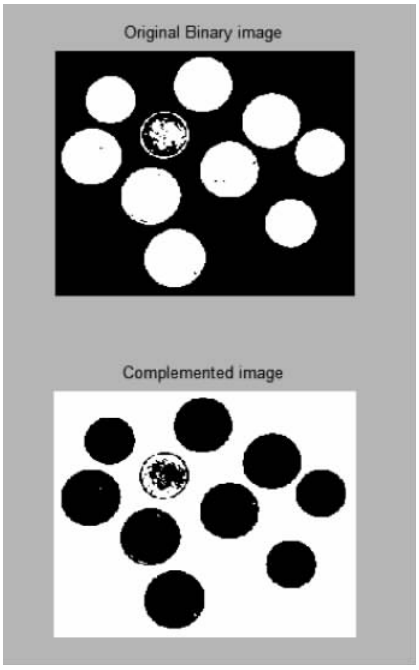


图 3.1.1　例程 3.1.1 的运行结果

例程 3.1.2 的功能是对输入的一段视频进行边缘检测，其运行结果如图 3.1.2 所示。

例程 3.1.2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%定义系统对象
hVideoSrc = vision.VideoFileReader('vipmen.avi','ImageColorSpace','Intensity');
hEdge = vision.EdgeDetector('Method','Prewitt','ThresholdSource','Property','Threshold',
15/256,'EdgeThinning',true);
%创建显示窗口
WindowSize = [190 150];
hVideoOrig = vision.VideoPlayer('Name','Original');
hVideoOrig.Position = [10 hVideoOrig.Position(2) WindowSize];
hVideoEdges = vision.VideoPlayer('Name','Edges');
hVideoEdges.Position = [210 hVideoOrig.Position(2) WindowSize];
%对视频的每一帧进行边缘检测并显示
while ~isDone(hVideoSrc)
   frame = step(hVideoSrc);                %读入视频
   edges = step(hEdge,frame);              %对视频的每一帧进行边缘检测
   step(hVideoOrig,frame);                 %显示输入视频的每一帧
   step(hVideoEdges,edges);                %显示边缘检测的结果
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

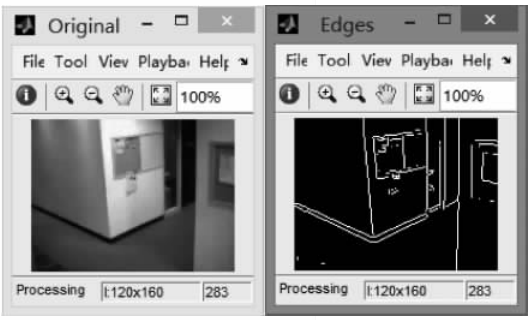


图 3.1.2　例程 3.1.2 的运行结果

采用基于系统对象 vision.X 的图像处理，与采用数字图像处理工具箱相比，其优势主要体现在以下两个方面：

(1) 运行速度更快。

(2) 绝大多数系统对象支持 MATLAB 的 C/C++代码转换功能，可以将其快速地转换为可以运行的 C 代码。

3.2　图像直方图的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision.Histogram 可实现对输入灰度图像的灰度直方图统计。

vision. Histogram 的具体使用方法如下：

功能：对输入的灰度图像输出其直方图矩阵。

语法：A＝step(vision. Histogram,Img)；

其中：Img 为输入图像；A 是输出的直方图矩阵。

属性：

LowerLimit：直方图的下限，默认值为 0。

UpperLimit：直方图的上限，默认值为 1。

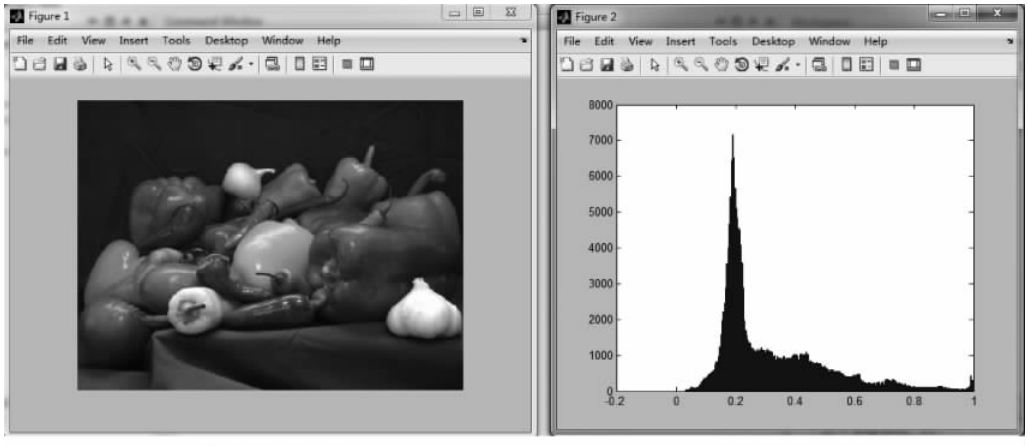
NumBins：直方图的条数，默认值为 256。

Normalize：是否对直方图进行归一化处理，可以将其设置成 true 或 false。

下面通过例程 3.2.1 来具体说明 vision. Histogram 的具体使用方法，其运行结果如图 3.2.1 所示。

例程 3.2.1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%读入待转换的彩色图像,并将其转换成灰度图像
I = rgb2gray(imread('peppers.png'));
%显示灰度图像
imshow(I)
%将图像数据类型转换成单精度型
img = im2single(I);
%定义系统对象
hhist2d = vision.Histogram;
%运行系统对象,求其灰度直方图
y = step(hhist2d,img);
figure
%显示灰度直方图
bar((0:255)/256,y);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



(a) 输入的灰度图像　　(b) 灰度图像直方图

图 3.2.1　例程 3.2.1 的运行结果

3.3 图像色彩空间变换的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision. ColorSpaceConverter 可实现对输入图像的色彩空间变换。

vision. ColorSpaceConverter 的具体使用方法如下：

功能：对输入的图像进行色彩空间转换。

语法：A＝step(vision. ColorSpaceConverter，Img)；

其中：Img 为输入图像；A 是经转换在其他色彩空间的图像。

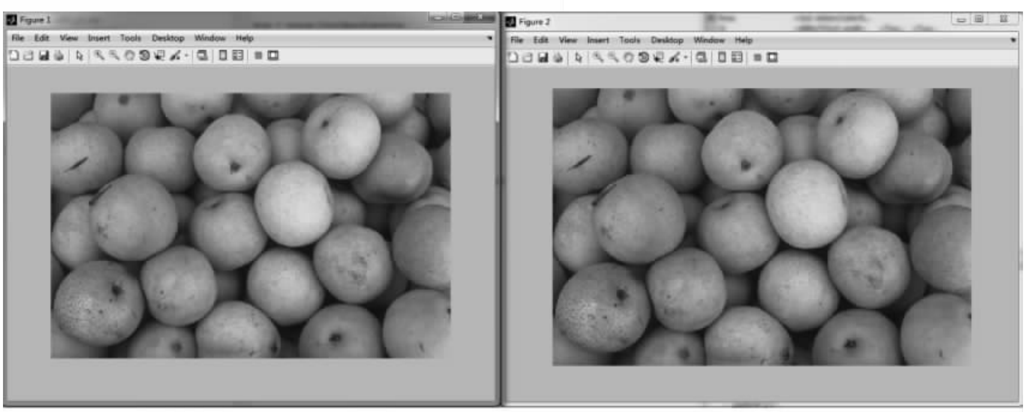
属性：

Conversion：通过对该属性进行设置，可以实现不同图像空间的转换，其包括 RGB to YCbCr、YCbCr to RGB、RGB to intensity、RGB to HSV、HSV to RGB、sRGB to XYZ、XYZ to sRGB、sRGB to L＊a＊b＊、L＊a＊b＊ to sRGB。

下面通过例程 3.3.1 来具体说明 vision. ColorSpaceConverter 的具体使用方法，其运行结果如图 3.3.1 所示。

例程 3.3.1

```
*****************************************************************
%读入图像并显示
i1 = imread('pears.png');
imshow(i1);
%创建系统对象
hcsc = vision.ColorSpaceConverter;
%设置系统对象属性
hcsc.Conversion = 'RGB to intensity';          %将 RGB 空间转换成灰度空间
%进行转换
i2 = step(hcsc,i1);
%显示转换后的结果
figure
imshow(i2);
*****************************************************************
```



(a) RGB空间中的图像　　(b) 灰度空间中的图像

图 3.3.1　例程 3.3.1 的运行结果

3.4 图像缩放的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision.GeometricScaler 可实现对输入图像的缩放变换。

vision.GeometricScaler 的具体使用方法如下：

功能：对图像进行几何尺寸的缩放。

语法：A = step(vision.GeometricScaler,Img)；

其中：Img 为原始图像；A 是缩放后的图像。

SizeMethod：图像尺寸缩放的方法。Output size as a percentage of input size，对输入的图像按照一定比例缩放；Number of output columns and preserve aspect ratio，按照输出图像的列数以及由其确定的比例进行缩放；Number of output rows and preserve aspect ratio，按照输出图像行数以及由其确定的比例进行缩放；Number of output rows and columns，按照输出图像的行数和列数进行缩放。

ResizeFactor：行列缩放比例，只有将 SizeMethod 设置为 Output size as a percentage of input size 时，ResizeFactor 属性才有效。可用一个数组[a,b]对 ResizeFactor 进行设置，a 为图像行的缩放系数，b 为图像列的缩放系数，默认值为[200,150]。

NumOutputColumns：输出图像列的值。只有将 SizeMethod 设置为 Number of output columns and preserve aspect ratio 时，NumOutputColumns 属性才有效。其默认值为 25。

NumOutputRows：输出图像行的值。只有将 SizeMethod 设置为 Number of output rows and preserve aspect ratio 时，NumOutputRows 属性才有效。其默认值为 25。

Size：输出图像的大小。只有将 SizeMethod 设置为 Number of output rows and columns 时，Size 属性才有效。可用一个数组[a,b]对 size 进行设置，a 为输出图像的行数，b 为输出图像的列数，默认值为[25,35]。

InterpolationMethod：插值方法选择。Nearest neighbor，最邻近插值；Bilinear，双线性插值；Bicubic，立方插值；Lanczos2，16-邻域插值；Lanczos3，36-邻域插值。

Antialiasing：当缩放图像时低通滤波器使能。当 Antialiasing 被设置为 true 时，在缩放图像之前，采用低通滤波器对图像进行滤波。

例程 3.4.1 是调用系统对象 vision.GeometricScaler 对图像进行缩放的程序，其运行结果如图 3.4.1 所示。

例程 3.4.1

```
*****************************************************************
%读入图像
I = imread('cameraman.tif');
%创建系统对象
 hgs = vision.GeometricScaler;
%设置系统对象属性
hgs.SizeMethod = ...
```

```
    'Output size as a percentage of input size';      % 对输入的图像按照一定比例缩放
hgs.InterpolationMethod = 'Bilinear';                 % 采用双线性插值
% 运行系统对象
J = step(hgs,I);
% 显示原始图像与处理后的结果
 imshow(I); title('Original Image');
 figure,imshow(J);title('Resized Image');
 ****************************************************************
```

图 3.4.1　例程 3.4.1 的运行结果

3.5　图像平移的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision.GeometricTranslator 可实现对输入图像的平移变换。

vision.GeometricTranslator 的具体使用方法如下：

功能：将输入图像进行平移。

语法：A = step(vision.GeometricTranslator,Img)

其中：Img 为原始图像；A 是平移后的图像。

属性：

OutputSize：输出图像的尺寸。当该属性设置为 Full 时，输出的图像比输入图像尺寸大，保证会将平移后的图像显示完整；当该属性设置为 Same as input image 时，输出图像的尺寸和输入图像的尺寸相同，但只能显示图像的一部分。该属性的默认值为 Full。

OffsetSource：选择平移值通过何种方式输入。当设置为 Property 时，通过设置属性

Offset 的值确定平移量；当设置为 Input port 时，则通过输入接口进行平移量设置。

Offset：平移量。可以用数组[a，b]对其进行设置，a 为竖直偏移量（以向下移动为正），b 为水平偏移量（以向右移动为正）。默认值为[1.5 2.3]。

MaximumOffset：最大偏移量。可以用数组[a，b]对其进行设置，a 为最大竖直偏移量，b 为最大水平偏移量。只有 OutputSize 设置为 Full 且 OffsetSource 设置为 Input port 时此属性才有效。默认值为[8，10]。

BackgroundFillValue：背景图像填充值。默认值为 0。

InterpolationMethod：插值方法设置。Nearest neighbor，最邻近插值；Bilinear，双线性插值；Bicubic，立方插值。默认值为 Bilinear。

例程 3.5.1 是调用系统对象 vision.GeometricTranslator 对图像进行平移的程序，其运行结果如图 3.5.1 所示。

例程 3.5.1

```
*****************************************************************
%创建系统对象
htranslate = vision.GeometricTranslator;
%设置系统对象属性
htranslate.OutputSize = 'Same as input image';        %输出图像的大小与输入相同
htranslate.Offset = [30 30];                          %在 X、Y 轴上的偏移量各为 30 个像素
%读入图像，并转换成单精度性
Img = imread('cameraman.tif');
I = im2single(Img);
%运行系统对象
Y = step(htranslate,I);
%显示结果
subplot(1,2,1),imshow(Img);
subplot(1,2,2),imshow(Y);
*****************************************************************
```

图 3.5.1　例程 3.5.1 的运行结果

3.6 图像旋转的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision. GeometricRotator 可实现对输入图像的旋转变换。

vision. GeometricRotator 的具体使用方法如下：

功能：按照指定的角度旋转图像。

语法：A = step(vision. GeometricRotator,Img)；

其中：Img 为原始图像；A 是旋转后的图像。

其参数性能如下：

OutputSize：输出图像的尺寸。若设置为 Expanded to fit rotated input image，则将旋转后的图像进行扩充；若设置为 Same as input image，则旋转后输出图像的尺寸与原输入图像相同。默认值为 Expanded to fit rotated input image。

AngleSource：选择旋转角度来源。若选择 Property，则旋转角度来源于该系统对象的性质 Angle，此时通过 Y = step(vision. GeometricRotator,IMG)来运行该系统对象；若选择 Input port，则通过在输入接口中 Angle 的值来旋转图像，通过 Y = step(HROTATE,IMG,ANGLE)来运行该系统对象。

Angle：图像旋转角度，默认值为 pi/6。

MaximumAngle：最大旋转角度，默认值是 pi。

RotatedImageLocation：设定如何旋转。若设定为 Top-left corner，则以左上角的顶点为旋转点进行旋转；若设定为 Center，则以该图像的中心点为旋转点进行旋转。默认值为 Center。

SineComputation：设定如何计算旋转。若设定为 Trigonometric function，则为计算法；若设定为 Table lookup，则为查表法。

BackgroundFillValue：设定背景填充值。默认值为 0(黑色)。

InterpolationMethod：设定插值方式。可设置为最邻近插值 Nearest neighbor、双线性插值 Bilinear、三次插值 Bicubic。

例程 3.6.1、例程 3.6.2 是调用系统函数 vision. GeometricRotator 对图像进行旋转的程序，其运行结果分别如图 3.6.1 和图 3.6.2 所示。

例程 3.6.1

```
*****************************************************************
%读入图像并转换成为双精度型
img = im2double(imread('peppers.png'));
%创建系统对象
hrotate = vision.GeometricRotator;
%设定旋转角度为 pi / 6
hrotate.Angle = pi / 6;
%执行系统对象
rotimg = step(hrotate,img);
%显示旋转后的图像
imshow(rotimg);
*****************************************************************
```

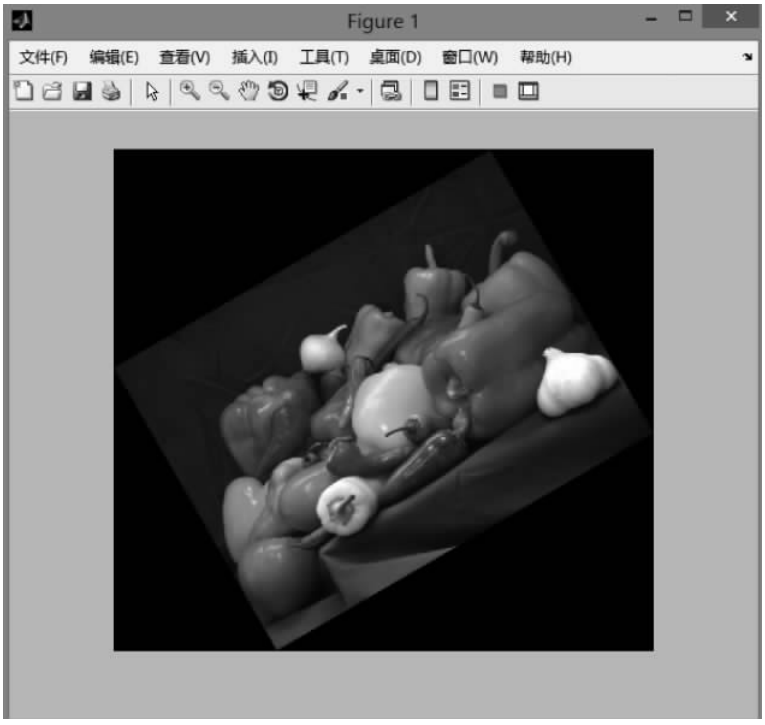


图 3.6.1　例程 3.6.1 的运行结果

例程 3.6.2

```
*****************************************************************
%创建系统对象
hrotate2 = vision.GeometricRotator;
%设定系统对象的属性
hrotate2.AngleSource = 'Input port';                %从输入接口中输入旋转角度
hrotate2. OutputSize = 'Same as input image';       %设定旋转后的图像大小与输入相同
%读入 RGB 图像并将其转换成为双精度灰度图像
img2 = im2double(rgb2gray(imread('onion.png')));
%显示
figure,imshow(img2)
%运行系统对象
rotimg2 = step(hrotate2,img2,pi/4);
figure,imshow(rotimg2);
*****************************************************************
```

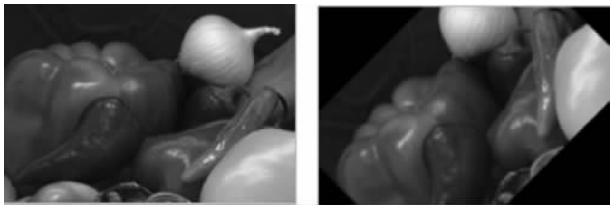


图 3.6.2　例程 3.6.2 的运行结果

3.7 图像傅里叶变换的编程实现

在 MATLAB 中，可以调用计算机视觉工具箱中的 vision.FFT 来实现对输入灰度图像的快速傅里叶变换。

vision.FFT 的具体使用方法如下：

功能：对输入的灰度图像进行快速傅里叶变换。

语法：A＝step(vision.FFT,Img)；

其中：Img 为原始图像；A 是傅里叶变换后的图像。

属性：

FFTImplementation：FFT 的执行方式。可设置为 Auto、Radix-2、FFTW。默认值为 Auto。若将其设置为 Radix-2，则输入图像矩阵的行数、列数必须为 2^n。

BitReversedOutput：可以将其设置为 false 或 true，其默认值为 false。

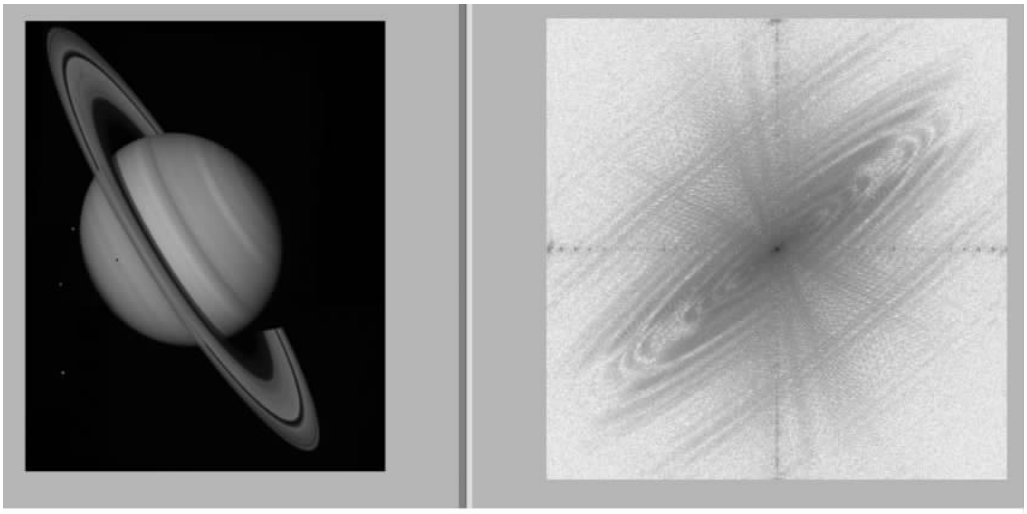
Normalize：是否对输出图像进行归一化处理，可以将其设置为 false 或 true，默认值为 false。

例程 3.7.1 是调用 vision.FFT 进行二维图像傅里叶变换的例子，其运行结果如图 3.7.1 所示。

例程 3.7.1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%定义系统对象
hfft2d = vision.FFT;                          %用于进行傅里叶变换
hcsc = vision.ColorSpaceConverter('Conversion','RGB to intensity');    %用于色彩空间转换
hgs = vision.GeometricScaler('SizeMethod','Number of output rows and columns','Size',[512 512]);
                                              %用于改变图像的大小
%读入 RGB 图像
x = imread('saturn.png');
imshow(x)
%将读入的图像转变为 512×512 大小的图像
x1 = step(hgs,x);
%将 RGB 图像转换为灰度图像
ycs = step(hcsc,x1);
%对图像进行傅里叶变换
y = step(hfft2d,ycs);
%使变换后的零频率分量位于中心
y1 = fftshift(double(y));
figure
%显示结果
imshow(log(max(abs(y1),1e-6)),[]);
colormap(jet(64));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

在 MATLAB 中，可以调用计算机视觉工具箱中的 vision.IFFT 来实现对输入频域图像的快速傅里叶逆变换。



(a) 输入图像　　(b) 傅里叶频谱

图 3.7.1　例程 3.7.1 的运行结果

vision.IFFT 的具体使用方法如下：

功能：对输入的频域图像进行傅里叶逆变换。

语法：A＝step(vision.IFFT,Img)

其中：Img 为原始图像；A 是傅里叶逆变换后的图像。

属性：

FFTImplementation：FFT 的执行方式。可设置为 Auto、Radix-2、FFTW。默认值为 Auto。若将其设置为 Radix-2，则输入图像矩阵的行数、列数必须为 2^n。

BitReversedOutput：可以将其设置为 false 或 true，其默认值为 false。

ConjugateSymmetricInput：可以将其设置为 false 或 true，其默认值为 true。

Normalize：是否对输出图像进行归一化处理，可以将其设置为 false 或 true，默认值为 false.

例程 3.7.2 是调用 vision.IFFT 进行二维图像傅里叶逆变换的例子，其运行结果如图 3.7.2 所示。

例程 3.7.2

```
*****************************************************************
%定义系统对象
hfft2d = vision.FFT;                %用于进行傅里叶变换
hifft2d = vision.IFFT;              %用于进行傅里叶逆变换

%读入图像,并转换成单精度型
   xorig = single(imread('cameraman.tif'));

%将时域图像转换到频域
   Y = step(hfft2d,xorig) ;         %运行系统对象 hfft2d
   imshow(Y)
```

```
%将频域图像转换到时域
xtran = step(hifft2d,Y) ;        %运行系统对象 hifft2d

%显示结果
   figure
   imshow(abs(xtran),[]);
*****************************************************************
```

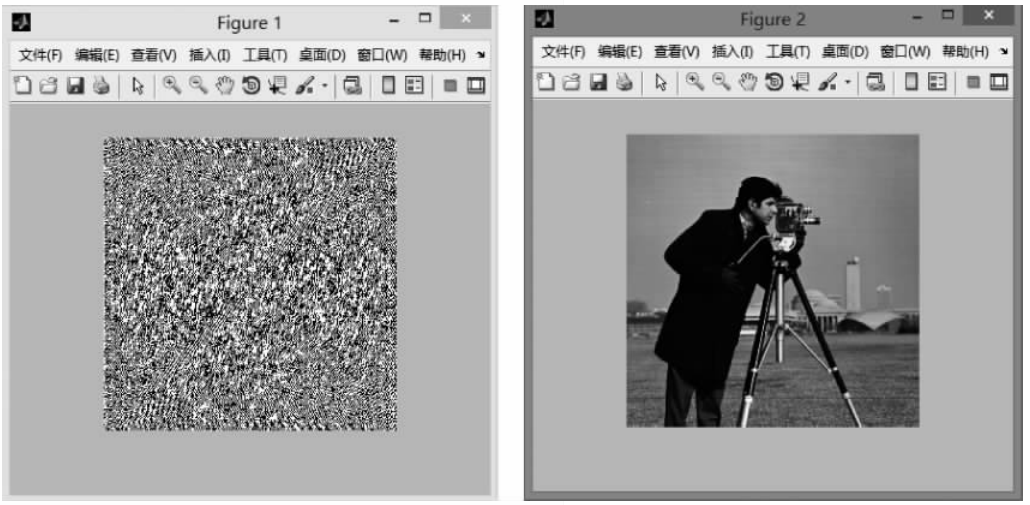


图 3.7.2　例程 3.7.2 的运行结果

3.8　图像余弦变换的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision.DCT 可实现对输入图像的离散余弦变换；调用 vision.IDCT 可实现离散余弦逆变换。

vision.DCT 的具体使用方法如下：

功能：对输入的灰度图像进行离散余弦变换。

语法：A＝step(vision.DCT,Img)；

其中：Img 为原始图像；A 是离散余弦变换后的图像。

属性：

SineComputation：如何计算正弦、余弦值。若设定为 Trigonometric function，则为计算法；若设定为 Table lookup，则为查表法。

vision.IDCT 的具体使用方法如下：

功能：对输入的灰度图像进行离散余弦逆变换。

语法：A＝step(vision.IDCT,Img)

其中：Img 为原始图像；A 是离散余弦逆变换后的图像。

属性：

SineComputation：如何计算正弦、余弦值。若设定为 Trigonometric function 则为计

算法；若设定为 Table lookup，则为查表法。默认值为 Table lookup 查表法。

例程 3.8.1 是调用系统函数对图像进行离散余弦变换及重构的程序，其运行结果如图 3.8.1 所示。

例程 3.8.1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%创建系统对象
hdct2d = vision.DCT;
%读入图像并转换成双精度型
I = double(imread('cameraman.tif'));
%运行系统对象，将输入图像进行离散余弦变换
J = step(hdct2d,I);
%显示原始图像及变换后的余弦系数
subplot(1,3,1),imshow(I,[0 255]),title('原始图像')
subplot(1,3,2),imshow(log(abs(J)),[]),colormap(jet(64)),colorbar,title('离散余弦系数')
%创建系统对象
hidct2d = vision.IDCT;
%将小于 10 的部分置零
J(abs(J) < 10) = 0;
%进行余弦逆变换(重构)
It = step(hidct2d,J);
%显示重构后的图像
subplot(1,3,3),imshow(It,[0 255]),title('重构后的图像')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

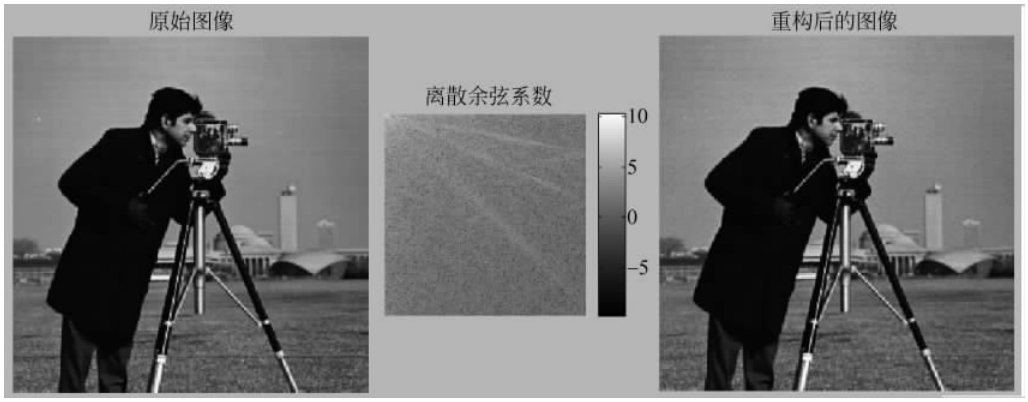


图 3.8.1　例程 3.8.1 的运行结果

3.9　图像腐蚀、膨胀的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的系统对象 vision.MorphologicalDilate 可实现对输入图像的膨胀运算，调用 vision.MorphologicalErode 可实现对输入图像的腐蚀运算。

vision. MorphologicalDilate 的具体使用方法如下：

功能：对输入的图像进行膨胀操作。

语法：A＝step(vision. MorphologicalDilate,Img)；

其中：Img 为原始图像；A 是膨胀操作后的图像。

属性：

NeighborhoodSource：结构元素输入的方式。如果设置为 Property,则通过设置系统属性参数 Neighborhood 实现；如果设置为 Input por,则在运行系统对象时,通过输入接口矩阵输入,具体方式为 A＝step(vision. MorphologicalDilate,Img,B),B 为输入接口矩阵。该属性的默认值为 Property。

Neighborhood：结构元素矩阵。当 NeighborhoodSource 的属性设置为 Property 时,该属性参数有效。该属性的默认值为[1 1 1 1]。

例程 3.9.1 是调用系统函数 vision. MorphologicalDilate 实现对输入图像的膨胀操作的程序,其运行结果如图 3.9.1 所示。

例程 3.9.1

```
**********************************************************************
%读入图像
x = imread('peppers.png');
%设置系统对象属性
hcsc = vision.ColorSpaceConverter;
hcsc.Conversion = 'RGB to intensity';
hautothresh = vision.Autothresholder;
hdilate = vision.MorphologicalDilate('Neighborhood',ones(5,5));
%运行系统对象
x1 = step(hcsc,x);                    %将 RGB 图像转换成灰度图像
x2 = step(hautothresh,x1);            %将灰度图像转换成二值图像
y = step(hdilate,x2);                 %对二值图像进行膨胀运算
%显示结果
figure;
subplot(1,3,1),imshow(x); title('原始图像');
subplot(1,3,2),imshow(x2); title('二值图像');
subplot(1,3,3),imshow(y); title('膨胀图像');
**********************************************************************
```

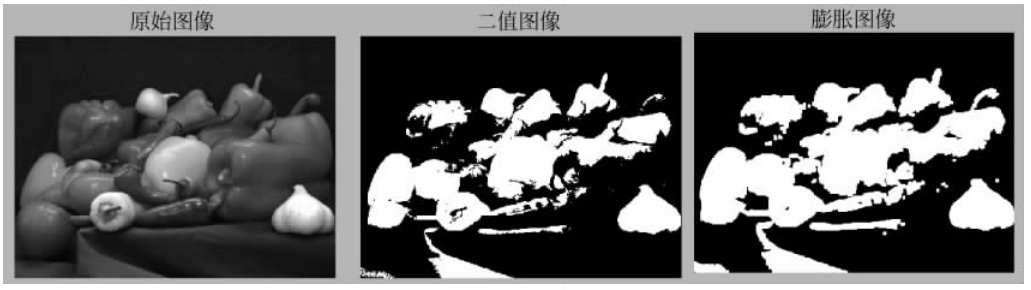


图 3.9.1　例程 3.9.1 的运行效果

vision. MorphologicalErode 的具体使用方法如下：

功能：对输入的图像进行腐蚀操作。

语法：A＝step(vision. MorphologicalErode,Img)；

其中：Img 为原始图像；A 是腐蚀操作后的图像。

属性：

NeighborhoodSource：结构元素输入的方式。如果设置为 Property,则通过设置系统属性参数 Neighborhood 实现；如果设置为 Input port,则在运行系统对象时,通过输入接口矩阵输入,具体方式为 A＝step(vision. MorphologicalErode,Img,B),B 为输入接口矩阵。该属性的默认值为 Property。

Neighborhood：结构元素矩阵。当 NeighborhoodSource 的属性设置为 Property 时,该属性参数有效。该属性的默认值为 strel('square',4)。

例程 3.9.2 是调用系统函数 vision. MorphologicalErode 实现对输入图像的腐蚀操作的程序,其运行结果如图 3.9.2 所示。

例程 3.9.2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%读入图像
x = imread('peppers.png');
%设置系统对象属性
hcsc = vision.ColorSpaceConverter;
hcsc.Conversion = 'RGB to intensity';
hautothresh = vision.Autothresholder;
herode = vision.MorphologicalErode('Neighborhood',ones(5,5));
%运行系统对象
x1 = step(hcsc,x);                          %将 RGB 图像转换成灰度图像
x2 = step(hautothresh,x1);                  %将灰度图像转换成二值图像
y = step(herode,x2);                        %对二值图像进行腐蚀运算
figure;
subplot(1,3,1),imshow(x); title('原始图像');
subplot(1,3,2),imshow(x2); title('二值图像');
subplot(1,3,3),imshow(y); title('腐蚀图像');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

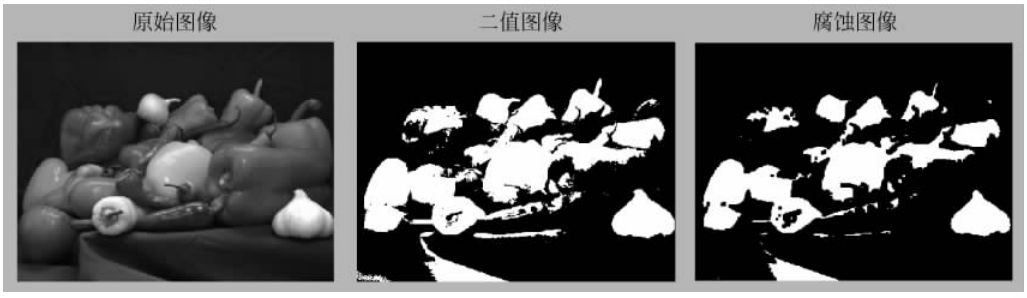


图 3.9.2　例程 3.9.2 的运行效果

3.10 图像开运算、闭运算的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的系统对象 vision. MorphologicalOpen 可实现对输入图像进行开运算，调用 vision. MorphologicalClose 可实现对输入图像进行闭运算。

vision. MorphologicalOpen 的具体使用方法如下：

功能：对输入的图像进行开运算。

语法：A＝step(vision. MorphologicalOpen，Img)。

其中：Img 为原始图像；A 是开运算操作后的图像。

属性：

NeighborhoodSource：结构元素输入的方式。如果设置为 Property，则通过设置系统属性参数 Neighborhood 实现；如果设置为 Input port，则在运行系统对象时，通过输入接口矩阵输入，具体方式为 A＝step(vision. MorphologicalOpen，Img，B)，B 为输入接口矩阵。该属性的默认值为 Property。

Neighborhood：结构元素矩阵。当 NeighborhoodSource 的属性设置为 Property 时，该属性参数有效。该属性的默认值为 strel('disk'，5)。

例程 3.10.1 是调用系统对象 vision. MorphologicalOpen 实现对输入图像进行开运算操作的程序，其运行结果如图 3.10.1 所示。

例程 3.10.1

```
**********************************************************************
%读入图像并转换为单精度型
img = im2single(imread('blobs.png'));
%设置系统对象属性
  hopening = vision.MorphologicalOpen;
 hopening.Neighborhood = strel('disk',5);
%运行系统对象
opened = step(hopening,img);
%显示实验结果
figure;
subplot(1,2,1),imshow(img); title('原始图像');
subplot(1,2,2),imshow(opened); title('开运算后的图像');
**********************************************************************
```

vision. MorphologicalClose 的具体使用方法如下：

功能：对输入的图像进行闭运算。

语法：A＝step(vision. MorphologicalClose，Img)。

其中：Img 为原始图像；A 是闭运算操作后的图像。

属性：

NeighborhoodSource：结构元素输入的方式。如果设置为 Property，则通过设置系统属性参数 Neighborhood 实现；如果设置为 Input port，则在运行系统对象时，通过输入接口矩阵输入，具体方式为 A＝step(vision. MorphologicalClose，Img，B)，B 为输入接口

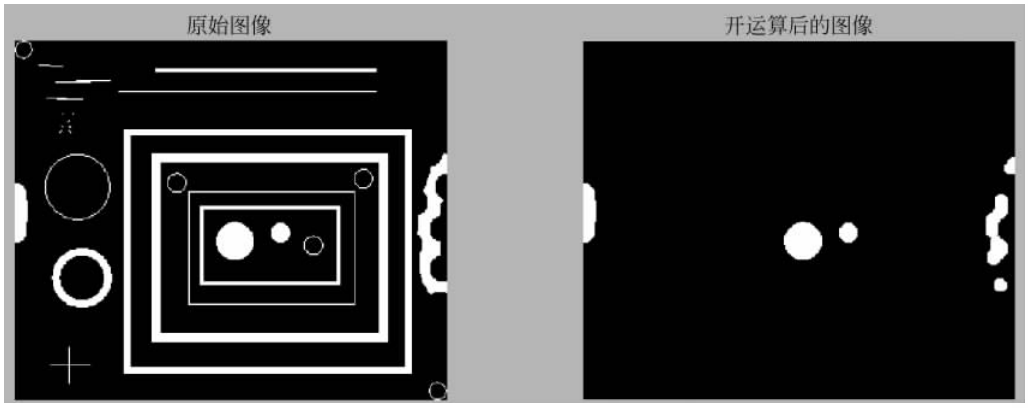


图 3.10.1　例程 3.10.1 的运行结果

矩阵。该属性的默认值为 Property。

Neighborhood：结构元素矩阵。当 NeighborhoodSource 的属性设置为 Property 时，该属性参数有效。该属性的默认值为 strel('line',5,45)。

例程 3.10.2 是调用系统对象 vision.MorphologicalClose 实现对输入图像进行闭运算操作的程序，其运行结果如图 3.10.2 所示。

例程 3.10.2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%读入图像并转换为单精度型
img = im2single(imread('blobs.png'));
%设置系统对象属性
hclosing = vision.MorphologicalClose;
hclosing.Neighborhood = strel('disk',10);
%运行系统对象
closed = step(hclosing,img);
%显示实验结果
figure;
  subplot(1,2,1),imshow(img); title('原始图像');
  subplot(1,2,2),imshow(closed); title('闭运算操作后的结果');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

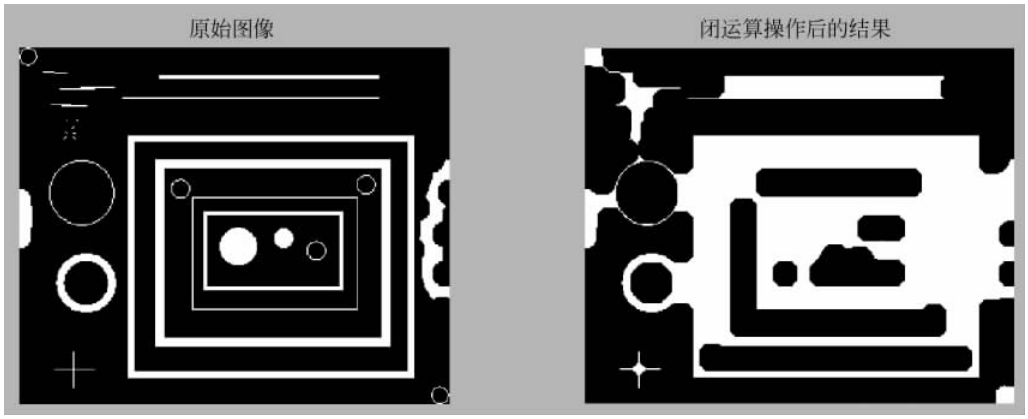


图 3.10.2　例程 3.10.2 的运行结果

3.11 图像中值滤波的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的系统对象 vision.MedianFilter 可实现对输入图像的中值滤波。

系统对象 vision.MedianFilter 的使用方法如下：

功能：对输入的二维图像进行中值滤波。

属性：

NeighborhoodSize：中值滤波器的邻域尺寸。当将其设置为一个整数时，表示邻域尺寸为行、列均为该尺寸的方形矩阵；当将其设置为一个二元素向量时，则表示邻域尺寸为该向量元素数值为行列数的矩阵。其默认值为[3 3]。

OutputSize：输出尺寸。可以将其设置为 Same as input size 或 Valid，默认值为 Same as input size。当将 OutputSize 属性设置为 Valid 时，输出图像的尺寸为

输出图像的行数 = 输入图像的行数 − 邻域行数 + 1

输出图像的列数 = 输入图像的列数 − 邻域列数 + 1

PaddingMethod：输入图像扩充方法。可以将其设置为 Constant、Replicate、Symmetric、Circular。其默认值为 Constant。

PaddingValueSource：输入图像扩充值。当 PaddingMethod 属性设置为 Constant 时，该属性可调。可以将其设置为 Property 或 Input port。其默认值为 Property。

PaddingValue：当 PaddingMethod 属性设置为 Constant 且 PaddingValueSource 设置为 Property 时有效，该属性可调。其默认值为 0。

例程 3.11.1 是调用系统对象 vision.MedianFilter 对噪声图像进行滤波的程序，其运行结果如图 3.11.1 所示。

例程 3.11.1

```
*********************************************************************
%读入图像
img = im2single(rgb2gray(imread('peppers.png')));
%添加噪声
img = imnoise(img,'salt & pepper');
%显示噪声图像
subplot(1,2,1),imshow(img),title('噪声图像');
%定义系统对象
hmedianfilt = vision.MedianFilter([5 5]);
%对图像进行滤波处理
filtered = step(hmedianfilt,img);
%显示滤波后的图像
subplot(1,2,2),imshow(filtered),title('滤波图像');
*********************************************************************
```

图 3.11.1　例程 3.11.1 的运行结果

3.12　图像角点检测的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision.CornerDetector 可实现对输入灰度图像的角点检测。

vision.CornerDetector 的具体使用方法如下：

功能：对输入的灰度图像进行角点检测。

语法：A＝step(vision.CornerDetector,Img)；

其中：Img 为灰度图像；A 是角点位置矩阵。

属性：

Method：角点检测算法设置。可以将其设置为 Harris corner detection、Minimum eigenvalue 或者 Local intensity comparison。默认值为 Harris corner detection。

Sensitivity：角点检测敏感因子。只有将 Method 属性设置为 Harris corner detection 时，Sensitivity 属性才可调，角点检测敏感因子的取值范围为 0＜k＜0.25，默认值为 0.01。

SmoothingFilterCoefficients：平滑滤波器系数。

CornerLocationOutputPort：该属性的功能为角点位置输出使能。当该属性设置为 true 时，输出角点的位置矩阵。其默认值为 true。

MetricMatrixOutputPort：该属性的功能为角点相应输出使能。当该属性设置为 true 时，输出角点的响应值矩阵。其默认值为 false。CornerLocationOutputPort 属性与 MetricMatrixOutputPort 不能同时设置为 false。

MaximumCornerCount：检测角点数量的最大值。当 CornerLocationOutputPort 属性设置为 true 时，MaximumCornerCount 属性才有效。该属性的默认值为 200。

CornerThreshold：角点判别阈值。只有大于该阈值时才被认为是角点。当 CornerLocationOutputPort 属性设置为 true 时，CornerThreshold 属性才有效。

NeighborhoodSize：邻域大小设置。当 CornerLocationOutputPort 属性设置为 true 时，NeighborhoodSize 才有效。该属性的默认值为[11 11]。

下面通过例程 3.12.1 来具体说明 vision.CornerDetector 的具体使用方法，其运行结果如图 3.12.1 所示。

例程 3.12.1

```
*****************************************************************
%读入图像并转换成单精度型
I = im2single(imread('hongkong.jpg '));
%创建角点检测系统对象
hcornerdet = vision.CornerDetector;
%对输入的图像进行 Harris 角点检测
pts = step(hcornerdet,I);
%设置角点标记
color = [1 0 0]; %[red green blue],将标志点的颜色设置为红色
hdrawmarkers = vision.MarkerInserter('Shape','Circle','Size',10,'BorderColor','Custom',
'CustomBorderColor',color);          %创建用于标记的系统对象
J = step(hdrawmarkers,J,pts);        %在图像上标注角点
imshow(J); title ('角点检测结果');
*****************************************************************
```

图 3.12.1　例程 3.12.1 的运行结果

3.13　图像边缘检测的编程实现

在 MATLAB 中，调用计算机视觉工具箱中的 vision.EdgeDetector 可实现对输入灰度图像的边缘检测。

vision.EdgeDetector 的具体使用方法如下：

功能：对输入的灰度图像进行边缘检测。

语法：A＝step(vision.EdgeDetector,Img)；

其中：Img 为灰度图像；A 是边缘检测后的二值图像。

属性：

Method：通过对该属性进行设置，可以采用不同的边缘检测算法进行检测，可以设置的算法包括 Sobel、Prewitt、Roberts、Canny，默认值为 Sobel。

BinaryImageOutputPort：在采用 Sobel、Prewitt 或 Roberts 边缘检测算子进行边缘检测时，须对 BinaryImageOutputPort 属性进行设置。如果将该属性设置为 true，则边缘

检测后的结果将输出逻辑二值数组。该属性的默认值为 true。

GradientComponentOutputPorts：如果将该属性设置为 true，则输出梯度元素，该属性的默认值为 false。

ThresholdSource：该属性的功能是如何确定阈值。可以将该属性设置为 Auto、Property、Input port，其默认值为 Auto。

Threshold：该属性的功能用于阈值设定。只有当将 ThresholdSource 属性设置为 Property 时，Threshold 属性才可以设置。当采用 Sobel、Prewitt 或 Roberts 算子进行边缘检测时，如果需要对 Threshold 属性进行设置，则需要输入一个具体的数值作为阈值；当采用 Canny 算子进行边缘检测时，则需要输入一个二元素向量作为阈值，向量的一个元素为低阈值，向量的第二个元素为高阈值。当采用 Sobel、Prewitt 或 Roberts 算子进行边缘检测时，Threshold 属性的默认值为 20；当采用 Canny 算子进行边缘检测时，Threshold 属性的默认值为[0.25 0.6]。

ThresholdScaleFactor：阈值缩放因子。

GaussianFilterStandardDeviation：高斯滤波器标准差。当采用 Canny 算子进行边缘检测时，可以对该属性进行设置。

下面通过例程 3.13.1 来具体说明 vision.EdgeDetector 的具体使用方法，其运行结果如图 3.13.1 所示。

例程 3.13.1

```
**********************************************************************
%定义系统对象
hedge = vision.EdgeDetector;                                          %用于边缘检测
hcsc = vision.ColorSpaceConverter('Conversion','RGB to intensity'); %用于颜色空间转换
hidtypeconv = vision.ImageDataTypeConverter('OutputDataType','single');%用于数据转换
%读入图像并将其转换成灰度图像
img = step(hcsc,imread('peppers.png'));
%将其转换成单精度型
img1 = step(hidtypeconv,img);
%进行边缘检测
edges = step(hedge,img1);
%显示边缘检测后的结果
imshow(edges);
**********************************************************************
```

图 3.13.1　例程 3.13.1 的运行结果